LATVIJAS
UNIVERSITĀTE

PROMOCIJAS DARBS

Rīga
2025

UNIVERSITY OF
LATVIA

FACULTY OF SCIENCE
AND TECHNOLOGY

DEPARTMENT OF COMPUTING

Maksims Ivanovs

DEEP LEARNING FOR APPLIED COMPUTER VISION: SOLVING IMAGE UNDERSTANDING TASKS WITH CONVOLUTIONAL NEURAL NETWORKS

DOCTORAL THESIS

Field: Computer Science

Subfield: Artificial Intelligence

Thesis Advisor:

Senior Researcher, Dr. sc. ing. Roberts Kadiķis

Riga 2025

Abstract

This PhD thesis focuses on the application of deep learning methods to solving key image understanding tasks: image classification, object detection, and semantic segmentation.

In the literature review, I provide the background for the experimental work presented in the rest of the thesis and establish that the most promising type of deep neural architecture for image understanding tasks is convolutional neural networks (CNNs). I also discuss the challenges related to the availability of data for training CNNs and outline how transfer learning and synthetic data can be leveraged as potential solutions to mitigate this issue.

The practical part of the thesis focuses on various applications of CNNs to real-world image understanding problems.

In Chapter 2, I describe the use of CNNs for recognising hand-washing movements, with the goal of designing a system to monitor hand hygiene. This work led to the collection of the largest dataset of labelled hand-washing videos to date; subsequently, lightweight CNN models were trained on it. Although the accuracy of these models was not yet sufficient for implementing a real-world monitoring system, the experimental findings represent a significant step toward that goal. Additionally, the experiments with CNNs reported in Chapter 2 highlight the differences between simplified datasets and complex, real-world datasets.

In Chapter 3, I report the study on the use of CNNs for semantic segmentation of street views, an essentials task for the perceptual module of self-driving cars. The results indicate that augmenting real-world datasets with photorealistic synthetic images can significantly improve the accuracy of semantic segmentation, as the best MobileNetV2 and Xception-65 models trained on the augmented data outperformed their counterparts trained solely on real-world images.

In Chapter 4, I present the use of CNN-based object detectors for identifying plastic bottles suitable for being picked up by a robotic arm. The object detectors were trained on synthetic images generated using Blender; by enhancing these images with Generative Adversarial Networks (GANs), the mean average precision of object detection improved compared to the models trained on the original synthetic images.

In Chapter 5, I describe the use of CNNs to classify images of cell tissue with the goal of automating the growth of organs-on-a-chip (OOC). The accuracy of the EfficientNet-B7 and MobileNetV3Large classifiers was improved by training them on datasets augmented with synthetic data generated with the Stable Diffusion large generative model.

The goal of the thesis – to provide efficient solutions for applied image understanding tasks – was achieved for all tasks except the classification experiments on the PSKUS dataset, the most complex and noisy dataset of hand-washing videos. Additionally, the findings contributed to a better understanding of some methodological challenges in deep learning, such as augmenting real-world datasets with synthetic data. The results reported in the thesis have been published in six scientific articles indexed in Elsevier Scopus and/or Web of Science databases and presented at four conferences. The approbation of the results was conducted in seven research projects at the Institute of Electronics and Computer Science (EDI), where this thesis was developed. The results support the four thesis statements that I propose for the defence.

To my family.

Acknowledgements

I would like to express my deepest gratitude to a number of people whose support and encouragement have been pivotal in bringing this thesis to completion.

First and foremost, I would like to thank my thesis advisor, Dr. Roberts Kadiķis, for his invaluable guidance, expertise, and patience. His insights and advice have been essential in shaping my research career in general and this thesis in particular.

I am immensely grateful to my family for their unwavering support throughout this journey: to my wife Ilze, for her love, patience and companionship; to my daughters Laura, Alise, Adriāna, Emīlija, and Karolīna, who fill my life with joy and inspiration; to my mother Ņina, for her belief in my success.

A special thank you goes to my colleagues at the Institute of Electronics and Computer Science (EDI), especially Dr. Modris Greitāns and Dr. Kaspars Ozols for creating a supportive research environment, and Dr. Atis Elsts, Dr. Jānis Judvaitis, and fellow PhD candidates Krišjānis Nesenbergs, Didzis Lapsa, and Anatolijs Zencovs for the opportunities to engage in stimulating discussions.

I am very grateful to all the co-authors of my publications, with whom I have had the pleasure and honour to collaborate. Their contributions have been instrumental in my research.

I would also like to acknowledge my colleagues at the University of Latvia (UL), especially Professor Juris Borzovs, Professor Zane Bičevska, Professor Jānis Zuters, Professor Laila Niedrīte, and Professor Guntis Arnicāns, for providing me with the opportunities to teach at UL and invaluable advice and kind help in various academic and administrative matters. I wish to thank the administrative staff at the UL, particularly Ārija Sproģe, Dace Mileika, and the late Anita Ermuša, for their kind assistance in various organisational matters. I am also very grateful to Professor Jānis Zuters and Associate Professor Edgars Celms for being examiners at my PhD exam, to Professor Uldis Straujums, Professor Guntis Arnicāns, and Professor Juris Borzovs for providing valuable feedback during the public discussion of an earlier version of this thesis, and to Associate Professor Jevgēnijs Vihrovs for kindly sharing with me the LaTeX template of his PhD thesis, thus saving me a lot of time and effort on formatting this work.

I extend my heartfelt thanks to my students at the University of Latvia, who have greatly contributed to my development as an educator, thus also making me a better researcher.

Last but certainly not least, I wish to thank my dogs Pērle, Kara (oh, sweet little Kara!), Lesija, and Bella, and my pet parrots Rons, Lira, Solo, Frodo, and the late Taira for their companionship and the joy they bring into my life. They have been a source of comfort and relief for me during the most stressful times.

This academic journey has been challenging, rewarding, and at times, challenging once more, and I am thankful to have had such a supportive network of family, colleagues, students, and friends. Once again, thank you all!

Contents

List of Abbreviations

A549	Human lung adenocarcinoma alveolar basal epithelial cell line
AGI	Artificial general intelligence
AI	Artificial intelligence
ANN	Artificial neural network
AP	Average precision
AR	Average recall
ASPP	Atrous spatial pyramid pooling
BERT	Bidirectional encoder representations from transformers (DNN model)
Caco-2	colorectal adenocarcinoma epithelial cell line
CamVid	The Cambridge-driving Labeled Video Database
CARLA	Car Learning to Act (driving simulator)
CCM	Cityscapes-CARLA Mixed (dataset)
CFG	Classifier-free guidance
CIoU	Complete Intersection over Union
CLIP	Contrastive Language-Image Pre-training
CmBN	Cross mini-Batch Normalization
CNN	Convolutional neural network
CPU	Central processing unit
CRF	Conditional Random Field
CSPD	Cross-stage partial connections
CycleGAN	Cycle-Consistent Generative Adversarial Network
DIoU-NMS	Distance IoU Non-Maximum Suppression
DL	Deep learning
DNN	Deep neural network
DPM	Deformable Parts Model
DPM++ 2M	Diffusion Probabilistic Model Second-Order Multistep Improved [Sampler]
ECDC	European Centre for Disease Prevention and Control
EDI	Institute of Electronics and Computer Science
FID	Frechet Inception Distance
FLOP	Floating-point operation
FPN	Feature Pyramid Network
FPS	Frames per second
GAN	Generative Adversarial Network
GAP	Global average pooling
GPU	Graphics processing unit
GRU	Gated recurrent unit
GTA	Grand Theft Auto (video game)

HPC	High performance computing
HPMEC	Human pulmonary microvascular endothelial cell line
HSAEC	Human small airway epithelial cell line
HSV	Hue, saturation and value
h-swish	Hard-swish
HUVEC	Human umbilical vein endothelial cell line
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
IoT	Internet-of-Things
IoU	Intersection over union
kNN	K-Nearest Neighbours
LDM	Latent diffusion model
LoRA	Low-rank adaptation
LSTM	Long short-term memory
mAP	Mean average precision
MDR	Multidrug-resistant [bacteria]
METC	Medical Education Technology Centre
MICC-SRI	Media Integration and Communication Center – Semantic Road Inpainting (dataset)
mIoU	Mean intersection over union
MiWRC	Multi-input weighted residual connections
ML	Machine learning
MOT	Moving objects tracking
MRI	Magnetic resonance image
MS COCO	Microsoft Common Objects in Context (dataset)
MiWRC	Multi-input weighted residual connections
NAS	Neural architecture search
NHBE	Normal human bronchial epithelial cell line
NLP	Natural language processing
OOC	Organ-on-a-chip
OS	Operating system
PA	Pixel accuracy
PAN	Path Aggregation Network
PSKUS	Pauls Stradins Clinical University Hospital
R-CNN	Regions with CNN (DNN model)
RGB	Red, green, and blue
RL	Reinforcement learning
ReLU	Rectified linear unit
SiLU	Sigmoid linear unit
SPP	Spatial pyramid pooling
SPPF	Spatial Pyramid Pooling Fusion
SSD	Single Shot Detector (DNN model)
SVM	Support Vector Machine
SYNTHIA	Synthetic collection of Imagery and Annotations (dataset)
TORCS	The Open Racing Car Simulator
TSD	Traffic Signalisation Detection
VAE	Variational Autoencoder
VGG	Visual Geometry Group
WHO	World Health Organisation
YOLO	You Only Look Once (DNN model)

Introduction

From time immemorial, there has been a dream about artificial aides to humans, capable of performing various physical and intellectual tasks on their own. Creations of that kind were already mentioned in the earliest works of European literature: in particular, Nilsson [1] considers 'attendants' that, according to Homer's Iliad, were crafted by the blacksmith god Hephaestus to help him get around, to be one of the earliest fictional artificial intelligence (AI) agents. Folklore and literature abound in other characters brought to life from inanimate matter by human ingenuity, from Galatea to Golem to Čapek's robots, yet it was not until the advent of the modern science and technology that the dreams about AI began to become more tangible. Advances in neurobiological research on the brain structure and function and in psychological investigations of intelligence, development of relevant mathematical methods, research on the theory of computation, and, finally, the invention of the first computers have all contributed to the inception of the AI revolution around the middle of the twentieth century. Since then, AI methods have progressed from the very simple artificial models of neurons [2] to the development of large language models such as GPT-4 [3] capable of assisting humans in a variety of tasks, from debugging code to composing poetry. However, the level of human-like artificial general intelligence (AGI) has not been reached yet, and the pace of AI progress varies significantly across different fields.

As an example of disparities, let us compare and contrast achievements of AI in chess and computer vision tasks. In chess, which has long held the 'traditional status [...] [of] an exemplary demonstration of human intellect' [4], AI-based engines gained the ultimate superiority over the best human players more than two decades ago and have maintained that status since then, demonstrating that computers are extremely good at solving well-defined rule-based tasks. The exemplary opposite case is computer vision: since even simple organisms are capable of perceiving their visual environment and navigating in it, it may appear at first glance (pun intended) that developing human-level computer vision should not pose too great a challenge compared to other areas of AI research.

As Szeliski observes in his overview of the history of computer vision [5], this belief was shared by at least some early pioneers of AI and robotics: in particular, according to a story that has been a part of AI folklore for a long time now, in 1966, Marvin Minsky asked Gerald Jay Sussman, his undergraduate student at MIT at that time, to 'spend the summer linking a camera to a computer and getting the computer to describe what it saw' [6]. However, it turned out that teaching computers to see and understand what they see is a somewhat more complicated task than a summer project for a group of undergraduates, and more than half a century later, research on computer vision in general and image understanding (i.e., 'getting the computer to describe what it saw') in particular still presents many unresolved challenges and exciting tasks. It is also one of the most active research areas in computer science today, as there are many possible applications for automated systems with human or even superhuman capacity for understanding visual scenes.

In this thesis, I[1] am concerned with the three general kinds of image understanding problems, namely:

- image classification: categorising images into one of several predefined classes;
- image segmentation: segmenting an image into different objects and background areas by categorising each its pixel into one of several predefined classes [7];
- object detection: detecting instances of objects in images and categorising each instance into one of several predefined classes [8].

My work on these problems belongs to the domain of applied computer vision, meaning the application of computer vision methods to practical tasks in science and industry. In particular, I use computer vision methods to solve the following real-world image understanding tasks:

- to classify hand-washing movements in the videos filmed in clinical settings for monitoring hand hygiene (Chapter 2);
- to perform semantic segmentation of street views for improving the navigation systems of self-driving cars (Chapter 3);
- to detect graspable bottles in a pile for the bin-picking task carried out by a robotic arm (Chapter 4);
- to classify microscopy images for automated monitoring of growing organs-on-a-chip (OOC) (Chapter 5).

Furthermore, from a broader perspective, my work is also concerned with several general methodological problems in computer vision, that is, the problems that pertain not only to the specific task that I solve in a particular case, but to computer vision research in general. Thus, in Chapter 2, I highlight the need for large datasets when training state-of-the-art image classifiers. When the task is rather specific – such as classifying hand-washing movements – the number of publicly available image datasets is often rather small, and even those may be only partially available or originally labelled in a way that requires relabelling for the task. As a result, in such circumstances, one needs to acquire and label the data first, which is time- and effort-consuming and therefore makes alternative approaches such as the use of synthetic data, which is discussed in the following, particularly topical. Furthermore, as I point out in Chapter 2 characterising the datasets collected for hand-washing movement classification, annotating videos with such continuous and rather complex movements as washing hands inevitably involves discrepancies between the different human annotators of the data, which should be accounted for and resolved in a methodologically sound way. In Chapters 3, 4, and 5 I address the problem of the availability (or lack thereof) of real-world data for training AI models by resorting to the use of synthetic (i.e., artificially generated) data to augment real-world datasets. The use of synthetic data has become increasingly popular in recent years,

[1]The conventional way to refer to oneself in a thesis is to use the third person singular ('the author') or the first person plural ('we'). However, in this thesis, I avoid the former, as I find it unnecessarily convoluted (pun intended); as for 'we', I typically use it when referring to myself and the reader (as in 'as we can see') or when discussing collaborative efforts. Some examples of using such a collaborative 'we' can be found when I describe acquiring and labelling videos of hand-washing episodes in Chapter 2 or images of cell culture in Chapter 5. Otherwise, I prefer the bold, honest and straightforward 'I', as that allows me to underscore that my opinion is my opinion, the work that I did is the work that I did, and my mistakes and omissions are, well, my mistakes and omissions.

not only in computer vision but also in many other AI domains (see [9] for a comprehensive overview) such as natural language processing (NLP) and reinforcement learning (RL), as it allows to reduce the cost, effort, and tedium of collecting and labelling real-world data. On the other hand, it also necessitates dealing with the gap between real-world and synthetic data [10], which, in case of synthetic images, is primarily caused by them being less photorealistic than real-world images. I address the problem of using synthetic data to augment real-world datasets in several ways. Thus, in Chapters 3 and 5, I investigate how the proportion of synthetic data used to augment real-world image datasets affects the accuracy of semantic segmentation and image classification, respectively, that is, whether using more synthetic data always improves performance. I also explore various approaches to generating synthetic data: in the research reported in Chapter 3, I generate images of street views and their segmentation masks with the out-of-the-box open-source driving simulator Cars Learning to Drive (CARLA, [11]); in Chapter 4, I describe how Generative Adversarial Networks (GANs, [12]), potentially powerful but also not-so-easy-to-train and often fragile AI models, can be used to generate synthetic images of plastic bottles; in Chapter 5, I report my work on generating synthetic images of cell culture with the powerful generative AI model Stable Diffusion [13].

In all studies presented in this thesis, I address image understanding challenges by means of deep learning (DL; [14]), that is, using deep neural networks (DNN), which are complex ensembles of artificial neurons – abstract units capable of jointly learning underlying data representations. At an early age of computer vision research, image classification, image segmentation, and object detection problems were dealt with by means of traditional (or classical) computer vision methods [15], which relied on feature extraction and subsequent use of classical machine learning algorithms such as Support Vector Machines (SVM; [16]) and k-Nearest Neighbours (kNN; [17]). However, since 2012, when the Convolutional Neural Network (CNN) AlexNet [18] won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2012, outstripping its competitors – traditional computer vision models – by a large margin on the ImageNet [19] multiclass dataset, the popularity of DNNs has begun to increase steadily, and they are currently regarded as state-of-the-art for most computer vision tasks. Since DNNs have also been successfully applied in other AI domains such as time series analysis, NLP, and content generation, research on general methodological problems in deep learning – such as the already mentioned need for large datasets for training models, quality control issues in large-scale datasets, and the use of synthetic data for augmenting real-world datasets – can potentially contribute to the advancements in these fields as well.

The goal of this thesis is to provide efficient solutions for applied image understanding tasks. **The central premise** of the thesis is that convolutional neural networks can successfully solve the image understanding tasks considered in this work. **The research objectives** and **hypotheses** depend on the specific task and therefore are defined in the chapters of the thesis reporting respective research. **The research methods** that I use in this thesis are those commonly employed in AI and computer vision research: exploration and analysis of relevant literature, data cleaning and preprocessing, synthetic data generation, design and implementation of experiments involving deep neural networks, and analysis and validation of the results of experiments.

As a result of the work presented in this thesis, I propose the following **thesis statements** for defence:

- **Thesis statement one:** In applications of CNNs to real-world image understanding tasks, data availability and quality present greater challenges than model selection and customisation.

- **Thesis statement two:** CNNs that perform well when trained and evaluated on datasets acquired in laboratory conditions may struggle to achieve similar success when trained and evaluated on more complex real-world data.

- **Thesis statement three:** While state-of-the-art CNN-based image classifiers and object detectors with a larger number of parameters typically demonstrate higher accuracy on benchmark datasets than their counterparts with a smaller number of parameters, this accuracy gap narrows or even vanishes when these models are trained and evaluated on smaller, more complex real-world datasets.

- **Thesis statement four:** While augmenting real-world datasets with photorealistic synthetic images is an efficient way to improve the accuracy of CNNs trained on such data, increasing the amount of synthetic data does not directly correlate with improved accuracy on image understanding tasks.

The above thesis statements are primarily grounded in the results found in the following chapters: thesis statement one – in Chapters 2, 3, 4, and 5; thesis statement two – in Chapter 2; thesis statement three – in Chapters 4 and 5; thesis statement four – in Chapters 3 and 5.

Research findings reported in this thesis have been published in the following scholarly articles indexed in Elsevier Scopus and/or Web of Science databases:

1. M. Lulla, A. Rutkovskis, A. Slavinska, A. Vilde, A. Gromova, **M. Ivanovs**, A. Skadins, R. Kadikis, and A. Elsts, "Hand-washing video dataset annotated according to the World Health Organization's hand-washing guidelines," *Data*, vol. 6, no. 4:38, 2021.

   The length of the publication: 6 pages.

   My approximate contribution: $10 - 15\%$.

2. A. Elsts, **M. Ivanovs**, R. Kadikis, and O. Sabelnikovs, "CNN for hand washing movement classification: What matters more – the approach or the dataset?," in *2022 Eleventh International Conference on Image Processing Theory, Tools and Applications (IPTA)*, pp. 1–6, IEEE, 2022.

   The length of the publication: 6 pages.

   My approximate contribution: $30 - 40\%$.

3. **M. Ivanovs**, K. Ozols, A. Dobrajs, and R. Kadikis, "Improving semantic segmentation of urban scenes for self-driving cars with synthetic images," *Sensors*, vol. 22, no. 6: 2252, 2022.

   The length of the publication: 13 pages.

   My approximate contribution: 90%.

4. D. Duplevska, **M. Ivanovs**, J. Arents, and R. Kadikis, "Sim2Real image translation to improve a synthetic dataset for a bin picking task," in *2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1–7, IEEE, 2022.

   The length of the publication: 7 pages.

   My approximate contribution: $30 - 40\%$.

5. **M. Ivanovs**, L. Leja, K. Zviedris, R. Rimsa, K. Narbute, V. Movcana, F. Rumnieks, A. Strods, K. Gillois, G. Mozolevskis, A. Abols, and R. Kadikis, "Synthetic image generation with a fine-tuned latent diffusion model for organ on chip cell image classification," in *2023 Signal Processing: Algorithms, Architectures, Arrangements, and Applications (SPA)*, pp. 1–6, IEEE, 2023.

   The length of the publication: 6 pages.

   My approximate contribution: 90%.

6. V. Movčana, A. Strods, K. Narbute, F. Rūmnieks, R. Rimša, G. Mozoļevskis, **M. Ivanovs**, R. Kadikis, K. Zviedris, L. Leja, A. Zujeva, T. Laimiņa, and A. Abols, "Organ-On-A-Chip (OOC) Image Dataset for Machine Learning and Tissue Model Evaluation," *Data*, vol. 9, no. 2:28, 2024.

   The length of the publication: 10 pages.

   My approximate contribution: $15 - 20\%$.

Research findings included in this thesis have also been reported in the following scholarly publications that are not indexed in Elsevier Scopus or Web of Science databases:

1. **M. Ivanovs**, R. Kadikis, M. Lulla, A. Rutkovskis, and A. Elsts, "Automated quality assessment of hand washing using deep learning," *arXiv:2011.11383*, 2020.

   The length of the publication: 8 pages.

   My approximate contribution: $\geq 50\%$.

2. O. Zemlanuhina, M. Lulla, A. Rutkovskis, A. Slavinska, A. Vilde, A. Melbarde-Kelmere, A. Elsts, **M. Ivanov**[2], and O. Sabelnikovs, "Influence of different types of real-time feedback on hand washing quality assessed with neural networks/simulated neural networks," in *SHS Web of Conferences*, vol. 131, p. 02008, EDP Sciences, 2022.

   The length of the publication: 13 pages.

   My approximate contribution: 10%.

I have presented research findings reported in this thesis at the following conferences:

1. IEEE International Conference on Image Processing Theory, Tools & Applications – IPTA 2022, Salzburg, Austria, 2022.

   Presentation *CNN for Hand Washing Movement Classification: What Matters More — the Approach or the Dataset?*

2. 27$^{\text{th}}$ IEEE International Conference on Emerging Technologies and Factory Automation – EFTA 2022, Stuttgart, Germany.

   Presentation *Sim2Real Image Translation to Improve a Synthetic Dataset for a Bin Picking Task*.

3. SEMICON Europa 2022 Conference, Munich, Germany, 2022.

   Presentation *Synthetic Data for Robotics: Opportunities and Challenges*.

[2]Note the misspelled surname: *Ivanov* rather than *Ivanovs*.

4. 26th IEEE Signal Processing: Algorithms, Architectures, Arrangements and Applications Conference – SPA 2023, Poznan, Poland, 2023.

   Presentation *Synthetic Image Generation With a Fine-Tuned Latent Diffusion Model for Organ on Chip Cell Image Classification.*

In addition to the above publications, during my doctoral studies, I have also contributed to the following publications that are not included in this thesis:

1. J. Judvaitis, A. Mednis, V. Abolins, A. Skadins, D. Lapsa, R. Rava, **M. Ivanovs**, and K. Nesenbergs, "Classification of actual sensor network deployments in research studies from 2013 to 2017," *Data*, vol. 5, no. 4:93, 2020.

2. A. Skadins, **M. Ivanovs**, R. Rava, and K. Nesenbergs, "Edge pre-processing of traffic surveillance video for bandwidth and privacy optimization in smart cities," in *2020 17th Biennial Baltic Electronics Conference (BEC)*, pp. 1–6, IEEE, 2020.

3. R. Rava, **M. Ivanovs**, A. Skadins, and K. Nesenbergs, "World coordinate virtual traffic cameras: Edge-based transformation and merging of multiple surveillance video sources," in *2020 7th International Conference on Soft Computing & Machine Intelligence (ISCMI)*, pp. 233–236, IEEE, 2020.

4. **M. Ivanovs**, R. Kadikis, and K. Ozols, "Perturbation-based methods for explaining deep neural networks: A survey," *Pattern Recognition Letters*, vol. 150, pp. 228–234, 2021.

5. **M. Ivanovs**, B. Banga, V. Abolins, and K. Nesenbergs, "Methods for explaining CNN-based BCI: A survey of recent applications," in *2022 IEEE 16th International Scientific Conference on Informatics (Informatics)*, pp. 137–141, IEEE, 2022.

6. B. Cugmas, E. Štruc, I. Berzina, M. Tamosiunas, L. Goldberga, T. Olivry, K. Zviedris, R. Kadikis, **M. Ivanovs**, M. Bürmen, and P. Naglič, "Automated classification of pollens relevant to veterinary medicine," in *2024 IEEE 14th International Conference Nanomaterials: Applications & Properties (NAP)*, pp. 1–4, IEEE, 2024.

7. B. Cugmas, E. Štruc, M. Tamosiunas, L. Goldberga, I. Berzina, R. Kadikis, **M. Ivanovs**, S. Warshaneyan, and P. Naglič, "Comparison of two fixation methods in automated pollen classification on whole slide images," in *Latin America Optics and Photonics Conference*, Optica Publishing Group, 2024.

I have presented the results of these studies at the following conferences:

1. Latvijas Ārstu kongress: Zinātniski praktiskā sesija "Lielie dati medicīnā", Riga, Latvia, 2022.

   Presentation *Mākslīgais intelekts un attēlu apstrāde medicīnas pielietojumiem.*

2. IEEE 16th International Scientific Conference on Informatics – Informatics 2022, Poprad, Slovakia, 2022.

   Presentation *Methods for Explaining CNN-Based BCI: A Survey of Recent Applications.*

I conducted and approbated my research for this thesis at the Institute of Electronics and Computer Science (EDI – *Elektronikas un datorzinātņu institūts*). The research was part of several scientific projects at EDI and was financially supported by their funding. The following is the list of these projects:

1. *Programmable Systems for Intelligence in Automobiles – PRYSTINE* (Horizon 2020 ECSEL Joint Undertaking funding under grant agreement 783190).

2. *Efficient module for automatic detection of people and vehicles using video surveillance cameras – VAPI* (ERDF project No. 1.2.1.1/18/A/006 research No. 1.5).

3. *Automated hand washing quality control and hand washing quality evaluation system with real-time feedback – Handwash* (project No. lzp-2020/2-0309).

4. *Integration of reliable technologies for protection against Covid-19 in healthcare and high risk areas – COV-CLEAN* (project No. VPP-COVID-2020/1-004).

5. *Intelligent Motion Control under Industry 4.E – IMOCO4.E* (Horizon 2020 ECSEL Joint Undertaking funding under grant agreement 101007311).

6. *AI-Improved Organ on Chip Cultivation for Personalised Medicine – AImOOC* (contract with Central Finance and Contracting Agency of Republic of Latvia no. 1.1.1.1/21/A/079; the project was cofinanced by REACT-EU funding for mitigating the consequences of the pandemic crisis).

7. *Holographic microscopy- and artificial intelligence-based digital pathology for the next generation of cytology in veterinary medicine – VetCyto* (project No. lzp-2023/1-0220).

The thesis consists of five chapters, followed by a conclusion and bibliography. In Chapter 1, I provide the background of my work, discussing the fields of computer vision and deep learning and their intersection. Chapter 2 focuses on the use of CNNs for hand-washing movement classification; I present the datasets collected in the course of research as well as the results of training and evaluating CNN models on them. Chapter 3 addresses the use of CNNs for semantic segmentation of urban street views. In Chapter 4, I detail the use of CNN-based object detectors for a bin-picking task carried out by a robotic arm. In Chapter 5, I report on the application of CNNs to classifying cell culture images with the goal of automating the process of growing organs-on-a-chip. Finally, in Conclusion, I revisit the key findings of the research presented in this thesis, reflect on the challenges encountered throughout the work, substantiate the four thesis statements I propose for defence, and suggest directions for future research aimed at advancing computer vision methods and overcoming the current limitations in image understanding tasks.

Chapter 1

Background

The goal of this chapter is to outline the relevant background for research on image understanding within the context of applied computer vision. The structure of the chapter is as follows. In Section 1.1, I define computer vision – first descriptively and informally, then more rigorously and formally – and discuss its scope as well as explore some of its highlights. In Section 1.2, I narrow the focus of the discussion to the subfield of computer vision – image understanding – and discuss three major tasks in that subfield that are particularly relevant to the work presented in this thesis: image classification, image segmentation, and object detection. Furthermore, I present the key metrics for evaluating the performance of algorithms and systems on these tasks. In Section 1.3, I examine methods for solving image understanding tasks. I briefly introduce classical methods, but primarily focus on deep learning-based approaches, as these are central to the research in this thesis. In Section 1.4, I explore the importance of datasets for training machine learning (ML) models and the challenges related to their availability. I also discuss how transfer learning, data augmentation, and synthetic data can help mitigate the problem of the scarcity of training data. Finally, in Section 1.5, I offer some concluding remarks.

1.1 Computer vision: definition, scope, and highlights

Providing a comprehensive overview of a field as vast, complex, multifaceted, and rapidly evolving as computer vision is a truly daunting task; attempting such an overview in a single chapter of a PhD thesis would be overly ambitious and likely unfeasible. Therefore, instead of trying to survey the field of computer vision in its entirety, I limit myself to the more modest task of defining what computer vision is and highlighting the aspects most relevant to this thesis. For a more detailed treatment of the subject, I refer readers to the fundamental works by Jähne et al. [20], Szeliski [5], Hartley and Zisserman [21], and Forsyth and Ponce [22] as well as more recent, though less comprehensive, surveys of the state of the art in computer vision by O'Mahony et al. [15] and Feng et al. [23].

Regardless of the scope of the discussion, a rigorous approach dictates that I must define the term 'computer vision' in a manner that is both comprehensive and formal. However, to motivate my choice of the definition, I first consider the notion of computer vision more informally. A suitable starting point is Minsky's famous suggestion[1] for a research project that I already mentioned in the Introduction to this thesis: to link a camera to a computer

[1] As a caveat, one should not oversimplify Minsky's proposal, which in fact was quite elaborated – see the memo of the 'Summer Vision Project' drafted by Seymour Papert [24]. All in all, I mention it here as a convenient way to initiate the discussion rather than to belittle Minsky's ambition.

and get the computer to describe what it is seeing [6]. Indeed, it appears that this is largely what researchers and software engineers in computer vision do: with the help of computers, they transform input from visual sensors into some meaningful representation of the original visual scene. While such a setup is ubiquitous, it is far from trivial, as there are many aspects to consider, such as:

- *What kind of input does the system receive?* As human thinking tends to be anthropocentric, the first association when considering input to a camera linked to a computer might be that it captures a typical visual environment, that is, an RGB video stream from the part of the electromagnetic spectrum visible to humans, ranging from ≈400 nm to ≈700 nm. However, it does not seem reasonable to exclude other segments of electromagnetic spectrum, such as ultraviolet (UV) light, infrared light, or X-rays, from the scope of computer vision. While these wavelengths are not visible to the human eye per se, the same methods as the ones used for processing and analysing visible light can often be applied to them.

- *Is the input treated as a continuous stream or as divided into discrete units?* While humans perceive their visual environment as a continuous flow of information, the temporal dimension of such input makes it more challenging to process and analyse in computer vision. Circumventing this challenge, much of computer vision research focuses on images rather than video: there are more image datasets than video datasets for training and benchmarking models, and image generation, processing, and analysis methods are more developed. While the focus on images is practical for these reasons, it makes computer vision markedly different from biological vision, where treating static snapshots of visual scenes as the primary units of analysis may seem artificial.

- *What kind of output do we expect from the system?* The output of the human visual system is, essentially, *seeing* the world, and understanding what 'seeing' actually means has kept philosophers and cognitive scientists occupied for a long time. In contrast, the typical output of computer vision systems has historically often been discrete and rather simple, such as matching an entire image to a single label. The same as with the previous point, this approach has been customary due to practical considerations, as simple output makes it easier to design, benchmark and analyse computer vision systems. However, it also makes these systems markedly different from their biological counterparts. Conversely, developing systems that produce more complex output, such as elaborate and complex descriptions of visual scenes, brings computer vision closer to other fields of AI such as NLP.

- *How do we evaluate the quality of the output?* Simple evaluation metrics (e.g., the number of correctly classified images) are easier to implement, yet they can oversimplify tasks compared to the complexity of real-world visual environments.

- *Where (if at all) do we draw the line between vision and cognition?* This question arguably arises whenever understanding is required in a computer vision task. Furthermore, similar to biological visual systems, which incorporate acquired experience into their function rather than operate as blank slates, their digital counterparts also integrate world knowledge, and as a consequence, vision and cognition become intrinsically intertwined.

- *To what extent are the inner workings of the system comprehensible and transparent?* The human visual system is highly efficient but notoriously complex, and only partly

understood. While some of the difficulty in studying it arises from the impossibility of conducting invasive experiments on humans, it is also true that even when such experiments are possible, e.g., in case of laboratory animals, scientific endeavours have so far provided only partial understanding of how vision functions, and the brain, to a large extent, still remains a 'black box'. In case of computer vision system, the best results on many computer vision tasks have been achieved with deep neural networks, which in their essence are also 'black boxes': while it is easier to pry them open (e.g. to extract the state of a particular artificial neuron or its response to a stimulus) than their biological counterparts, analysing these systems is still challenging due to the large number of parameters in a typical DNN model. As a result, computer vision systems often match their biological counterparts in terms of their opacity.

The questions posed above are best treated not as requiring immediate and straightforward answers, but rather as signposts guiding research paths in computer vision. At the same time, they lead me to conclude the following:

- Computer vision deals not only with input in the visible light spectrum, but also with wavelengths invisible to humans, such as ultraviolet light, infrared light, and X-rays.

- The three main components of a computer vision system are input (including input formation), information processing algorithms, and output. These three components of computer vision are interrelated: thus, as Jähne et al. [25] observe, in their work, they regard computer vision 'from image formation to measuring, recognition, or reacting' as an integral process. Overall, this is a reasonable approach: for instance, the type of input data may affect the choice of information processing algorithms, and the desired output format may influence the choice of the visual sensors for acquiring input. However, I would also like to note that in practice, it is quite common in computer vision to focus on some particular stage(s) of the said integral process at the expense of taking other stages for granted. For instance, a researcher aiming to improve the performance of a particular algorithm on a specific dataset is unlikely to spend too much (if any at all) time pondering the physics of image formation, as it bears no particular relevance to their immediate task. Such a reductionist approach certainly has its advantages, but it also may result in oversights when some aspects of computer vision are actually important but are not accounted for in a particular study or setup.

- The interrelation of different aspects of computer vision makes it a multidisciplinary research field that brings together physics, mathematics, computer science, biology, physiology, cognitive sciences, and other disciplines.

- The boundaries between vision and other perceptual and cognitive processes, such as language processing, are not clear-cut but rather blurry. This is because computer vision tasks often involve at least some level of understanding of the visual world, and the output of a computer vision system is frequently expressed in the form of a verbal description. Therefore, the said boundaries should be drawn cautiously, if at all, depending on the specific research problem.

- While evaluation metrics are necessary and useful for comparing and improving computer vision algorithms, they should be taken critically[2], as they tend to oversimplify the challenges compared to those faced by visual systems operating in real-world environments.

[2]Cf. Goodhart's Law, which states that a measure ceases to be useful when it becomes a target itself.

Although the above observations do not provide a comprehensive analysis of computer vision but rather highlight some of its noteworthy characteristics, I believe that they convincingly demonstrate the complexity and scope of computer vision as a field of research and engineering, and, by extension, show that producing a rigorous and sufficiently informative academic definition of computer vision is not straightforward. To demonstrate the obstacles that one may encounter in a quest for such a definition, let us consider how computer vision is defined in a reputable source for the general audience, such as The Encyclopædia Britannica. According to Britannica, it is '[a] field of artificial intelligence in which programs attempt to identify objects represented in digitised images provided by cameras, thus enabling computers to "see"' [26]. While this definition is essentially correct, it includes several contentious points, namely:

- *Computer vision is treated as a subfield of artificial intelligence.* While most modern computer vision methods are indeed rooted in AI, there are also algorithms – for instance, for optical flow estimation [27, 28], colour-based segmentation [29, 30], and edge detection [31] – that are based on explicit programming and predefined mathematical (especially geometrical) principles.

- *Objects are mentioned as the only type of entities that computer vision is concerned with.* To provide just a single yet convincing counterexample, segmentation algorithms aim to identify not only objects but also various background elements (e.g., sky, terrain, water surface, or wall) in images.

- *Images are considered the primary data source for computer vision systems.* However, as previously mentioned, images are artificial entities, and working with them differs from processing the continuous visual stream characteristic of the real world. Moreover, an excessive focus on images as the primary units of analysis may, in my opinion, hinder the overarching goal of computer vision to enable computers to truly *see*.

In my search for a better and more precise definition of computer vision, I have found and henceforth adopt the one given in the seminal work by Jähne et al. [25], which defines computer vision as 'the host of techniques to acquire, process, analyze, and understand complex higher-dimensional [visual] data from our environment'. Note that I have added the term '[visual]' in square brackets to clarify the meaning, as the original definition is given in a computer vision textbook, where the context makes the phrase 'complex higher-dimensional data' clear, whereas for standalone use, including 'visual' may be helpful.

As the last note here on the definition and scope of computer vision, I would like to differentiate it from digital image processing. While some (admittedly, rather few) works, e.g., O'Mahony et al. [15], treat these terms as equivalent, I follow the more prevalent approach in this thesis by distinguishing them. Specifically, I adopt the approach of Pitas et al. [32] and maintain that digital image processing refers to low-level operations with images such as image enhancement (e.g., adjusting brightness or contrast) or colour image processing. In contrast, the broader term 'computer vision' (and the similar but less frequently used 'digital image analysis') also includes higher-level operations with visual data such as object detection and image segmentation. Depending on usage, computer vision may subsume digital image processing, but not the other way around, as the latter is substantially narrower in scope.

1.2 Image understanding: definition, main tasks, metrics

As established in the previous section, computer vision spans multiple disciplines, from physics to psychology to philosophy, and is concerned with a wide range of research problems. I also argued that it is a common practice in the field of computer vision to focus on some of its aspects rather than trying to encompass the full breadth of research questions in it. As this thesis also follows the common practice of narrowing the scope of research, I will introduce a narrower term within the domain of computer vision to refer to the research problems that my work is primarily concerned with. For that purpose, I henceforth use the term **'image understanding'**, which has been used in a number of well-regarded studies, e.g. [32], [33], [34], [35], though it is notably less prevalent in the literature than the term 'computer vision' (cf. Figure 1.1).

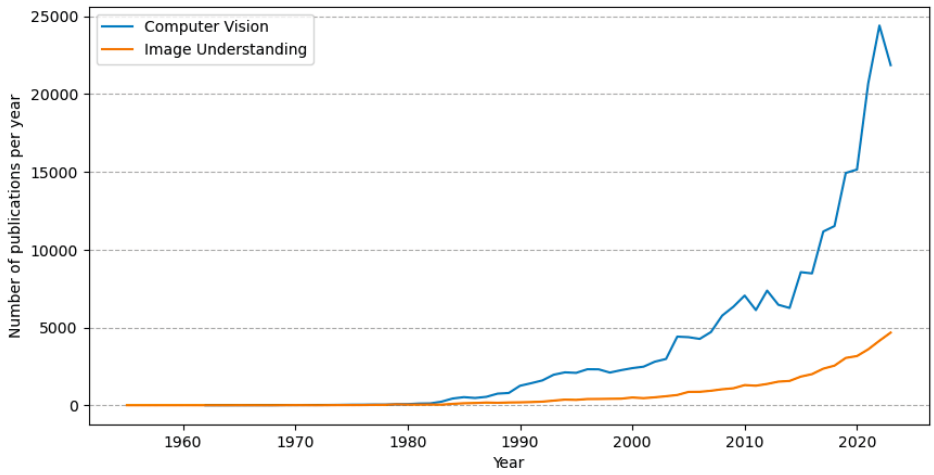


Figure 1.1: Annual trends in Scopus-indexed publications on computer vision and image understanding (1955-2023). The graph shows the number of publications per year, filtered by the search terms `computer AND vision` and `image AND understanding` within the 'Computer Science' and 'Engineering' subject areas. The data were retrieved on 20 January 2024.

I use the term 'image understanding' as defined by Zhang [36], that is, as a suite of methods and techniques that 'attempt to interpret the meaning of image at a high level to provide semantic information closely related to human thinking, and help further to make decisions and to guide the actions according to the understanding of scenes.' Some of the primary tasks within image understanding are image classification, image segmentation, object detection, object localisation, image captioning, optical character recognition, and pose estimation. My research that forms the foundation of the present thesis is concerned with the first three of these tasks, that is, image classification, image segmentation, and object detection. Therefore, I dedicate the remainder of this section to defining and briefly discussing these tasks. However, prior to that, I will address potential objections to my choice of the term 'image understanding' to refer to them.

First, I address the possible question of why I use the term 'image understanding' instead of directly referring to image classification, image segmentation, and object detection. The practical reason is that 'image understanding' is more concise and therefore more convenient, avoiding the need to list all three tasks each time I refer to them collectively. Another, more general reason is my belief that research on image classification, image segmentation,

object detection, and other similar tasks should not only aim at improving the performance of computer systems on these specific tasks, but also contribute to the overarching goal of enabling computers to *understand* images (and visual data in general) at a level comparable to human understanding of our visual environment. While the contributions of my work in this thesis to the advancement of this goal are incremental rather than fundamental, and while the convolutional neural networks that I employed for solving practical computer vision tasks are far from actually *understanding* their input in the way humans do, I believe that it is still valuable to use the term 'image understanding' so that, figuratively speaking, it would remind us about the forest looming behind the trees.

Second, I address the potential question of why I prefer the term 'image understanding' over another umbrella term for the three tasks in question, namely, 'classification tasks'. Indeed, the latter term is sometimes used to jointly refer to image classification, image segmentation, and object detection. For image classification, this is straightforward, as the task inherently involves classification; image segmentation can be viewed as a classification task at the pixel level; finally, object detection is often carried out using a two-stage process: first, object localisation, followed by object classification. However, there are several reasons why I find the term 'classification tasks' less appropriate in the given context. To begin with, this term can easily be misunderstood as referring solely to image classification – just one out of the three tasks it aims to encompass. Furthermore, the term 'classification tasks' does not emphasise the ultimate goal of understanding the semantics of the visual world as effectively as the term 'image understanding' does. Finally, while many state-of-the-art object detection methods consist of two stages and do involve classification in the second stage, not all object detection methods rely on classification – for instance, objects can be detected by detecting changes (cf. e.g. [37]). Therefore, categorising all object detection methods as a subtype of classification task would be an overgeneralisation. As a consequence, using the term 'classification tasks' for a group of tasks that includes object detection would arguably be rather imprecise.

Having defined image understanding and justified my use of the term, I now turn to the three image understanding tasks central to this thesis: image classification, image segmentation, and object detection (see examples in Figure 1.2). In addition, I will introduce the common evaluation metrics for these tasks.

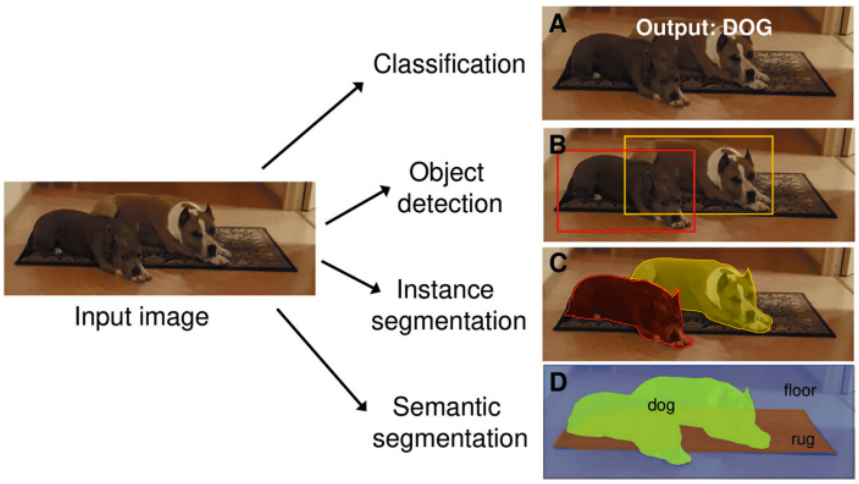


Figure 1.2: Examples of image classification, object detection, instance segmentation, and semantic segmentation. Reproduced from [38].

Image classification, in its essence, is a labelling procedure [39]: a classifier labels a given image I as belonging to a single class C_i, which is an element of a fixed set of considered classes $C = \{C_1, C_2, C_3, ..., C_N\}$. Several variations can arise from this general definition. Thus, if $|C| = 2$, the classification is said to be *binary*; to distinguish the situation with $|C| > 2$ from binary classification, it is commonly called *multiclass* classification. Furthermore, in some situations, it is more appropriate to assign two or more labels to an image rather just a single label, as the image may feature more than one object in it; in such a case, classification becomes *multilabel*. As Jähne et al. [25] note, other variations in the general labelling scheme can be implemented depending on the task. For instance, we might assign a single label 'cat' to all images of cats, or classify them by breeds. Similarly, objects of the same shape but different colours might be classified as belonging to either a single class or different classes, depending on the specific criteria.

Image classification is often considered the most basic task in the field of image understanding – for instance, it is often said that the 'Hello World' task of deep learning for computer vision is training a neural network to classify hand-written digits in the MNIST dataset [40]. However, as Rawat and Wang [41] point out, image classification still poses a number of challenges for automated systems; examples of such challenges include variability in the appearance of objects depending on the viewpoint (e.g., the rear of the car looks completely different from its front) and the high variability of objects within the same class (e.g., within the broad class of 'plants')[42]. Overcoming these challenges and limitations is crucial for advancing image understanding, since image classification serves as the foundation for several other tasks, particularly semantic segmentation and object detection.

The main metrics for image classification are accuracy, precision, recall, and F_1 score. Among those, *accuracy* is the most straightforward and commonly used metric. Given that true positives are denoted as TP, true negatives as TM, false positives as FP, and false negatives as FN,

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}. \tag{1.1}$$

Precision quantifies the ability of the classifier to identify true positives while producing a low rate of false positives. The formula for calculating precision is as follows:

$$\text{Precision} = \frac{TP}{TP + FP}. \tag{1.2}$$

Recall (also called *sensitivity*), which is often reported alongside precision, quantifies the ability of the classifier to identify true positives while producing a low rate of false negatives. Therefore, it may be interpreted as a measure of the completeness of a classifier. The formula for calculating recall is as follows:

$$\text{Recall} = \frac{TP}{TP + FN}. \tag{1.3}$$

Finally, F_1 *score*, a weighted average of precision and recall, is calculated as

$$F_1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2TP}{2TP + FP + FN}. \tag{1.4}$$

F_1 score is often more informative and less misleading than accuracy, particularly in case

of datasets with imbalanced classes. This is because it ensures that both the precision and recall of the model are accounted for. In contrast, accuracy can indeed be deceptive in such cases: for instance, a classifier may achieve an ostensibly impressive accuracy of 90% on a dataset where 90% of the data belongs to the largest class, simply by labelling every test input as belonging to that class. However, the outcome of such 'learning' can hardly be considered a success, and an resulting 'naive' classifier[3] is essentially useless. That said, the F_1 score also has the drawback of being less intuitive than accuracy.

Object detection has a wide range of real-world applications, including autonomous driving, intelligent video surveillance, robotics, and security [43]. It is a more complex task than image classification, because an object detector needs to both assign each object in the image a label C_i from the fixed set of considered classes $C = \{C_1, C_2, C_3, ..., C_N\}$, and identify the positions of these objects. In other words, while an image classifier aims to answer the question *What object is there?*, an object detector, as Zou et al. [44] put it, needs to answer the question *What objects are where?*

Localisation of objects is typically done by drawing a bounding box around each of them. A bounding box is a rectangular[4] frame that fully encloses the object. As Padilla et al. [46] point out, the most common way to represent a bounding box is by providing its top left and bottom right coordinates, i.e, $(x_{init}, y_{init}, x_{end}, y_{end})$. However, they also note that the one of the most popular families of object detection algorithms, YOLO detectors [47], uses a different representation. In YOLO, a bounding box is defined by providing the x and y coordinates of its center and its width and height as:

$$\left(\frac{x_{\text{center}}}{\text{image width}}, \frac{y_{\text{center}}}{\text{image height}}, \frac{\text{box width}}{\text{image width}}, \frac{\text{box height}}{\text{image height}}\right) \quad [46].$$

According to a comprehensive survey of object detection metrics by Padilla et al. [48], the primary metrics for this task include precision, recall, and several metrics derived from them, such as average precision, mean average precision, and average recall. All of these metrics are generally based on measuring how closely predicted bounding boxes B_p correspond to ground truth bounding boxes B_{gt}. The overlap between the former and the latter is calculated using the intersection over union (IoU) measurement:

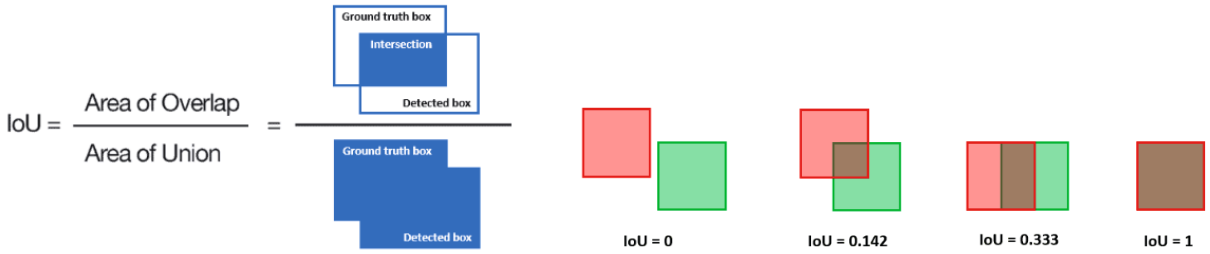
$$\text{IoU} = \frac{\text{area}(B_p \cap B_{gt})}{\text{area}(B_p \cup B_{gt})} \tag{1.5}$$

The visual representation of calculating IoU is shown in Figure 1.3 (a). Figure 1.3 (b) illustrates different IoU values, ranging from no overlap at all with IoU = 0 (left) to a perfect overlap with IoU = 1 (right).

For the calculation of *precision* and *recall* in the context of object detection, the same formulas are used as for image classification, i.e., the formulas given by Equations 1.2 and 1.3, respectively. However, the key difference is that in case of object detection, a detected object is treated as a true positive or a false positive depending on whether it meets a predefined IoU threshold. For instance, if the IoU threshold is set to 0.5, detected objects with IoU $\geq$ 0.5 are counted as true positives, those with lower IoU values are treated as false positives,

[3]Here and in the following, a 'naive' classifier refers to a model that operates in a primitive fashion, either assigning classes at random, or, in the case of imbalanced datasets, always assigning the majority class. This should not be confused with the naive Bayes classifier – a well-established machine learning method.

[4]Polygons are also used in some cases (see e.g. [45]), as they may capture the shapes of real objects better than rectangles.



(a) (b)

Figure 1.3: Illustration of the intersection over union (IoU): (a) visualisation of the IoU calculation; (b) visual examples of different IoU values. Adapted from [49].

and any undetected objects are counted as false negatives. The *average precision (AP)* and *average recall (AR)* for class C_i in a dataset consisting of n images are calculated as follows:

$$AP_{C_i} = \frac{1}{n} \sum_{j=1}^{n} P_{C_{ij}} \tag{1.6}$$

$$AR_{C_i} = \frac{1}{n} \sum_{j=1}^{n} R_{C_{ij}}. \tag{1.7}$$

Since datasets typically contain multiple classes $C_1, C_2, C_3, ...C_N$, the average precision values for each class can be summarised by calculating the *mean Average Precision (mAP)* across all classes:

$$mAP = \frac{1}{N} \sum_{i=1}^{N} AP_{C_i}. \tag{1.8}$$

While the mAP is often calculated with a fixed IoU threshold set to 0.5, another common approach is to compute several mAP values with IoU thresholds ranging from 0.5 to 0.95 in steps of 0.05, and then average these values. This is expressed as:

$$\text{mAP}_{\text{avg}} = \frac{1}{|T|} \sum_{t \in T} \frac{1}{N} \sum_{i=1}^{N} \text{AP}_{C_i}^{t} \tag{1.9}$$

where $|T|$ is the number of IoU thresholds $T = \{0.5, 0.55, 0.6, \ldots, 0.95\}$, N is the number of classes, and $\text{AP}_{C_i}^{t}$ is the average precision for class C_i at IoU threshold t. For clarity, the resulting value, mAP_{avg}, is often denoted as mAP(avg for IoU $\in [0.5 : 0.05 : 0.95]$).

Image segmentation has applications across various domains, from medical image analysis to self-driving cars. The goal of segmentation is to partition an input image into multiple segments (i.e., continuous groups of pixels), sometimes referred to as 'superpixels'. Naturally, the partitioning should be meaningful in some sense rather than random. Thus, one of the main types of image segmentation, *semantic segmentation*, has the goal of labelling each pixel of the resulting segments with a single class label C_i belonging to a fixed set of considered classes $\{C_1, C_2, C_3, ...C_N\}$. Therefore, semantic segmentation is essentially multi-class classification at the pixel level; as it is performed for each pixel of an input image, it is generally considered a more challenging task than image classification [50]. An even more

challenging variant is *instance segmentation*, which distinguishes not only between different classes but also between different instances of the same class. For instance, for the example provided in Figure 1.2, a semantic segmentation algorithm will assign the label `dog` to some pixel (Figure 1.2, D), but for instance segmentation, more detailed labelling is needed to indicate whether the algorithm considers the pixel in question as belonging to `dog1` or `dog2`. (Figure 1.2, C).

The most common metrics for evaluating image segmentation are pixel accuracy, mean pixel accuracy, IoU, and mIoU.

Pixel accuracy (PA) is the ratio of correctly classified over the total number of pixels. For N classes, it is calculated as follows:

$$PA = \frac{\sum_{i=1}^{N} p_{ii}}{\sum_{i=1}^{N} \sum_{j=1}^{N} p_{ij}}, \tag{1.10}$$

where p_{ij} is the number of pixels of class i predicted as belonging to the class j (where $i = j$ or $i \neq j$).

Mean Pixel accuracy (mPA) is the average PA computed across all classes in the dataset, that is,

$$mPA = \frac{1}{N} \sum_{i=1}^{N} \frac{p_{ii}}{\sum_{j=1}^{N} p_{ij}}. \tag{1.11}$$

While both PA and mPA are simple and intuitive metrics, they can misrepresent the performance of a segmentation algorithm when dealing with datasets containing imbalanced classes. In particular, an algorithm may achieve high values for these metrics by accurately predicting the most common class while performing poorly on less represented classes. Furthermore, neither PA nor mPA takes into consideration the localisation of the predicted pixels, making these metrics less robust. Therefore, they appear less frequently in the literature than IoU and mIoU, which are the leading metrics for evaluating semantic segmentation algorithms. IoU for segmentation is calculated similarly to IoU for object detection, with the key difference being that, in segmentation, the intersection between the ground truth and the prediction is computed for segmentation masks rather than bounding boxes. The formula for IoU for semantic segmentation with N classes is:

$$IoU = \frac{\sum_{i=1}^{N} p_{ii}}{\sum_{i=1}^{N} \sum_{j=1}^{N} p_{ij} + \sum_{i=1}^{N} \sum_{j=1}^{N} p_{ji} - \sum_{i=1}^{N} p_{ii}} \quad [51]. \tag{1.12}$$

mIoU, which averages the IoU across all classes in the dataset, is calculated as follows:

$$mIoU = \frac{1}{N} \sum_{i=1}^{N} \frac{p_{ii}}{\sum_{j=1}^{N} p_{ij} + \sum_{j=1}^{N} p_{ji} - p_{ii}} \quad [51], \tag{1.13}$$

where, as mentioned above, p_{ij} is the number of pixels of class i predicted as belonging to class j (where $i = j$ or $i \neq j$).

1.3 Methods for solving image understanding tasks

Having outlined the main image understanding tasks and the metrics for evaluating performance of computer systems on these tasks, I proceed with a brief overview of the methods

for solving these tasks. Broadly speaking, these methods can be divided into two principal categories: classical methods, and deep learning-based methods.

1.3.1 Classical methods

The defining characteristic of **classical methods** for image understanding is that they are fully or partially based on explicitly programmed algorithms and rely on hand-crafted features and rules designed by human experts in specific domains (see Figure 1.4, (a)).

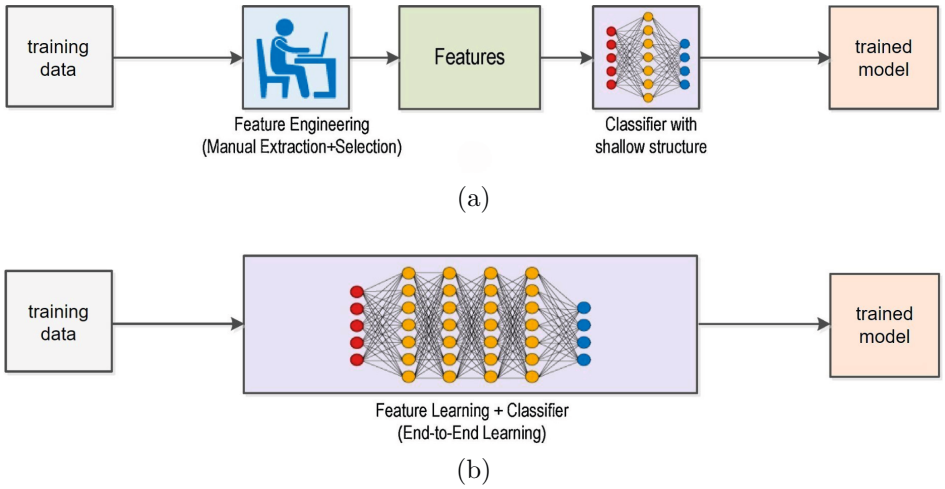


Figure 1.4: Comparison between the workflows for model design: (a) the workflow of classical computer vision methods; (b) the deep learning workflow. Adapted from [52].

As Szeliski [5] points out, some examples of classical methods for **image classification** are bag-of-words algorithms and parts-based algorithms. Bag-of-words (also known as bag-of features, bag-of-keypoints, and bag-of-keypatches [53, 54]) algorithms in computer vision were originally inspired by the bag-of-words approaches to text classification in NLP. In image classification, the 'words' in question are visual words, i.e., handcrafted visual features. Bag-of-words algorithms classify images by computing the distribution of visual words in the target image and comparing it to the distribution that the algorithm learned from the training data using a classifier such as k-Nearest Neighbours (kNN) or Support Vector Machine (SVM)[5](see Figure 1.5).

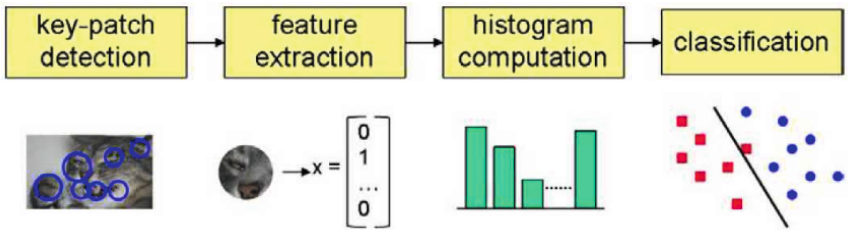


Figure 1.5: Example of bag-of-words algorithm pipeline. Reproduced from [54].

While bag-of-words algorithms are conceptually simple and intuitive, they suffer from two major shortcomings: first, the visual words are not particularly semantically meaningful; second, these algorithms do not take into account the spatial location of the visual words

in the image. To address these limitations and mitigate their effect on classifier accuracy, parts-based algorithms have emerged. These algorithms differ from bag-of-words approaches by relying on more semantically meaningful constituent parts of the classified objects (e.g., the wheels of a motorcycle) and making use of the geometric relationships between these parts. Different parts-based methods vary substantially in how they model these geometric relationships – for instance, as constellations, stars, trees, hierarchical models, or sparse flexible models.

In the early stages of **object detection** research, the primary approach was the sliding window method, which involves dividing the target image into subwindows and applying a classifier to each of them to determine whether it contains an instance of the object (see Figure 1.6).

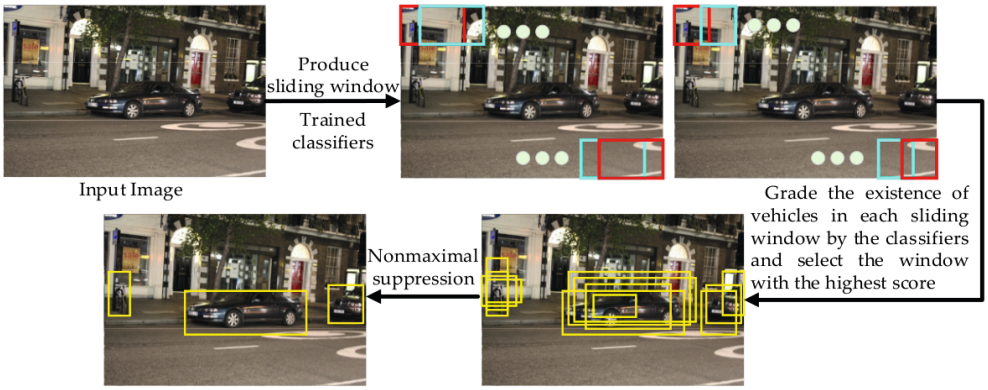


Figure 1.6: Example of a sliding window pipeline for object detection. Reproduced from [55].

To implement this approach successfully, several challenges need to be addressed, namely, an algorithm has to classify the object(s) in each subwindow while also handling variations in object size and overlap between image windows [56]. As Szeliski [5] points out, the development of classical methods for general object detection was largely driven by attempts to solve the PASCAL Visual Object Classes (VOC) challenge [57]. An early approach that achieved comparative success on the PASCAL VOC dataset involved running an SVM classifier on features extracted from subwindows obtained with branch-and-bound search [58] or selective search based on hierarchical segmentation [59]. However, while classical methods made notable progress in detecting specific types of objects such as faces [60] and pedestrians [61], the task of detecting general object categories was not only more challenging than image classification but was also considered nearly intractable with classical methods.

Due to the complexity of **semantic segmentation**, it poses an even greater challenge for classical methods than image classification or object detection. To tackle this, early studies often employed bottom-up approaches, relying on hand-crafted low-level image features such as the smoothness and continuity of image region boundaries [62], texture [63], or colour [64]. An alternative to that is a top-down approach: for instance, Borenstein and Ullman [65] proposed matching putative segments in target images with stored representations of the shapes of objects from a particular class. Furthermore, since semantic segmentation is essentially classification at the pixel level, the concept of visual words – as previously mentioned, first applied to image classification in the domain of computer vision – found its use for semantic segmentation as well. Thus, in Schroff et al. [66], each class was modelled with a single histogram of visual words; afterwards, during classification, a label was assigned to each pixel by generating visual words in its neighbouring region, calculating a histogram

for them, and then identifying the closest match among class histograms to that histogram by finding the shortest Euclidean distance. However, as Yu et al. [67] note, while the method of single-histogram class models is easy to implement, its disadvantage is that it treats each pixel individually, whereas the very concept of dividing image into segments or superpixels implies that they are associated with each other. Another pixel-wise method, TextonBoost [68, 69], sought to better capture local information by employing a Conditional Random Field (CRF, [70]). The CRF took extracted image features such as texture, layout, colour distribution, location, and the edges of putative objects as its input; the use of these features allowed the CRF to model contextual relationships between pixels. However, as Sturgess et al. [71] observe, the rough shape and texture model of TextonBoost resulted in imprecise representation of object boundaries.

Overall, CRF-based, top-down, and bottom-up classical semantic segmentation methods were not sufficiently accurate or robust for many real-world applications.

1.3.2 Deep learning-based methods

The rise in popularity of **deep learning-based methods** began in 2012, when the convolutional neural network (CNN; a particular class of DNN optimized for analysing visual imagery; details follow) AlexNet [18] won the ILSVRC2012 [72] image classification competition by a landslide, achieving a Top-5 classification error of 16.4% and thus outrunning the closest competitor by 9.8% [72]. In just a few years, image classification became dominated by DNN-based methods (cf. Figure 1.7). A similar shift occurred in object detection by 2014, when the Regions with CNN features (R-CNN)[73] model was introduced and convincingly outperformed its closest competitor, the classical methods based Deformable Parts Model (DPM) v5 [74]. On the PASCAL VOC07 dataset, R-CNN achieved a mean average precision of 58.5%, compared to 33.7% by DPM-v5. The same transition occurred in the domain of semantic segmentation: today's state-of-the-art methods are DNN-based, and as demonstrated by a recent benchmarking study by Plaksyvyi et al. [75], they substantially outperform their classical forerunners.

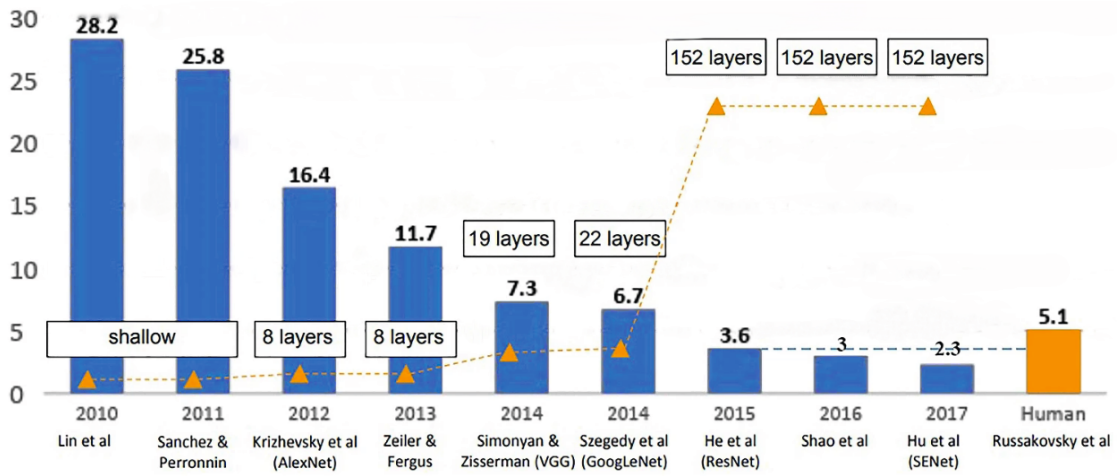


Figure 1.7: Top-5 error rate of image classifiers that won ILSVRC. The depth of the models is provided as well; note that the error rate decrease corresponds to increase in the depth of the models. Also note that since ResNet [76], the state-of-the-art models have outperformed humans (the last bar) on this challenge. Adapted from [5].

As a result of the expansion of deep learning, DNNs have largely replaced classical methods for the majority of image understanding tasks. As O'Mahony et al. [15] point out in their comprehensive comparison of deep learning with classical ML methods, the latter are still used nowadays when image understanding tasks can be solved by simple means, such as pixel counting or colour thresholding, when algorithms must run on low-power devices incapable of supporting resource-hungry DNNs, or as an auxiliary method alongside DNNs. However, in most other cases, DNNs are preferred over classical approaches. The primary reason is their superior performance, yet several other factors contribute to their popularity. One key advantage is that it is not necessary to hand-craft features for them (cf. Figure 1.4 (b)), as DNNs learn directly from data through end-to-end learning. Additionally, their accuracy scales with the increase of the data available for training DNNs, and transfer learning (see Section 1.4.2) allows DNNs to apply knowledge learned from large datasets to smaller ones. Finally, the availability of open-source frameworks for deep learning, such as TensorFlow [77], Keras [78], and PyTorch [79] has also facilitated the rise of deep learning to popularity. As for the drawbacks of deep learning, the most notable of them are that DNNs require a large amount of data and computing power for training. However, the vast amounts of data available on the Internet and the powerful graphics processing units (GPUs) available on the hardware market help mitigate these challenges.

Due to its popularity, deep learning is currently a vast and very rapidly evolving field of research in both academic and industrial domains, with a wealth of literature and Internet sources available. Some of the best surveys of this field have been written by LeCun et al. [80], Pouyanfar et al. [81], and Alom et al. [82]; furthermore, excellent introductions to the field can be found in textbooks by Goodfellow et al. [14], Howard and Gugger [83], Chollet [84], Glassner [85], and Zhang et al. [86], among many others. In the following brief overview of deep learning, I follow the path commonly found in these sources and approach the task of outlining the underlying principles of deep neural networks by starting with the simplest building block of a neural network: the single artificial neuron.

Single neuron model

The development of the artificial neuron model was originally inspired by the structure and functionality of its biological counterpart (cf. Figure 1.8 (a)). The first such model, which implemented simple logic gates, was proposed by Pitts and McCulloch [2]; later, Rosenblatt [87] advanced it by improving the architecture and the learning algorithm of the model, thus making it capable of performing linearly separable classification tasks. The direct descendant of the McCulloch–Pitts neuron and the perceptron, the modern artificial neuron (cf. Figure 1.8 (b)), is described by the formula

$$y = f\left(\sum_{i=1}^{n} w_i x_i + b\right), \tag{1.14}$$

where x_i is the i-th input to the neuron, w_i is the weight that the i-th input to the neuron is multiplied by, b is the bias term, f is the activation function, and y is the output of the neuron. Remarkably, the artificial neuron retains some similarities to biological neurons: the inputs x_1, x_2, x_3, ..., x_n of the artificial neuron are similar to the signals from other neurons that a biological neuron receives via dendrites, the weights w_1, w_2, w_3, ..., w_n that these inputs are multiplied by – to the strength of synaptic connectivity, the summation function $\sum$ of the artificial neuron – to the aggregation of the activations received via dendrites in the body of the neuron, and the activation function f of the artificial neuron – to the threshold

for firing in a biological neuron.

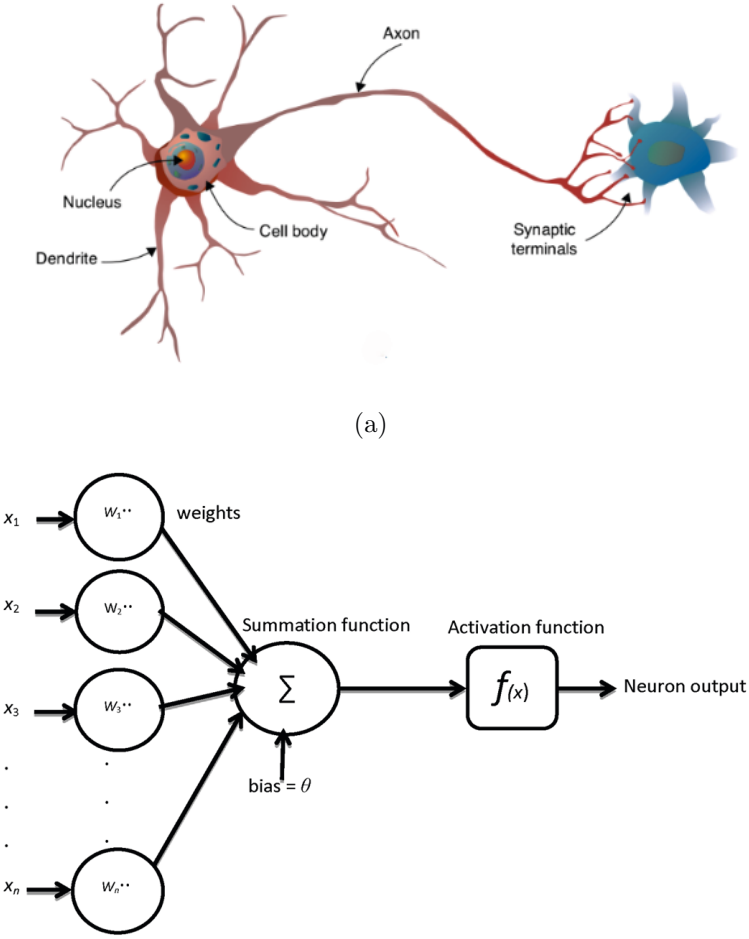


Figure 1.8: Schematic representation of (a) a biological neuron; (b) an artificial neuron. Reproduced from [85] and [88].

However, it should be emphasised that these similarities are observed at a rather high level of abstraction; moreover, artificial neurons significantly diverge from the structure and function of biological neurons to meet their specific functional and computational requirements. In particular, two essential features of the model of the modern artificial neuron, its activation function and optimisation algorithm, are driven by practical considerations, such as the desired output range and learning efficiency, rather than by striving for biological plausibility. The most commonly used activation function in a modern artificial neuron is the Rectified Linear Unit (ReLU) function:

$$f(x) = max(0, x). \tag{1.15}$$

ReLU was originally introduced in the context of artificial neural networks by Fukushima [89] and gained popularity through the work of Nair and Hinton [90]. As Bhumbra [91] observes, while earlier popular activation functions, such the Heaviside step function (Figure 1.9 (a)) and the sigmoid function (Figure 1.9 (b)), were biologically inspired – the former resembling the 'all-or-none' firing property of a biological neuron, and the latter resembling the firing rate of a biological neuron – ReLU (Figure 1.9 (c)) has no known physiological correlates.

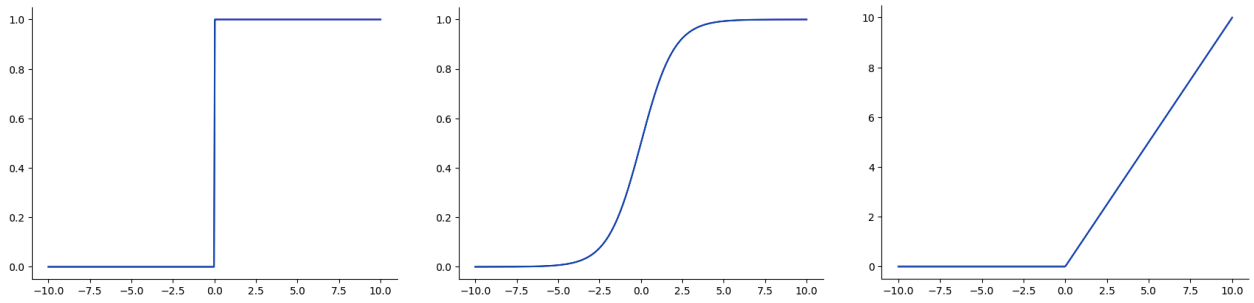
Nevertheless, ReLU offers several advantages over both the Heaviside step function and the sigmoid function, especially in neural networks with multiple artificial neurons. In particular, compared to the Heaviside step function,

$$f(x) = \begin{cases} 1 & \text{if } \sum_{i=1}^{n} w_i x_i + b \geq 0 \\ 0 & \text{otherwise,} \end{cases} \tag{1.16}$$

the advantage of ReLU is that its output is continuous rather than discrete, allowing to leverage efficient gradient descent-based optimisation approaches (discussed later). In comparison with the sigmoid function,

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \tag{1.17}$$

ReLU is less computationally expensive and helps mitigate the vanishing gradient problem, which can cause learning to stall in neural networks.



(a) (b) (c)

Figure 1.9: Activation functions of artificial neurons: (a) the Heaviside step function; (b) the sigmoid function; (c) the ReLU function.

The foundational algorithm for optimising artificial neurons, gradient descent, updates the weights w_1, w_2, w_3, ..., w_n[5] of a single neuron as follows:

$$w_{i,\text{new}} = w_i - \alpha \cdot \frac{\partial L}{\partial w_i} \tag{1.18}$$

where $w_{i,\text{new}}$ is the updated weight w_i after a single iteration of gradient descent, α is the learning rate – a small (typically in the range from 0.1 to 0.001, depending on such factors as the size of the model and characteristics of a training dataset) constant determining the size of the update step – and $\frac{\partial L}{\partial w_i}$ is the partial derivative of the loss function L (e.g., binary cross-entropy loss, or mean squared error loss) with respect to the weight w_i. The loss function measures the error between the predicted output of the neuron and the ground truth, i.e., the actual target values. Although gradient descent has no direct neural correlates, it has proven to be an efficient optimisation algorithm for artificial neurons and is widely used today as a constituent part of the backpropagation algorithm [92, 93] to implement learning in artificial neural networks of different scale, from a single neuron to large and complex DNNs.

[5]For the sake of simplicity, the bias term b is treated as yet another weight for the input $x = 1$.

Fully connected artificial neural networks

As previously mentioned, an artificial neuron is an essential concept for understanding deep learning. However, a single neuron alone does not have the capacity to solve image understanding tasks[6] due to their high complexity. The capacity to solve such tasks emerges when artificial neurons are assembled into a network. A simple instance of such a network consists of an input layer, an output layer, and one or more intermediate layers, known as hidden layers. As the information flows in one direction, from input to output, the network is called a feedforward neural network; moreover, as each neuron $n_1^{(l)}$, $n_2^{(l)}$, $n_3^{(l)}$, ... $n_n^{(l)}$ in layer l is connected both to every neuron $n_1^{(l-1)}$, $n_2^{(l-1)}$, $n_3^{(l-1)}$, ... $n_k^{(l-1)}$ in layer $l-1$ and to every neuron $n_1^{(l+1)}$, $n_2^{(l+1)}$, $n_3^{(l+1)}$, ... $n_m^{(l+1)}$ in layer $l+1$ (where $k, m = n$ or $k, m \neq n$), the network is fully connected (Figure 1.10).

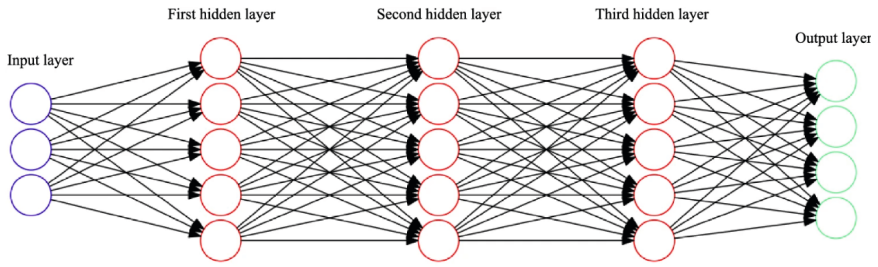


Figure 1.10: Schematic representation of the architecture of a fully connected feedforward neural network. Reproduced from [94].

A good example of the capacity of a simple fully connected feedforward network can be found in Nielsen's now-classic textbook on neural networks [95]: an artificial neural network (ANN) with a single hidden layer of 30 neurons achieves an accuracy of about 95% on the task of classifying hand-written digits in the MNIST dataset [40]. A sceptic might argue that, for a number of reasons, the simplicity of this example ANN is only ostensible, both architecture- and usage-wise. For instance, the number of the neurons in the hidden layer has be determined heuristically, finding the optimal point between underfitting and overfitting. While there are now relatively clear guidelines for choosing an activation function, this decision has historically been more complex. Additionally, the backpropagation algorithm becomes more intricate in a multi-neuron, multi-layer network, and its learning rate requires adjustment by trial and error. While these arguments are valid[7], the remarkable simplicity of neural networks lies in the fact that once their building blocks – such as artificial neurons and learning algorithms – are designed, they can be applied to many other tasks without requiring any fundamental changes.

Indeed, there is nothing about the design of the above ANN that indicates it is specifically intended for classifying hand-written digits. While the sizes of the input layer, hidden layer, and output layer had to be adjusted to match the input image size, the complexity of the data, and the number of classes in the dataset, there was no need for feature engineering. In other words, it was not necessary to explicitly indicate that the digit zero resembles an oval, or that the digit nine looks like a circle with a squiggle under it, etc.

[6] Apart from some very simple cases, such as pixel intensity-based decision making.

[7] The importance of these architectural and learning considerations is one of the reasons why it took several decades for the ANN and deep learning (DL) paradigms to become practically operational for such complex tasks as image understanding.

As a result, the same architecture can be successfully applied to classify very different images. To exemplify that, I implemented the same architecture as above in the Keras framework [78] and trained it in the Google Colab environment[8] on an NVIDIA A100 GPU, following the same procedure as in Nielsen [95] – 30 epochs of training, stochastic gradient descent learning algorithm with $\alpha = 3.0$, the mini-batch size of 10 – on a more recent and complex alternative to MNIST, the Fashion MNIST dataset [96](see the visual comparison between the two in Figure 1.11).

To ensure comparability, I also reproduced Nielsen's MNIST experiment [95] and repeated each experiment 30 times. As a result, similar to what Nielsen reported, the model in the MNIST experiment achieved a mean accuracy of 94.6% (SD = 0.0036, variance = 1.31e-05) on the test data. As expected, the performance of the non-adapted model on the Fashion MNIST dataset was substantially lower, achieving a mean accuracy of 84.8% (SD = 0.0085, variance = 7.24e-05) on the test data. However, this is still significantly higher than the accuracy of a putative 'naive' classifer (100%/10 classes with equal sample size = 10%) and serves as a fitting example of the generalisability of ANNs to new data.

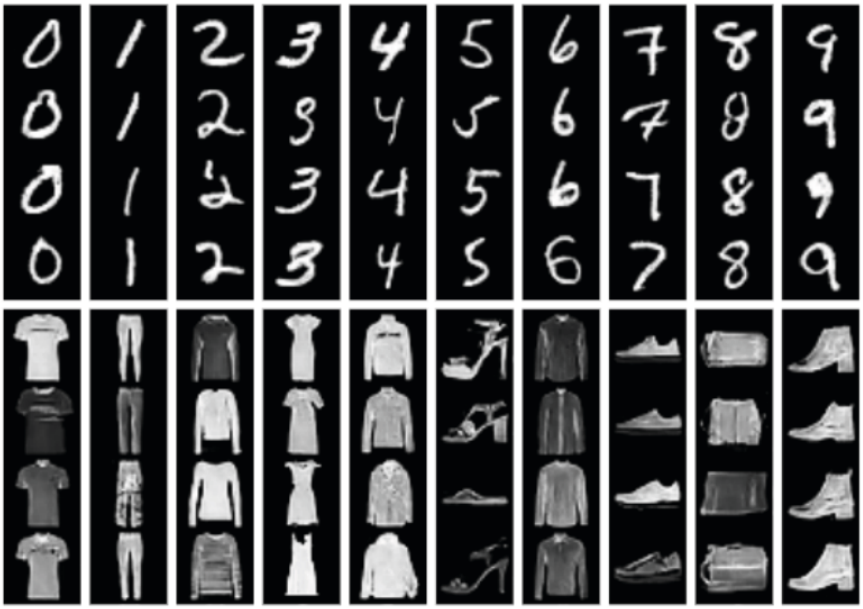


Figure 1.11: Examples from MNIST (top) and Fashion MNIST (bottom) datasets. Fashion-MNIST columns represent the following classes: `T-shirt`, `trousers`, `pullover`, `dress`, `coat`, `sandal`, `shirt`, `sneaker`, `bag`, and `ankle boot`. Reproduced from [97].

Convolutional neural networks

Despite the advantages of fully connected ANNs, they also suffer from certain drawbacks when applied to image understanding tasks. In particular, they do not take into account the hierarchical patterns or spatial relationships in data, as they treat each input feature – such as a pixel value in a grayscale image, or a channel value in an RGB image – independently rather than considering its relation to spatially adjacent features. This drawback may not be particularly evident when fully connected ANNs are trained on simple datasets like MNIST, where the target object occupies the entirety of the image and is centred. However, it

[8] https://colab.research.google.com/; accessed 10 September 2024.

negatively affects the performance of such architectures on more complex datasets, such as ImageNet [19], or in real-world applications. Additionally, the dense connectivity in fully connected ANNs is not particularly computationally efficient due to the high number of parameters, which may impact both training and inference performance of such networks.

Convolutional neural networks (CNNs), the most popular architecture for image understanding, have revolutionised the use of ANNs in computer vision [98] and are also the primary tool used in the research reported in this thesis. CNNs successfully address the challenges of utilising spatial patterns in image data and making network connectivity more sparse, thereby improving efficiency.

Historically, the design of CNN was inspired by Hubel and Wiesel's [99] experimental research on the processing of the visual spatial information in the cat brain. Fukushima's Neurocognitron [100] was the earliest attempt to integrate the principles of visual shift invariance discovered by Hubel and Wiesel in biological systems into the design of artificial neural networks. Building on this foundational work to develop robust applications, LeCun – often dubbed the father of CNNs – along with his collaborators, improved the methodology of training models invariant to pattern shifts by integrating the backpropagation algorithm into it. This led to state-of-the-art results at the time in image understanding tasks such as document [101] and handwritten zip code [102] recognition. Finally, as previously mentioned, a revolutionary breakthrough in the development of CNNs occurred in 2012, when the AlexNet [18] won the ILSVRC competition [72] and sparked an exponential surge in the creation of new CNN architectures and their practical applications.

While the range of available CNN architectures is broad, from the early pioneering LeNet [101] to the most recent state-of-the-art models, a typical CNN architecture consists of three main types of layers: convolutional, pooling, and fully connected layers (Figure 1.12).

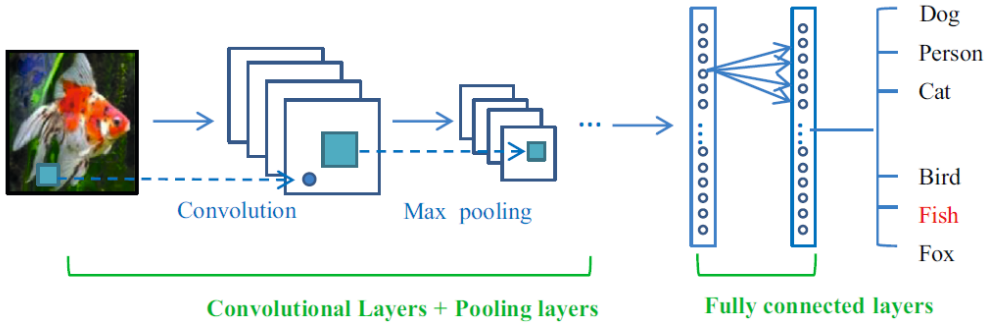


Figure 1.12: Architecture of a convolutional neural network. Reproduced from [98].

Convolutional layers are the central building blocks of CNNs. Each layer consists of a number of kernels – small matrices with weights that a CNN learns during training. A kernel operates by sliding across the input that it receives from the previous layer and performing dot product multiplication with the part of the input it covers[9] (Figure 1.13). As a result, each kernel produces a feature map that indicates the presence of specific features, such as edges or textures, in its input layer (Figure 1.14). Feature maps tend to increase in complexity in deeper layers of CNNs: while activations in the initial layers resemble edge detection, activations in deeper layers respond to more complex patterns. Mathematically, the basic form of a convolution applied to a two-dimensional image can be expressed by the following formula:

[9]Which, to revisit neurophysiological analogies once again, corresponds to the receptive field of a biological neuron.

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n), \tag{1.19}$$

where I denotes the image, $I(m,n)$ is the pixel value of the input image at position (m,n), K represents the kernel, and $S(i,j)$ is the output of the convolution operation at position (i,j).

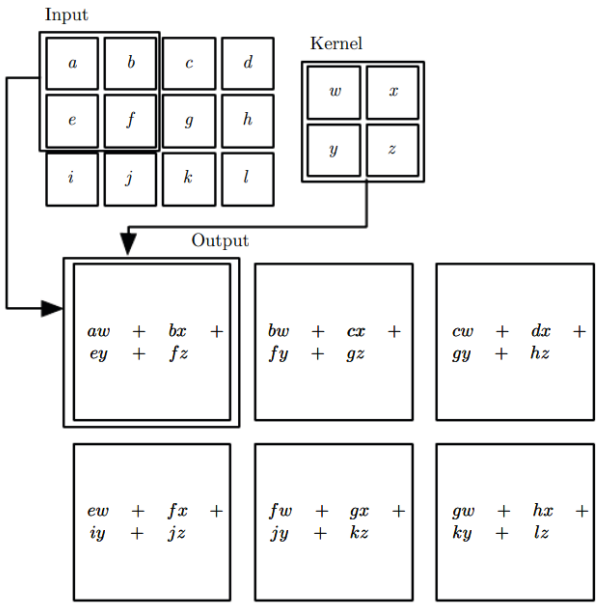


Figure 1.13: The basic form of a convolution of a two-dimensional image. The boxes with arrows are drawn to demonstrate how the dot product of the kernel with the upper left element of the input forms the corresponding element of the output. Reproduced from [14].

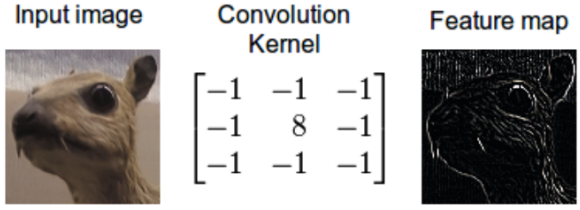


Figure 1.14: An example of edge detection with a convolutional kernel. Note that, for illustrative purposes, the kernel used in the example is composed of predefined integers. Such an approach is characteristic of traditional image processing, whereas CNNs employ floating-point kernels that learn their weights during training. Reproduced from [103].

Some essential aspects of implementing convolutional layers include the choice of activation function, the number of channels in a kernel, and the selection of padding and stride size. To introduce non-linearity, ReLU or other similar activation function is typically applied to the feature map before it is passed to the next layer. Furthermore, since the input to a convolutional layer typically has multiple channels, each kernel must have as many channels as the input. During convolution, each channel is convolved separately, and the resulting values are summed to produce a single output value for each location in the feature map

(Figure 1.15). Padding the image with zeros or values of neighbouring pixels is often used to ensure that image size does not shrink after each convolution. While the default stride size for a kernel is 1, larger strides can be used for the sake of dimensionality reduction or to increase the computational efficiency of the network.

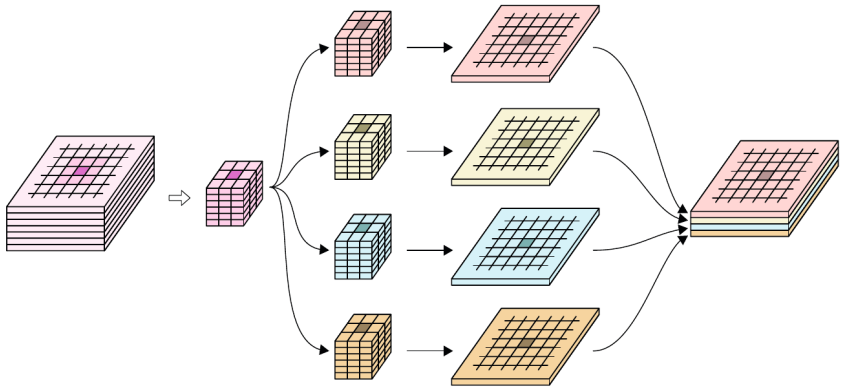


Figure 1.15: Application of convolutions to an input that has multiple channels. Note that each kernel must have as many channels as the input, but the output consists of one channel per kernel. Reproduced from [85].

Overall, the use of convolutional layers provides the advantage of sparsity, as the weights in a given convolution kernel are the same for all the input values it is applied to. This results in fewer weights for the network to learn compared to fully connected layers. Additionally, applying the same kernel across the entire input allows the network to detect features corresponding to the given kernel regardless of their location – an important advantage over networks relying solely on fully connected layers.

Pooling layers typically follow convolutional layers in CNN architectures. Their primary function is to reduce feature maps produced by the preceding convolutional layer, thereby decreasing the number of parameters and reducing the overall complexity of the network. The two main types of pooling in CNN are average pooling and max pooling. Average pooling outputs the average value of all inputs in a rectangular window of a certain size (Figure 1.16 (a)) and is expressed by the following formula:

$$P_{avg}(x, y) = \frac{1}{N} \sum_{i=1}^{N} x_i \tag{1.20}$$

Max pooling, on the other hand, outputs the largest value of all inputs within a rectangular window of a certain size (Figure 1.16 (b)) and is expressed as:

$$P_{max}(x, y) = \max(x_1, x_2, ..., x_N) \tag{1.21}$$

Fully connected layers are typically the final layers in a CNN. Their purpose is to map the features learned by the preceding layers – mainly convolutional layers followed by pooling layers – onto the output of the network, making it possible for CNNs to perform tasks such as image classification, object detection, semantic segmentation, and other image understanding tasks.

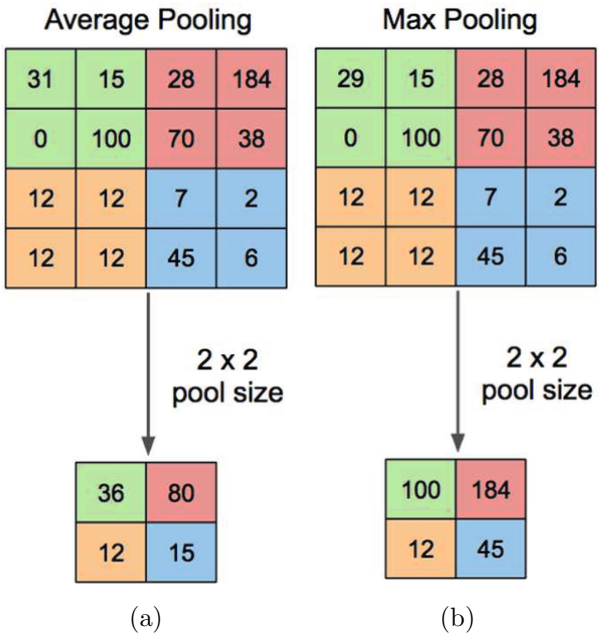


Figure 1.16: Visual examples of pooling operations: (a) average pooling; (b) max pooling. Reproduced from [104].

In the remainder of this section, I will describe the specific CNN models used in the research presented in this thesis.

Image classification models. For image classification tasks reported in this thesis, I used three CNN models: MobileNetV2 [105], MobileNetV3Large [106], and EfficientNet-B7 [107].

MobileNets is a family of compact CNNs designed to balance accuracy and latency. To date, there have been three generations of MobileNets: the original MobileNet [108], MobileNetV2 [105], and MobileNetV3 [106]. As the name suggests, they were developed with the goal of making it possible to deploy them on mobile and other low-power devices, such as edge and embedded devices.

The first generation, the original *MobileNet*, utilises depthwise separable convolutions, originally introduced by Sifre [109] and later implemented in Inception CNN models [110]. While both Inception models and MobileNet use depthwise separable convolutions for the same purpose – to reduce the computational load – there is a substantial difference in how extensively this type of convolution is applied in the two architectures. Namely, while depthwise separable convolutions are only used in the few initial layers of Inception CNNs, MobileNet is primarily build from layers implementing them.

The core concept of depthwise separable convolutions is to split the two main components of standard convolution operations – convolving the input to the layer with kernels, and combining the results of convolutions to obtain new representations – into two distinct steps. To do that, two layers are used: a depthwise convolutional layer, and a pointwise convolutional layer, thereby reducing the computational cost. To elaborate, the computational cost C of a standard convolution is:

$$C_{std} = D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F \quad [108], \tag{1.22}$$

where $D_K \times D_K$ is the size of the kernel, $D_F \times D_F$ is the size of the input to the convolutional layer, M is the number of channels in the input, and N is the number of output channels. The cost of depthwise separable convolution is the sum of the costs of depthwise convolution and 1×1 pointwise convolution:

$$C_{dw} = D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F \quad [108] \tag{1.23}$$

Since the steps of convolving the input and combining the results of the convolutions are performed jointly, a reduction in computation is achieved:

$$\frac{C_{dw}}{C_{std}} = \frac{D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F}{D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F} = \frac{1}{N} + \frac{1}{D_K^2} \quad [108] \tag{1.24}$$

To sum up, depthwise separable convolutions reduce computations in comparison to standard convolutions by the factor of $\approx K^2$, where K is the size of the kernel $D_K \times D_K$ [105]. Thus, in case of MobileNet, which uses 3×3 depthwise separable convolutions, the computational cost is reduced by ≈ 9 times.

Another means of improving performance in MobileNet is the use of the ReLU6 activation function instead of the standard ReLU, that is, confining the output to a maximum of 6, ensuring it remains within the range [0, 6]:

$$f(x) = min(max(0, x), 6). \tag{1.25}$$

The purpose of using ReLU6 rather than ReLU is to improve the robustness of low-precision computations.

As a result of these improvements, MobileNet, whose architecture consists of an initial 3×3 standard convolution layer with stride 2, followed by 13 depthwise separable convolution blocks (see Figure 1.17 (a)), and then a global average pooling layer and a fully connected layer, contains far fewer parameters than a model with the same architecture using full convolutions – 4.2 million vs 29.3 million – yet still achieves comparable accuracy, 70.6% vs 71.7%, on the ImageNet classification task [108].

The next generation of MobileNets, *MobileNetV2*, features substantial modifications to the main building blocks of the network. While the original MobileNet has 2 layers in each building block, in MobileNetV2, each building blocks consists of three layers (Figure 1.17, (b) and (c)):

- The first layer is a 1×1 convolution layer; it increases the channel dimension of the input feature map. This expansion is necessary to improve the capacity of depthwise convolution, as otherwise, its capacity would be lower than that of regular convolution.
- The second layer is a 3×3 depthwise convolution layer.
- The third layer is another 1×1 convolution layer, which shrinks the expanded feature map back to its original dimension. Unlike the original MobileNet, this layer in MobileNetV2 does not use an activation function, as the authors found that using linearity rather than nonlinearity in this layer prevents excessive loss of information.

As the overall narrow-wide-narrow sequence is known as the inverted bottleneck, this three-layer building block in MobileNetV2 is respectively referred to as the mobile inverted

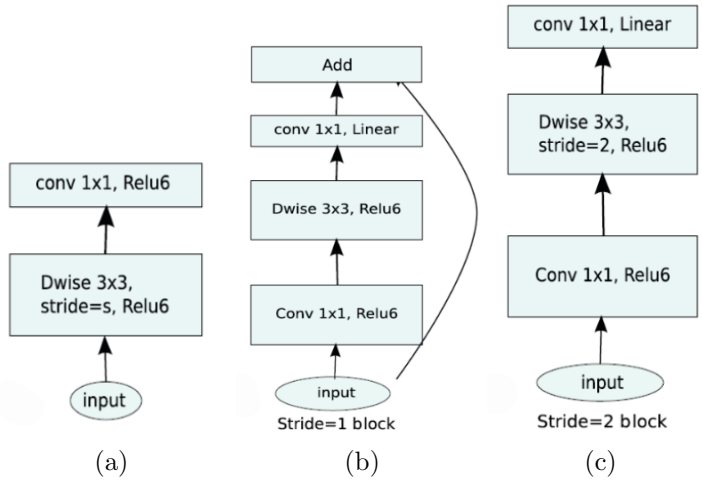


Figure 1.17: Building blocks of MobileNetV1 (a) and MobileNetV2: (b) with stride = 1; (c) with stride = 2. Note the skip connection in (b). Reproduced from [108].

bottleneck block [111]. The stride of depthwise convolutions in the mobile inverted bottleneck block is either 1 or 2. In the case of stride, an additional feature – a skip connection

$$output_{\text{block}} = f(input_{\text{block}}) + input_{\text{block}} \tag{1.26}$$

– is added (see Figure 1.17 (b)) to improve gradient flow across multiple layers [105].

With these improvements, MobileNetV2 achieves a better accuracy on ImageNet – 72.0% vs 70.6% – than its predecessor, MobileNetV1, while using less parameters, 3.4 million vs 4.2 million.

The architecture of *MobileNetV3*, available in two sizes – Small[10] and Large – was designed by Howard et al. [106] through a two-stage search.

The first stage of the search was platform-aware neural architecture search (NAS) following the methodology outlined by Tan et al. [112]. Since both studies used the same RNN-based optimizer and architecture search space, Howard et al. [106] found similar results to Tan et al. [112] for their Large model, for which the target inference latency was set to $\approx$ 80 ms. Therefore, the Large architecture selected after the first search stage was one of the architectures designed by Tan et al. [112], MnasNet-A1. MnasNET models build upon the MobileNetV2 architecture by incorporating squeeze and excitation operations [113] into the mobile inverted bottleneck blocks. The squeeze operation is applied to the feature map $\mathbf{X}$ of a convolutional layer with the dimensions $H \times W \times C$ (where H is height, W is width, and C is the number of channels) to obtain a channel descriptor – a vector $\mathbf{z}$ with dimensions $1 \times 1 \times C$. Each entry z_c of $\mathbf{z}$ is obtained by shrinking the respective channel X_c, with dimensions $H \times W$, into a single value using global average pooling (GAP):

$$z_c = \text{GAP}(X_c) = \frac{1}{H \times W} \sum_{i=1}^{H} \sum_{j=1}^{W} X_c(i, j) \tag{1.27}$$

Next, the excitation operation uses $\mathbf{z}$ to recalibrate the channels of $\mathbf{X}$. To do this, $\mathbf{z}$ is first

[10] As I did not use MobileNetV3Small in the research reported in this thesis, I will not discuss it in detail here.

passed through two fully connected (FC) layers, W_1 and W_2, to obtain the scaling vector $\mathbf{s}$:

$$s = \sigma(\mathbf{W_2}\text{ReLU}(\mathbf{W_1}\,\mathbf{z})), \tag{1.28}$$

where σ denotes the sigmoid function. The vector $\mathbf{s}$ is then used to rescale $\mathbf{X}$ by multiplying each channel X_c by a scalar s_c:

$$\tilde{X}_c = s_c \times X_c \tag{1.29}$$

The goal of the entire $\mathbf{X} \to \tilde{\mathbf{X}}$ transformation, using squeeze and excitation, is to improve the representative power of the CNN [113], thereby improving its performance without adding significant computational overhead.

The second stage of architecture search employed the NetAdapt algorithm [114], which was tuned to meet the needs of the study. The modified NetAdapt algorithm refined the architecture obtained in the first stage by iteratively generating a set of new proposals, each of which modified the architecture obtained in the previous iteration in a way that reduced the inference latency by at least $\delta = 0.01|L|$, where L was the latency of the seed model. After generating a proposal, the pretrained model from the previous step was adjusted to the proposed architecture, fine-tuned for 10 000 steps, and evaluated on the target metric, which was maximise the ratio $\frac{\Delta\text{Acc}}{\Delta\text{latency}}$. Here, ΔAcc is the change in the accuracy of the model, and Δlatency is the change in the latency of the model (where $\Delta\text{latency} \geq \delta$). Once the target inference latency of 80 ms was achieved, the weights of the final model were obtained by training the resulting architecture from zero on ImageNet [19].

In addition to the two-stage architecture search, Howard et al. [106] improved performance by redesigning expensive layers, e.g., the last few layers of the network and the initial convolutional layers. They also replaced ReLU with the swish activation function in some blocks of MobileNetV3. Swish, introduced by Ramachandran et al. [115], is given by the formula

$$\text{swish}(x) = x \cdot \sigma(x).$$

Swish has been shown to improve accuracy over ReLU-based layers. However, it can incur computational overheads when used on mobile devices due to the cost of computing the sigmoid function, which is resource-intensive for mobile CPUs [106]. Therefore, Howard et al. [106] replaced swish with hard-swish (h-swish), its piece-wise linear analogue:

$$\text{h-swish}(x) = x \cdot \frac{\text{ReLU6}(x+3)}{6}$$

Furthermore, as Howard et al. [106] experimentally discovered that most of the benefits from using swish come when this activation function is used in the deeper layers of the network, MobileNetV3Large uses h-swish in the first block, blocks 2 to 7, and block 19, while blocks 8 to 17 retain the same activation function as in MobileNetV2, i.e., ReLU6.

As a result of these improvements, MobileNetV3Large achieves 75.2% Top-1 accuracy on ImageNet, compared to 72% for MobileNetV2, all while reducing latency by 20% compared to the latter.

EfficientNet-B7 is the largest model in the EfficientNet family of CNNs, developed by Tan and Le [107] with the goal of achieving the same accuracy as state-of-the-art classifiers while having fewer parameters (hence 'efficient' in 'EfficientNet'). Similar to MobileNetV3, the baseline model, EfficientNet-B0, was developed through an automated neural architecture

search proposed by Tan et al. [112], optimising the model for both accuracy and computational efficiency. The goal of the optimisation was set as:

$$\underset{m}{\text{maximize}} \quad Accuracy(m) \times \left[\frac{FLOPs(m)}{T}\right]^{w} \quad [112,\ 107] \tag{1.30}$$

where m is the model, $Accuracy(m)$ is the accuracy of the model, $FLOPs(m)$ is the number of the floating-point operations (FLOPs), which measures the computational demands of the model, T is the target number of FLOPs (set to 400 million by the authors), and w is a hyperparameter controlling the trade-off between accuracy and FLOPs, set to -0.07 by the authors.

The same as with MobileNetV2, the main building blocks of EfficientNet-B0 are mobile inverted bottlenecks, with added squeeze-and-excitation optimisation [113]. Overall, EfficientNet-B0 baseline model consists of an initial standard convolutional layer with 3×3 kernels, 16 mobile inverted bottleneck layers with kernel sizes of either 3×3 or 5×5, and the final layers – a convolutional layer with 1×1 kernel size, a pooling layer, and a fully connected layer.

To design the rest of the models in the EfficientNet family, a compound scaling method was used. As Tan and Le observe [107], the typical approaches to scaling CNNs to achieve better accuracy are to increase their depth, width, or, less commonly, image resolution. Usually, only one of these dimensions is scaled at a time, as scaling two or three arbitrarily would require many experiments based on trial and error approach, making it computationally expensive. Instead, Tan and Le [107] propose a more advanced approach – to uniformly scale up the depth, width, and image resolution using a set of fixed scaling coefficients. To do that, one needs to specify the compound coefficient ϕ to indicate how many more computational resources are available for the scaled-up model compared to the baseline model:

$$\phi = \log_2 \left(\frac{FLOPs_{\text{scaled}}}{FLOPs_{\text{baseline}}}\right), \tag{1.31}$$

where $FLOPs_{scaled}$ is the number of FLOPs allocated to the scaled-up model, and $FLOPs_{baseline}$ is the number of FLOPs for the baseline model. Next, the coefficients for depth d, width w, and image resolution r of the scaled-up model are set as:

$$\begin{aligned} &\text{depth:} && d = \alpha^{\phi} \\ &\text{width:} && w = \beta^{\phi} \\ &\text{resolution:} && r = \gamma^{\phi} \qquad\qquad [107] \\ & && \text{s.t. } \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2 \\ & && \alpha, \beta, \gamma \geq 1 \end{aligned} \tag{1.32}$$

where $\alpha, \beta, \text{and} \gamma$ are constants determined via a small-scale grid search optimising for accuracy:

$$\begin{aligned} &\max_{d,w,r} && Accuracy(\mathcal{N}(d,w,r)) \\ &\text{s.t.} && \mathcal{N}(d,w,r) = \bigodot_{i=1,\ldots s} \hat{F}_i^{d\cdot\hat{L}_i}\left(X_{\langle r\cdot\hat{H}_i, r\cdot\hat{W}_i, w\cdot\hat{C}_i\rangle}\right) \quad [107] \\ & && \text{Memory}(\mathcal{N}) \leq \text{target\_memory} \\ & && \text{FLOPs}(\mathcal{N}) \leq \text{target\_flops}. \end{aligned} \tag{1.33}$$

Here d, w, and r, as in Equation 1.32, are the coefficients for network depth, network width, and image resolution, respectively; $F_i^{L_i}$ is layer F_i repeated L_i times in the i-th block of the network; $\langle H_i, W_i, C_i \rangle$ is the shape of input tensor X for layer i.

As a result, after setting ϕ to 1 for the baseline model EfficientNet-B0, which implied search for EfficientNet-B1 with $\frac{FLOPs_{\text{scaled}}}{FLOPs_{\text{baseline}}} = 2$, the grid search returned the following optimal coefficients for EfficientNet-B0: $\alpha = 1.2$, $\beta = 1.1$, and $\gamma = 1.15$. These values of α, β, and γ were then fixed as constants, and by increasing the value of ϕ, EfficientNet-B1 to B7 were derived. The largest model, EfficientNet-B7, achieved a marginally better accuracy of 84.4% on the ImageNet dataset compared to 84.3% by its closest competitor at the time, the GPipe model [116], all while having considerably fewer parameters – 66 million versus 557 million.

Semantic segmentation models. For semantic segmentation tasks, I use two image classification models – Xception [117], and MobileNetV2 [105] – both extended with a DeepLabv3 [118] segmentation head.

Xception, released by Chollet in 2017, is a modification of the Inception type of architecture of CNNs, originally introduced by Szegedy et al. [119] and further developed as Inception-v2 [110], Inception-v3 [120], Inception-v4 [121], and Inception-ResNet [121]. The main building blocks of Inception models are the eponymous modules. While the implementation details of these modules vary from one generation of Inception networks to another, an Inception module, in its simplified form, consists of a set of 1×1 kernels followed by 3×3 kernels, with the resulting feature maps concatenated (Figure 1.18).

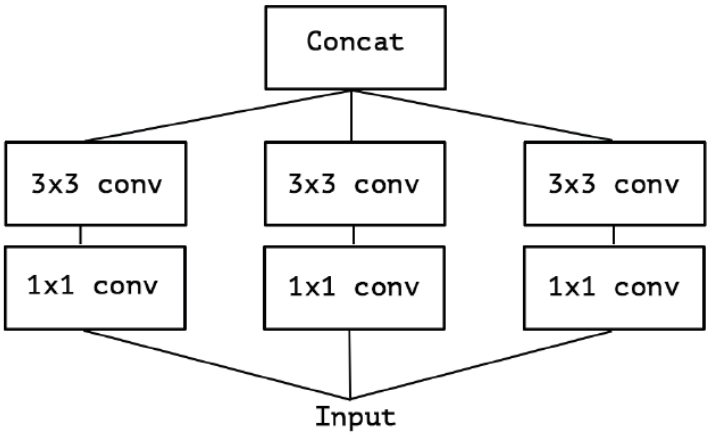


Figure 1.18: Architecture of a simplified Inception module. Reproduced from [117].

The function of 1×1 kernels is to find cross-channel correlations, while the function of 3×3 kernels (or larger, e.g., 5×5 kernels, in other versions of Inception modules) is to find spatial correlations. Since these operations are decoupled in Inception modules, the underlying hypothesis of Inception, as Chollet [117] observes, is that these operations are sufficiently independent from each other that it is better to keep them separate rather than combine them, as a regular convolutional layer would. In Xception, this hypothesis is taken further (hence 'Xception', which stands for 'Extreme Inception') by postulating that these operations can be entirely decoupled by replacing Inception modules with depthwise separable convolutions. The resulting Xception architecture consists of 36 depthwise separable convolutional layers with residual connections. These layers form three major parts of the model: the entry flow,

through which the input data passes first, the middle flow, repeated eight times, and the exit flow (Figure 1.19). Xception has the same number of parameters as its predecessor, Inception-v3 [120]; it demonstrated marginally better accuracy than Inception-v3, 79% vs 78.2%, on the ImageNet dataset, and substantially better accuracy than Inception-v3 on JFT, an internal Google dataset [117].

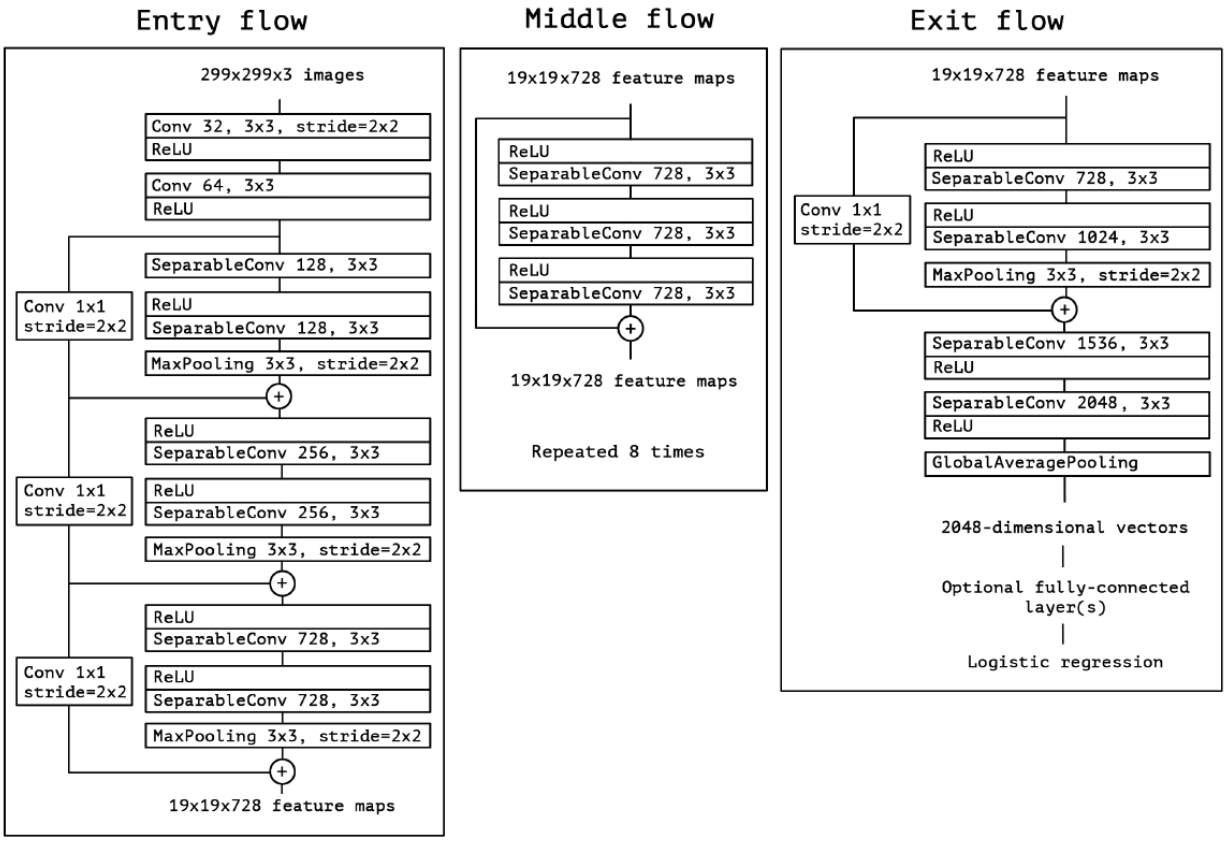


Figure 1.19: Architecture of Xception [117]. Reproduced from [117].

For semantic segmentation, Xception is used as the backbone of the DeepLabv3 [118] model. The models in the DeepLab family are designed for repurposing image classifiers for semantic segmentation tasks. The first DeepLab model was introduced by Chen et al. [122] in 2014 and was soon followed by DeepLabv2 [123], v3 [118], and v3+ [124][11].

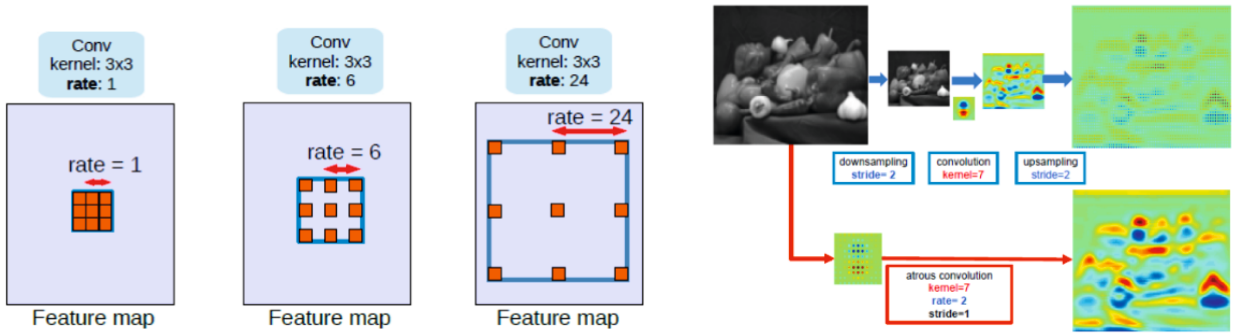
The backbone of *DeepLabv1* is VGG-16[12] [125] – a 16-layer CNN image classifier, which was a state-of-the-art CNN at the time DeepLabv1 was released. To adapt VGG-16 for semantic segmentation, the last max pooling layers are removed, and its fully connected layers are replaced by convolutional layers to preserve dense spatial information that would otherwise be lost, as fully connected layers would collapse it into a single vector. However, these changes alone are not sufficient, as standard convolutions would yield very sparse feature maps, a common issue when repurposing CNN-based image classifiers for semantic segmentation. One way to address this is by using deconvolutional layers to produce more refined segmentation masks [126]. However, as Chen et al. point out [122], this substantially in-

[11] As Deeplabv3+ was not used in the research reported in this thesis, I do not discuss it here.
[12] Named after the Visual Geometry Group at the University of Oxford, where the model was developed.

creases the complexity and therefore the training time of the model. Therefore, DeepLabv1 employs an alternative approach: atrous (from the French *à trous* – 'with holes') convolutions. Atrous convolutions, also called the hole algorithm and dilated convolutions, were originally introduced in the field of signal processing to efficiently compute the undecimated wavelet transform [127]. In the context of computer vision, this method involves either inserting zeros in convolutional kernels to increase their size, or, more efficiently, keeping the kernels unchanged but sparsely sampling the feature map they convolve (Figure 1.20). Atrous convolutions are given by the formula:

$$y[i] = \sum_k x[i + r \cdot k]w[k] \quad [118], \tag{1.34}$$

where i is each location on the output y and the feature map x, w is the kernel, and r is the atrous rate, that is, the stride with which the input signal is sampled. In VGG-16, as the backbone of DeepLabv1, three last convolutional layers are expanded by a factor of 2 (i.e., $r = 2$), and the subsequent convolutional layer (which was a fully connected layer in the original model) is expanded by a factor of 4 (i.e., $r = 4$).



(a) (b)

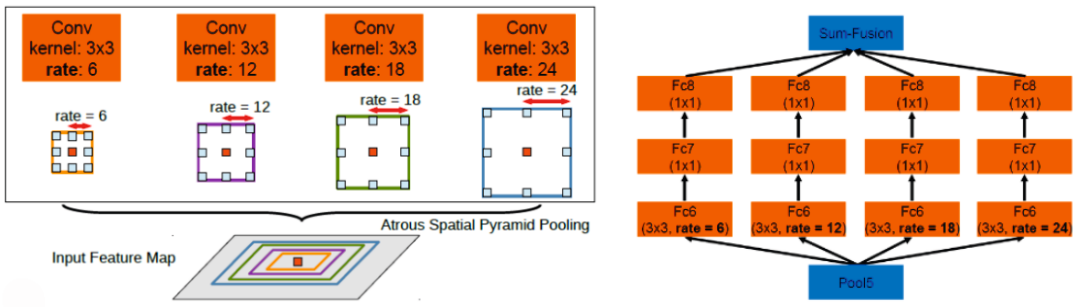
Figure 1.20: Illustration of atrous convolutions: (a) different rates of atrous convolutions for a kernel size of 3×3; (b) atrous convolution in 2D. Top row of (b): using standard convolution on a feature map with low resolution leads to the extraction of sparse features. Bottom row of (b): using atrous convolution (rate = 2) on a feature map with high resolution leads to the extraction of dense features. Adapted from [123] and [118].

Another hallmark of DeepLabv1 is the use of a fully connected CRF [70] to address the inherent problem that arises when CNN-based image classifiers are repurposed for semantic segmentation, namely, that classifiers need to be invariant to spatial transformation, whereas semantic segmentation models require spatial accuracy [122]. The use of the CRF is an efficient solution for this problem, as it can capture edge details while accounting for long-range dependencies. Furthermore, at the time of the release of DeepLabv1, the CRF was more computationally efficient than its alternatives [122].

By integrating these features with VGG-16, DeepLabv1 achieved both better speed and accuracy, improving by 7.2% in mIoU compared to the nearest competitor on the PASCAL VOC dataset, while keeping its overall architecture of the system simple, as it essentially consists of two modules: a CNN, and a CRF module [122].

DeepLabv2 further improves the accuracy of semantic segmentation by efficiently addressing the issue that that objects can appear at different scales in input images – e.g., a tree

might take up only a part of an image, or its entirety. To handle this, a model needs filters with fields of view of different size to segment objects efficiently. As the authors of DeepLabv2 observe [123], the standard approach prior to their work was to run a model on several rescaled versions of the same image and then aggregate the resulting feature maps. However, as they note, while this indeed improved segmentation accuracy, it also introduced significant computational overhead. To solve this, DeepLabv2 employs multiple atrous convolutional layers with different sampling rates in parallel. The outputs from these layers are processed separately and then eventually fused to obtain the final result. Chen et al. [123] call this approach 'atrous spatial pyramid pooling' (ASPP; Figure 1.21) and use it not only with VGG-16, as in the first version of DeepLab, but also with the more efficient ResNet-101 [76], demonstrating the compatibility of DeepLabv2 with various architectures. As Chen et al. [123] report, these improvements enable DeepLabv2 to achieve a 79.7% mIoU on the PASCAL VOC dataset and deliver state-of-the-art results on three other datasets: PASCAL-Context [128], PASCAL-Person-Part [129], and Cityscapes [130].



(a) (b)

Figure 1.21: Atrous spatial pyramid pooling (ASPP): (a) Schematic representation of ASPP. Different colours of the rectangles correspond to the different fields of view of kernels with different rates. (b) Actual architecture of the ASPP module in DeepLab V2. Adapted from [123].

The major change in *DeepLabv3*, compared to previous versions, is that it does not employ the CRF anymore, allowing the model to be trained end-to-end[13] and increasing its computational efficiency, as the CRF had introduced substantial computational overhead [118]. To replace the CRF, DeepLabv3 implements two key changes. First, batch normalisation [110] is introduced, which standardises the inputs to the layers of the model across each mini-batch. Second, the ASPP module is enhanced by incorporating image-level features that encode global context, helping to mitigate the issue of a decreasing number of valid kernel weights in ASPP. This issue arises because, as larger sampling rates are used in ASPP, fewer weights are applied to the valid feature region, with more being applied to the padding with zeros [118]. In practice, that entails two main changes to the implementation of the ASPP module (Figure 1.22):

- adding a 1×1 convolution layer;
- applying global average pooling to the feature map produced by the layer preceding the ASPP module, passing the obtained image-level features through a 1×1 convolution with 256 kernels, and then bilinearly upsampling the output back [118].

[13] Previously, with the CRF, the training consisted of two stages.

As a result of the above improvements, DeepLabv3 achieved an mIoU of 85.7% on the PASCAL VOC 2012 test set while also improving computational efficiency by eliminating the CRF.

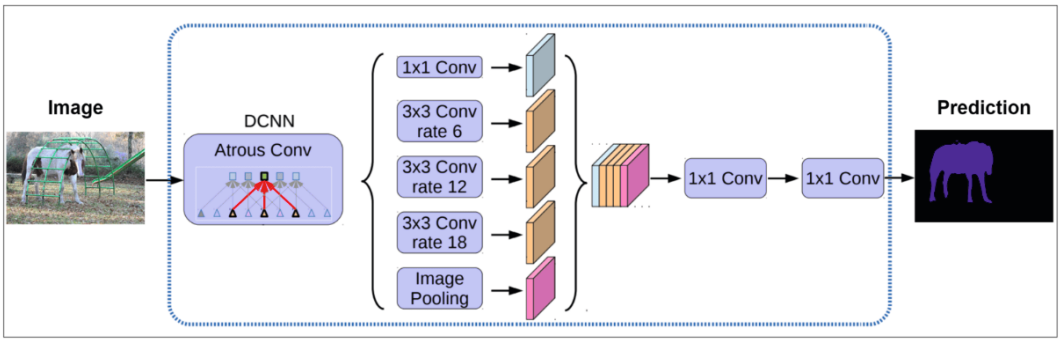


Figure 1.22: Overview of the DeepLab V3 architecture. Reproduced from [131].

While the original DeepLabv3 paper reports only experiments with ResNet-50 and ResNet-101 [76] CNNs, the proposed system is compatible with a broad range of CNN-based classifiers. In particular, the DeepLabv3 repository[14] includes models based on Xception [117] and MobileNetV2 [105]. To repurpose the original Xception classifier for semantic segmentation, the authors of the repository implemented the following changes:

- increased the number of the layers, making the model available in three versions – with 41, 65, and 71 layers, respectively;
- replaced max pooling operations with atrous separable convolutions;
- added batch normalisation and ReLU activation after each 3×3 depthwise convolution.

In the adaptation of MobileNetV2 for semantic segmentation, the ASPP module is omitted to enable faster computation.

Object detection models. For research involving object detection tasks reported in this thesis, I use the *YOLOv5* model. Models from the YOLO (You Only Look Once) family, as the name suggests, can detect objects after just a single forward pass of the input image. This makes such single-stage object detectors particularly fast and suitable for real-time applications, especially in comparison with two-stage object detectors such as R-CNN [73] and its successors, Fast R-CNN [132], Faster R-CNN [133], and Mask R-CNN [134]. While single-stage object detectors initially tended to be less accurate than their two-stage counterparts, over time, many versions of YOLO models have been introduced, from the original YOLOv1 introduced in 2015 by Redmon and Farhadi [47], to the most recent YOLOv8 [135] and YOLO-NAS (where NAS stands for 'neural architecture search') [136], and as each generation has introduced improvements over previous versions, the accuracy of the YOLO models has improved substantially.

In the following, I provide a brief overview of YOLO models from the foundational v1 to the v5 model used in the research reported in this thesis.

[14] https://github.com/tensorflow/models/blob/master/research/deeplab/g3doc/model_zoo.md; accessed 24 September 2024.

YOLOv1 [47] operates by dividing an input image into a $S \times S$ (by default, $S = 7$) grid of cells (Figure 1.23) with the notion that when the centre of some object in an image is located within one of these cells, that cell is responsible for detecting the object, whereas other cells are supposed to disregard it to prevent multiple detections. The output of each grid cell includes:

- B bounding boxes (by default, $B = 2$). For each bounding box, 5 predictions are given: x, y, w, h, and confidence scores. (x, y) are the coordinates of the centre of the bounding box, whereas w and h are its width and height; importantly, the former pair of coordinates is given relative to the boundaries of the grid cell, whereas the latter pair of values is given relative to the entire image. The confidence score S_{conf} is defined as:

$$S_{\text{conf}} = p(\text{object}) \times IoU_{\text{pred}}^{\text{truth}} \quad [137], \tag{1.35}$$

  where $p(\text{object})$ is the probability that an object is present is a given bounding box, and $IoU_{\text{pred}}^{\text{truth}}$ is the IoU between the predicted bounding box and the ground truth bounding box.

- C class probabilities, conditioned as $Pr(\text{Class } i \mid \text{Object})$ on the grid cell containing an object. Regardless of the value of B, only one set of C values is predicted per grid cell.

Architecture-wise, YOLOv1 consists of 24 convolutional layers (with kernel sizes of either 1×1 to reduce feature space, or 3×3) followed by two fully connected layers. Apart from the final layer, which uses a linear activation function, all other layers use a leaky ReLU [138] activation:

$$f(x) = \begin{cases} x & \text{if } x \geq 0 \\ 0.1x & \text{otherwise.} \end{cases} \tag{1.36}$$

Overall, YOLOv1 was the fastest object detection model at the time of its release, with the capacity to detect objects in real time. However, while it achieved an mAP of 63.4% on the PASCAL VOC2007 dataset, its localisation error was substantially higher than that of slower but more precise two-stage state-of-the-art detectors at that time such as Fast R-CNN [132]. Furthermore, YOLOv1 suffered from low recall [139].

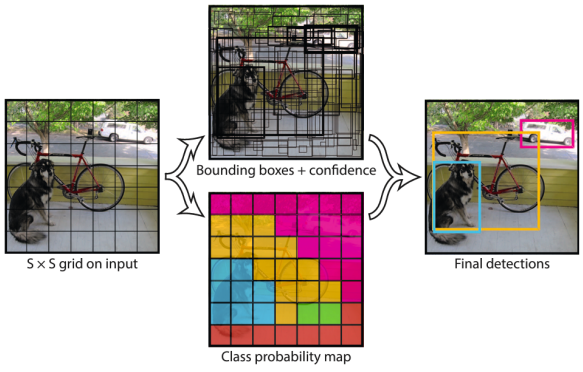


Figure 1.23: General scheme of YOLO. Reproduced from [47].

YOLOv2 [139] features a number of improvements over v1, including batch normalisation for all convolutional layers, a higher input resolution of the backbone classifier (448×448 in v2 vs 224×224 pixels in v1), and the adoption of a fully convolutional architecture by removing dense layers. Yet another substantial change is the introduction of anchor boxes – boxes of predefined size for predicting bounding boxes. Multiple anchor boxes are defined for each grid cell, allowing for more precise capture of the varying sizes and shapes of objects. To use anchor boxes efficiently, their size is found by k-means clustering on the bounding boxes from the training split of the PASCAL VOC2007 dataset, rather than being hand-picked.

The architecture of YOLOv2 is based on the custom backbone architecture Darknet-19, which contains 19 convolutional layers and 5 max-pooling layers. The detection head, which is added to the backbone, consists of four convolutional layers and a passthrough layer. The passthrough layer brings features with finer resolution of 26×26 from an earlier layer to concatenate them with the coarser resolution grid of 13×13, on which detections are predicted, thus improving the access of the detector to fine-grained features of the input image.

Due to the above improvements, YOLOv2 achieved an mAP of 78.6% on the PASCAL VOC2007 dataset, a substantial improvement over YOLOv1.

To reduce localisation errors and improve the detection of small objects, *YOLOv3* [140] transitioned to the new generation of the backbone, Darknet-53. In addition to the substantially increased depth of the model, which consists of 53 convolutional layers vs 19 layers in the YOLOv2 backbone, Darknet-53 features the adoption of strided convolutions instead of max-pooling layers and the addition of residual connections. Another hallmark of YOLOv3 is multi-scale prediction: instead of predicting detections on a single grid, as in YOLOv1 and YOLOv2, there are three different branches of the output of the backbone with increasing resolution – a 13×13, 26×26 grid, and 52×52 grid – which improves the detection of small objects.

Due to the advancements in object detection around the time when YOLOv3 was released, this and the following versions of YOLO have been evaluated on the Microsoft Common Objects in Context (MS COCO) dataset [141], a more challenging benchmark for object detection than the previously used PASCAL VOC 2007, which had become too easy a challenge. YOLOv3-SPP, a version of YOLOv3 that uses a spatial pyramid pooling (SPP) block with different kernel sizes $k \times k$ (where $k \in \{1, 5, 9, 13\}$) and thus has a larger receptive field, achieved an mAP of 36.2% on MS COCO. This was a state-of-the-art result at that time, while YOLOv3-SPP also demonstrated substantially faster inference times than its competitors – RetinaNet [142] and SSD (Single Shot Detector) variants [143, 144].

YOLOv4 and subsequent versions were released by research groups other than the original team led by Farhadi and Redmon, as Redmon stepped away from computer vision research due to concerns about its potential use for military purposes and privacy invasion. YOLOv4, released by Bochkovsky et al. [145], features a number of improvements over previous versions; these improvements are based on extensive experiments conducted by the authors. Thus, to identify an optimal backbone, they explored a range of architectures, including ResNeXt50 [146], already mentioned Darknet-53, and EfficientNet-B3 [107]. By means of conducting experiments as well as due to theoretical reasoning that the most suitable candidate for the backbone would have a larger receptive field size and number of parameters [145], they selected the Darknet-53 architecture, enhanced with cross-stage partial connections (CSPD; [147]), which they named CSPDarknet53. Similar to YOLOv3-SPP, YOLOv4 adds an SPP block to the backbone and uses multi-scale prediction; however, the SPP of

YOLOv4 is further modified, and while in YOLOv3-SPP the features from different layers of the backbone are aggregated for multi-scale prediction using the Feature Pyramid Network (FPN) method [148], YOLOv4 uses the Path Aggregation Network (PAN) method [149] for that. The object detection head in YOLOv4 remains anchor-based, as in YOLOv3.

In addition to architectural changes, YOLOv4 introduces several further improvements, divided into two categories: Bag-of-Freebies, and Bag-of-Specials. The Bag-of-Freebies includes methods that improve the accuracy of inference without slowing it down; cost-wise, they either change the strategy of training the model, or increase the cost of training – but, to emphasise again, not the cost of inference. Such methods include data augmentation techniques such as CutMix [150] and Mosaic data augmentation and other techniques such as Complete IoU (CIoU) loss [151], Cross mini-Batch Normalization (CmBN), and self-adversarial training. The methods in the Bag-of-Specials are the ones that slightly decrease the speed of inference but substantially improve the accuracy of object detection. Some of such methods implemented in YOLOv4 are the Mish activation function [152], Multi-input weighted residual connections (MiWRC) for the backbone, and Distance IoU Non-Maximum Suppression (DIoU-NMS; [151]) for the object detection head.

As a result of these improvements, YOLOv4 achieved an mAP of 43.5% on the MS COCO dataset while maintaining real-time inference speed.

YOLOv5 [153], released shortly after YOLOv4 by Ultralytics under the leadership of Glenn Jocher, marks a shift from Darknet to PyTorch [79] as its primary framework. The adoption of PyTorch, with its larger and more established ecosystem and infrastructure, has facilitated the wider use of YOLO (especially, as Hussain [137] notes, for mobile applications), and has increased the number of open-source contributors to it. YOLOv5 offers four models of different size, from the small YOLOv5s with 7.5 million parameters to the extra-large YOLOv5x with 86.7 million parameters. Other notable changes in this version include the use of the Sigmoid Linear Unit (SiLU; [154]) activation functions in the convolutional layers, the adoption of advanced data augmentation techniques (including those from the open-source Python library Albumentations [155]), and the introduction of the AutoAnchor algorithm. AutoAnchor fine-tunes anchor boxes on the training dataset prior to training the network per se, making them more suitable for a particular dataset, compared to the previous approach of using anchor boxes tuned on the PASCAL VOC2007 dataset for all tasks.

Architecture-wise (see Figure 1.24 for an overview), YOLOv5 employs a modification of the CSPDarknet53 backbone called New CSP-Darknet53, which connects to the object detection head via the Spatial Pyramid Pooling Fusion (SPPF) module – a more computationally efficient equivalent of the SPP module – and the New CSP-PAN module, an updated version of the PAN used in YOLOv4. The object detection head itself remains largely the same as in YOLOv3 and YOLOv4.

Thanks to the above improvements, the largest version of YOLOv5, YOLOv5x, achieved an mAP of 50.7% on the MS COCO dataset with an image input size of 640×640 pixels. Moreover, with an increased input size of 1536×1536 pixels and the use of test-time augmentation, it achieved an mAP of 55.8% on MS COCO.

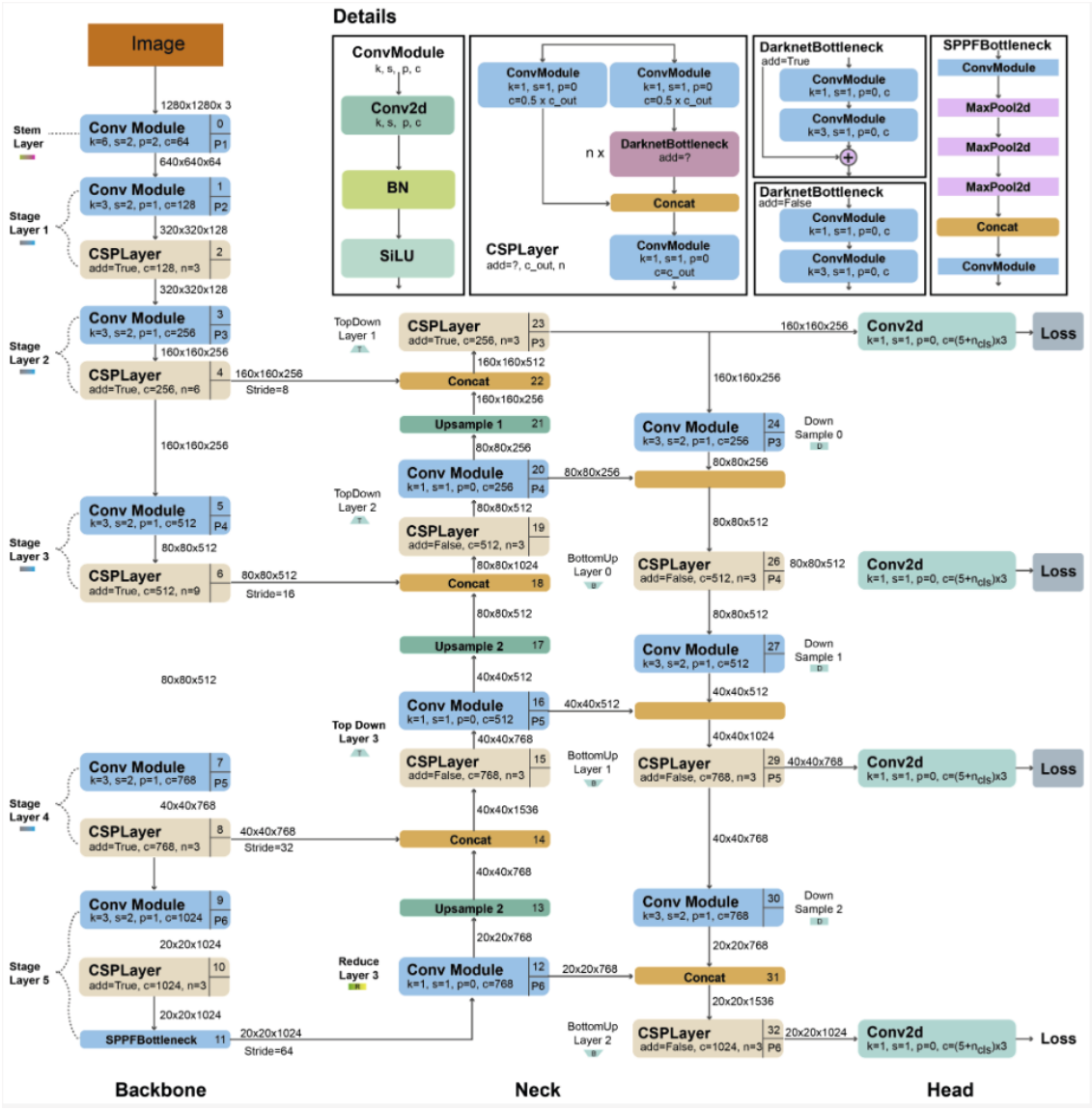


Figure 1.24: Schematic representation of YOLOv5. Reproduced from [156].

1.4 Datasets for image understanding tasks

Having explored image understanding models, I now turn to the discussion of the datasets for training and evaluating them. As the reader will observe, the focus of the discussion broadens, as many insights pertain not only to image understanding but also to deep learning as a whole.

1.4.1 The fundamental role of the data

Much of the attention of the general public and ML researchers and practitioners alike to the progress in the field of AI has been captured by the emergence and performance of new AI models, be those large language models-powered chatbots such as ChatGPT [3], text-to-image generative models such as Midjourney [157] and DALL·E [158], or the image understanding models discussed in earlier sections. While it is hardly surprising, as these models are entities that actually carry out AI tasks, it appears that such a strong focus on the models

may overshadow another cornerstone of AI: datasets for training and evaluation. Neglecting the role of datasets would be particularly erroneous at the current historical stage of AI development, since many AI subfields are presently dominated by deep learning approaches, which are notoriously data-hungry. Therefore, while AI models are indeed at the forefront of AI research, as Denton et al. observe, datasets 'form the background conditions upon which ML research and development operates: they structure how ML practitioners frame and approach problems, inform how progress is defined and tracked within research communities, and create the grounds upon which algorithmic techniques are developed, tested, and ultimately deployed in industry contexts' [159].

As various authors acknowledge (see e.g. [14, 85]), the availability of large datasets for training, evaluating and benchmarking models has been a major enabling factor for the advancements of deep learning. Without such datasets, it would not be possible to fully leverage the capabilities of DNNs. Open access to these datasets has sparked competition to develop better models, democratising deep learning research and enabled researchers to work on image understanding tasks even without the resources to acquire and label data independently.

However, while open datasets such as CIFAR-10 and CIFAR-100 [160], MNIST [40], ImageNet [19], PASCAL VOC [57], MS COCO [141], and Cityscapes [130] have accelerated the development and improvement of image understanding models, real-world applications often require specialised datasets. This leads to several data-related challenges when applying DNNs to real-world problems, such as:

- datasets must be collected, curated and labelled, with the difficulty of labelling increasing from image classification to object detection to semantic segmentation;
- datasets need to be large enough to prevent overfitting and diverse enough to avoid bias;
- data collection may be impeded or made impossible by moral, ethical, and legal concerns;
- due to concept drift, that is, the change between the class distribution at the time of training vs the current class distribution [161], it may be necessary to acquire new data and retrain the model on them.

Characteristic examples of such challenges are biomedical datasets (see Chapter 5): they tend to be comparatively small, making overfitting a significant issue; acquiring additional data is often time-consuming and expensive; labelling may require scientific expertise and a lot of time, as images tend to be large and complex; class distribution tends to be imbalanced [162], with more negative samples (e.g., images of cancer-free tissue samples) than positive samples.

To mitigate the problems of data insufficiency and imbalance, one can use such approaches as transfer learning, fine-tuning, data augmentation, and the use of synthetic data, as discussed in the following.

1.4.2 Transfer learning and fine-tuning

Transfer learning aims to improve the performance of a model on a target domain by leveraging knowledge learned from another domain (or several domains), referred to as the source domain [163]. This allows models to learn effectively even when there is a limited supply

of training data from the target domain; furthermore, the time and computational costs for training a model can also be substantially reduced [164]. As surveys on transfer learning observe [165, 163], this type of knowledge transfer is quite similar to our everyday experience: for instance, if one has already learned to play some musical instrument, it will likely be easier to learn a new instrument, as the skills needed for both tasks are similar, albeit not the same. In the context of deep learning, the typical workflow of implementing transfer learning is as follows [166]:

- retrieve layers from a model trained on the source domain;
- freeze them to prevent information loss during the next steps;
- add several new trainable layers on top of the frozen layers;
- train the new layers on the target dataset;
- optionally: unfreeze the entire model (or part of it) and retrain it on the target dataset using a low learning rate.

The last of these steps, called fine-tuning, allows to gradually fit the knowledge in the unfrozen layers to the target dataset; due to the use of a low learning rate, it is expected that the model adaptation will not result in loss of the information in unfrozen layers.

Transfer learning is often facilitated by the availability of open-access CNN models trained on large datasets. Due to the advantages it offers, it is used in various DL application domains including the ones that the following chapters of the thesis are concerned with, namely, gesture recognition [167], autonomous driving [168], robotics [169], and medical image classification [170].

1.4.3 Data augmentation

Another approach to dealing with the problem of data insufficiency is to use data augmentation techniques. The underlying assumption of data augmentation for image understanding tasks is that it is possible to extract more information for training a model from the original data by artificially inflating the training dataset by means of either warping images or oversampling them [171]; importantly, the manipulations of the images should not change the labelling, as that would make the augmented dataset unusable for training the model. Some examples of image warping (see Figure 1.25) are geometric transformations such as image flipping, random rotations, image cropping, image translation, random erasing, and noise injection; furthermore, data warping may include colour transformations such as colour space changes [172, 171]. The simplest approach to oversampling is image replication, that is, duplicating images belonging to the underrepresented classes; some more advanced oversampling methods are feature space augmentations and mixing images [171]. Furthermore, some authors consider such approaches as GANs [12] to belong to the domain of data augmentation; however, it appears more reasonable to follow the distinction suggested by Nikolenko [9] and reserve the use of the term 'data augmentation' for the methods involving recombination and adaptation of real data while using the term 'synthetic data' for the methods involving creating new data. In any case, the boundary between these two groups of methods is sometimes quite blurry.

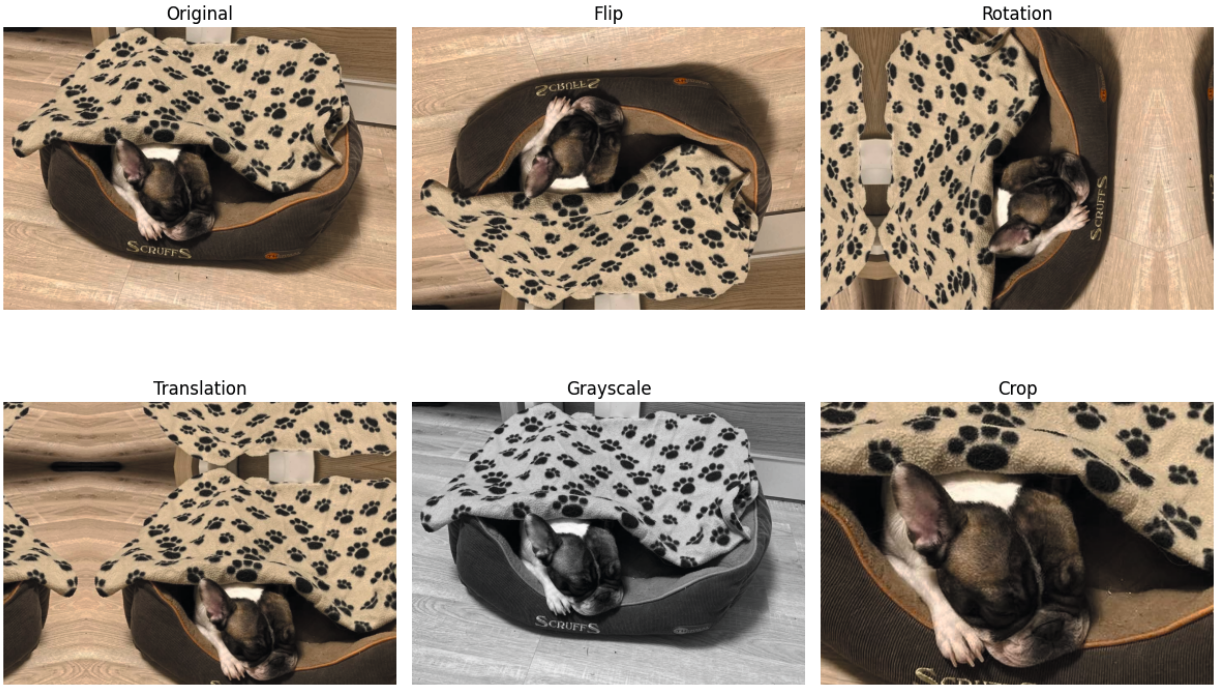


Figure 1.25: Examples of applying image augmentation techniques.

1.4.4 Synthetic data

The most promising approach to solving the data availability problem for training DNNs is arguably the use of synthetic data, that is, artificially generated data that are similar to the data from the target domain. Nowadays, synthetic data is used for a broad range of deep learning applications for image understanding tasks, from the navigation systems of autonomous vehicles and unmanned aerial vehicles to medical image analysis to crowd counting (see Nikolenko [9] for a comprehensive overview). The means of generating synthetic data vary widely as well, including GANs, 3D computer graphics software toolsets such as Blender[15], 3D gaming engines such as Unity[16] and Unreal Engine[17], and assets (e.g., screenshots) from videogames. Importantly, in many cases, it is not necessary to label synthetic data, as labelling can be obtained automatically when generating a synthetic dataset. That makes synthetic data much more convenient to use and often less expensive to acquire than real-world data, which typically need labelling. Other potential advantages of using synthetic data include reducing reliance on real-world data, addressing the underrepresentation of some classes in real-world datasets, and obtaining data that would be challenging to acquire in the real world due to legal obstacles or privacy concerns.

In the context of deep learning, there are several main ways to use synthetic data [9], namely:

- to train DNN models such as classifiers or object detectors solely on synthetic data and then run inference on the target domain data, i.e., on the real-world data;
- to refine (e.g., by means of using generative models such as GANs) synthetic data and then train DNN models, as mentioned in the previous point, on the refined data;

[15] https://www.blender.org/; accessed 10 September 2024.
[16] https://unity.com/; accessed 10 September 2024.
[17] https://www.unrealengine.com/en-US; accessed 10 September 2024.

- to augment real-world datasets with synthetic data in order to obtain a better (e.g., more diverse, or larger, or both) training dataset.

The main obstacle for the applications of synthetic data in deep learning is the domain gap between the real-world and synthetic data, which may lead to a decrease in the performance of the models trained solely on synthetic images. To overcome this issue, one can adhere to the augmentation approach rather than to rely on synthetic data alone, or try to make synthetic images as photorealistic as possible, or employ simulation-to-reality (Sim2Real) domain transfer techniques.

1.5 Concluding remarks

In this chapter, I provided the background for the research on image understanding presented in the rest of this thesis. I began with a broad outline of the field of computer vision, emphasising its connection with various disciplines and subsequently focusing the discussion on the narrower (yet still vast) topic of image understanding. Since my experimental work reported in this thesis was mainly concerned with the three main image understanding tasks, image classification, object detection, and semantic segmentation, I outlined the main metrics for measuring the performance of the models on these tasks and then discussed methods for solving them, in particular, deep learning-based methods, which nowadays have largely succeeded their so-called classical predecessors. I offered an in-depth overview of the particular models I used for my experiments, namely, MobileNetV2, MobileNetV3Large, and EfficientNet-B7 image classifiers, DeepLabv3 semantic segmentation models, and YOLOv5 object detector. My goal was to capture the essential features of these models such as (to mention just a few examples) the use of depthwise separable convolutions in MobileNet models, the notion of scalability underlying EfficientNet models, the use of atrous convolutions in DeepLab models, and predicting both class probabilities and bounding boxes with a single network in YOLO models. To present these models adequately, it was necessary to place them in their historical context, that is, to compare them either with their predecessors from the same model family (e.g., MobileNetV3 with MobileNetV2, and MobileNetV2 with MobileNetV1, or YOLOv5 with the earlier versions of YOLO), or with related architectures (e.g. Xception – 'Extreme Inception' – with Inception). In doing all that, I had to tread a fine line between presenting the information in a rather schematic manner on the one hand and attempting at transforming the chapter into something akin to a deep learning primer on the other hand. As a consequence, tradeoffs were necessary, and while I hope that I have presented some of the concepts and operations underlying the models in sufficient detail as to allow the reader to understand them without resorting to additional sources, some other features of the models were just briefly mentioned, as elaborating on them would require overly lengthy detours. The same considerations apply to the discussion of another essential aspect of deep learning – data for training and validating models: while I highlighted the crucial role of data in deep learning and the impact they have on shaping the practices and goals of the AI community as well as outlined the main means of mitigating the problems of insufficient data and dataset bias such as transfer learning, data augmentation, and the use of synthetic data, the said discussion was unavoidably limited in scope and depth. On the whole, while I would prefer the background chapter of this thesis to be as self-contained as possible, it appears to me that an uneven level of detail was hardly avoidable, as while quite a few deep learning concepts can be understood even without having background in that field, many of the current state-of-the-art CNN models, e.g. the recent versions of YOLO

object detectors, and many advanced approaches to data augmentation and synthetic image generation incorporate a lot of complex solutions and therefore do not lend themselves well to brief yet exhaustive explanations *ab initio*.

Chapter 2

Hand-Washing Movement Classification

In this chapter, I explore applications of CNNs to the classification of hand-washing movements with the goal of designing an automated system of monitoring the quality of hand washing to evaluate and improve compliance with hand hygiene guidelines in a hospital setting. The chapter is based on the following scholarly articles and conference papers:

[173] **M. Ivanovs**, R. Kadikis, M. Lulla, A. Rutkovskis, and A. Elsts, "Automated quality assessment of hand washing using deep learning," *arXiv:2011.11383*, 2020;

[174] M. Lulla, A. Rutkovskis, A. Slavinska, A. Vilde, A. Gromova, **M. Ivanovs**, A. Skadins, R. Kadikis, and A. Elsts, "Hand-washing video dataset annotated according to the world health organization's hand-washing guidelines," *Data*, vol. 6, no. 4, p. 38, 2021;

[175] O. Zemlanuhina, M. Lulla, A. Rutkovskis, A. Slavinska, A. Vilde, A. Melbarde-Kelmere, A. Elsts, **M. Ivanov**, and O. Sabelnikovs, "Influence of different types of real-time feedback on hand washing quality assessed with neural networks/simulated neural networks," in *SHS Web of Conferences*, vol. 131, p. 02008, EDP Sciences, 2022;

[176] A. Elsts, **M. Ivanovs**, R. Kadikis, and O. Sabelnikovs, "CNN for hand washing movement classification: What matters more – the approach or the dataset?," in *2022 Eleventh International Conference on Image Processing Theory, Tools and Applications (IPTA)*, pp. 1–6, IEEE, 2022.

My contribution to these studies was as follows. Although I did not personally conduct data collection or labelling, I was involved in planning and overseeing these stages of the dataset design. Furthermore, I actively participated in processing the datasets and conducted experiments with CNNs; however, it should be mentioned that in the study by Elsts et al. [176], my senior colleague Atis Elsts lead the work. Additionally, I was actively involved in writing both the drafts and the final versions of the manuscripts of all the above studies except the study by Zemlanuhina et al. [175], where my involvement in writing was less direct.

2.1 Introduction

To better understand the need for an automated system of monitoring the quality of washing hands, it is essential to consider the role of hand hygiene in healthcare. Poor hand hygiene is a primary cause of the two major epidemiological issues: the spread of multidrug-resistant (MDR) bacteria, and the spread of infections in a clinical setting [177]. The literature underscores that these issues are of a considerable scale: it has been estimated that infections caused by MDR bacteria result in more than 35 000 deaths in the EU/EEA [178] and 1 270 000 deaths worldwide [179] annually. Given the severity of this problem, the European Centre for Disease Prevention and Control (ECDC) and the World Health Organisation (WHO) have designated limiting the spread of anti-microbial resistance as one of the global healthcare priorities. Furthermore, hands are the main pathway of germ transmission in the healthcare environment [180], which is a large-scale problem as well: it has been estimated that 8.9 million cases of healthcare-associated infections occur each year in hospitals and long-term care facilities in the EU/EEA [181]. More than half of these cases are considered preventable [181], with better hand hygiene being one of the primary means of prevention. To further emphasise the scale of the problems caused by poor hand hygiene, I would like to note that the above estimates predominantly date back to the period before the COVID-19 pandemic, and although it has eventually been established that the SARS-CoV-2 virus is mainly transmitted by air rather than via contaminated surfaces [182], the global health crisis caused by the pandemic has made the issue of proper hand hygiene even more pressing.

As better compliance with hand hygiene recommendations reduces the prevalence of healthcare-associated infections [183], it is crucially important to follow them. The most well-known and widely adopted hand-washing technique is provided in the WHO guidelines [184]; it comprises the six key hand-washing movements (see Figure 2.1) that should be performed for the overall duration from 40 to 60 seconds. Although this technique is often promoted in educational campaigns [185] and is arguably easy to follow, epidemiological research indicates that both general public and medical professionals tend to neglect it [186, 187, 188], thus highly increasing the risk of spreading infections. While healthcare workers are usually well aware that they should maintain high standards of hand hygiene, they may (actually, as observational data indicate, typically tend to) wash hands for a period of time that is too short, omit some of the movements mentioned in the guidelines, or even forget to wash their hands altogether. That often happens due to the notoriously heavy workload and time pressure in healthcare; some other likely reasons for the non-compliance with the guidelines are lack of knowledge, hand skin irritation, and poor accessibility to disinfectants [184]. All in all, the data on the compliance with the requirements for hand hygiene are quite alarming, as research indicates that healthcare workers wash their hands properly in only about 40% of cases [187, 188], which can hardly be considered a satisfactory performance. To give a reference point, the goal of some educational campaigns was to attain compliance with the WHO guidelines of at least 90% [189].

To improve hand hygiene, it is recommended to use a comprehensive approach [190, 186, 191], which includes organising awareness-raising campaigns, posting up reminders in workplaces, ensuring that hand sanitisers are easily available, carrying out regular audits, and assessing and providing feedback on how well hands are washed. However, it should be noted that interventions reported in the literature have not always resulted in sustainable results [192, 193], and improving hand hygiene compliance in healthcare settings still remains a challenge.

One of the ways to address the issue of the quality of hand washing is to monitor it with the purpose of both observing the level of compliance with the WHO protocol and

Figure 2.1: Steps of washing hands with soap and water according to the WHO guidelines. Reproduced from [184].

improving hand hygiene habits of medical personnel [194]. That can be done by such means as tracking hand disinfectant consumption [195], direct real-time observation, or filming and subsequently assessing the videos of how medical personnel wash their hands. Currently, the established standard of monitoring is the direct observation approach [194, 196, 183]; although it offers multiple advantages, it also suffers from certain shortcomings, namely, it is time- and resource-consuming, the observer may fail to notice important details, and the whole direct observation paradigm is potentially subject to the Hawthorne Effect, which implies that a person often changes their behaviour when they know that they are being observed but may revert to their previous behaviour when they know that the observations are over [194, 196].

To address the shortcomings of hand-washing monitoring, it can be automated, i.e., a human observer can be replaced with a system functioning on its own. Such a system would allow to collect a larger amount of observational data with better precision [197]; even more importantly, it would be capable of functioning continuously and uninterruptedly around the clock. While it is possible to use such data modalities as accelerometer data for monitoring hand washing movements, a more prevalent approach (see Section 2.2) is to use video capture from a camera installed above a sink. The main objectives of a monitoring system based

on the video stream processing are to capture visual information (real-time video) of hand-washing episodes and recognise and classify hand-washing movements; furthermore, it would be helpful to enable the system to provide feedback to the user during each hand-washing episode as well as to store the data (either recordings or their summaries) of all hand washing episodes for further analysis [175]. To function successfully, such a system should be capable of:

- recognising and classifying hand washing episodes with sufficient accuracy;
- tackling the domain shift problem, that is, performing well in new locations and with new users;
- preserving the privacy of the users: for instance, it should not transfer the data in such a way that they could be intercepted, as video footage can contain potentially sensitive information;
- running on devices with low power consumption such as edge or mobile devices, as that would facilitate the installation and practical usage of the monitoring system in a real-world scenario.

A promising approach to implementing a system with all the features listed above is to use a smartphone or an edge device as its central unit. The cameras of modern smartphones allow for capturing video with a rather high resolution and frame rates; furthermore, their screens and speakers make it possible to provide different kinds of audiovisual feedback, while their haptic actuators can be used to provide tactile feedback by means of vibration. As for edge devices, they have the advantages of being comparatively inexpensive and more portable and therefore easier to install than traditional computers.

Regarding the software implementation part of a video-based hand-washing monitoring system, the most challenging part of it is an algorithm for recognising and classifying hand-washing movements. As state-of-the-art results in gesture and movement classification have been achieved with CNNs [198], it appears promising to use these algorithms for the purpose of classifying hand-washing movements. Therefore, **the goal** of the research described in this chapter was to develop a CNN-based hand-washing movement classifier for a hand-washing monitoring system. **The main hypothesis** was that lightweight CNNs, i.e., CNNs capable of running on mobile and edge devices, can successfully, i.e., with the accuracy above that of a putative 'naive' classifier, classify hand-washing movements in a real-world hospital setting.

The rest of the present chapter is organised as follows. In Section 2.2, I give an overview of related work, discussing methods for monitoring hand washing. In Section 2.3, I describe datasets containing hand-washing recordings: first, I characterise some such datasets that are publicly available, then I proceed with the descriptions of two datasets – PSKUS, and METC – that were acquired and processed by the team that I was a part of. In Section 2.4, I report the initial experiments on these two datasets, and in Section 2.5 – the experiments in a cross-dataset study, which comprised not only these two datasets, but also the Kaggle dataset [199]. Finally, in Section 2.6, I finish this chapter with some concluding remarks.

2.2 Related work: methods for monitoring hand-washing

I begin a brief overview of related work by highlighting the key points in the surveys on the digital methods for monitoring hand hygiene. Several such surveys that are worth noting are

works by Ward et al. [200], Srigley et al. [197], and a more recent publication by Wang et al. [201].

Of the 42 scientific articles surveyed by Ward et al. [200], fewer than 20% included precise estimates of the efficiency or accuracy of the presented systems, which indicates that this crucial aspect of the automated hand-washing monitoring systems was insufficiently explored at that time. In general, the surveyed systems typically evaluated the quality of hand washing with a simple binary metric such as `done/not done` or used a single timer to detect its total duration, yet neither approach is sufficient to ensure that all palmar surfaces have been properly cleaned [200]. Furthermore, although some of the systems surveyed in that study were classified as fully autonomous, they also included a wearable and mobile component, which could impede their use in a real-world scenario. Ward et al. [200] also identified the issues with cost, patient privacy, and lack of validation of the designed systems.

Srigley et al. [202] provided a systematic efficacy review of hand hygiene monitoring. Most of the studies they surveyed only monitored general compliance with hand hygiene guidelines; only one study also monitored the duration of hand-washing movements. In terms of feedback, only two systems provided individualised feedback and real-time reminders. All in all, Srigley et al. [202] considered the evidence for clinical adoption of electronic and video-based hand hygiene monitoring systems insufficient. It is also worth mentioning that for the future work, they suggested focusing on video-based monitoring approaches.

Wang et al. [201] conducted a comprehensive bibliographic search and identified 89 studies of interest. In 73 of these studies, electronic systems of monitoring hand hygiene were used, which had the following features:

- application-assisted direct observation: 5 out of 73 or 7%;
- camera-assisted observation: 10 out of 73 or 14%;
- sensor-assisted observation: 29 out of 73 or 40%;
- real-time locating system: 32 out of 73 or 44%.

Furthermore, in 21 of the surveyed studies, some type of evaluation of hand hygiene quality was carried out, namely, the compliance with the WHO protocol was evaluated in 14 studies (67%), and the evaluation by means of applying fluorescent substances to the hand surfaces and observing changes in their illumination, which reflects the quality of hand washing, was done in the remaining 7 studies (33%). The authors identified such limitations of electronic hand hygiene monitoring systems as insufficient accuracy, privacy, confidentiality, and usability; in addition to that, they observed that there was a lack of standardised metrics to evaluate performance across such systems.

Before discussing computer vision-based approaches, I would like to note that the surveys mentioned above indicated that it is possible to monitor hand washing using data acquired from non-visual sensors. Some examples of that are found in Wang et al. [203] and Galluzzi et al. [204, 205]. Wang et al. [203] used armband sensor data and achieved high recognition accuracy of the different movements according to the WHO guidelines, especially for the user-dependent model, namely, 96% vs. 82% for the user-independent model. Galluzzi et al. [204, 205] reported the accuracy of 93% using consumer-grade wrist-worn accelerometers when recognising the movements defined by the WHO. It is worth noting that in their experiments, the devices were worn on both wrists, likely making the setup rather cumbersome.

All in all, approaches based on the wearable sensors have several drawbacks: first, they require medical staff to put in extra effort (e.g., to wear a personal wristband device), which can arguably reduce compliance; second, wearable devices can interfere with the hand washing

procedure itself. Therefore, it appears reasonable to concur with the already mentioned viewpoint in Srigley et al. [202] and consider computer vision-based systems to hold a better promise for the design of hand-washing monitoring systems.

One of the earliest approaches to monitoring hand washing based on computer vision was described by Hoey et al. [206], who focused on developing an assistant to aid dementia patients with hand washing. They implemented a particle filter-based classification approach and provided real-time feedback to the user; however, they did not try to distinguish between the different types of hand washing movements. Llorca et al. [207] presented another vision-based system with the explicit goal to provide an automatic hand-washing quality assessment and recognise six different washing 'poses'. As that study predates the era of deep learning, it employed a classical machine learning approach with a complex pipeline involving skin color detection, hand segmentation, a particle filtering model for hand tracking, and an SVM-based classifier for the movement recognition. Although the reported accuracy was high (70.1% to 97.8%, depending on the motion) on a dataset with 4 test subjects, the generalizability of the approach raises some doubts if more test subject were included, especially ones with a darker skin colour, or if the approach were applied to videos taken in real-world conditions.

Since DNNs have achieved good results on many perceptual data-based tasks, there have been a number of studies using them for hand-washing movement classification according to the six-step WHO guidelines. Thus, Yeung et al. [208] presented a computer vision and deep learning-based system for hand hygiene monitoring. The system used depth sensor modality instead of the full video data to preserve privacy, and the data were classified using a CNN. However, the authors aimed to recognise adherence to hand hygiene in general rather than evaluating whether the user performs all specific hand movements. The study by Li et al. [209] investigated the applications of CNN for gesture recognition in general and achieved high accuracy, showing that this approach is suitable for the task. Prakasa and Sugiarto [210] extracted frames from videos, converted the resulting images from RGB channels to hue, saturation, and value (HSV) channels to obtain image component with high contrast on the skin human region (the hue channel), and classified them using a custom CNN classifier. However, the dataset they used consisted of a single instructional video; therefore, the capacity of the model to generalise was not adequately tested. Nagaraj et al. [211] designed a three-stream network architecture, based on classical works on two-stream CNN fusion [212, 213]. The three streams utilized RGB frames, optical flow frames, and histogram of gradients as the inputs, thus incorporating spatial, temporal, and object-level information from the videos. The authors used the full Kaggle dataset [199] to show that their approach performed better than any of the three modalities used separately, and achieved 86.6% accuracy. Remarkably, such accuracy was achieved for 12-class separation[1] and is likely to be even higher if a lesser number of classes were considered. Furthermore, the authors also provided a GitHub repository with an implementation of the fusion classifier. Another recent study by Cikel et al. [214] used the publicly available subset of the Kaggle dataset [199] with hand-washing movements to train and evaluate 3 models consisting of a ResNet-152 [76] CNN encoder and a decoder based on a 3-layer long short-term memory (LSTM; [215]), using as input the RGB frames of the videos for the first one, the optical flow for the second one, and a two-stream input made up of both RGB frames and optical flow for the third one. The RGB network achieved an accuracy of 97.33%. Yamamoto et al. [216] used vision-based systems and a CNN for a different problem, namely, for estimating how well hands were washed. They compared the quality score of the automated system with the

[1]The 12-class problem arises when left-hand and right-hand washing movements are treated as separate classes.

ground-truth data obtained by applying on hands a substance that was fluorescent under UV light. Their results demonstrated that the CNNs are able to classify the hand washing quality with high accuracy.

All in all, as follows from the above brief survey of literature, there have been a number of studies reporting high hand-washing movement recognition accuracy using CNN classifiers based on pretrained models, which are often extended by implementing a multi-stream network architecture, or by incorporating recurrent elements of DNN architecture such as LSTM. At first glance, these results seem to suggest that the problem of automated monitoring of hand-washing is on the verge of being solved, if not already resolved, yet they should be taken with some caution, since the surveyed studies typically utilised datasets collected in lab environments, such as the Kaggle challenge dataset [199], and therefore it is far from certain whether their findings can translate well into applications in real-world scenarios.

2.3 Hand-washing recording datasets

Only a few datasets featuring recordings of hand washing were publicly available at the time when the research reported in this chapter was conducted. Thus, the Kinetics Human Action Video Dataset [217] by Google contains 916 videos of washing hands, whereas the STAIR Actions dataset [218], which consists of more than 100 000 videos, contains around 1 000 videos that are related to washing hands. The Kaggle data science website[2] hosts the already mentioned Kaggle Hand Wash Dataset [199], which consists of 292 hand-washing episodes labelled according to the WHO guidelines; however, only 25 of these episodes are publicly available. The episodes were recorded at a resolution of 720×480 pixels at the frame rate of 30 frames per second (FPS); the recordings were done in lab conditions with good lighting and the movements being meticulously correct. Altogether, these three datasets have several substantial limitations: none of them has more than 1000 hand-washing videos, they do not feature medical professionals or a clinical setting, and only the Kaggle dataset, the publicly available part of which is rather small, comes along with labelling according to the WHO guidelines. Therefore, it appears that the public availability of the data for training ML-driven classifiers of hand-washing movements is rather limited.

To address the lack of data for training and evaluating hand-washing movements classifiers, a group of Latvian epidemiologists and their support team collected and annotated two datasets of hand-washing videos in a clinical setting: the PSKUS dataset, and the METC dataset. To make the datasets suitable for ML tasks, the data collection and annotation were done in cooperation with the research team at EDI, of which I was a part. Afterwards, I conducted classification experiments by training and evaluating CNN models on these datasets. Therefore, I describe both these datasets in the following.

2.3.1 PSKUS dataset

Data acquisition

The PSKUS dataset was collected in the summer of 2020 at one of the largest hospitals in Latvia, Pauls Stradins Clinical University Hospital (*Paula Stradiņa klīniskā universitātes slimnīca (PSKUS)* in Latvian, hence the name of the dataset) using a custom Internet-of-Things (IoT) system (Figure 2.2). Each instance of the system consisted of one or several

[2] https://www.kaggle.com/; accessed 10 June 2024.

(a) (b)

Figure 2.2: Data acquisition setup: (a) prior to deployment (b) deployed in hospital.

AirLive IP POE 100CAM cameras or Axis M3046V IP cameras installed above sinks for washing hands and connected to Netgear 5-Port PoE Gigabit Ethernet switch GS305P, and a Raspberry Pi 4 device with a microSD card that stored the Raspberry operating system, a custom data acquisition program, and acquired video files. Such IoT systems were deployed in nine different locations simultaneously, with one location corresponding to one sink. In total, there were 12 cameras, as some of the Raspberry Pi devices had more than one camera attached to them, which made it possible to record hand washing at a single sink simultaneously from different angles. The locations where the cameras were deployed included a hospital neurology unit, surgery unit, an intensive care unit, and other hospital units in Pauls Stradins Clinical University Hospital.

The cameras recorded all continuous movements within their field of view; the video stream was captured at the frame rate of 30 FPS and the resolution of either 640×480 or 320×240 pixels. To filter out irrelevant movements of short duration, e.g., a person passing by, a recording was only started in the case when motion was detected for 3 seconds continuously; as a result, the videos in the dataset may miss up to the first 3 seconds of each hand-washing episode. Furthermore, videos shorter than 20 seconds were not saved by the recording system to reduce the number of false positives in motion detections.

The recorded data were manually transferred to a central server at Pauls Stradins Clinical University Hospital by bringing microSD cards from the Raspberry Pi devices and uploading the data from them onto the server.

Overview and structure

The PSKUS dataset consists of video files along with their annotations in CSV and JSON formats. Table 2.1 presents an overview of the dataset.

The folders and files in the dataset are structured as follows:

```
DataSets
\- Dataset1
  \- Videos
    \- 2020-06-27_11-57-25_camera104.mp4
    \- 2020-06-28_18-28-10_camera102.mp4
```

```
    \- ...
  \- Annotations
    \- Annotator1
      \- 2020-06-27_11-57-25_camera104.csv
      \- 2020-06-27_11-57-25_camera104.json
      \- ...
    \- Annotator2
      \- 2020-06-27_11-57-25_camera104.csv
      \- 2020-06-27_11-57-25_camera104.json
      \- ...
\- Dataset2
  \- Videos
    \- ...
  \- Annotations
    \- ...
...
summary.csv
statistics.csv
```

Each video file has one or more annotations created by one or several annotators. For the sake of convenience, annotations are provided in two formats, although most of the information in the CSV and JSON files overlaps. Video files and their annotations have matching names: thus, the video file *name*.mp4 has annotations in both *name*.csv and *name*.json. Additionally, there are several files providing the overview information: the file summary.csv contains a summary of the dataset, and the file statistics.csv contains the key metrics for each hand-washing episode in the dataset.

Table 2.1: Overview of the PSKUS dataset.

Property	**Value**
Frame rate (FPS)	30
Resolution	320 × 240 or 640 × 480
Number of videos	3 185
Number of annotations	6 690
Total washing duration (seconds)	83 804
Movement 1–7 duration (seconds)	27 517
Episodes with ring present	440
Episodes with armband or watch present	127
Episodes with long nails present	58

Labelling

Annotators – infectious disease experts involved in the research project, other medical professionals, and volunteers, including Riga Stradins University students – were given access to

the video files on the server to label the data. At the preliminary stage of analysis, the annotators vetted the videos to remove the files that did not include actual hand-washing episodes. Each annotator did that independently based on the guidelines provided to them; as a result, some files have been vetted out by one annotator, but not by the other one(s). Therefore, in the final dataset, some videos may have annotations in one folder (e.g., `Annotator1`) but not in another (e.g., `Annotator2`). Furthermore, the annotators are anonymised in the final version of the dataset, and the folder `Annotator1` in one part of the dataset is not necessarily annotated by the same person as the folder `Annotator1` in a different part of the PSKUS dataset.

Afterwards, annotators labelled hand-washing movements in the videos using a custom annotation program developed in the Python programming language using OpenCV computer vision library, which allowed to assign to each video frame the following information: (1) whether hand washing was visible in the frame, and (2) which (if any) of the movements defined in the WHO guidelines hand washing corresponded to. The annotation guidelines were developed by epidemiologists involved in data collection and acquisition on the basis of the WHO guidelines. In total, seven different hand washing movements were defined as recommended by the WHO; with the addition of the `other` movement, the list of movement labels is given in Table 2.2.

Table 2.2: Movement codes for hand-washing movements.

Movement code	**Movement description**
1	Palm to palm
2	Palm over dorsum, fingers interlaced
3	Palm to palm, fingers interlaced
4	Backs of fingers to opposing palm, fingers interlocked
5	Rotational rubbing of the thumb
6	Fingertips to palm
7	Turning off the tap with a paper towel
0	Other hand-washing movement

According to the annotation guidelines provided for the annotators:

- Code from 1 to 6 is used to denote a correctly performed hand-washing movement that corresponds to one of the movements from the WHO guidelines.
- Code 0 is used to denote either a movement from the WHO guidelines that is not performed correctly, or any washing movement that is not defined in the WHO guidelines.
- Code 7 is used to denote the process of correctly terminating the hand-washing episode. Specifically, for `movement 7` to be recorded, the tap has to be turned off in the following way: the person washing their hands has to take the towel, dry their hands with it, and then turn off the tap with the towel rather than with a bare hand, thus preventing potential contamination from the tap handle.
- Frames that do not capture hand-washing are labelled accordingly, i.e., with `is_washing` set to zero.

Additional annotations applied to the whole video rather than to separate frames indicate whether the person that is washing hands is wearing a ring, watch, armband, or has long

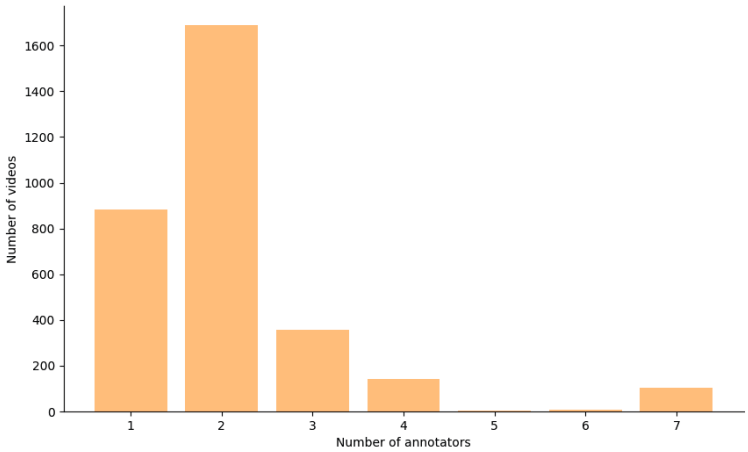


Figure 2.3: Number of annotators per video in the PSKUS dataset.

artificial nails, as these items may interfere with the quality of hand washing [219, 184, 220]. These annotations are as follows:

- `A` - armband or watch.
- `R` - ring.
- `N` - long nails.

To increase the reliability of the annotations, the majority of files in the dataset are labelled by more than one annotator (Figure 2.3). In particular, out of 3 185 videos, 882 were labelled by one annotator, 1 691 by two annotators, 355 by three annotators, 141 by four annotators, 4 by five annotators, 8 by six annotators, and 104 by seven annotators. Some statistics on the inter-annotator agreement are as follows: frames that were annotated by two annotators have 91.23% agreement on the `is_washing` label, and for those frames where both annotators have assigned a value of 1 to `is_washing`, there is further agreement of 90.06% on the movement code. As for cases of inter-annotator disagreement, the following reasons were identified in the dataset:

- short-term disagreement between the labels typically exists at time points when washing movements change;
- `movement 1` and `movement 3` look quite similar and can be hard to distinguish when filmed at an angle;
- the interpretation of what constitutes `movement 7` has been different between the different annotators;
- some of the videos have low light levels.

Data distribution

After splitting the videos into frames and keeping only those where two or more annotators assigned the same label, the data in the PSKUS dataset has the following distribution by classes: class 0 - 922 724 frames, class 1 - 61 844 frames, class 2 - 93 924 frames, class 3 - 41 489 frames, class 4 - 53 188 frames, class 5 - 50 591 frames, class 6 - 51 112 frames (Figure 2.4). Since class 0 is heavily overrepresented, only 20% of its frames, i.e., 207 732 frames, were used for training the models (see Sections 2.4 and 2.5).

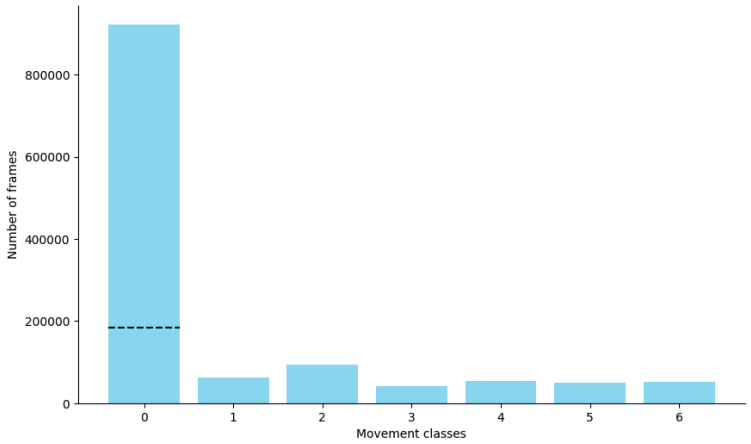


Figure 2.4: Distribution of the frames in the PSKUS dataset by movement classes. The dotted line indicates that only 20% of the frames from class 0 were used in the experiments.

2.3.2 METC dataset

As a key part of the study by Zemlanuhina et al. [175], another dataset of hand-washing videos, the METC dataset, was acquired and labelled. The primary objective of that study was to determine which type of real-time feedback most efficiently motivates medical staff to follow the WHO guidelines for washing hands; while this objective is essential for the development of an efficient hand-washing monitoring system, it lies beyond the scope of this thesis, and I therefore report only the relevant parts of the research in Zemlanuhina et al. [175] in the present section.

Data acquisition

The METC dataset was collected during a user feedback evaluation study [175] in July and August 2021. The study was conducted at the Medical Education Technology Centre (METC; hence the name of the dataset) of Riga Stradins University and involved 72 participants. All participants were healthcare specialists: employees of Riga Stradins University, physicians, and medical students; therefore, they were familiar with the requirements for maintaining proper hand hygiene in a clinical setting. Each participant took part in one hand-washing session, in the course of which they performed 3 hand-washing trials, receiving different types of feedback on how they washed hands each time, namely:

- In the first trial, there was *no guidance* for participants on how to wash their hands, i.e., they washed their hands using any sequence of hand washing movements they considered appropriate for as long period of time as they considered sufficient.

- In the second trial, participants washed hands in a *semi-guided mode*, that is, they were assisted by a custom-made smartphone application that provided real-time feedback indicating which hand-washing movements were recognised and displayed visual information when a specific movement was performed for more than 7 seconds, i.e., for a sufficient length of time. However, the application did not guide the participants or informed them that the procedure should be performed for a certain period of time.

- In the third trial, participants washed hands in a *guided mode*, that is, the application explicitly instructed them which movements and for how long a period of time to

perform.

Before each hand-washing session, participants were familiarised with the functionality of the application and the feedback (if any) it would provide in each of the trials. Afterwards, participants treated hands with an UV-active gel covering the surface of the palm, the dorsum of the hand, the regions between fingers, and fingertips; subsequently, images of the treated hands placed under the UV lamp uncovering all contaminated regions of the hand were taken. After each washing trial, new images with the hands after washing were taken under UV light exposure. As a result, by comparing the images of hands before and after the washing trial, it was possible to evaluate the quality of hand washing.

Videos of the hand-washing sessions were recorded for further use; notably, all experiments took place in the same location, i.e., over the same sink. As the use of the CNN classifier in the hand-washing monitoring system was still in the process of development at the time when the study was conducted, hand-washing movement recognition was performed by a human operator, an expert in hand hygiene, rather than the automated system. The human operator monitored how participants washed their hands in real time; the monitoring was done on a PC located in a nearby room and relied on a live video stream from the camera of the monitoring setup. The video streaming was facilitated by a PC application based on a Flask web server, which was also used to annotate the videos.

Overview and structure

The METC dataset consists of video files along with their annotations in JSON format. Table 2.3 presents an overview of the dataset.

The folders and files in the dataset are structured as follows:

```
DataSets
\- Interface_number_1
    \- Annotations
      \- 2021-06-30_09-56-19-11labots.json
      \- 2021-06-30_10-28-33-2-1+labots.json
      \- ...
    \- Videos
      \- 2021-06-30_09-56-19-11labots.mp4
      \- 2021-06-30_10-28-33-2-1+labots.mp4
      \- ...
\- Interface_number_2
    \- Annotations
      \- ...
    \- Videos
      \- ...
summary.csv
statistics.csv
```

Each video file has an annotation, provided in a JSON file. Video files and their annotations have matching names: thus, the video file *name*.mp4 has annotations in *name*.json.

Additionally, there are several files providing overview information: the file `summary.csv` contains a summary of the dataset, and the file `statistics.csv` contains the key metrics for each hand-washing episode in the dataset.

Table 2.3: Overview of the METC dataset.

Property	**Value**
Frame rate (FPS)	≈ 16
Resolution	640×480
Number of videos	212
Number of annotations	212
Total washing duration (seconds)	13 870
Movement 1–7 duration (seconds)	9 144
Episodes with ring present	0
Episodes with armband or watch present	0
Episodes with long nails present	0

Note that the frame rate of the videos was slightly variable, as they were reconstructed from sequences of JPG images taken with the maximum frame rate supported by the capturing devices.

Labelling

The ground truth annotation of the videos was done in real time by a human operator, an expert in hand hygiene. The annotation was carried out as follows: when the operator recognised a hand-washing movement defined in the WHO guidelines, he would mark it accordingly. In general, labelling was done in the same manner as for the PSKUS dataset. In addition to that, the operator also simultaneously evaluated the quality of each recognised movement: the operator marked the movement with `OK` if the movement was performed correctly or with `!` if there were any inaccuracies in performing the movement[3].

It should be noted with respect to the quality of hand washing in the dataset that while the participants in the study were knowledgeable medical staff and were instructed to complete the task of washing hands to the best of their ability, imperfect execution of the protocol was still present: some of the participants did not even complete all six basic hand-washing movements, and none of the 72 participants performed movement 7 properly.

Data distribution

The data in the METC dataset have the following distribution by classes: class 0 - 64 980 frames, class 1 - 19 967 frames, class 2 - 22 651 frames, class 3 - 20 221 frames, class 4 - 17 576 frames, class 5 - 19 915 frames, class 6 - 24 682 frames (see Figure 2.5). Since class 0 is overrepresented, only 50% of class 0 frames, i.e., 32 490 frames, were used for training the models (see Sections 2.4 and 2.5).

[3]This type of labelling, i.e., `OK` vs. `!`, is not provided in the publicly available version of the METC dataset, but its summary and analysis are provided in [175].

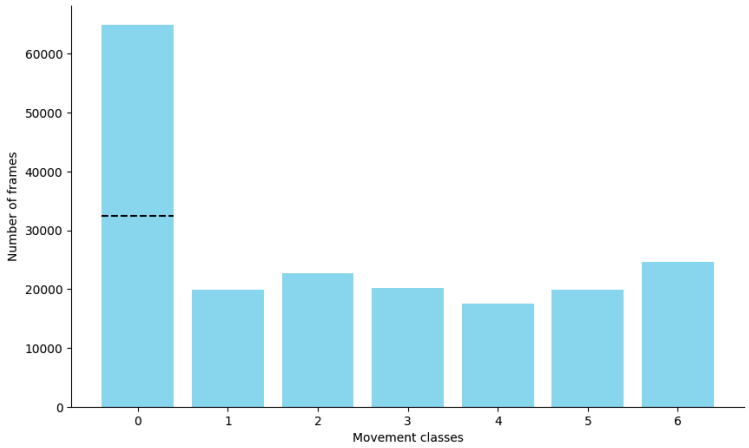


Figure 2.5: Distribution of the frames in the METC dataset by movement classes. The dotted line indicates that onlly 50% of the frames from class 0 were used in the experiments.

2.4 Initial experiments on PSKUS and METC datasets

To obtain initial classification results on the PSKUS and METC datasets, I conducted experiments by training and evaluating MobileNetV2 [105] CNN models on them. I describe the methodology and results of these experiments in the following.

To preprocess the PSKUS dataset for experiments, I first split the data into trainval and test subsets. The data for the test subset were images from one particular location, the emergency ward, and the rest of the data was used as the trainval subset. Such a split of the data was done to make sure that the data for training and evaluating the model do not come from the same location. Afterwards, during training the model, the trainval subset was randomly split for training and validation with the ratio of 80/20. As the METC dataset was acquired in a single location, preprocessing it for experiments was more straightforward: first, I divided the dataset into trainval and test subsets with the ratio of 80/20 and then, during the training of the models, the trainval subset was further randomly split for training and validation with the ratio of 80/20.

The training was done on a PC with Intel Core i7-12700K CPU, NVIDIA RTX 3090 GPU, and Windows 11 OS using TensorFlow v.2.16.1 and Keras v3.0.5. I conducted two experiments per each dataset. The architecture of the model was identical across all experiments: the base MobileNetV2 model with the weights pretrained on the ImageNet dataset was extended with a preceding data augmentation layer featuring random rotations by up to 20 degrees and horizontal flips, and three follow-up layers:

- `GlobalAveragePooling2D` layer;
- `Dense` (i.e., fully connected) layer with 128 neurons, ReLU activation, and dropout rate of 0.5 to avoid overfitting;
- `Dense` layer with 7 neurons and softmax activation function.

In the first experiment, I froze the base MobileNetV2 model and only trained the layers added on top of it for 30 epochs with the Adam optimiser (learning rate = 0.001), the categorical crossentropy loss function, and early stopping enabled so that the model would stop training if there was no validation loss improvement for 10 consecutive epochs. When the training was finished, the best (in terms of accuracy on the validation subset) checkpoint was retrieved and evaluated on the test data. The results of evaluating the model trained on the PSKUS

dataset on the test subset of the same dataset were as follows: test accuracy = 55.65%, precision = 24.06%, recall = 17.36%, and F_1 score = 16.72%. See also the confusion matrix in Figure 2.6.

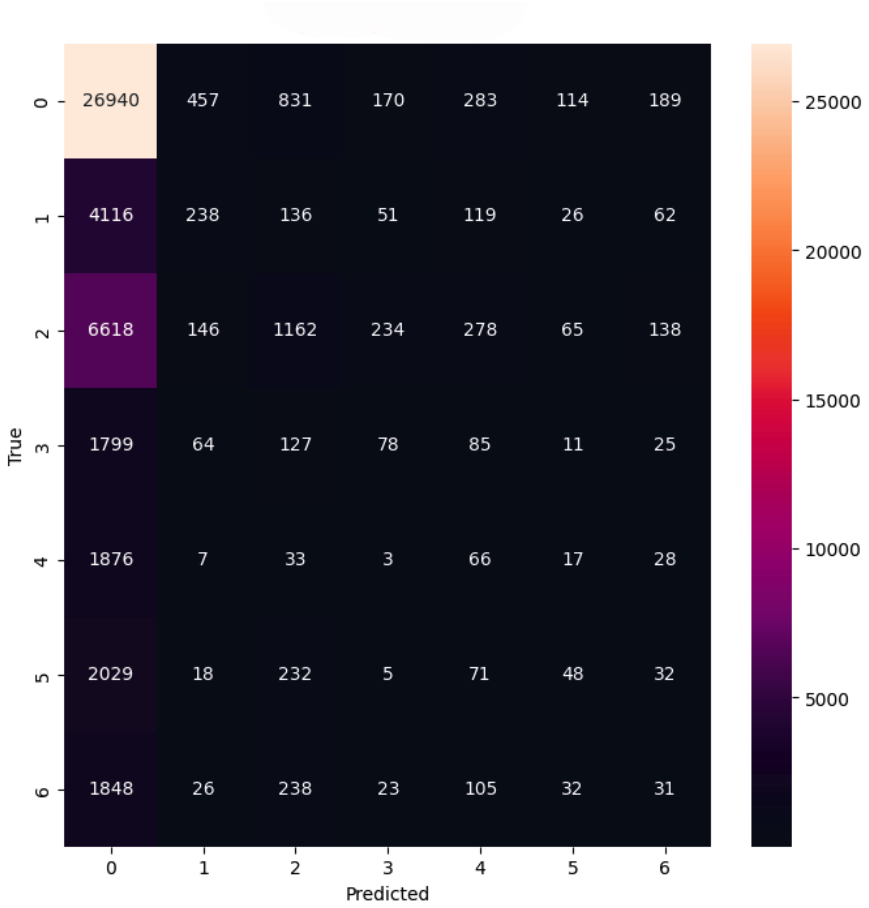


Figure 2.6: Results of the initial experiment training MobileNetV2 (top layers only) on the PSKUS dataset: confusion matrix showing the evaluation on the test subset.

Evaluation of the model trained on the METC dataset on the test subset yielded test accuracy of 50.96%, precision of 53.83%, recall of 49.12%, and an F_1 score of 49.74%. The confusion matrix from this experiment is shown in Figure 2.7.

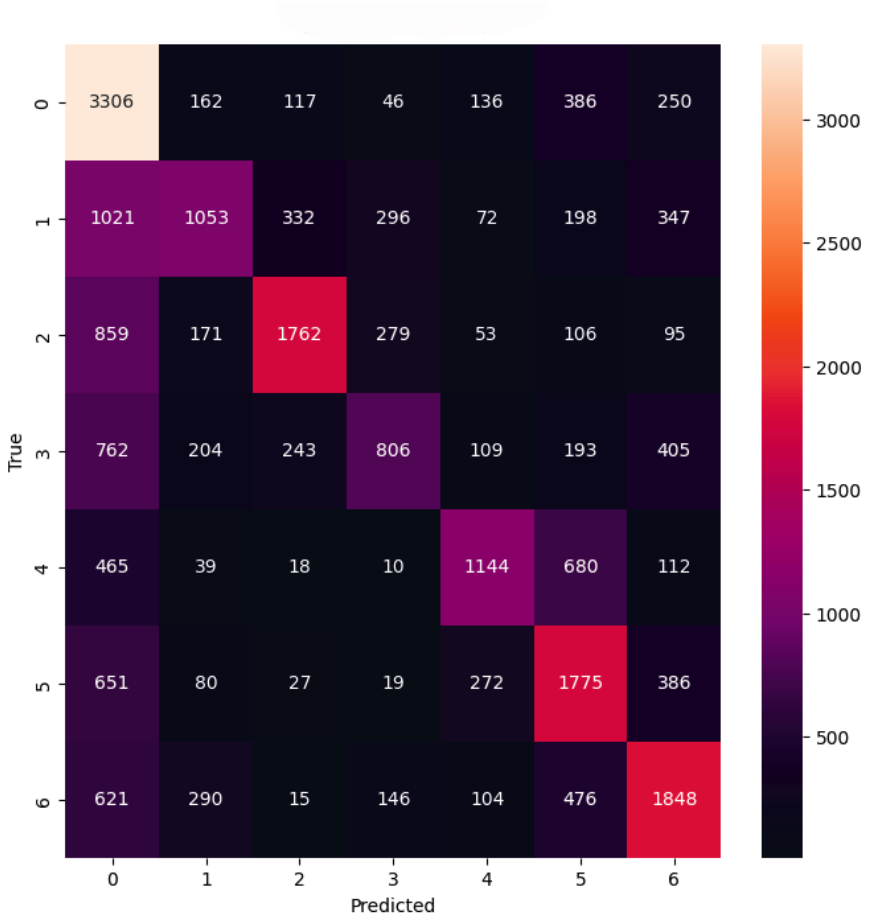


Figure 2.7: Results of the initial experiment training MobileNetV2 (top layers only) on the METC dataset: confusion matrix showing the evaluation on the test subset.

Some observations that follow from these results are as follows. First, it is obvious that F_1 score is a more suitable metric than accuracy for such an unbalanced (due to the predominance of class 0) dataset as PSKUS. Second, it appears that the capacity of the models for generalisation on the unseen data is starkly different for the PSKUS vs METC dataset: while the F_1 score of the model trained on the former dataset was just slightly above random guessing, the performance of the model trained on the latter dataset was substantially better.

In the second experiment, I trained MobileNetV2 models for 10 epochs with the same hyperparameters as in the first experiment and then continued training for additional 30 epochs with all model layers unfrozen, i.e., being trainable. To avoid the instability of the model after unfreezing, the learning rate was decreased from 0.001 to 0.0001. The results of the experiment with such parameters on the PSKUS dataset were as follows: test accuracy = 56.71%, precision = 35.27%, recall = 16.73%, and F_1 score = 15.52%. See also the confusion matrix in Figure 2.8.

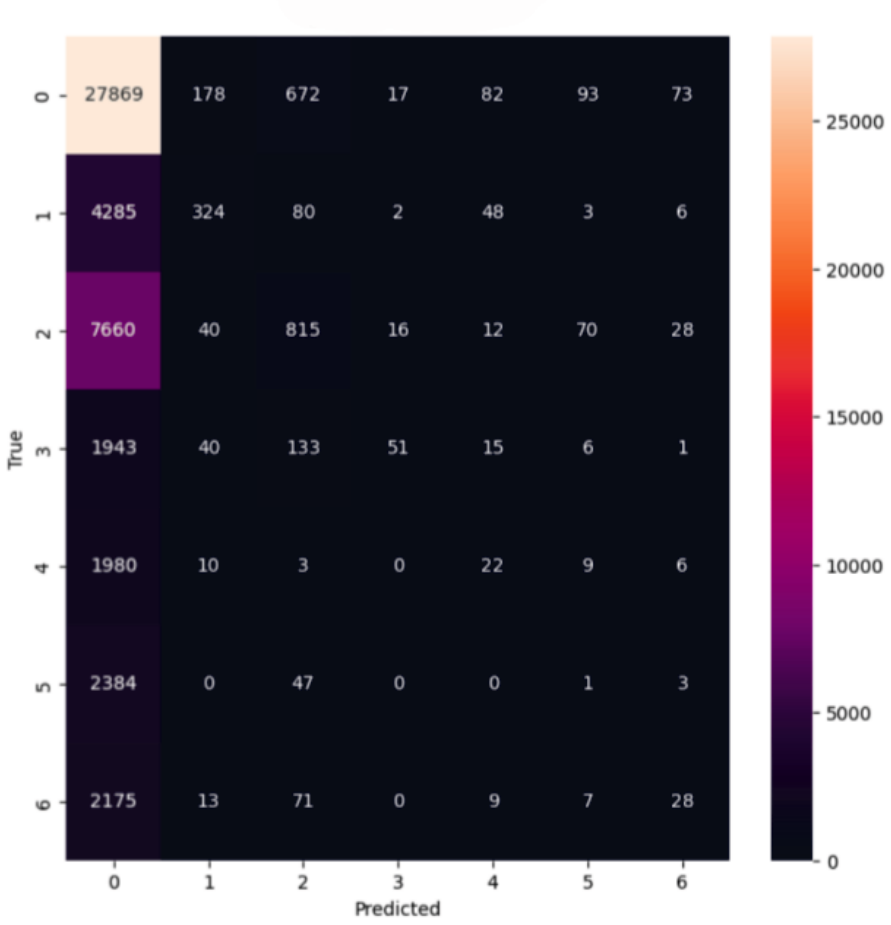


Figure 2.8: Results of the initial experiment training MobileNetV2 (the model fully retrained) on the PSKUS dataset: confusion matrix showing the evaluation on the test subset.

Similar experiment on the METC dataset yielded the following results: test accuracy = 63.75%, precision = 68.37%, recall = 62.56%, and F_1 score = 63.89%. The confusion matrix from this experiment is shown in Figure 2.9.

As it follows from the results of the second experiment, the MobileNetV2 model trained on the METC dataset benefited from additional training with all the layers being unfrozen, as its F_1 score improved by more than 14% in comparison with the model trained with just added layers being unfrozen. However, for the MobileNetV2 model trained on the PSKUS dataset, the outcome of the same additional training was entirely different, as it led to a decrease in the F_1 score by $\approx 1\%$. Remarkably, it is not the case that the model in question did not learn anything during the training, as its loss and accuracy improved consistently during the training phase. However, the model seemed to be incapable of generalising the learned knowledge when it came to classifying images coming from the new, previously unseen locations, i.e., the data in the test dataset. Another likely explanation for the worse performance of the model trained on the PSKUS dataset is that the consistency of the labelling by multiple annotators was lower than that of a single annotator, as it was in the case of the METC dataset. As a consequence, less consistent labelling made it more difficult for the model to learn how to classify the images.

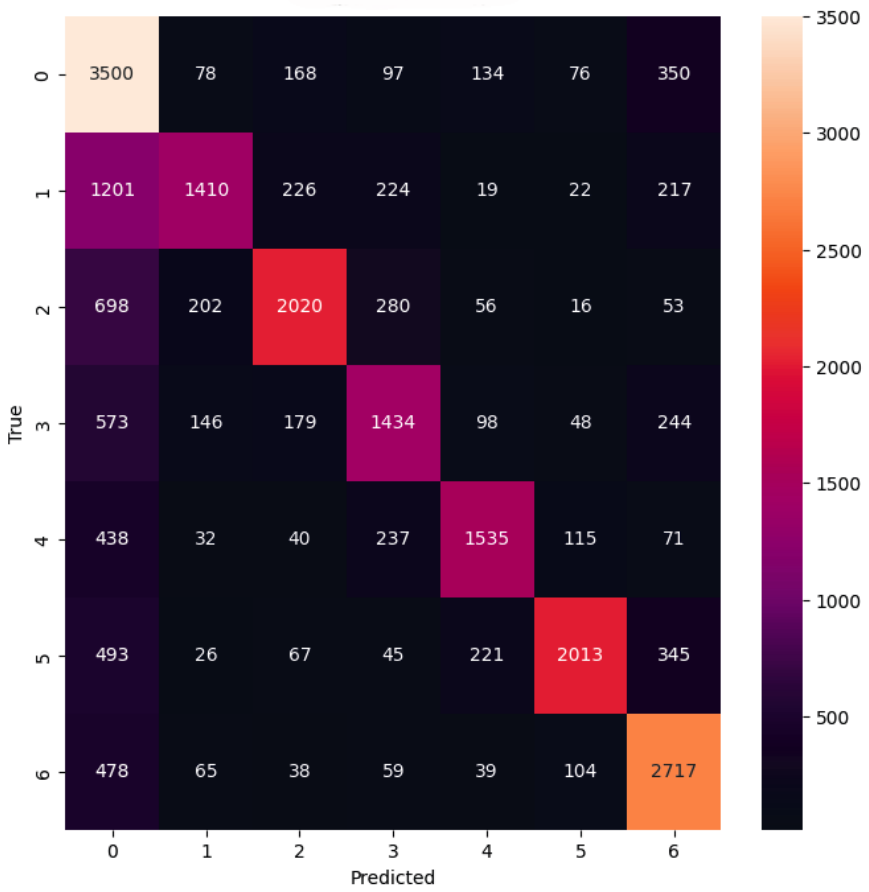


Figure 2.9: Results of the initial experiment training MobileNetV2 (the model fully retrained) on the METC dataset: confusion matrix showing the evaluation on the test subset.

2.5 Cross-dataset study of CNN performance

As reported in the previous section, I conducted a number of experiments by training CNNs on the two real-world datasets, PSKUS and METC, which were of different sizes and levels of complexity. The MobileNetV2 model trained on the PSKUS dataset demonstrated a performance similar to that of a putative 'naive' classifier on the test data, whereas its counterpart trained on the METC dataset demonstrated substantially better results. However, even the performance of the latter model was still not as impressive as the results reported in the literature (see Section 2.2). One possible explanation for such a discrepancy is that there is a substantial gap between what a CNN classifier can achieve on datasets collected in a lab conditions with ideal lightning, the same position of camera and the same sink being used across all trials, and emphatically correct and uniform performance of hand-washing movements on the one hand, and datasets collected in far-from-ideal real-world conditions (PSKUS) or nearly real-world conditions (METC) on the other hand. To investigate whether that was the case, we[4] conducted a study [176], which I report in the present section; the goal of the study was to investigate whether the same model architectures that, as reported in the literature, perform well on smaller and simpler datasets would also perform well on larger and more complex datasets. Since the CNN models reported in the literature on the classification

[4]In this section of the chapter, I mainly use 'we' rather than 'I', as the experiments reported here were done collaboratively by my colleague Atis Elsts and me.

of hand-washing movements are often extended with a multi-stream network architecture or recurrent elements such as LSTM, we also utilised such extended architectures in our experiments in addition to the single-frame classifiers that were used in the experiments reported in the previous section. Furthermore, in this study, we continued using lightweight classifiers such as MobileNets that can run on mid-range smartphones in inference mode without requiring powerful hardware such as dedicated graphics accelerators, as we considered such an approach to be more suitable for designing the real-world hand-washing monitoring systems in the future.

2.5.1 Datasets and data preprocessing

(a) (b) (c)

Figure 2.10: Sample images featuring movement class 1 (rubbing palm to palm) from: (a) the Kaggle hand-washing dataset; (b) the METC dataset; (c) the PSKUS dataset.

In our experiments, we used three datasets: the PSKUS dataset, the METC dataset, and the publicly available part of the Kaggle hand-washing dataset. Sample images from each datasets are shown in Figure 2.10; furthermore, the main statistics of each dataset are summarised in Table 2.4.

Table 2.4: Main characteristics of the datasets used in the cross-dataset study.

Parameter	**Kaggle**	**METC**	**PSKUS**
Washing episodes	25	213	3 185
Users	$\leq$25	72	many
Locations	$\leq$25	1	9
Environment	Lab	Lab	Real-life
Resolution	720×480	640×480	640×480, 320×240
Frame rate (FPS)	30	≈ 16	30

To further expand on the differences between the three datasets, the Kaggle dataset features high-quality scripted hand-washing videos corresponding to each of the hand-washing steps defined by the WHO [221]. That makes it markedly different from the METC dataset, where some mistakes when executing hand-washing movement are still present despite preliminary instructions given to the participant. The difference is even more conspicuous in

the case of the PSKUS dataset featuring medical staff washing their hands as part of their normal job duties, as the videos in it were filmed in the real-life conditions and therefore include hand washing positions that are partially out of the frame or partially occluded as well as low and variable lighting conditions. All in all, in the PSKUS dataset, imperfect and incomplete execution of the washing steps is a rule rather than an exception.

To make it possible to compare the results across the datasets, the latter were preprocessed accordingly. The original labelling in the Kaggle dataset distinguishes between washing left hand and right hand; since there is no such distinction in the two other datasets, the respective classes were merged so that left-hand and right-hand movements would belong to the same class. Furthermore, as the wrist washing movement (step 7, according to the WHO guidelines) is not labelled in the other two datasets, the respective class was merged with class 0, which corresponds to the `other` movement. As the videos in the METC dataset were annotated by a human operator in real time, there was some delay between the actions in the videos and the operator's response. To eliminate it, a 1-second long video segment was removed every time the class label would change in the data stream. Finally, as for most videos in the PSKUS dataset there were annotations available by multiple annotators, the same approach as in the experiments reported in Section 2.4 was used: only those parts of that dataset for which two or more annotators had assigned matching class labels were used for training and evaluating CNN models.

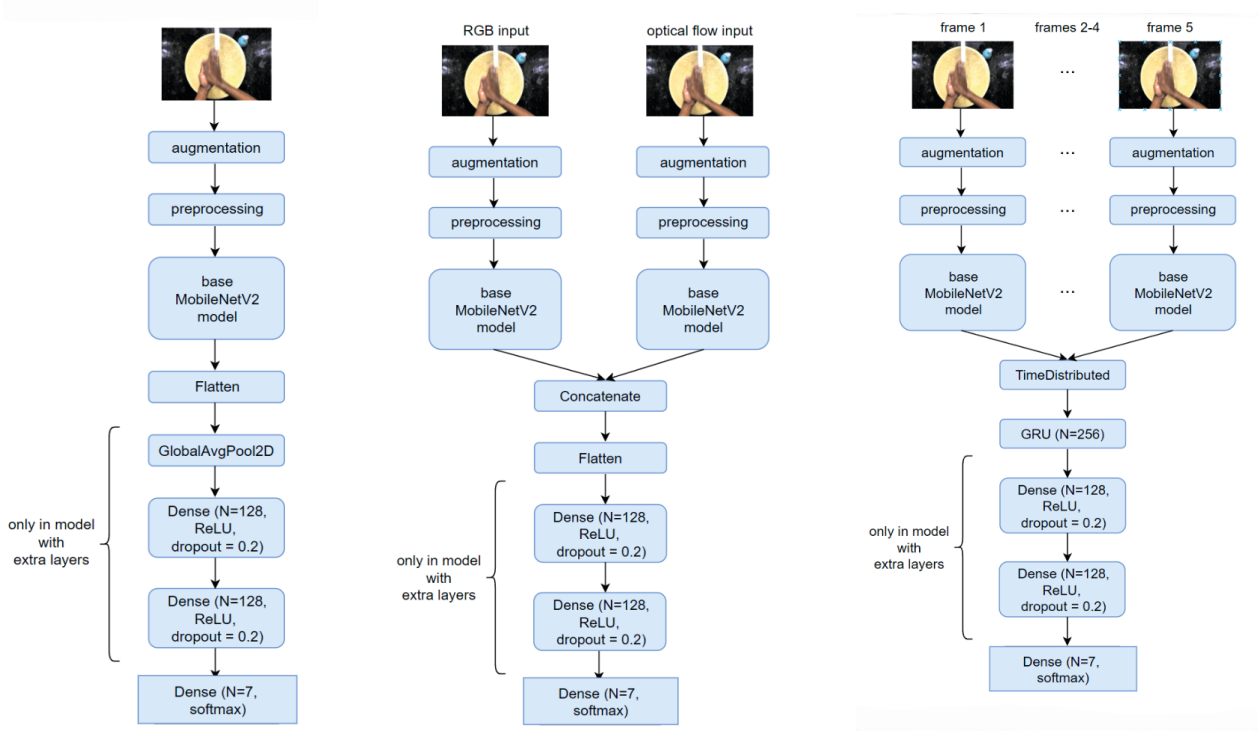
2.5.2 Experiments: methodology and results

Model architectures

The baseline model (see Figure 2.11 (a)) used in the experiments was MobileNetV2 [105] with the weights pretrained on ImageNet [19] and the following architecture:

- input layer;
- data augmentation layer featuring random rotations by up to 20 degrees and horizontal flips;
- preprocessing layer;
- baseline MobileNetV2 model;
- `Flatten` layer;
- included only in the model with extra layers: `GlobalAveragePooling2D` layer and two `Dense` layers, each with 128 neurons, ReLU activation function, and a dropout rate of 0.2 to avoid overfitting;
- `Dense` layer with 7 neurons and softmax activation function.

In addition to the baseline model, two more complex types of architectures were used in the experiments. The first one was a two-stream network (Figure 2.11 (b)) consisting of two MobileNetV2 models, one of which processed the RGB input, while the other one processed the optical flow input. The goal of adding the latter type of input to the model was to represent motion between the consecutive frames and thus improve the capacity of the model to capture the temporal aspect of the input. Relevant studies (see e.g. [211]) suggest that temporal information is required for accurate movement recognition, as it is arguably



(a) (b) (c)

Figure 2.11: Architectures of the CNN models used in the cross-dataset study: (a) baseline CNN; (b) two-stream CNN; (c) recurrent CNN.

not possible to differentiate between `movement 1` and `movement 3` using just a single image. Overall, the architecture of the model was as follows:

- two parallel input layers;
- two parallel data augmentation layers;
- two parallel baseline MobileNetV2 models;
- `Concatenate` fusion layer;
- `Flatten` layer;
- included only in a model with extra layers: `GlobalAveragePooling2D` layer and two `Dense` layers, each with 128 neurons, ReLU activation function, and a dropout rate of 0.2 to avoid overfitting;
- `Dense` layer with 7 neurons and softmax activation function.

The second complex type of architecture used in experiments was a recurrent CNN (Figure 2.11 (c)), with a time-distributed layer joining together five base MobileNetV2 models and the Gated Recurrent Unit (GRU; [222]) used as the memory unit. Similarly to the two-stream network described above, the goal of incorporating the GRU in the architecture was to improve the capacity of the model to classify hand-washing movements by capturing the temporal aspect of hand washing. The overall architecture of the model was as follows:

- five parallel input layers;

- five parallel data augmentation layers;
- five parallel baseline MobileNetV2 models;
- `TimeDistributed` fusion layer;
- `GRU` layer with 256 neurons;
- included only in a model with extra layers: two `Dense` layers, each with 128 neurons, ReLU activation function, and a dropout rate of 0.2 to avoid overfitting;
- `Dense` layer with 7 neurons and softmax activation function.

The summary of the configuration, hyperparameters, and training procedure (details follow) of the baseline, the two-stream, and the recurrent CNN models is given in Table 2.5.

Table 2.5: Configuration and hyperparameters of CNNs for the cross-dataset study.

Parameter	**Value**
All networks / default values	
Base model	MobileNetV2
Initial weights	ImageNet, 224×224
Input image resolution	320×240
Data augmentations	Rotations, flips
N of fully connected layers	1
Layers retrained	1 ('top') or all ('full')
Epochs	20
Batch size	32
Optimiser	Adam
Learning rate	default (0.001)
Loss function	Cross-entropy
Classes	7
Two-stream networks	
Streams	RGB & optical flow
Fusion	Before dense layers
Optical flow type	Farneback
Optical flow step (seconds)	0.33
Recurrent networks	
Recurrent element	GRU
Frame step (seconds)	0.2
Frames	5
Baseline and Recurrent CNN with extra layers	
Additional fully connected layers	2

Training procedure

Prior to training the models, the datasets were split into trainval and test subsets as follows:

- the Kaggle dataset was split in a 70/30 ratio, making sure that frames from a particular video appear in either trainval or test split, but not both;
- similarly, the METC dataset was split to maintain the same condition, but in a 75/25 ratio;
- the PSKUS dataset was split in the same way as in the study reported in Section 2.4, that is, the data for the test subset were images from one particular location, the emergency ward, and the rest of the data were used as the trainval subset.

Furthermore, since the data from the three different datasets had different original resolution, the images were rescaled to the same resolution of 320×240 pixels using the built-in TensorFlow [77] functions with bilinear interpolation as the resize method. Afterwards, the models were trained for 20 epochs each with enabled early stopping if there was no decrease in the validation loss for 10 epochs; class weighting was used to deal with the class imbalance in the datasets. In addition to experiments with training and evaluating models on the same dataset, we conducted additional experiments employing transfer learning to measure the generalization performance across datasets both before and after 10 epochs of fine-tuning. In these experiments, we investigated only knowledge transfer from less complex to more complex datasets, i.e., from the Kaggle dataset to the METC dataset, from the Kaggle dataset to the PSKUS dataset, and from the METC dataset to the PSKUS dataset.

Results

As two of the datasets used in this study, the METC dataset, and particularly the PSKUS dataset, are imbalanced in class representation, I report the results as F_1 scores rather than accuracy. While I report the results of the evaluation of the models both on the validation and test data, note that these findings are not equally important: the most important results are those achieved on the latter dataset split, as they demonstrate the performance of the model on unseen test data, whereas the results on the former dataset split are important insofar as they allow us to understand whether the model learns anything at all, which is a topical question in cases when the performance of the model on the test data is unsatisfactory.

The results of **the main experiments** are shown in Figure 2.12. While it was expected that the best results on the test subsets of respective datasets would be achieved by the more complex models, i.e., either the two-stream network, or the recurrent CNN, the best F_1 score on the Kaggle and METC datasets was achieved by the baseline model, which only used a single frame as the input: it achieved 96% F_1 score on the Kaggle dataset and 64% F_1 score on the METC dataset. Another notable result is that full retraining usually improved the accuracy of the single-frame models while decreasing it in the case of the more complex classifiers: thus, it improved the performance of the baseline classifier from 93% to 96% on the Kaggle dataset and from 45% to 64% on the METC dataset, whereas it resulted in a deterioration in performance of the two-stream model and the recurrent model from 94% to 61% and from 90% to 36% respectively on the Kaggle dataset, and from 46% to 38% and from 55% to 33% respectively on the METC dataset. A possible explanation for that is that full retraining caused overfitting in more complex models.

Most concerningly, none of the approaches showed even average performance on the test data on the most complex among the datasets, the PSKUS dataset. Namely, the best classifier, which was the two-stream model with only the top layers retrained, achieved the F_1 score of only 21%; remarkably, the poor performance was not caused by a failure to learn, as the F_1 score on the validation data was above 95% in some of the experiments (fully retrained

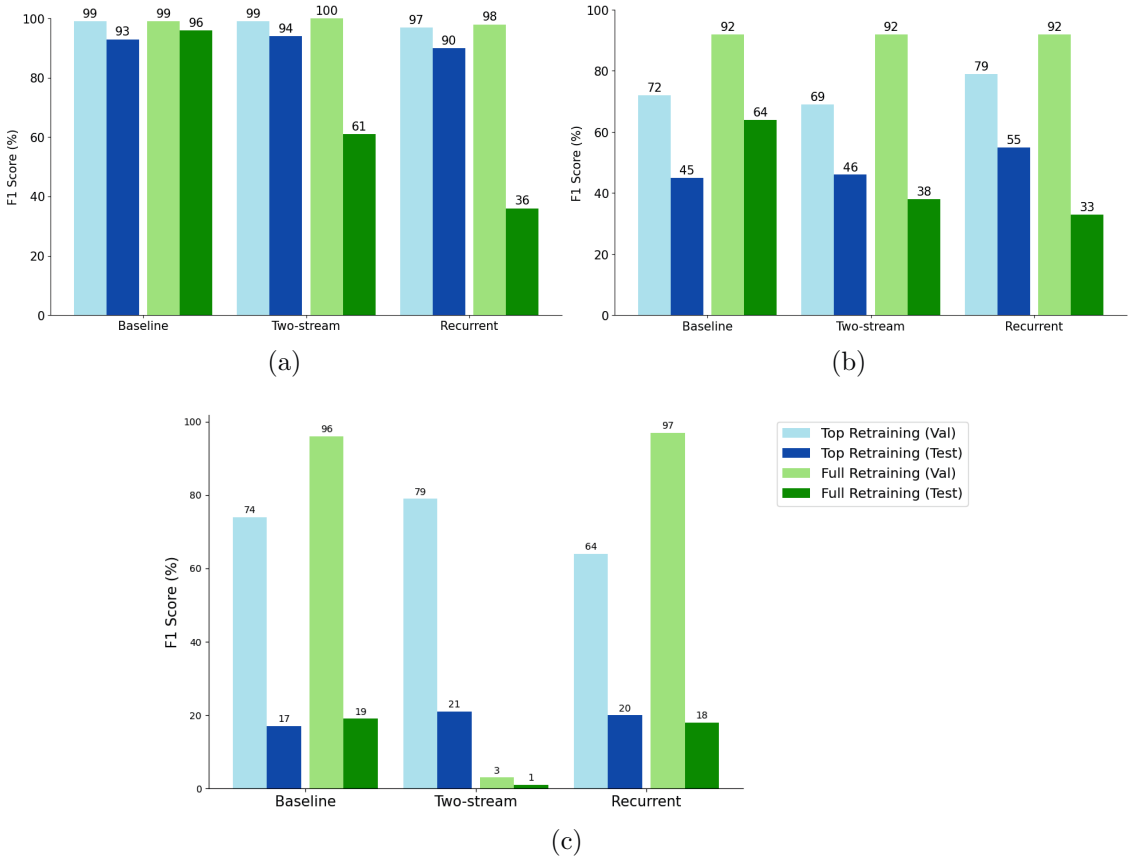


Figure 2.12: F_1 scores of different CNN architectures evaluated in the cross-dataset study: (a) on the Kaggle dataset; (b) on the METC dataset; (c) on the PSKUS dataset.

single-frame model and RNN); arguably, the problem lies in the poor generalisation ability of the models to the more complex data.

The results of the experiments with the baseline and recurrent CNN classifiers[5] with **two extra Dense layers** (see Figure 2.13) showed the same pattern as the results in the main experiments: the best-performing type of architecture was still the fully retrained single-frame model, which performed very well (96% F_1 score) on the Kaggle dataset and satisfactorily (64% F_1 score) on the METC dataset, but showed no improvement in comparison with the same model without extra layers. While adding extra layers improved the performance of the single-frame model on the PSKUS dataset both for the model with top only trained (+3%) and for the fully retrained model (+5%), the F_1 score of 25% achieved by the best among them, the single-frame fully retrained model, was far from satisfactory.

Finally, with respect to the results of **transfer learning** experiments (Figure 2.14), it is worth noting that one of the retrained models, the Kaggle–to–METC model, achieved the best performance among all experimental groups on the METC dataset (65% F_1 score), while another such model, the Kaggle–to–PSKUS model, achieved the best performance among all experiment groups on the PSKUS dataset (27% F_1 score). However, aside from this, the lessons from knowledge transfer attempts are not encouraging, as none of the classifiers showed acceptable performance before being retrained on the new dataset, and the recurrent

[5]As the two-stream network did not show a major improvement on the two other approaches in any of the experiments, we excluded this type of architecture from further experiments.

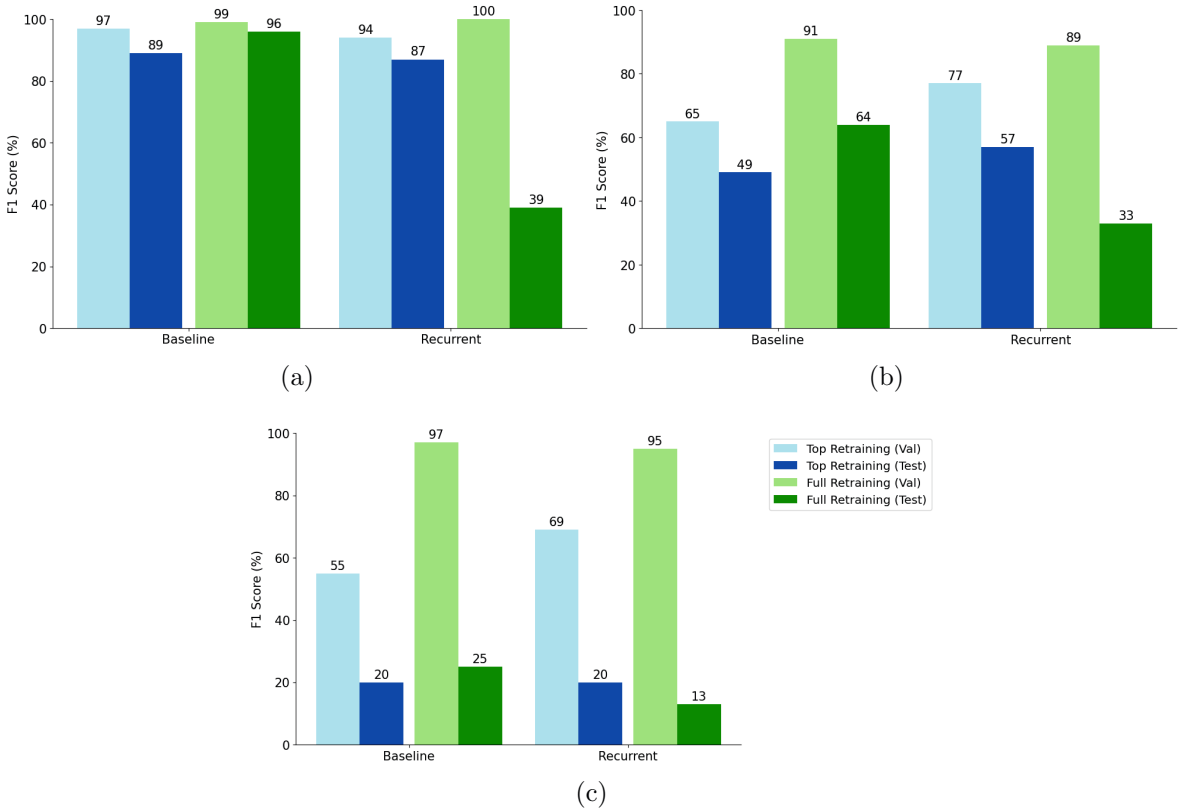


Figure 2.13: F_1 scores of the different CNN architectures with two extra Dense layers in the cross-dataset study: (a) on the Kaggle dataset (b) on the METC dataset (c) on the PSKUS dataset.

CNN initially trained on the Kaggle data completely failed to learn on the PSKUS dataset. Furthermore, after retraining, the performance on the new dataset was all in all quite similar to the performance when the classifier was trained straight from the MobileNetV2 base. This suggests the classification knowledge learned by the models in the experiments reported in this section was not transferable to new data contexts.

To summarise the results, lightweight CNN-based classifiers that performed well on the Kaggle dataset (>95% F_1 scores) demonstrated mediocre performance on the more complex lab-based METC dataset (50–60% F_1 scores), and failed to generalise on the real-life PSKUS dataset. Adding temporal information typically reduced generalisation performance, likely because the more complex models were more susceptible to overfitting, adding extra Dense layers was helpful only in certain cases such as training the single-frame model on the PSKUS dataset, and the effect of full retraining depended on the architecture of the model, generally improving performance of single-frame models and deteriorating performance of more complex models. In summary, these results demonstrate that the dataset is in fact more important than the approach when evaluating hand washing movement classification accuracy, and that translating the existing work on hand washing movement classification from the lab to the real-life conditions is not straightforward at all.

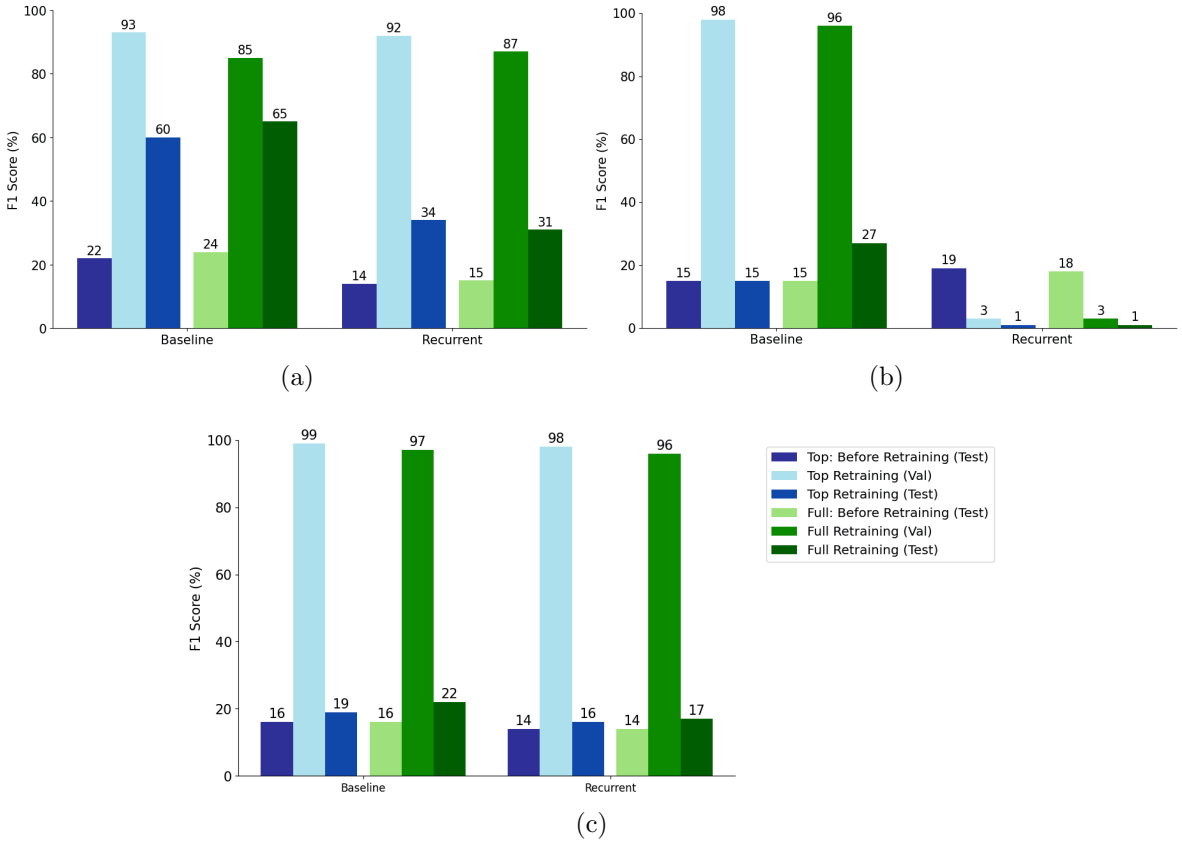


Figure 2.14: F_1 scores of the different CNN architectures with transfer learning in the cross-dataset study: (a) from the Kaggle dataset to the METC dataset (b) from the Kaggle dataset to the PSKUS dataset (c) from the METC dataset to the PSKUS dataset.

2.6 Concluding remarks

In this chapter, I described the research on the use of CNNs for the classification of hand-washing movements with the primary goal of creating an automated system for monitoring hand hygiene in a clinical setting. This goal is particularly relevant for healthcare: as poor hand hygiene in hospitals causes numerous infection cases and even deaths, an automated monitoring system would be of substantial help for improving the hand hygiene habits of the medical staff. While there have been several studies aiming at recognising hand-washing movements according to the WHO guidelines, ML models in these studies were trained on datasets collected in lab conditions. Since there had not been any publicly available large real-world datasets of hand-washing videos with labelling corresponding to the WHO guidelines, the research reported in this chapter started with collecting the data: first, the large real-world PSKUS dataset, and then the METC dataset, which can be positioned between simpler lab-collected datasets such as the Kaggle dataset and more complex real-world datasets.

Preliminary experiments on the PSKUS and METC datasets demonstrated that the lightweight CNN classifier MobileNetV2 did not outperform a putative 'naive' classifier on the former dataset; as to its performance on the latter dataset, it was substantially better, but still much worse than that typically reported in the literature on the use of CNN-based classifiers for hand-washing movement classification. The first conclusion from this is that the impressive classification results that have recently been reported in the literature on hand-

washing movement classification should be taken with a grain of salt, as the methods they have been achieved with may not translate well to complex real-world applications. The second conclusion is that the problem of the domain shift regarding the datasets for CNN should be taken seriously, as the research reported in this chapter demonstrated that changing the task from classification of the video stream from over one sink (the METC dataset case) to the classification of the stream from over several sinks (the PSKUS dataset case) dramatically degrades the performance of classifiers with the same architecture. Taking into account that the best performance of the CNN-based classifiers on the target dataset, the PSKUS dataset, is far from satisfactory, the third conclusion is that the problem of classification on this real-world dataset has not been solved yet, and further work is needed.

Furthermore, it is worth noting that the PSKUS dataset has become publicly available three years ago, in February 2021, and since then it has attracted some attention from the research community working on the problem of hand-washing movement classification: according to the statistics provided by the host website[6], it has been viewed 2422 times and downloaded 2410 times. In addition, according to Google Scholar metrics, the article in *MDPI Data* journal describing the dataset was cited 12 times[7], and the publication [176] on the cross-dataset study was cited 6 times[8]. However, somewhat surprisingly, the use of the PSKUS dataset in the publications citing it appears to be limited only to the *Related Work* (or equivalent) section, that is to say, the authors of the publications merely acknowledge the existence of the PSKUS dataset but never actually try to solve the classification problem on it. Therefore, designing a classifier with a sufficiently high accuracy on the PSKUS dataset remains a task for the future.

[6] https://zenodo.org/record/4537209; accessed 16 August 2024

[7] https://scholar.google.lv/scholar?oi=bibs&hl=en&cites=4959999994190408658; accessed 16 August 2024

[8] https://scholar.google.lv/scholar?oi=bibs&hl=en&cites=13054653446600211893; accessed 16 August 2024

Chapter 3

Semantic Segmentation of Street Views

In this chapter, I present research on the use of CNNs for semantic segmentation of street views, which is a pivotal task for the design of self-driving cars. The primary focus of the research I report here is on the promising approach of augmenting the dataset of real-world images with synthetic data, which helps tackle the problem of the limited availability of labelled data for training semantic segmentation models.

A large part of the insights and material in this chapter comes from the following publication in a scientific journal:

[223] **M. Ivanovs**, K. Ozols, A. Dobrajs, and R. Kadikis, "Improving semantic segmentation of urban scenes for self-driving cars with synthetic images," *Sensors*, vol. 22, no. 6: 2252, 2022.

As the first author of the above publication, I was responsible for planning and conducting experiments for synthetic data generation as well as for training and validating CNN-based semantic segmentation models. I also took the lead in analysing the experimental results, documenting the findings, and drafting the manuscript. Finally, I played a principal role in revising and finalising the manuscript in collaboration with my co-authors.

3.1 Introduction

There are numerous applications for methods used in the semantic segmentation of images of urban areas, such as estimating building footprints [224], mapping urban green spaces [225], and detecting water bodies [226] and slums [227]. For these purposes, aerial or satellite images are typically used, as they provide a bird's-eye view of cityscapes. However, when it comes to the images of street views – by which I mean panoramic images taken from a position close to the ground, i.e., from the vantage point of a driver or a pedestrian – there appears to be a single major application for semantic segmentation methods, namely, for use in the navigation system of self-driving cars. Therefore, to underscore the relevance of the work reported in this chapter, I begin with a brief exploration of the topic of self-driving cars.

Self-driving cars, which are also referred to as robotic cars [228], autonomous vehicles [229, 230], and driverless vehicles [231], are currently one of the most promising emerging technologies and a lively area of academic and industrial research. Research laboratories, universities, and companies have been actively working on designing self-driving cars since

the mid-1980s [232]; in the last decade, research on self-driving cars and development of their prototypes with varied degrees of autonomy have gained momentum with an increasing focus on technologies for data acquisition and processing [233]. However, the task of developing self-driving cars with the highest level of autonomy – i.e., autonomous to such an extent that no human intervention during driving is required in any circumstances [234] – still remains an unsolved challenge.

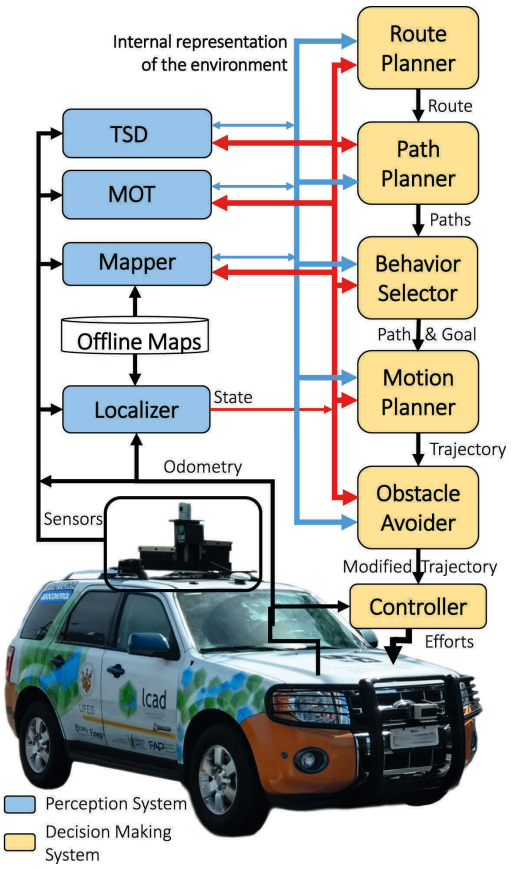


Figure 3.1: Schematic overview of the autonomy system of a self-driving car, including Traffic Signalization Detection (TSD) and Moving Objects Tracking (MOT). Reproduced from [232].

The architecture of the autonomy system in self-driving cars is typically divided into two main modules (Figure 3.1): the perception system, and the decision-making system [235, 232]. The perception system deals with image understanding tasks such as object recognition, object localisation, and, which is particularly relevant to the topic of this chapter, semantic segmentation [236]. Similar to other visual domains, the best results in semantic segmentation of street views have been achieved using CNNs [237]. However, the issue of data availability limits the feasibility of using CNNs for this purpose, as acquiring images in an urban setting is both expensive and time-consuming; in addition to that, sharing the acquired data publicly may encounter legal obstacles due to privacy concerns. Furthermore, pixel-wise labelling of complex urban scenes takes a lot of time and effort: thus, it was reported that for the creation of Cambridge-driving Labeled Video Database (CamVid; [238]), labelling took around 1 hour per image, whereas in the case of the Cityscapes dataset [130], fine pixel-level annotation and quality control of a single image required on average more than 1.5 hours.

As outlined in Section 1.4.4, one of the approaches to tackling the problem of data availability for training DNNs is to resort to the use of synthetic data, i.e., artificially gener-

ated datasets that are to some extent similar to the real-world data. Using synthetic data eliminates or at least decreases the need for real-world data acquisition as well as provides additional advantages. A synthetic image generation pipeline can be modified to produce more diverse data – e.g., by changing the weather conditions, or increasing the number of particularly salient objects such as cars, pedestrians, traffic lights, and traffic signs. This allows the production of diverse synthetic data on a large scale. Furthermore, such pipelines usually eliminate the need for manual image annotation, as the contours of both the objects and the background can be obtained automatically and with a high degree of precision.

The goal of the work reported in the present chapter was to improve the accuracy of semantic segmentation of street views by augmenting a dataset of real-world images with synthetic data. In particular, I investigated whether it is possible to improve the accuracy of semantic segmentation by using synthetic data generated with an open-source driving simulator CARLA [11], which can be done in a relatively simple, fast, and largely automated manner. **The main hypothesis** of the study was that augmenting real-world data with synthetic data would result in the improved accuracy of semantic segmentation of street views.

The rest of the chapter is structured as follows. In Section 3.2, I give an overview of related work, surveying datasets and methods for semantic segmentation of street views. In Section 3.3, I describe three datasets of street views that I used for training CNN models in the study. Section 3.5.2 presents the results of the experimental study and their discussion; it is followed by Section 3.6, which offers concluding remarks.

3.2 Related work: datasets and methods for semantic segmentation of street views

Since labelling semantic segmentation datasets is labour-intensive, only a limited number of publicly available semantic segmentation datasets exist, and they tend to be comparatively small in size. The CamVid dataset [238] consists of 700 annotated images obtained from a video sequence of 10 minutes; the pixel-wise labelled subset of the Daimler Urban Segmentation Dataset [239] contains 500 images; the KITTI semantic instance segmentation benchmark dataset [240] consists of 200 semantically annotated training and 200 test images; finally, in the Cityscapes dataset [130], there are 5 000 fine-labelled and 20 000 coarse-labelled images. Due to its comparatively large size and the high quality of the annotations and documentation, Cityscapes is the best-known and widely used dataset of semantically annotated street views. In particular, according to the Cityscapes benchmark suite for pixel-level semantic segmentation[1], at the time of writing, there were 294 models listed in the benchmark table, each submitted to the automated benchmarking server. However, it should be noted that the code for many of these models is not publicly available, which makes reproducing their results challenging. Among the models with publicly available code, those from the DeepLab library [123] were particularly noteworthy when the study reported in the present chapter was conducted, as they were made publicly available via a well-documented and popular DeepLab repository[2] and demonstrated high accuracy. Specifically, according to the benchmarks published by the authors of the repository, the Xception-65 model achieved a

[1] https://www.cityscapes-dataset.com/benchmarks/#instance-level-results; accessed 6 September 2023

[2] https://github.com/tensorflow/models/tree/master/research/deeplab; accessed 6 September 2023

78.79% mIOU on the test subset after training on Cityscapes fine-labelled images[3].

While the results achieved on the Cityscapes dataset with the state-of-the-art methods for semantic segmentation are rather impressive, there is still room for further improvement. One possible approach is to increase the amount of training data; however, this is difficult due to the scarcity of semantically labelled data for self-driving cars caused by 'the curse of dataset annotation' [241]. This is the major problem for training CNN models for semantic segmentation in general, which extends beyond the particular task of improving the accuracy of segmentation on the Cityscapes dataset and can potentially impede the development of self-driving cars with a high degree of autonomy.

In response to this problem, a number of studies have used synthetic data for augmenting segmentation datasets for self-driving cars. Ros et al. [242] used synthetic images from their Synthetic collection of Imagery and Annotations (SYNTHIA), which represents a virtual New York City modelled by the authors with the Unity platform, to augment real-world datasets and improve semantic segmentation; Richter et al. [243] extracted synthetic images and data for generating semantic segmentation masks from the video game Grand Theft Auto (GTA) V and used the acquired synthetic dataset for improving the accuracy of semantic segmentation; Hahner et al. [244] created a custom dataset of synthetic images of foggy street scenes to enhance semantic scene understanding under foggy road conditions. However, there are still several challenges in the use of synthetic data for self-driving cars. First, even high-quality synthetic images are not entirely photorealistic and therefore are less valuable for training than real-world images; therefore, it is often more reasonable to augment a real-world dataset with synthetic data rather than train CNN models solely on synthetic data [9]. Second, generating synthetic data can require quite a lot of effort – at least at the stage of initial design. Developing a pipeline for the generation of synthetic images can be challenging in terms of time and effort involved; another approach, acquiring synthetic images from video games, may also prove difficult, since the internal workings and assets of games are often hard to access [243].

A promising alternative path for the generation of synthetic images is to utilise open-source sandbox driving simulators such as the Open Racing Car Simulator (TORCS) [245] or Car Learning to Act (CARLA) [11]. While such simulators are said to lack the extensive content found in top-level video games [243], their open-source nature makes it easier to access and modify them as well as imposes fewer (if any) legal constraints on the use of the eventually generated data. Recently, Berlincioni et al. [246] made use of the data generated with a driving simulator: the authors created their Media Integration and Communication Center – Semantic Road Inpainting (MICC-SRI) dataset with CARLA to tackle the problem of image inpainting [247, 248], i.e., predicting missing or damaged parts of an image by inferring them from the context. However, it remains an open question whether the quality of synthetic images generated with open-source driving simulators is sufficient for improving the accuracy of CNNs on semantic segmentation, as this task demands particularly high-quality training data.

[3] https://github.com/tensorflow/models/blob/master/research/deeplab/g3doc/model_zoo.md; accessed 6 September 2023

3.3 Street views datasets for semantic segmentation: Cityscapes, MICC-SRI, and CCM

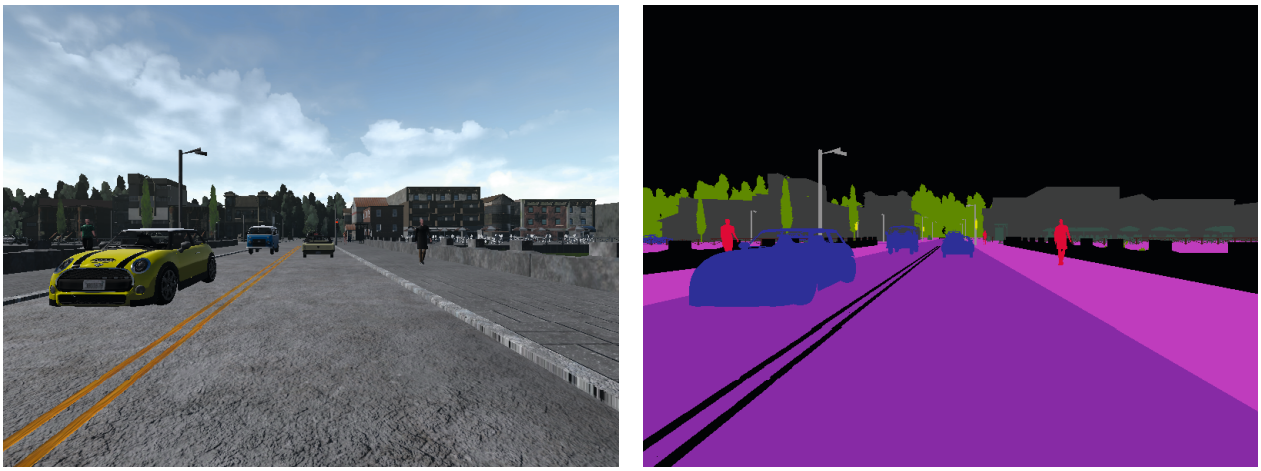
To investigate the use of synthetic data generated with open-source driving simulators to augment real-world street view datasets, I used three datasets: the Cityscapes dataset [130] of real-world images, the MICC-SRI dataset [246], composed of synthetic images generated with the CARLA simulator, and the CCM (Cityscapes-CARLA Mixed) dataset, which I created for the study reported in this chapter. These three datasets are described in detail below; sample images from each dataset and their corresponding pixel segmentation masks are shown in Figure 3.2.

Cityscapes [130] is one of the most well-known datasets of urban landscapes for self-driving cars. It comprises a diverse set of images at a resolution of 1024×2048 pixels taken on the streets of 50 different European (predominantly German, with some Swiss) cities using cameras mounted on a specially equipped car. Coarse semantic segmentation annotations are available for 20 000 images, while fine (pixel-level) annotations are provided for 5 000 images. In the present study, I used only the Cityscapes images with fine annotations. Furthermore, I had to change the division of the original dataset into training, validation, and test subsets, as due to the need to relabel the segmentation masks (see Section 3.4) to ensure compatibility between the Cityscapes images and the synthetic images generated with CARLA, it was not possible to benchmark the models used in this study on the original Cityscapes test set, which is withheld from public access to ensure impartial benchmarking. I created a custom split of the Cityscapes dataset as follows: out of the 3 475 Cityscapes fine-annotated images available in public access, I used 2 685 for training, 290 for validation, and 500 for testing CNN models. I adhered to the original Cityscapes policy that images from a particular location should only appear in one split, i.e. in training, *or* validation, *or* test set, but not in multiple splits. This approach ensures the strict separation between the data in each subset, which was necessary to prevent a flawed methodology of testing a CNN model on data that is too similar to the data it was trained on. Specifically, after randomly selecting the locations, I used images taken in Frankfurt, Lindau, and Münster for the test dataset, images taken in Bochum, Krefeld, and Ulm for the validation dataset, and the rest of the images for the purpose of training the models.

The MICC-SRI dataset [246] consists of 11 913 synthetic RGB frames of urban driving footage with the resolution of 600×800 pixels generated with the CARLA simulator [11]; for all RGB frames, semantic segmentation annotations are provided. The dataset was originally created for semantic road inpainting tasks, and the images are not photorealistic (cf. Figure 3.2, **b**). As Berlincioni et al. [246] report, the frames for MICC-SRI dataset were collected by running separate simulations for 1 000 frames from each spawning point on the two available maps in CARLA version 0.8.2, the most recent version at the time of their study. The simulations were run at 3 FPS; to ensure diversity of the generated data, the simulations were subsampled to take an image every 3 seconds. Further processing reported by the authors included removing occasional misalignment between the RGB frames and segmentation masks. RGB frames and corresponding semantic segmentation annotations in the MICC-SRI dataset are available in two versions, the one with the static objects only, and the one with both static and dynamic (cars and pedestrians) objects. For the study reported here, I used only the RGB images and segmentation masks containing both static and dynamic objects.

My custom-made CCM dataset consists of 2 685 Cityscapes images as well as 46 935 synthetic images that I generated with CARLA simulator. The resolution of synthetic images

(a)



(b)

(c)

Figure 3.2: Sample images and their segmentation masks from: (a) the Cityscapes dataset; (b) the MICC-SRI dataset; (c) the CCM dataset. Note that (a) and (c) are not to scale with respect to (b), as the actual resolution of (a) and (c) is 1024×2048 pixels vs 600×800 pixels for (b).

is 1024×2048 pixels, matching the resolution of the Cityscapes images. The synthetic images were collected by running simulations on several maps available in the latest stable release of CARLA (v0.9.12), namely, `Town 1`, `Town 2`, `Town 3`, `Town 4`, and `Town 10`. The simulations were run at 1 FPS, with an RGB image and its corresponding segmentation mask saved every second. To increase diversity, the simulation in `Town 1` was run with the weather conditions set to `Clear Noon`, whereas simulation in `Town 10` was run with the weather conditions set to `Cloudy Noon`; on the rest of the maps, the simulations were run with default settings. To acquire images with a large number of dynamic objects, the simulations were run with 100 spawned vehicles and 200 spawned pedestrians. The number of images acquired in each location is given in Table 3.1. The simulations took approximately 96 hours to complete on a desktop PC with Windows 10 OS, an Intel i5-6400 CPU, and an NVIDIA 1060 GPU.

Table 3.1: The number of CARLA-generated images for the CCM dataset by location.

Location	**Images**
`Town 1`	7 866
`Town 2`	3 838
`Town 3`	11 124
`Town 4`	11 484
`Town 10`	13 049

3.4 Data preprocessing

Data preprocessing consisted of relabelling semantic segmentation masks, resizing images, and augmenting real-world Cityscapes data with synthetic data.

Relabelling semantic segmentation ensured compatibility between the annotation labels of the Cityscapes images and the CARLA-generated images in both the MICC-SRI and CCM datasets. Different label mappings were required for the two datasets, because the MICC-SRI dataset was generated with CARLA v0.8.2, which had fewer labels than the more recent CARLA v0.9.12 used for the CCM dataset. The label mapping between Cityscapes and MICC-SRI was the same as in the experiments by Berlincioni et al. [246]; since it was not detailed in the original publication, I obtained the mapping through personal communication with Lorenzo Berlincioni, and it is provided in Table 3.2.

Table 3.2: Label mapping for experiments with the MICC-SRI dataset.

Cityscapes labels	**MICC-SRI labels**	**Resulting labels for augmented dataset**
unlabeled, ego vehicle, rectification border, out of ROI, static, dynamic, rail track, sky, license plate	none, other	other
road	road lines, roads	roads
ground, sidewalk, parking	sidewalk	sidewalk
building	buildings	buildings
wall, fence, guard rail, bridge, tunnel	fences, walls	fences, walls
pole, polegroup, traffic light, traffic sign	poles, traffic signs	poles, traffic signs
vegetation, terrain	vegetation	vegetation
person, rider	pedestrian	human
car, truck, bus, caravan, trailer, train, motorcycle, bicycle	vehicles	vehicles

Furthermore, I provide the mapping between the Cityscapes labels and CARLA v0.9.12 labels, which I designed for the creation of the CCM dataset, in Table 3.3.

Image resizing was necessary for experiments on the Cityscapes and MICC-SRI datasets, as the images in these datasets were of different sizes (1024 × 2048 pixels vs 600 × 800 pixels, respectively), whereas the input to a CNN typically needs to be uniform in size. One possible solution was to upscale MICC-SRI images; however, since they have different height-to-width ratios than Cityscapes images, 0.75:1 vs 0.5:1, upscaling would lead to significant distortion and likely reduce the accuracy of semantic segmentation. Therefore, I took a different approach, splitting each Cityscapes image into 9 smaller images, each sized 600×800 pixel, thus matching the size of the images in the MICC-SRI dataset. No resizing was necessary for the experiments involving the CCM and Cityscapes datasets, as the synthetic images for the CCM dataset were generated with the same size as the real-world images in the Cityscapes dataset.

Finally, to investigate how the amount of synthetic data used for augmentation affects the accuracy of semantic segmentation, I created three splits of the CCM dataset, each including all the Cityscapes real-world images designated for training, along with 100%, 50%, and 25% of the synthetic images that I generated with the CARLA simulator. The synthetic images for these splits were selected randomly; to avoid unnecessary repetition, I will refer to these splits as CCM-100, CCM-50, and CCM-25, respectively.

Table 3.3: Label mapping for experiments with the CCM dataset.

Cityscapes labels	**CARLA (v0.9.12) labels**	**Resulting CCM dataset labels**
unlabelled, ego vehicle, rectification border	unlabelled	unlabelled
building	building	building
fence	fence	fence
tunnel, pole group	other	other
pedestrian, rider	pedestrian, bike rider	pedestrian & rider
pole	pole	pole
road	road, roadline	road
sidewalk, parking	sidewalk	sidewalk & parking
vegetation	vegetation	vegetation
car, truck, bus, caravan, trailer, train, motorcycle, bicycle	vehicles	vehicles
wall	wall	wall
traffic sign	traffic sign	traffic sign
sky	sky	sky
ground	ground	ground
bridge	bridge	bridge
rail track	rail track	rail track
guardrail	guardrail	guardrail
traffic light	traffic light	traffic light
static	static	static
dynamic	dynamic	dynamic
terrain	water, terrain	water & terrain

3.5 Experiments: methodology and results

3.5.1 Methodology

I conducted semantic segmentation experiments using two CNN models from the DeepLabv3 library [118]: MobileNetV2 [105] and Xception-65 [117], which both were pretrained on the PASCAL VOC 2012 dataset [57]. As mentioned in Section 3.2, DeepLab is a well-known

state-of-the-art library for semantic segmentation; I chose these two particular models from it because MobileNetV2 is a compact and fast CNN, while Xception-65 is a larger CNN that offers better segmentation accuracy at the cost of longer training times and greater GPU memory requirements. The models were trained on Dell EMC PowerEdge C4140 high-performance computing (HPC) servers of the Riga Technical University HPC Center[4] equipped with Intel Xeon Gold 6130 CPUs and NVIDIA V100 GPUs with 16 GB VRAM. The models were trained using default settings: for MobileNetV2, the output stride was set to 8, and the training crop size was 769×769 pixels; for Xception-65 models, the atrous rates were set to 6, 12, and 18, the output stride was 16, the decoder output stride 4, and the training crop size was 769×769 pixels. The learning rate was set to 0.0001, and the training was optimised using the SGD optimiser with a momentum value of 0.9.

I used only real-world images for validating and testing the models, i.e., in all experiments, in all experiments, the Cityscapes validation set was used for validation, and the Cityscapes test set for testing. The batch size for training was the maximum possible given the GPU memory that was at my disposal: 4 images for MobileNetV2 models, and 2 images for Xception-65 models. For experiments on the MICC-SRI and Cityscapes datasets, the MobileNetV2 models were trained for 1200 epochs on each dataset, and the Xception-65 models were trained for 300 epochs on each dataset. For experiments on the CCM and Cityscapes datasets, MobileNetV2 and Xception-65 models were trained for 200 epochs on the Cityscapes dataset and on the CCM-100, CCM-50, and CCM-25 dataset splits. In total, training the four models on the MICC-SRI and Cityscapes datasets took $\approx$ 1480 hours of computing, while training the eight models on the Cityscapes dataset and CCM splits took $\approx$ 1415 hours of computing.

3.5.2 Results

I report the main results using the standard metrics for semantic segmentation: IoU and mIoU. Similar to other authors, e.g. Cordts et al. [130], I report and include in the calculations only semantically meaningful classes, excluding classes such as `Other` or `None`. Since I had to modify the labelling scheme from the one originally used in the Cityscapes dataset, I was unable to directly compare the performance of the CNN models trained on the augmented datasets with state-of-the-art results on the original Cityscapes dataset reported in the literature, e.g., in Chen et al. [123]. Therefore, I compare the accuracy of the models trained on the augmented datasets with CNN models of the same architecture that I trained on the Cityscapes dataset using the accordingly modified labelling scheme.

Results on Cityscapes and MICC-SRI datasets

I summarise the main results of training MobileNetv2 and Xception-65 DNN models on the Cityscapes and MICC-SRI datasets in Tables 3.4 and 3.5. As can be seen, augmentation of Cityscapes dataset with MICC-SRI images did not improve the accuracy of semantic segmentation; on the contrary, both MobileNetV2 and Xception-65 models performed slightly better when trained only on real-world images than on the dataset augmented with synthetic images, with an mIoU of 75.43% vs 75.11% for the MobileNetV2 model and an mIoU of 79.34% vs 78.81% for the Xception-65 model. The MobileNetV2 model trained only on the real-world images performed better than its counterpart trained on the augmented dataset across all segmentation classes, whereas in the case of the Xception-65 models, the only class on which the model trained on the augmented data achieved a better result than its

[4] `https://hpc.rtu.lv/`; accessed 20 September 2024.

Table 3.4: Comparison of the accuracy (IoU) of semantic segmentation: MobileNetV2 trained on the Cityscapes dataset vs. MobileNetV2 trained on the Cityscapes dataset augmented with the MICC-SRI dataset.

Class	Cityscapes	Cityscapes augmented with MICC-SRI
`Road`	**92.66**	92.62
`Sidewalk`	**67.02**	66.61
`Building`	**86.48**	86.18
`Fences and Walls`	**44.46**	43.21
`Poles and traffic signs`	**57.07**	56.72
`Vegetation`	**89.52**	89.45
`Pedestrians`	**76.59**	76.54
`Vehicles`	**89.65**	89.55
Mean IoU	**75.43**	75.11

Table 3.5: Comparison of the accuracy (IoU) of semantic segmentation: Xception-65 trained on the Cityscapes dataset vs. Xception-65 trained on the Cityscapes dataset augmented with the MICC-SRI dataset.

Class	Cityscapes	Cityscapes augmented with MICC-SRI
`Road`	**93.69**	93.60
`Sidewalk`	71.78	**72.70**
`Building`	**88.67**	88.30
`Fences and Walls`	**52.20**	49.16
`Poles and traffic signs`	**63.58**	62.52
`Vegetation`	**90.75**	90.58
`Pedestrians`	**81.75**	81.39
`Vehicles`	**92.29**	92.24
Mean IoU	**79.34**	78.81

counterpart trained on the real-world images alone was `Sidewalk`. The likely explanation for the worse performance of the models trained on the augmented dataset is the low photorealism of images in the MICC-SRI dataset: while the quality of these synthetic images was sufficient for the semantic road inpainting task, it proved inadequate for the more challenging task of semantic segmentation. It is also worth noting that the Xception-65 models trained on the Cityscapes and MICC-SRI datasets demonstrate better performance than MobileNetV2 models trained on the same datasets. This performance difference is likely due to the larger size (i.e., a greater number of parameters) and the resulting better learning capacity of the Xception-65 architecture.

Results on Cityscapes and CCM datasets

The results of training MobileNetV2 and Xception-65 models on the Cityscapes dataset and the three splits of the CCM dataset – CCM-100, CCM-50, and CCM-25 – are reported in Tables 3.6 and 3.7, respectively. As can be seen, for both CNN architectures, augmentation with synthetic data improved semantic segmentation accuracy. For MobileNetV2, the model trained on CCM-100 achieved an mIoU of 55.49%, the model trained on CCM-50 achieved an

Table 3.6: Comparison of the accuracy (IoU) of semantic segmentation: MobileNetV2 trained on the Cityscapes, CCM-100, CCM-50, and CCM-25 datasets.

Class	Cityscapes	CCM-100	CCM-50	CCM-25
Building	75.54	79.39	**80.17**	79.18
Fence	00.02	21.49	**24.47**	17.82
Pedestrian & Rider	69.23	67.94	68.92	**69.38**
Pole	10.48	38.51	**38.54**	36.81
Road	88.57	88.46	**89.72**	89.15
Sidewalk	54.51	57.24	**59.79**	58.62
Vegetation	83.90	86.14	**86.41**	85.54
Vehicles	82.20	82.72	82.72	**82.78**
Wall	0.00	19.90	**23.81**	15.95
Traffic Sign	0.00	**35.42**	34.10	25.30
Sky	82.80	85.32	85.83	**85.85**
Traffic light	0.00	**21.32**	15.38	00.13
Water & Terrain	33.61	37.50	**38.73**	38.20
Mean IoU	44.68	55.49	**56.05**	52.67

mIoU of 56.05%, and the model trained on CCM-25 achieved an mIoU of 52.67%, whereas the model trained solely on Cityscapes images achieved an mIoU of 44.68%. The same trend was observed in the experiments with the Xception-65 models: the model trained on CCM-100 achieved an mIoU of 63.14%, the model trained on CCM-50 achieved an mIoU of 63.87%, and the model trained on CCM-25 achieved an mIoU of 64.46%, whereas the model trained on Cityscapes images only achieved an mIoU of 57.25%. Interestingly, the best-performing MobileNetV2 and Xception-65 models were not those trained on the CCM splits with the largest amounts of synthetic data: the best-performing MobileNetV2 model was trained on CCM-50, while the best-performing Xception-65 model was trained on CCM-25, the split with the smallest amount of synthetic data. This suggests that using larger amounts of synthetic data for augmentation does not necessarily result lead to better performance than augmentation with smaller amounts of such data.

Another notable finding is that models trained on different splits of the CCM dataset showed the best results (i.e., in comparison to other models) on different classes: thus, among the MobileNetV2 models, the one trained on CCM-100 showed better accuracy than other models on the classes `Traffic Sign` and `Traffic Lights`; the model trained on CCM-50 outperformed others on the classes `Building`, `Fence`, `Pole`, Road, `Sidewalk`, `Vegetation`, `Wall`, and `Water and Terrain`; and the model trained on CCM-25 performed best on the classes `Pedestrian & Rider`, `Vehicles`, and `Sky`. These differences are exemplified in Figure 3.3: as shown, the MobileNetV2 model trained solely on the real-world Cityscapes images could not segment road signs and traffic lights; as for the Xception-65 model trained solely on the real-world Cityscapes images, while it was able segment road signs, it could not distinguish between traffic lights and road signs, mistakenly classifying the former as the latter. In contrast, the MobileNetV2 and Xception-65 models trained on the CCM-50 and CCM-25 splits, respectively, i.e., the best-performing models for these architectures, were capable of

Table 3.7: Comparison of the accuracy (IoU) of semantic segmentation: Xception-65 trained on the Cityscapes, CCM-100, CCM-50, and CCM-25 datasets.

Class	**Cityscapes**	**CCM-100**	**CCM-50**	**CCM-25**
`Building`	84.94	85.10	85.08	**85.62**
`Fence`	37.20	40.19	40.19	**43.44**
`Pedestrian & Rider`	**78.08**	76.42	76.94	77.92
`Pole`	45.08	48.50	48.75	**49.26**
`Road`	**92.31**	91.82	91.43	91.85
`Sidewalk`	65.80	67.21	67.09	**69.88**
`Vegetation`	87.71	87.00	87.59	**87.85**
`Vehicles`	**89.86**	88.82	89.63	89.63
`Wall`	23.29	28.88	27.69	**31.63**
`Traffic Sign`	44.42	50.89	55.83	**56.14**
`Sky`	85.13	88.79	89.70	**90.34**
`Traffic Light`	0.00	43.64	42.19	**44.13**
`Water & Terrain`	**46.86**	39.58	35.62	44.22
Mean IoU	57.25	63.14	63.87	**64.46**

segmenting road signs and traffic lights and distinguishing between them. These observations are consistent with the results in Table 3.6 and Table 3.7 regarding the capacity of these models to segment objects from the `Traffic Lights` and `Road Signs` classes.

(a)


(b)


(c)

Figure 3.3: Semantic segmentation of a sample image with different models: (a) the original image; (b) segmentation masks produced with MobileNetV2 models: trained on Cityscapes images only (left) and on the CCM-50 split (right); (c) segmentation masks produced with Xception-65 models: trained on Cityscapes images only (left) and on the CCM-25 split (right). Note the differences in the ability of the models to accurately segment traffic lights and road signs.

3.6 Concluding remarks

Semantic segmentation models are an essential part of the perception module of a self-driving car: albeit they tend to have higher latency than object detectors due to their higher computational complexity, they provide finer-grained information about the shapes of surrounding objects, allowing the decision-making system of the car to better sense the environment. One of the main challenges in developing such semantic segmentation models is the availability of data for training them: in addition to the need to acquire data, which is a notorious problem in deep learning in general, data for supervised training of semantic segmentation models need to be meticulously labelled, which consumes many human-hours. To tackle this problem, I trained the MobileNetV2 and Xception-65 semantic segmentation models from the DeepLabv3 library on a mix of real-world data and data generated with CARLA, an open-source simulator for autonomous driving. I conducted two series of experiments. In the first series, I trained the models on the Cityscapes dataset augmented with low-photorealism images from the MICC-SRI dataset, generated using an older version of CARLA. In the second series, I trained them on the Cityscapes dataset augmented with more photorealistic synthetic images generated using a more recent version of CARLA. The main hypothesis of the study, namely, that augmenting real-world data with synthetic data would result in the improved accuracy of semantic segmentation of street views, was not confirmed in the first series of experiments but was confirmed in the second series of experiments. As the main difference between the data for augmentation in these series of experiments was the degree of photorealism, I conclude that the crucial factor in these experiments that determined whether synthetic data decreases or improves the performance of the CNN-based segmentation models was how similar these images were to the real-world images in the target dataset, Cityscapes.

This study also demonstrated that setting up a pipeline for generating synthetic data does not have to be costly or difficult, as CARLA allowed for generating a large amount of synthetic data quickly and without much meddling with out-of-the-box installation of that open-access simulator. However, there was also a downside to the use of the ready-made solution for generating data: due to the disparities in the labelling systems of Cityscapes and CARLA, it was necessary to merge some class labels to create the CCM dataset, and because of that, it was not possible to submit the models trained on CCM for evaluating on the official Cityscapes semantic segmentation benchmarks. While that did not prevent me from comparing the accuracy of the models trained on non-augmented vs augmented datasets, as I resorted to setting aside a part of the available data for testing the models, it was an unfortunate obstacle for comparing these models with their counterparts developed by other researchers. Although it was not possible to overcome that obstacle at the time when the study was conducted, the more recent releases of CARLA[5] do not pose this problem anymore, as the labels in this simulator are now fully compatible with the labels of Cityscapes. As the versions of CARLA released since then also feature numerous improvements of assets such as new town maps and vehicles, the use of CARLA-generated data for improving semantic segmentation on Cityscapes has become even more relevant and remains a promising direction for future work.

Another research problem worth investigating further is determining the optimal amount of synthetic data for augmenting a real-world dataset to achieve the best semantic segmentation accuracy. One possible answer to this question is ubiquitous in experimental science, which deep learning indeed is: 'it depends'. Indeed, since the optimal amount of the synthetic data depends on many factors – for instance, whether its distribution covers well the classes

[5]Cf. e.g. release notes for v0.09.14 - https://carla.org/2022/12/23/release-0.9.14/; accessed 10 April 2024.

underrepresented in the real-world dataset, or whether synthetic data represent both foreground classes (i.e., objects) and background classes, or just background classes (cf. [249]) – it is possible that a trial-and-error approach will always be necessary. On the other hand, post hoc explanations of what ratio of synthetic data was the best are suboptimal both from the intellectual standpoint and because of practical considerations, as training multiple model instances on different ratios of the augmented data is time-consuming and computationally expensive. Therefore, discovering general trends – such as the already mentioned observation that more photorealistic data typically yield better results – appears to be a promising direction for future research. While it seems unlikely that a robust, definitive solution to the problem of how much synthetic data to use for augmentation will emerge, I hypothesise that a promising approach is to modify the training of the model so as to take into account the domain gap between the real-world and synthetic data. In a way, that was already done in the study I detailed in this chapter, as I used only real-world data for validating the model during the training in order to ensure that the model performs robustly on the data in the target domain. More complex approaches worth investigating further may involve additional modifications to the training procedure of semantic segmentation models.

Chapter 4

Object Detection for a Bin-Picking Task

In this chapter, I present a study on the use of CNN-based models for detecting graspable items in a pile of objects. Detected coordinates of such items can then be utilised by an industrial robot to pick these objects up, a task commonly referred to as bin-picking [250] in robotics.

As visual object detectors are often an essential component of the perceptual system of industrial robots, the work reported in this chapter is relevant to the design and development of efficient robotic systems for manufacturing, sorting, and packing goods. Furthermore, as the same approach was used to obtain the data for training object detectors as the approach to obtain the data for training semantic segmentation models described in the previous chapter, that is, synthetic data were used for that purpose, the research that I report here is relevant to the broader challenge of overcoming the problem of the limited availability of data for training CNN models.

The present chapter is based on the following conference paper:

[251] D. Duplevska, **M. Ivanovs**, J. Arents, and R. Kadikis, "Sim2Real image translation to improve a synthetic dataset for a bin picking task," in *2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1–7, IEEE, 2022.

As a co-author of the above study, I was primarily responsible for training and validating CNN-based object detectors as well as contributed substantially to writing and editing of the manuscript. The experiments aimed at enhancing the quality of the synthetic data using Generative Adversarial Networks (GANs; [12]) were led by my collaborator Diana Duplevska.

4.1 Introduction

Adoption of robotic systems in manufacturing and other industrial operations has been growing steadily for decades now [252, 253], as they help to increase productivity, provide consistent output, ensure quality control, save costs, and improve safety in industrial settings. An essential component of such a system is a perception module (Figure 4.1), which supplies it with information about the environment, enabling its efficient functioning. The perception module typically handles image understanding tasks such as object localisation, image segmentation, and, particularly relevant for the topic of this chapter, object detection.

Recent advancements in object detection tasks for robotics, as detailed in a survey by Bai et al. [255], have been driven by the adoption of well-established and widely recognized

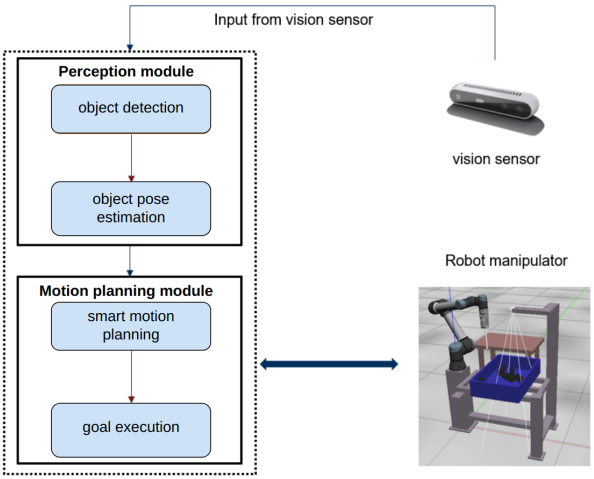


Figure 4.1: Overview of the architecture of a robotic arm . Adapted from [254].

CNN-based visual object detectors, such as Faster R-CNN [133], SSD [143], and YOLO [47]. However, for the successful design and deployment of a perception module based on a deep neural network, a large amount of training data is needed. Due to the specialised nature of tasks for industrial robots, acquiring such large training datasets poses a challenge, since images for these tasks are often not readily available in the public domain. Furthermore, after the initial training of a CNN model, the need for additional training data can recur: in particular, this can happen when changes in the environment of the deployed system cause a domain shift, making the original training data inadequate for representing the task the robot needs to perform.

A promising solution to the problem of the availability of data for designing perceptual systems in robotics is the use of simulations and synthetic data [253]. Leveraging synthetic data offers several advantages, including accelerating the design cycle, generating large amounts of data at low cost, and providing safe, fully controlled testing environments [256]. However, a number of unresolved challenges may affect robotic systems trained on synthetic data when they are deployed in real-life settings: in particular, their efficiency tends to decrease due to the differences between a simulation or synthetic data on the one hand and the real world on the other hand.

The problem of the gap between synthetic and real-world data is particularly relevant for the visual domain. On the one hand, the photorealism of rendered images, videos, and computer games has been improving steadily, and the tools for creating virtual environments with integrated physics simulation are becoming more user-friendly and accessible: for example, since such tools as Blender[1], Unity[2], and Unreal Engine[3] are available free of charge, researchers working on AI, computer vision, and robotics use them more frequently to generate data for training models or train systems directly in virtual environments. On the other hand, a decrease in precision is often observed in models trained exclusively on synthetic data compared to those trained on real data [257]. In robotic systems, this concern is closely related to the issue that is commonly referred to as 'the reality gap' [258]: even though good performance can be achieved in simulations, trained models may perform unreliably when transferred to a real environment. Therefore, developing methods for a more robust

[1] https://www.blender.org; accessed 19 September 2024.
[2] https://unity.com; accessed 19 September 2024.
[3] https://www.unrealengine.com; accessed 19 September 2024.

sim-to-real translation is a pivotal research topic in robotics.

The research reported in the present chapter is concerned with improving the accuracy of detecting plastic bottles with high visibility (i.e., those on the top of the pile of bottles) so that a robotic arm would be able to grasp them more efficiently. This specific task was set forth in the EDI part of the project *Intelligent Motion Control under Industry 4.E – IMOCO4.E*; from the practical perspective, it was envisaged as a contribution to the design of a robotic arm operating on a real-world production line, whereas from the scientific perspective, this work continued the exploration of the use synthetic data in robotics at EDI (see e.g. [259, 260, 261]). To detect graspable bottles, I used YOLOv5 [153], one of the most popular CNN-based object detectors; due to the already mentioned challenges with acquiring real-world data for training DNNs for robotics, I used synthetic data for that purpose. **The main goal** of the study reported in this chapter was to improve the accuracy of detecting plastic bottles with YOLOv5 object detector by enhancing the degree of photorealism of synthetic images. In particular, to make synthetic images more photorealistic, image-to-image translation with GANs [12] was leveraged. **The main hypothesis** of the study was that enhancing synthetic images of plastic bottles with GANs before training the YOLOv5 object detector on them would result in higher object detection accuracy compared to using unmodified synthetic images for training.

The rest of the present chapter is organised as follows. In Section 4.2, I outline related work; in Section 4.3, I describe the initial datasets used in the present study; in Section 4.4, I detail the methodology and the results of the experiments; finally, in Section 4.5, I offer concluding remarks.

4.2 Related work: object detection and sim-to-real translation for robotics

Despite some initial scepticism about the prospects of adopting deep learning to help robots to make sense of their environment (see [262] for an overview), DNN-based approaches to image understanding tasks rapidly became prevalent in robotics soon after their emergence in computer vision. Since industrial robotic systems operate in dynamic environments and need real-time information to function efficiently and safely, the object detection component of the perception module of such systems needs to be both accurate and fast. The second of these requirements, the need for fast inference with an object detector deployed as part of a robotic system, suggests considering single-stage rather than two-stage object detectors when designing an efficient robot. By design, single-stage object detectors prioritise speed by detecting objects in a single pass of the input image through the network, which makes them particularly suitable for applications involving real-time processing. Although single-stage object detectors may be less accurate than two-stage object detectors (see e.g. a comprehensive comparison by Carranza-García et al. [263]), as the latter employ a separate region proposal step followed by object classification and bounding box refinement, the critical requirement for fast decision-making in industrial robotic systems may favour the adoption of single-stage detectors despite the potential trade-off in accuracy.

One of the most popular single-stage object detectors is the YOLO family, starting with YOLOv1 in 2016 [47] and continuing up to the most recent (at the time of writing) YOLOv8 [135] and YOLO-NAS [136] (see Section 1.3.2 and a comprehensive survey by Terven et al. [156] for an overview). YOLO object detectors have been successfully applied to a wide variety of robotics use cases: to mention just a few examples, Tian et al. [264] used YOLOv2 to develop an object grasping system for a humanoid robot; Cao et al. [265] customised a

lightweight Tiny YOLOv2 [139] to detect shuttlecock for a badminton-playing robot; Zhaoxin et al. [266] designed a robotic system for picking tomatoes based on YOLOv5 and validated it through simulation. A number of studies (see, e.g., [267, 268, 269]) have also used various versions of YOLO as a component of the perception module of a robotic arm system. In the study reported in this chapter, I followed their approach and employed YOLOv5, the most recent version of YOLO at the time the study was conducted, as a state-of-the-art CNN model for object detection.

Regardless of how advanced the architecture of a CNN model for image understanding is, its performance can degrade when trained only on synthetic data, as the reality gap is a topical issue for the applications of ML [270]. To bridge or at least narrow the gap in the visual domain, one can attempt to transfer the appearance of a real-world environment to artificial data.

The main types of domain adaptation approaches are feature-level transfer [271] and pixel-level transfer [272, 273]. Feature-level transfer is concerned with learning domain-invariant features between source and target domains, whereas pixel-level transfer, which is typically based on image-conditioned GANs [12], focuses on image styling, i.e., images from a source domain are made to resemble images from a target domain. Overall, these methods can be used to address the simulation-to-reality domain shift in robotic manipulation tasks.

Despite the advantages offered by the domain adaptation approaches outlined above, they also have a substantial drawback, namely, they may alter images so as to cause the loss of information essential for a given task. This issue is particularly problematic for tasks involving object detection and robotic manipulation, since the complete or partial erasure of objects or their features in the training dataset can degrade the performance of the system. For this reason, it is common to combine domain adaptation with additional techniques such as semantic maps of simulated images [274], which preserve the semantics relevant for the task. Furthermore, additional loss functions can be introduced. Thus, reinforcement learning (RL) task loss [275] enforces consistency of task policy Q-values between the original and transferred images to preserve information important to a given task: RL-CycleGAN is trained jointly with the RL model and requires task-specific real-world episodes. Another approach is to add an object detector with perception consistency loss, which penalizes the generator for discrepancies in object detection between translations. RetinaGAN [276], which is based on this approach, uses object detection to ensure consistency across different domains.

In the work reported in this chapter, we[4] used pixel-level domain adaptation based on the use of GANs. To improve the quality of synthetic data, we transferred the domain from the synthetic to the real one (sim-to-real translation); our objective was to preserve objects in the images and keep them as recognisable as possible.

4.3 Initial datasets

The starting point for our experiments was two datasets: a real-world bottle image dataset and a synthetic bottle image dataset. I briefly describe both below.

[4]In the parts of this chapter concerned with synthetic data generation, I use 'we' rather than 'I', as work on that topic was led by my colleague Diana Duplevska.

(a) (b)

Figure 4.2: Sample images from the real-world (a) and original synthetic (b) dataset.

4.3.1 Real-world dataset

The dataset of real-world bottle images (see an example in Figure 4.2 (a)) was previously created by our colleagues at EDI. The real-world data were collected by placing bottles randomly in a plastic container and taking an image of the resulting visual scene. For each captured image, the positions of the bottles were altered by emptying the container and refilling it. Additionally, the camera exposure time and lighting intensity were systematically varied to acquire data with a high diversity of lighting conditions.

In total, the dataset consists of 2 060 real-world images with corresponding manual labelling of the objects. Of these images, 1 760 images were used for training GAN models for sim-to-real transfer (see Section 4.4.1), and the remaining 300 images comprised the test dataset for object detectors (see Section 4.4.3).

4.3.2 Synthetic dataset

For experiments in the study described in this chapter, a synthetic dataset, as described by Arents et al. [260], was generated; it consisted of 8800 photorealistic, high-resolution images (see an example in Figure 4.2 (b)) of bottles in a box. These images were used in object detection experiments (see Section 4.4.3) both in their initial form, i.e., exactly as rendered, and after sim-to-real translation with GANs as described in Section 4.4.1.

To generate the original synthetic images, the Blender physics simulation engine [277] was used. An initially empty box in the simulation was filled with bottles that were randomly dropped into it for each scene. This approach allowed for the realistic generation of random bottle configurations within the container. After filling the box with bottles, the intensity of four different light sources was varied in the simulation, and the scene was rendered from 16 different angles. Additionally, for increased realism, Blender shader nodes, realistic textures, reflections, and indirect light bounces were used.

Finally, an annotation file was generated for each rendered scene; it included every object in the scene, its rotation angle, coordinates, and visibility percentage. Bottles were considered

graspable if their visibility was above 60%.

4.4 Experiments: methodology and results

4.4.1 Sim-to-real transfer with CycleGAN

CycleGAN: overview of architecture and training procedure

We employed GANs [12] for image-to-image translation to enhance the photorealism of the synthetic dataset described in Section 4.3 and thus narrow the sim-to-real gap. For this purpose, we selected a specific GAN architecture, the Cycle-Consistent Adversarial Network (CycleGAN) [278], since it was designed to find a mapping from the domain X to the domain Y without requiring the training data to consist of matching image pairs. The absence of such a requirement was particularly important for our work, as otherwise, we would have had to create matching pairs of real-world and synthetic images, a task that would have been rather cumbersome.

CycleGAN employs two mapping functions: $G : X \rightarrow Y$ to translate images in domain X to domain Y, and an inverse mapping $F : Y \rightarrow X$ for translating images in domain Y back to domain X. Additionally, there are two discriminator functions D_X and D_Y, which are used to determine whether an image is in a respective domain; in this case, X corresponds to synthetic data, and Y corresponds to real-world data. Each mapping function, along with its associated discriminator function, has a generative adversarial loss. Furthermore, the inverse mapping introduces a cycle consistency loss to ensure $F(G(X)) \approx X$ and otherwise $G(F(Y)) \approx Y$. A schematic overview of the CycleGAN architecture is presented in Figure 4.3.

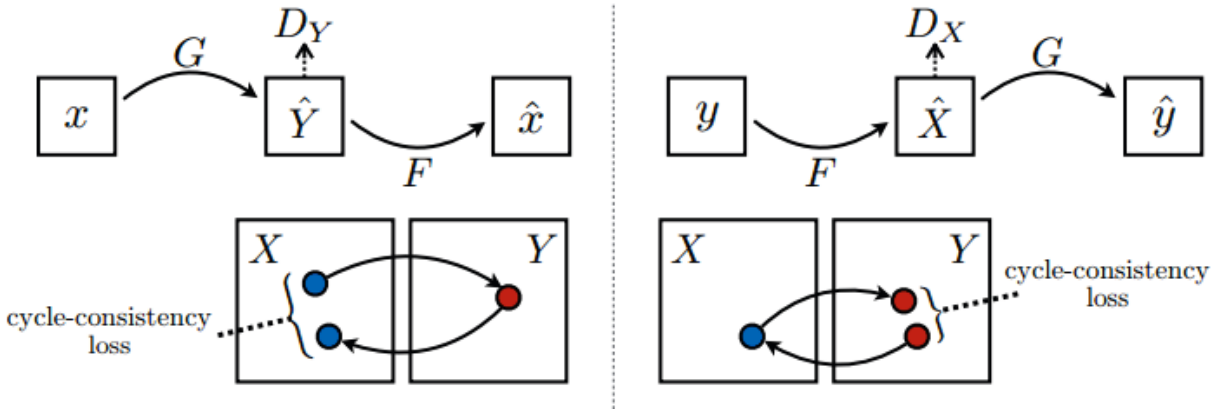


Figure 4.3: CycleGAN data generation algorithm. Reproduced from [278].

We began our experiments using the CycleGAN implementation in TensorFlow[5] on the datasets described in Section 4.3. The primary difference between the original CycleGAN proposed by Zhu et al. [278] and the TensorFlow implementation is that the former employs a modified ResNet-based generator [76], whereas the latter is based on a modified U-Net [279] generator for simplicity. U-Net (see Figure 4.4) is a convolutional autoencoder with skip connections: the encoder downscales the image using convolutional layers, whereas the decoder upscales the latent space back to the original dimensions. The goal of adding a skip connection to each transposed convolutional layer in the decoder is to mitigate the vanishing

[5] https://www.tensorflow.org/tutorials/generative/cyclegan; accessed 10 February 2024.

gradient problem by concatenating the output of a layer to multiple layers rather than just one.

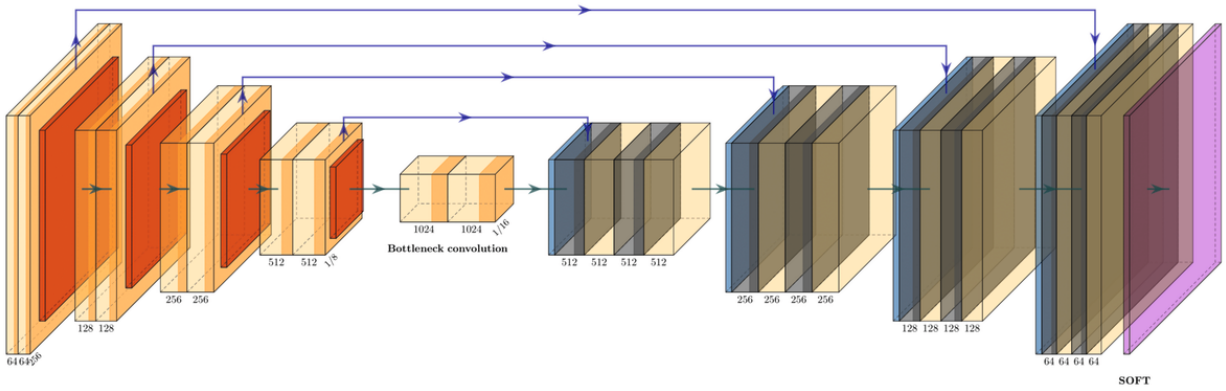


Figure 4.4: U-Net [279] network architecture. Reproduced from [251].

To conduct experiments with CycleGAN, several data preprocessing steps were necessary. First, from the synthetic dataset containing 8 800 images, we selected only those with good lighting to ensure that the features of the objects were clearly visible, facilitating training of neural networks. As a result, the synthetic dataset used for training CycleGAN consisted of 1 760 images. Second, we cropped the backgrounds in the synthetic images to preserve important parts of the images before resizing (see the next step). Third, we resized the images, as the CycleGAN requires images of the same size for training. The original real-world photos were 528×342 pixels, whereas the synthetic images were 1024×768 pixels; after resizing, all images were 256×256 pixels. This reduction in resolution was also necessary due to the limited GPU resources that were at our disposal for training CycleGAN.

The training procedure was consistent across all experiments with CycleGAN: each model was trained for 20 epochs using the Adam optimiser with an initial learning rate of 0.0002, $\beta = 0.5$ and $\lambda = 10$. The weights were initialized with a Gaussian distribution with a mean of 0 and a standard deviation of 0.02. For each epoch, the dataset was shuffled, and the buffer size was set to 1000.

Datasets generated with CycleGAN

As a result of training CycleGAN to transfer the style of the real-world photos of bottles to the synthetic images, several datasets were generated. These datasets are described below; sample images from each dataset are shown in Table 4.1.

Baseline CycleGAN dataset. In the first experiment, CycleGAN was trained with the parameters listed above without any additional adjustments. As shown in Figure 4.5, the resulting neural network correctly translated and drew the shape of the box; however, the bottles that were in the shadow in the original synthetic image were partially erased after the transfer. Furthermore, the neural network drew non-existent bottles in the top part of the resulting image. We considered the quality of such sim-to-real transfer insufficient for image understanding tasks, and for this reason, this dataset, which I refer to as the *Baseline* CycleGAN dataset, was not used for training object detectors.

***Augmented* CycleGAN dataset.** To improve the quality of the generated images, we added several data preprocessing functions: in addition to image resizing, jittering, mirroring

(a) (b)

Figure 4.5: Sim-to-real transfer results: (a) a synthetic image used as input; (b) an output image generated using Baseline CycleGAN. Note artefacts of sim-to-real translation – some non-existent bottles are drawn by CycleGAN in the top left corner of the generated image.

and data normalisation, we used colour data augmentation techniques, namely, random contrast, brightness, hue, and saturation. The aim of using these preprocessing functions was to reduce overfitting. We also added a central crop while maintaining the image size of 256×256, which allowed the neural network to focus on the bottles rather than the background and box.

***Augmented noise* CycleGAN dataset.** In an attempt to further improve the quality of the sim-to-real translation, we added Gaussian noise to the discriminator input to make it more difficult for the discriminator to evaluate images, thus allowing the generator to train longer. This method helps to avoid early overfitting of the discriminator; an overfitted discriminator would evaluate even high-quality images generated by the network as fake, leading to an imbalance in the neural network.

***Resized convolution* CycleGAN dataset.** While we considered the *Augmented* CycleGAN dataset suitable for training object detectors, there were still some observable image defects, eliminating which could potentially further improve the quality of generated images. The most noticeable problem was a small pixel grid in the image, known as checkerboard artifacts [280], which occurs because of the transposed convolution [281] operations in the decoder. Therefore, to solve this problem, it was necessary to replace the transposed convolution operation with some other operation. One possible solution was to resize the images and then add a regular convolutional layer; we chose a different approach, namely, we replaced the transposed convolution with a TensorFlow function `tf.keras.layers.UpSampling2D` to upscale the images. By using such resizing with convolution, we removed the checkerboard artifacts, yet the images became blurred, which is a substantial problem for object detection, because when the edges of objects blur, their boundaries become imprecise.

***Resized transpose* CycleGAN dataset.** To address the issue of the blurred edges that we identified in the images in *Resized convolution* CycleGAN dataset, we applied resized convolution on all layers except the last one, where we retained the transposed convolution. This method reduced the blurriness, and the checkerboard grid became hardly noticeable.

4.4.2 Evaluation of datasets generated with CycleGAN with FID score

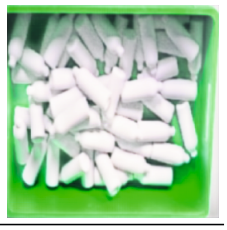
We used the Frechet Inception Distance (FID) score [282] to evaluate the quality of the images in the datasets generated with different versions of CycleGAN. The FID score is a performance metric that calculates the distance between the feature vectors of original images (in this case, either the real-world images, or the synthetic images as originally rendered with Blender) and the feature vectors of enhanced images, i.e., images generated by a GAN. It is given by the following formula:

$$\text{FID} = \|\mu_o - \mu_e\|^2 + \text{Tr}(\Sigma_o + \Sigma_e - 2\sqrt{\Sigma_o \Sigma_e}), \tag{4.1}$$

where μ_o is the mean vector of the feature distribution for the original images, μ_e is the mean vector of the feature distribution for the enhanced images, $\|\cdot\|$ is the Euclidean distance between the mean vectors, Σ_o is the covariance matrix of the feature distribution for the original images, Σ_e is the covariance matrix of the feature distribution for the enhanced images, and Tr is the trace of a matrix. Importantly, a lower FID score indicates a smaller difference between the original and the enhanced images, implying better sim-to-real transfer results.

To obtain FID scores, we used the TensorFlow-GAN (TF-GAN) library[6]. In total, 5 000 images were used; this amount of data was obtained using several augmentation techniques: random horizontal and vertical flips, rotations, and hue changes. The FID scores were calculated by comparing CycleGAN-generated images with the real-world images (FID A score) and the original synthetic images (FID B score); these comparisons allowed us to investigate the similarity between the enhanced images and their original synthetic versions as well as how similar they were to the real-world images, i.e., the target domain. All the FID scores and sample images from each dataset are shown in Table 4.1. As the results show, the worst FID scores for both FID A and FID B were obtained for the Baseline CycleGAN dataset. When comparing enhanced images with the real-world images (FID A), the best FID score was obtained for the *Resized transpose* CycleGAN dataset; however, when comparing the enhanced images with the original synthetic images (FID B), the best FID score was obtained for the *Resized convolution* CycleGAN dataset.

Table 4.1: Evaluation of sim-to-real transfer with CycleGAN using FID score.

Dataset	Baseline	*Augmented*	*Augmented noise*	*Resized convolution*	*Resized transpose*
Images					
FID A	230.28	171.41	137.27	141.03	**112.26**
FID B	264.81	169.73	152.29	**122.86**	127.25

FID A evaluates the distance between the images enhanced with CycleGAN and the real-world images; FID B evaluates the distance between the images enhanced with CycleGAN and the original (i.e., as rendered with Blender) synthetic images.

For further investigation, we chose two different types of CycleGAN modified from the

[6] https://github.com/tensorflow/gan; accessed 18 May 2024.

basic CycleGAN – *Augmented noise* and *Resized transpose* – and compared the quality of large images produced with them. These two types of CycleGANs have different methods for image resizing in the CNN part, which can affect image enhancement in different ways. Thus, on small input images, defects are imperceptible and do not affect the FID score, but on large images, defects and artefacts may appear that affect the image semantics. We applied *Augmented noise* and *Resized transpose* CycleGAN to the original synthetic images with a resolution of 1024×768 pixels and bright lighting. As a result, we created two more datasets: *Augmented noise* CycleGAN 1024×768 dataset, and *Resized transpose* CycleGAN 1024×768 dataset. We could not compare these datasets using the FID A score with the real-world images due to their differing sizes and semantics; therefore, we only compared them with the original synthetic images by calculating FID B score to investigate how the image quality changed after enhancement with CycleGAN. The results of the comparison and sample images are provided in Table 4.2.

Table 4.2: Evaluation of sim-to-real transfer with CycleGAN using the FID score for images with a resolution of 1024×768 pixels and constant brightness level.

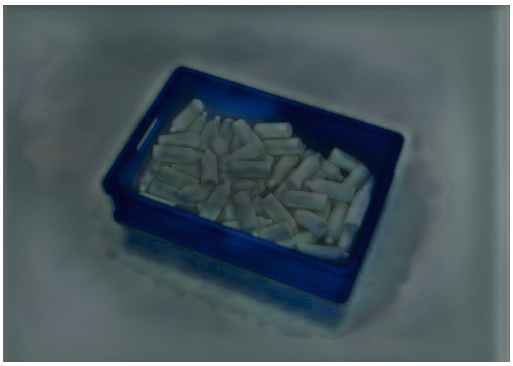
Dataset	*Augmented noise* 1024×768	*Resized transpose* 1024×768
Images		
FID B	**58.46**	69.24

As the lower FID scores in Table 4.2 compared to those in Table 4.1 indicate, the quality of images with the resolution 1024×768 pixels after sim-to-real transfer was better than that of the images with the resolution of 256×256 pixels. Furthermore, when comparing the *Augmented noise* CycleGAN 1024×768 dataset with the *Resized transpose* CycleGAN 1024×768 dataset, it can be seen that the pixel grid in the former was less noticeable, resulting in a smaller impact on the FID score than the blurred objects without the pixel grid in the latter dataset. As a result, the CycleGAN 1024×768 dataset had a better FID score than the *Resized transpose* CycleGAN 1024×768 dataset.

While the increase in the resolution of enhanced images resulted in an improved FID score, it should be noted that these results were obtained by conducting experiments only on large images with excellent lighting, yet the subsequent object detection experiments were to be conducted on images with different brightness parameters. To address this issue, two additional datasets were created by enhancing original synthetic images with various lighting conditions, ranging from very bright to dark, and different light source positions, using *Augmented noise* CycleGAN and *Resized transpose* CycleGAN. When applying *Augmented noise* CycleGAN to these images, the semantics of the images did not change, though some artifacts appeared in the background. However, when *Resized transpose* CycleGAN was used on images with medium brightness, artefacts appeared in the form of pixels around the objects (bottles and the box), which could interfere with the semantics of the images such as

the shapes of the objects. The dataset in question had a relatively high FID score, suggesting that the image quality was worse in this case and would likely result in decreased precision for the object detection task. Therefore, despite the consideration that it would be useful to account for varying lighting conditions, the datasets with the constant rather than varying brightness level were used for the object detection experiments reported in Section 4.4.3.

Table 4.3: Evaluation of sim-to-real transfer with CycleGAN using the FID score for images with a resolution of 1024 × 768 pixels and varying brightness level.

Dataset	*Augmented noise* 1024 × 768 with varying brightness levels	*Resized transpose* 1024 × 768 with varying brightness levels
Bright images		
Dark images		
FID B	**80.11**	124.83

4.4.3 Object Detection Experiments

Methodology

I conducted object detection experiments using the the YOLOv5 object detector [153] implemented in the Ultralytics library[7]. The experiments were conducted on each dataset with the YOLOv5 Small, Medium, and Extra Large models pretrained on the MS COCO dataset [141]; as suggested by the model names, they differ by the number of parameters: 7.2 million, 21.2 million, and 86.7 million, respectively.

The models were trained on three datasets: the original dataset of synthetic images, the *Augmented noise* CycleGAN 1024 × 768 dataset, and the *Resized transpose* CycleGAN 1024 × 768 dataset. During the training, 90 percent of the images in each dataset were used for training, and 10 percent were used for validation. The models were trained with the following parameters: an input image size of 640 × 640 pixels, a batch size of 16, a learning rate of 0.01, a momentum of 0.937, and a weight decay of 0.0005. Each model was trained for 300 epochs with early stopping after 100 epochs if there was no validation loss improvement.

[7] https://github.com/ultralytics; accessed 11 November 2024.

After training, the checkpoint with the best performance on the validation data was retrieved and tested on a set 300 real-world test images.

Results

I report the results of the object detection experiments in Table 4.4, using standard metrics for the task of object detection, namely, precision, recall, and mean average precision (mAP) for bounding boxes. mAP is calculated for an intersection over union (IoU) threshold of 0.5 as well averaged for IoU threshold from 0.5 to 0.95 in steps of 0.05.

As shown in Table 4.4, the models trained on the *Augmented noise* CycleGAN 1024×768 dataset consistently outperformed both the models trained on the original synthetic dataset and the models trained on the *Resized transpose* CycleGAN 1024×768 dataset in terms of precision, recall, and mAP for IoU with threshold of 0.5 and mAP averaged for IoU thresholds $\in [0.5 : 0.05 : 0.95]$. In contrast, the performance of the models trained on another enhanced dataset, *Resized transpose* CycleGAN 1024×768, was consistently worse than that of the models trained on the original synthetic data, with the sole exception being the better recall of the Extra Large model. Another noteworthy observation is that when comparing the performance of different model sizes on the same datasets, larger models did not consistently outperform their smaller counterparts: for instance, both the mAP for IoU with threshold of 0.5 and mAP averaged for IoU threshold $\in [0.5 : 0.05 : 0.95]$ were better for YOLOv5 Small trained on the *Augmented noise* CycleGAN 1024×768 dataset than for YOLOv5 Extra Large trained on the same dataset. This may indicate that the larger object detectors employed in the experiments were overfitting on the training dataset, which was comparatively small in size.

Table 4.4: Results of the object detection experiments with YOLOv5 on the original synthetic data and synthetic data enhanced with CycleGAN.

Model	**Dataset**	**Precision**	**Recall**	**mAP (threshold 0.5)**	**mAP (avg for IoU $\in$ [0.5 : 0.05 : 0.95])**
YOLOv5 Small	Original synthetic	68.9	89.4	74.2	44.7
	Resized transpose 1024×768	61.5	83.8	63.2	24.5
	Augmented noise 1024×768	**73.1**	**90.5**	**77.7**	**48.0**
YOLOv5 Medium	Original synthetic	69.3	80.2	72.0	42.0
	Resized transpose 1024×768	58.1	78.2	58.9	20.4
	Augmented noise 1024×768	**71.2**	**91.0**	**75.3**	**45.5**
YOLOv5 Extra Large	Original synthetic	71.8	78.5	75.0	41.3
	Resized transpose 1024×768	68.3	86.7	71.5	33.9
	Augmented noise 1024×768	**72.1**	**87.2**	**76.1**	**46.1**

4.5 Concluding remarks

The goal of the study reported in this chapter was to improve the accuracy of detecting plastic bottles for a bin-picking task by enhancing the photorealism of synthetic images using CycleGAN. In our initial experiment, we employed the original implementation of CycleGAN in TensorFlow for sim-to-real image translation and generated the Baseline CycleGAN dataset; in subsequent experiments, we used improved versions of the CycleGAN architecture and generated several other datasets: *Augmented* CycleGAN, *Augmented noise* CycleGAN, *Resized convolution* CycleGAN, and *Resized transpose* CycleGAN datasets. Evaluation of these datasets by means of the FID score demonstrated that the best sim-to-real transfer results were achieved with the *Resized transpose* CycleGAN and *Augmented noise* CycleGAN models. To further enhance the images, we used the *Resized transpose* CycleGAN and *Augmented noise* CycleGAN models to generate images with higher resolution, namely, 1024×768 pixels, compared to 256×256 pixels in prior experiments. As a result, improved FID scores were achieved for these new datasets, with the *Augmented noise* CycleGAN dataset obtaining a better FID score than the *Resized transpose* CycleGAN dataset. However, our attempts to further improve the FID score by applying sim-to-real transfer not only to bright images, as in the case of the *Resized transpose* CycleGAN 1024×768 and *Augmented noise* CycleGAN 1024×768 datasets, but also to images with varying lighting conditions and light source positions, were unsuccessful, yielding worse FID scores due to new artefacts. Therefore, the best results for sim-to-real transfer in our study were achieved with the *Resized transpose* CycleGAN 1024×768 and *Augmented noise* CycleGAN 1024×768 datasets, which were then used in object detection experiments.

The goal of the object detection experiments was to compare the object detection accuracy of models trained on the original Blender-generated synthetic data with those trained on enhanced synthetic images. I trained the YOLOv5 object detectors of three different sizes on the original synthetic dataset, the *Resized transpose* CycleGAN 1024×768 dataset, and the *Augmented noise* CycleGAN 1024×768 dataset; after training, the models were tested on real-world images. While the object detectors trained on the *Resized transpose* CycleGAN 1024×768 dataset performed worse than those trained on the original synthetic data, the models trained on the *Augmented noise* CycleGAN 1024×768 dataset outperformed the models trained on the original synthetic data across all metrics of interest: precision, recall, and mAP. These results demonstrated that CycleGAN can be successfully used for sim-to-real translation of synthetic datasets for bin-picking tasks.

The experiments reported in this chapter are the first step in the overall development of a bin-picking pipeline. While the object detectors trained in these experiment detect objects that are the most promising for a successful grasp in 2D images, the proposed approach can also be utilised for further steps such as instance segmentation and grasp pose estimation in 3D. Furthermore, the synthetic data generated with Blender contain sufficient 3D information to be used by different approaches, such as directly performing 6D object pose estimation (see e.g. [283]). However, a detailed plan of how the approach developed in this study can contribute to these steps is beyond the scope of this chapter and is envisaged as future work.

Chapter 5

Image Classification for Monitoring the Growth of Organs-on-a-Chip

In this chapter, I describe the development of CNN-based image classifiers for monitoring the growth of organs-on-a-chip (OOC; see [284] for a comprehensive survey), a promising emerging technology in biomedical research. Given that both the initial and final datasets of OOC microscopy images for training and evaluating CNN models were relatively small, as is often the case with studies involving biomedical image understanding, I augmented them with synthetic data, attempting to improve the accuracy of the classifiers. Therefore, in addition to exploring the applications of deep learning methods in biomedicine, this chapter also further examines the approach of training CNNs on a mix of real-world and synthetic data, which I began to explore in Chapters 2, 3, and 4.

This chapter builds upon the following scientific papers:

[285] **M. Ivanovs**, L. Leja, K. Zviedris, R. Rimsa, K. Narbute, V. Movcana, F. Rumnieks, A. Strods, K. Gillois, G. Mozolevskis, A. Abols, and R. Kadikis, "Synthetic image generation with a fine-tuned latent diffusion model for organ on chip cell image classification," in *2023 Signal Processing: Algorithms, Architectures, Arrangements, and Applications (SPA)*, pp. 148-153, IEEE, 2023.

[286] V. Movčana, A. Strods, K. Narbute, F. Rūmnieks, R. Rimša, G. Mozoļevskis, **M. Ivanovs**, R. Kadiķis, K. Zviedris, L. Leja, A. Zujeva, T. Laimiņa, and Arturs Abols, "Organ-On-A-Chip (OOC) Image Dataset for Machine Learning and Tissue Model Evaluation", *Data*, vol. 9, issue 2, 2024.

As the first author of [285], I was responsible for planning and conducting experiments with both real-world and synthetic data. I also led the analysis of experimental results as well as had a principal role in drafting, revising, and finalising the manuscript in collaboration with other co-authors. As the lead author from the field of Computer Science in [286], I ensured that the image dataset published as an integral part of that publication was suitable for ML purposes; I was also in charge of validating CNN models on that dataset.

5.1 Introduction

Recently, CNNs have successfully been applied to a broad range of biomedical image understanding tasks, such as the segmentation of magnetic resonance images (MRI) for diagnosing Alzheimer's disease [287], the segmentation of endoscopic and nuclei images [288], and the

classification of skin lesions [289] and chest X-ray images [290]. Remarkably, they have even outperformed human experts in some of these tasks [291]. In the work reported in the present chapter, I used CNNs to classify biomedical microscopy images to monitor the growth of OOC. To underscore the practical importance of this task, I provide an insight into the research on OOC in the following.

OOC technology combines tissue engineering and microfluidics to imitate key aspects of human physiology, with the aim of recreating the environment of particular human organs *in vitro*[1]. The primary assumption is that certain functions of a human organ can be replicated by growing respective cells (e.g., gut epithelial cells to imitate intestines, or lung epithelial and endothelial cells to imitate lungs) in horizontal microfluidic channels separated by a porous membrane. In an OOC setup, culture media flows over both sides of the membrane, ensuring that cells are supplied with nutrients and metabolic waste is removed. Furthermore, the flow exerts shear stress on the cells, similar to that exerted on cells in a living system, thereby creating a more physiologically plausible environment.

A successfully operating OOC setup needs to ensure that the tissue models are representative of their originals both morphologically and functionally; to achieve this, daily monitoring of the samples is essential. Such monitoring can be improved and automated by using a machine learning system-controlled rather than a human-supervised OOC cultivation system, which would reduce the operating costs of the system and increase its output.

One of the key requirements for the design of such a system as defined in the project *AI-Improved Organ on Chip Cultivation for Personalised Medicine – AImOOC*, in which the research reported in this chapter was carried out, was the ability to classify the condition of the cell sample on a chip (accessible via microscopy images acquired with the camera attached to an optical microscope) as 'good', 'acceptable', or 'bad' (Figure 5.1). Depending on the classification outcome, the intensity of the flow of the solution to the chip should be maintained, or increased, or the experiment should be stopped altogether if it is deemed unproductive to continue.

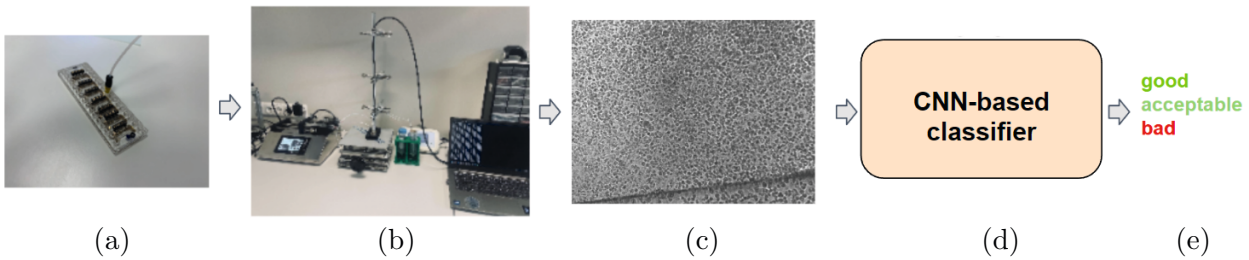


Figure 5.1: Schematic overview of the automated system for monitoring the growth of OOC: (a) a chip for growing OOC; (b) an OOC setup; (c) an image of OOC tissue acquired with a light microscope; (d) an image classifier; (e) an output of the classifier.

Given that state-of-the-art results on image classification tasks have been achieved with CNNs, using a CNN-based classifier as a part of the automated system in question is a promising solution. However, when implementing such a system, the issue of data availability for training a CNN model inevitably arises, as the number of images for training and evaluating CNN models in the *AImOOC* project was expected to be comparatively small, ranging from several hundred to several thousand, because of the current throughput of non-automated OOC systems. Furthermore, to the best of my knowledge, at the time when the *AImOOC* project was carried out, there were no publicly available datasets that would be suitable for

[1]I.e., not in a living organism, but rather in a controlled artificial environment.

that task, as the images obtained with the OOC setup designed by the EDI partners in the *AImOOC* project had a rather specific appearance (see a sample image in Figure 5.1).

To address the issue of limited data availability, I adopted the promising approach of using a large generative model for image synthesis. Such highly popular large text-to-image models as Midjourney [157], DALL·E [158] and Stable Diffusion [13] have demonstrated impressive capabilities in generating artificial images for various purposes – from book illustrations to visuals for advertising campaigns to synthetic X-ray images [292] – and are available in implementations that are quite easy to use. However, one should take into account that these models have not originally been trained on the data that would make them capable of generating such domain-specific data as OOC biomedical images; furthermore, retraining them from the ground up would require a lot of computational resources and therefore would be prohibitively expensive and time-consuming. Nevertheless, it is possible to produce synthetic counterparts of the real-world OOC images by such means as e.g. image-to-image translation as well as to fine-tune large generative models on a small amount of the data on consumer-grade computer hardware employing such recently proposed methods for that as low-rank adaptation (LoRA; [293]).

The goal of the research reported in the present chapter was to develop a classifier for OOC microscopy images. To achieve it, two studies were conducted. In the first study, I trained EfficientNet-B7 [107] CNN on both the initial real-world dataset of microscopy OOC images and the augmented dataset that included synthetic images generated with the Stable Diffusion model fine-tuned with LoRA. In the second study, the team of ML researchers that I led trained EfficientNet-B7 and MobileNetV3Large [106] CNNs on a larger real-world dataset of microscopy OOC images [286] and the augmented dataset, in which real-world images were supplemented with their synthetic counterparts generated with various generative AI methods available for the Stable Diffusion model, such as image-to-image translation, inpainting with masks, interpolation, and fine-tuning with LoRA. As in the research reported in Chapter 3, experiments with augmented datasets in both studies reported in this chapter involved adding different proportions of synthetic data to the real-world dataset to investigate how the proportion of synthetic data would affect the performance of the classifier. For both studies that I report in the following, I proposed the same two **hypotheses**, namely:

- **Hypothesis 1**: A CNN-based classifier achieves better accuracy on the real-world microscopy OOC image dataset than a putative 'naive' classifier.
- **Hypothesis 2**: The classification accuracy on the real-world microscopy OOC image dataset improves when a CNN-based classifier is trained on the dataset augmented with synthetic data generated with the Stable Diffusion model rather than solely on the real-world image dataset.

The structure of the rest of the chapter is as follows. In Section 5.2, I discuss related work; in Section 5.3, I describe the first study, which was conducted on the initial OOC image dataset; Section 5.4.1 is concerned with the second study, which was conducted on the final OOC image dataset; finally, in Section 5.5, I offer some concluding remarks.

5.2 Related work

5.2.1 Synthetic data for training CNN models for biomedical image understanding tasks

The use of synthetic data for training CNNs for various image understanding tasks in the biomedical domain has recently become increasingly popular, as it can help address the scarcity of real-world data, both in terms of overall dataset size and the underrepresentation of certain classes. Beyond the common challenges of using synthetic data, such as bridging the gap between synthetic and real-world domains, there are additional challenges specific to biomedical image understanding tasks. In particular, while many other domains have readily available image synthesis tools – e.g., it is possible to acquire artificial images of street views in video games [243, 294], or generate them with a car driving simulator, as I did in the research reported in Chapter 3 – such solutions are often unavailable for more niche biomedical purposes. Furthermore, biomedical images tend to be high-resolution, which may require more powerful hardware and longer computation times to generate synthetic data. Yet another challenge is evaluating generated synthetic images: while the quality of general-purpose synthetic images – such as faces, cars, street views, human bodies and other common scenes and objects – can at least preliminary be evaluated visually without additional expertise, evaluating artificial biomedical images may require expert knowledge, which can lengthen the iteration cycle for improving synthetic data.

Some of the popular methods for generating synthetic data for biomedical purposes are Variational Autoencoders (VAEs; [295]) and Generative Adversarial Networks (GANs). VAE is a generative model that combines the autoencoder architecture with Bayesian approaches to encode the inputs into a latent space and then generate new data by decoding from it. Since VAEs can efficiently model large datasets, they have been successfully used to generate biomedical data such as brain MRI [296] and endoscopic images [297]. However, VAEs also tend to produce blurry output due to learning non-informative latent codes [298] and unrealistic distributions of prior vs posterior data [299]. GANs, which were used in the research reported in Chapter 4, have also been successfully applied to a number of biomedical image understanding tasks, such as generating blood cell images [300] and images of retinal blood vessels [301]. However, as we could see in Chapter 4, adapting a GAN to a specific task can require a number of iterative changes by trial and error, and all in all, training GANs has a reputation in the ML community for being challenging due to the frequently occurring issues such as mode collapse, instability of the model, and non-convergence [302]. Therefore, while these approaches, especially the use of GANs, remain a lively area of research with substantial potential for applications in biomedicine, it may be beneficial to explore other robust and simpler means of image synthesis.

5.2.2 Large generative models for image synthesis

A promising alternative to the smaller generative architectures mentioned in the previous section is the use of large generative models for image synthesis. These DNN-based models, trained on very large datasets containing a broad range of imagery, can generate images with complex semantics and a high degree of photorealism in response to simple text prompts given in natural language. Some of the most popular large text-to-image models are Midjourney [157], DALL·E [158], and Stable Diffusion [13]. In addition to the high quality of output, another advantage of these models is their ease of use, as they are accessible via user-friendly interfaces, either through online platforms or, in the case of Stable Diffusion, locally as well,

and do not require any advanced skills to run them in a standard configuration.

Several studies have already leveraged large text-to-image models to generate biomedical images: Akrout et al. [303] generated images of skin diseases with Stable Diffusion; Ali et al. [292] created realistic X-ray and computed tomography synthetic images of lungs with Stable Diffusion and DALL·E 2; Chambon et al. [304] generated X-ray images of lungs with a fine-tuned Stable Diffusion model. The main challenge in applying large text-to-image models for generating biomedical images is adapting these models to the biomedical domain. This adaptation is necessary due to the specific semantics of target images, which differ from the general nature of images these models were originally designed to produce. Therefore, it may be more practical to use open-source large text-to-image models instead of proprietary ones, as the former offer more options for modifying (e.g., by means of fine-tuning) the foundational model. Since the most popular open-source generative AI model is currently Stable Diffusion [13], a latent text-to-image diffusion model, I chose to use it in the studies reported in this chapter and describe it in the following.

5.2.3 Stable Diffusion model for image generation

Diffusion models, first introduced in the context of generative modelling by Sohl-Dickstein et al. [305], are deep latent variable models that generate new data, such as images, by means of a two-stage process (see Figure 5.2): in the forward diffusion stage, Gaussian noise is added to the training data, slowly corrupting it, whereas in the reverse diffusion stage, the original input data is recovered by gradually reversing the diffusion process [306].

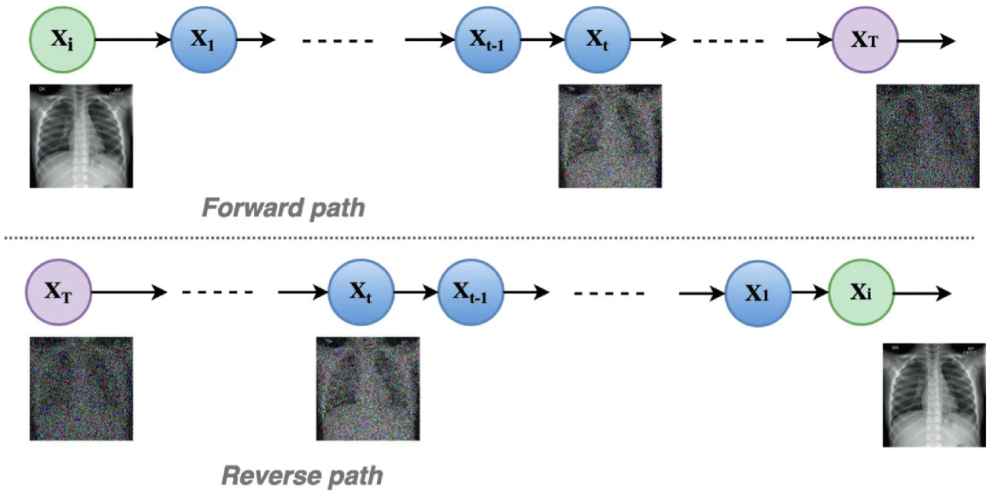


Figure 5.2: The two stages of diffusion model training: the forward and the reverse diffusion stage. Reproduced from [292].

More formally, the forward diffusion stage, starting from the original data $\mathbf{x}_0 \sim q(\mathbf{x}_0)$ and proceeding for T iterations, is defined as

$$q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) = \mathcal{N}(\mathbf{x}_t; \sqrt{1-\beta_t}\mathbf{x}_{t-1}, \beta_t \mathbf{I}) \text{ [307]}, \tag{5.1}$$

$$q(\mathbf{x}_{1:T} \mid \mathbf{x}_0) = \prod_{t=1}^{T} q(\mathbf{x}_t \mid \mathbf{x}_{t-1}) \text{ [307]}, \tag{5.2}$$

where the hyperparameter β_t (for $t \in \{1, \ldots, T\}$), the variance scheduler, is a small positive constant that controls the amount of noise added at step t and is set such that $\mathbf{x}_T$ approximates a standard Gaussian distribution [307]. The reverse diffusion stage is defined as

$$p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t), \Sigma_\theta(\mathbf{x}_t, t)) \text{ [307]}, \tag{5.3}$$

$$p_\theta(\mathbf{x}_{0:T}) = p(\mathbf{x}_T) \prod_{t=1}^{T} p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t) \text{ [307]}, \tag{5.4}$$

where θ are parameters learned by the diffusion model during the training.

To make it possible to impose conditions (for instance, a text prompt for text-to-image translation) on the data generation by the diffusion model, a set of conditions $\mathbf{c}$ can be incorporated into the reverse diffusion stage, which therefore becomes

$$p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{c}) = \mathcal{N}(\mathbf{x}_{t-1}; \mu_\theta(\mathbf{x}_t, t, \mathbf{c}), \Sigma_\theta(\mathbf{x}_t, t, \mathbf{c})) \text{ [307]}. \tag{5.5}$$

The training of the diffusion model involves minimising a variational lower-bound on the negative log-likelihood:

$$\mathcal{L}_{vlb} = -\log p_\theta(\mathbf{x}_0 \mid \mathbf{x}_1) + KL\left(p(\mathbf{x}_T \mid \mathbf{x}_0) \| \pi(\mathbf{x}_T)\right) + \sum_{t>1} KL\left(p(\mathbf{x}_{t-1} \mid \mathbf{x}_t, \mathbf{x}_0) \| p_\theta(\mathbf{x}_{t-1} \mid \mathbf{x}_t)\right) \text{ [306]}, \tag{5.6}$$

where KL stands for the Kullback-Leibler divergence between the true distribution p and the model distribution p_θ. As Ho, Jain, and Abbeel [308] demonstrated, the above training objective can be effectively replaced by a simpler one, namely,

$$\mathcal{L}_{simple} = \mathbb{E}_{t \sim [1,T]} \mathbb{E}_{\mathbf{x}_0 \sim p(\mathbf{x}_0)} \mathbb{E}_{\mathbf{z}_t \sim \mathcal{N}(0, \mathbf{I})} \left\| \mathbf{z}_t - \mathbf{z}_\theta(\mathbf{x}_t, t) \right\|^2 \text{ [306]}, \tag{5.7}$$

where $\mathbb{E}$ denotes the expected value, and $\mathbf{z}_\theta(\mathbf{x}_t, t)$ is the network's prediction of the noise at time step t.

While diffusion models have achieved impressive results in image generation tasks, they have a notable shortcoming: due to their complexity, substantial computational resources are required both for training and for performing inference (i.e., generating synthetic data) [13]. Therefore, as Esser et al. observe [309], work on making these stages of the life cycle of diffusion models less computationally demanding has recently become a growing area of research. One of the most notable recent studies on that topic was conducted by Rombach et al. [13], who replaced the computationally expensive pixel space that diffusion models typically operate upon with the more compact lower-dimensional space that is perceptually equivalent to the original one. To achieve this, the training was divided into two distinct phases: first, an autoencoder was trained on the original data, removing imperceptible details from the data space and producing its low-dimensional latent representation; second, a diffusion model was trained on this latent space. More formally, the encoder $\mathcal{E}$ encodes an image $x \in \mathbb{R}^{H \times W \times 3}$ into a latent representation $z = \mathcal{E}(x)$, and the decoder $\mathcal{D}$ subsequently reconstructs the image from z, returning $\tilde{x} = \mathcal{D}(z) = \mathcal{D}(\mathcal{E}(x))$ ($z \in \mathbb{R}^{h \times w \times c}$) [13]. $\mathcal{E}$ downsamples the input image by a factor $f = \frac{H}{h} = \frac{W}{w}$ [13]; Rombach et al. [13] empirically found that factor f in the range $4 \leq f \leq 16$ strikes a good balance between computational efficiency on the one hand and the quality of the output of their latent diffusion model (LDM) on the other hand. Additionally, to implement parametrization of the LDM with conditions

to enable text-to-image synthesis, they employed the Bidirectional encoder representations from transformers (BERT) tokenizer [310] and a transformer [311], which infer a latent code that can then be mapped onto the backbone of the LDM, the U-Net CNN [279], by means of a cross-attention mechanism [13].

Stable Diffusion [312] is an updated and more efficient version of the original implementation of the LDM [313]. It features a number of performance-improving differences over the original LDM; in particular, instead of the BERT tokenizer, it employs Contrastive Language-Image Pre-training (CLIP) text encoder [314]. Remarkably, this open-source model can be installed locally and run in inference mode on a machine with a GPU that has at least 10 GB of VRAM [312], which makes it accessible to a broad range of users. Furthermore, the release of Stable Diffusion with a browser-based user interface [315] has made interacting with the model even easier and allows users to employ various techniques for generating images (e.g., image interpolation, image-to-image translation, and inpainting; details follow) and methods for extending the LDM. Therefore, I chose the implementation in [315] for the experiments on augmenting real-world OOC image datasets with synthetic images reported in the rest of the chapter.

5.3 Experiments on the initial OOC image dataset

In the first study, I conducted experiments on the initial dataset of OOC microscopy images. In the following, I describe the real-world OOC image dataset, the process of generating synthetic data to augment it, and the methodology and results of experiments with CNN-based classifiers.

5.3.1 The initial OOC image dataset

To obtain data for the design and validation of a CNN-based classifier, cells were cultivated in a custom-made OOC setup. Several cell lines were used for that: to create a gut-on-a-chip model, Caco-2 (colorectal adenocarcinoma epithelial cells, HTB-37, ATCC, Manassas, VA, USA) and HUVEC (human umbilical vein endothelial cells, CRL-1730, ATCC) cell lines were used to mimic epithelial and endothelial cell layers; for developing a lung cancer-on-a-chip model, A549 (human lung adenocarcinoma alveolar basal epithelial cells, CCL-185, ATTC) and HPMEC (human pulmonary microvascular endothelial cells; 3000, ScienCell, Carlsbad, CA, USA) were used; for lung-on-a-chip modelling, the HSAEC (human small airway epithelial cells, PCS-301-010, ATCC) epithelial cell line was used. The cell seeding density varied and was specific to each cell type to achieve optimal attachment to the membrane of the chip channel and its coverage. Cells were typically cultivated in the OOC setup for $\approx$ 8 days, although some chips were cultivated for up to 22 days. Various flow rates ranging from 0.7 µL/min to 2.77 µL/min were used for cell cultivation.

For imaging the cell tissue, an automated OOC brightfield microscopy setup developed by Cellbox Labs Ltd. (Riga, Latvia) was used. The imaging system consisted of a high-resolution camera IM Compact M (IC10-05q32MU3101, Opto GmbH), and precise control of chip movement was achieved using a precision XYZ motion system. Live imaging and image acquisition were done using OptoViewer software (Opto GmbH), while the movement of the XYZ stage was controlled with Zaber Launcher software (Zaber Technologies).

The acquired images were saved in a private repository and indexed in a table containing a unique identifier for each image, along with data on initial cell seeding density, media flow rate, and cultivation length. The resulting initial dataset of OOC images was rather small,

consisting of 822 images, which were labelled into three classes with imbalanced distribution: the 'good' class with 500 images, the 'acceptable' class with 212 images, and the 'bad' class with 110 images. The ground truth classification of the images was performed by experienced cell biologists based on expected cell morphology and density, while also considering and taking into account the corresponding cell line and the duration of cultivation. The distribution of the images by classes and cell types is shown in Figure 5.3.

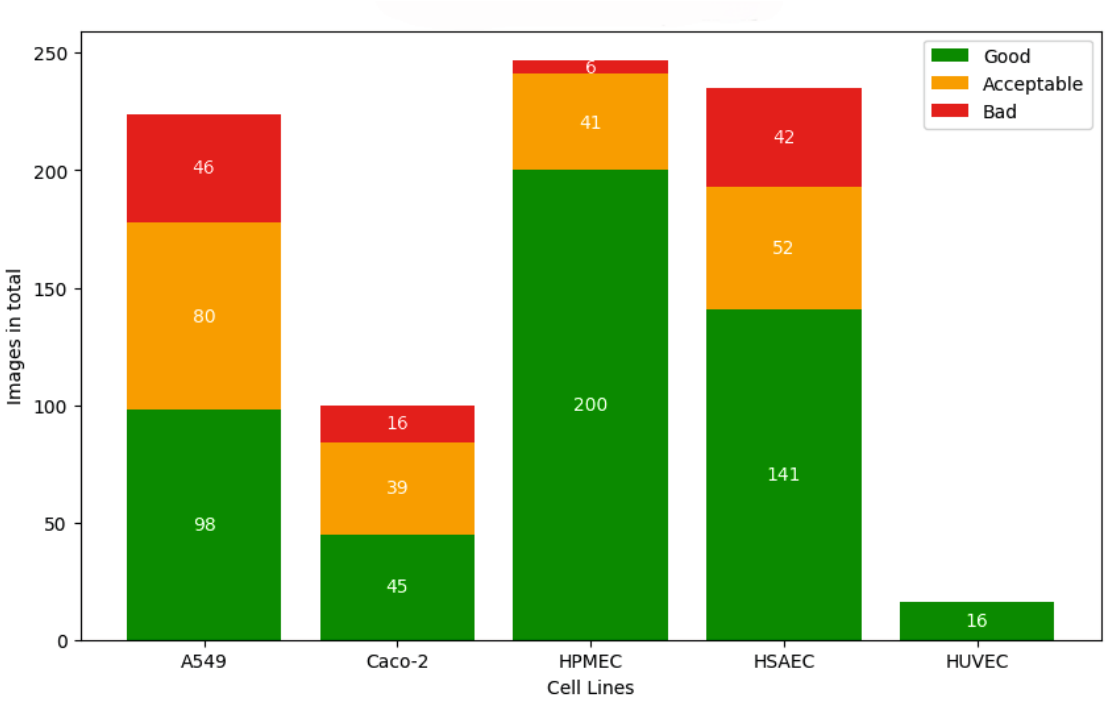


Figure 5.3: Distribution of images by cell lines and classes in the initial OOC dataset.

5.3.2 Synthetic data for augmenting the initial OOC image dataset

To augment the initial OOC image dataset with synthetic data, I fine-tuned Stable Diffusion using the method of low-rank adaptation (LoRA; [293]). A remarkable advantage of LoRA is that even a small number of images – just a few dozen – is typically sufficient for fine-tuning a large generative model. Fine-tuning the Stable Diffusion model with LoRA targets the cross-attention layers by incorporating low-rank matrices (hence the name of the method) that specifically adjust a subset of the weights in these layers. In particular, after fine-tuning, the original weight matrix $W \in \mathbb{R}^{d \times k}$ is transformed into its updated version $W' = W + \Delta W$, where ΔW is the update matrix. The efficiency of fine-tuning with LoRA is due to the low rank of the update matrix, which is achieved by rank decomposition $\Delta W = B \times A$ with $B \in \mathbb{R}^{d \times r}, A \in \mathbb{R}^{r \times k}$, and the rank $r \ll \min(d, k)$ [293]. Since W is frozen during fine-tuning, and only A and B are trained, and since the size of A and B is smaller than that of W, LoRA is very efficient. Thus, while the original Stable Diffusion-v-1-1 model was trained on a cluster consisting of 32 nodes with 8 NVIDIA A100 GPUs per node[2], it is possible to fine-tune it with LoRA using the free version of the Google Colab environment[3]. The fine-tuned model

[2] https://huggingface.co/CompVis/stable-diffusion-v-1-1-original#training; accessed 29 September 2024.

[3] https://colab.research.google.com/; accessed 29 September 2024.

can then be used for generating images in response to the text prompts provided by the user.

The procedure for generating synthetic data involved splitting the dataset of the real-world images into 5 folds (see Section 5.3.3) and using each fold to create three LoRA models (i.e., one for each of the classes, 'good', 'acceptable', and 'bad') using a popular open-source implementation[4] of LoRA for that. All in all, 15 LoRA models were generated with the following parameters: 2 repeats for each image, training for 10 epochs, training batch size equal to two, U-Net learning rate equal to 0.0005, text encoder learning rate equal to 0.0001. These LoRA models were then used together with the Stable Diffusion web UI model [315] to generate synthetic data. The Stable Diffusion web UI model was used with the following parameters: Euler A sampler, 20 sampling steps, classifier-free guidance (CFG) scale of 7, random seed. I generated two datasets of synthetic images with these parameters, the difference between them being that for one dataset, I set the LoRA weight to 1.0, whereas for the other dataset, I set this parameter to 0.8. A higher LoRA weight results in generated images that are more similar to the original ones, whereas a lower weight increases the variability of the generated images, making them less similar to the original ones. The creation of these two distinct datasets made it possible to explore the impact of different levels of LoRA weights on the performance of the image classifier trained on the augmented dataset.

The examples of the generated images are shown in Figure 5.4. Fine-tuning Stable Diffusion with LoRA made it possible to generate images that are somewhat similar to their authentic counterparts. In particular, the granularity of both authentic and synthetic 'good' images in the left column differs from that of the 'bad' images in the right column, which corresponds to the presence of the cells attached to the medium they were grown on in the former case compared to cells that have not attached to the medium in the latter case. However, due to the rather specific nature of the OOC images, it was unclear from visual inspection alone whether the degree of similarity would be sufficient to improve the accuracy of a CNN-based classifier trained on these data.

5.3.3 Experiments with CNNs on the initial dataset

I conducted experiments by training the EfficientNet-B7 [107] CNN model, available in the Keras framework [78], which was pretrained on the ImageNet [19] dataset. The architecture of the model was as follows:

- input layer with a resolution of 600 × 600 pixels;
- data augmentation layer with random rotations (factor=0.25), random translations (height factor=0.1, width factor=0.1), random flips, and random contrast (factor=0.1);
- baseline EfficientNet-B7 model with its weights frozen;
- `GlobalAveragePooling2D` layer;
- `BatchNormalization` layer with dropout (rate=0.2);
- `Dense` layer with 3 neurons with the softmax activation function.

The training of each model consisted of 30 epochs, using the Adam optimiser (learning rate=0.001), with sparse categorical crossentropy loss as the loss function. The dataset was partitioned into five folds, with each fold serving as a holdout fold for 5-fold cross-validation.

[4] https://civitai.com/models/22530; accessed 1 September 2024.

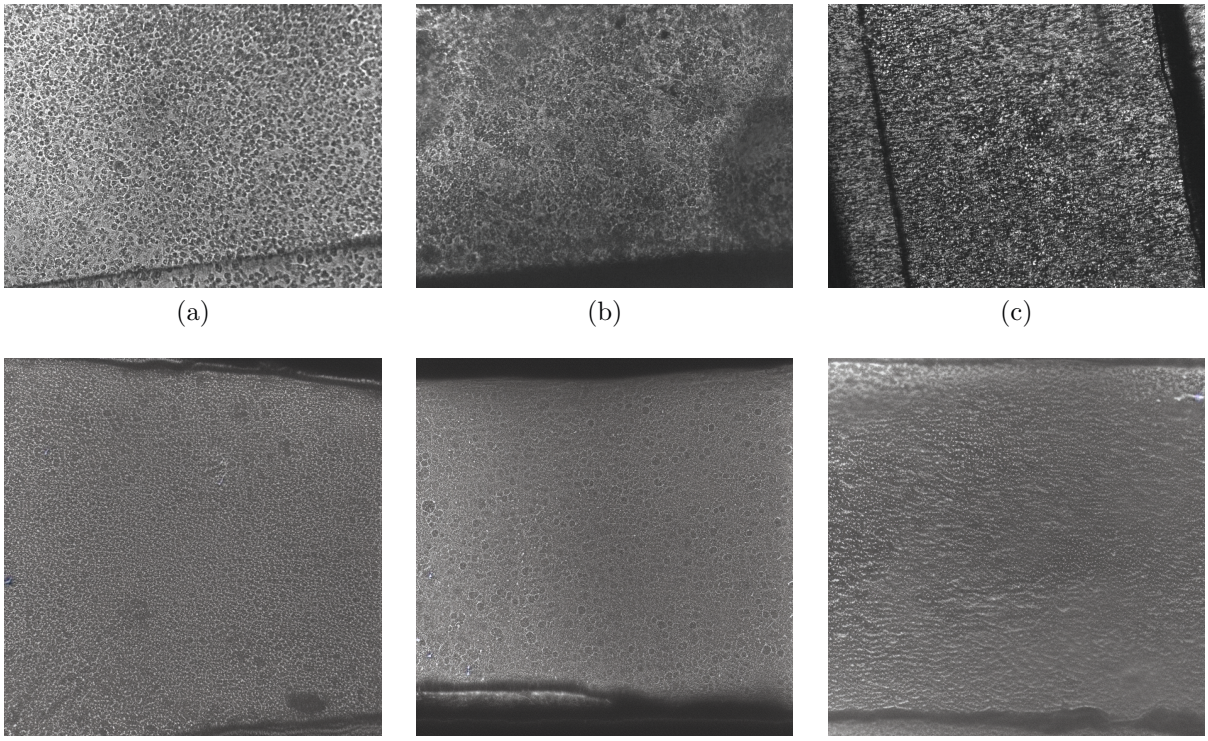


Figure 5.4: Sample OOC images. Top row – real-world images from the initial OOC image dataset: (a) class 'good'; (b) class 'acceptable'; (c) class 'bad'. Bottom row – synthetic images generated with Stable Diffusion fine-tuned with LoRA: (d) class 'good'; (e) class 'acceptable'; (f) class 'bad'.

When training on the augmented dataset, I ensured that synthetic data was generated using LoRA models created on only the training data, excluding the data from the respective holdout fold to prevent information leakage from the training data to the validation data. The training was conducted on a PC with 8 GB RAM, an Intel i5-2500K CPU, an NVIDIA RTX 3090 GPU with 24 GB VRAM, and Ubuntu 18.04.6 LTS OS.

The results of experiments on datasets augmented with synthetic data generated using a LoRA weight of 1.0 are reported in Table 5.1, while those using a LoRA weight of 0.8 are reported in Table 5.2. Furthermore, the results of the experiments with the model trained solely on the real-world alone are also provided in both tables for reference. As can be seen, the baseline CNN model achieved an accuracy of 72.9%, which is better than the accuracy of a putative 'naive' classifier on the given dataset, namely, 60.8%, corresponding to the percentage of the largest class. However, augmenting the real-world image dataset with synthetic images resulted in the deterioration rather than an improvement in classification accuracy, with a trend of decreasing accuracy as the percentage of synthetic data used for augmentation increased.

5.4 Experiments on the final OOC image dataset

In the second study, the team of EDI researchers that I led conducted experiments by training CNNs on the final dataset of OOC microscopy images. Below, I describe the dataset, the

Table 5.1: The results of evaluating EfficientNet-B7 image classifiers trained on the datasets augmented with the synthetic data generated with the LoRA weight of 1.0.

Dataset	**Precision**	**Recall**	**Accuracy**
Synthetic only (100%)	61.7	60.8	61.4
Real-world & 100% synthetic	70.7	68.7	69.9
Real-world & 75% synthetic	70.2	68.0	69.6
Real-world & 50% synthetic	72.0	69.7	71.0
Real-world & 25% synthetic	71.9	69.9	70.7
Real-world & 10% synthetic	72.9	70.1	72.1
Baseline (real-world only)	**73.1**	**71.5**	**72.9**

Table 5.2: The results of evaluating EfficientNet-B7 image classifiers trained on the datasets augmented with the synthetic data generated with the LoRA weight of 0.8.

Dataset	**Precision**	**Recall**	**Accuracy**
Synthetic only (100%)	63.0	59.5	62.1
Real-world & 100% synthetic	70.9	68.5	70.1
Real-world & 75% synthetic	71.8	69.6	70.7
Real-world & 50% synthetic	69.8	67.9	69.3
Real-world & 25% synthetic	70.9	69.0	70.4
Real-world & 10% synthetic	72.9	70.2	71.8
Baseline (real-world only)	**73.1**	**71.5**	**72.9**

methodology of generating synthetic data to augment it, and the methodology and results of the experiments with CNN classifiers.

5.4.1 The final OOC image dataset

The final dataset of OOC images consists of 3072 images, incorporating the initial dataset described in Section 5.3.1, and was acquired using the same methodology as the initial dataset. However, it also features some key differences, namely:

- based on the recommendation of the biology experts involved in the *AImOOC* project, the three-class labelling ('good', 'acceptable', and 'bad') was replaced by a straightforward binary classification: 'good' and 'bad'.
- in addition to the five cell lines in the initial dataset, the final dataset includes an additional line: NHBE (normal human bronchial epithelial cells, CC-2541, Lonza, Basel, Switzerland).

The distribution of the images by classes and cell types is shown in Figure 5.5.

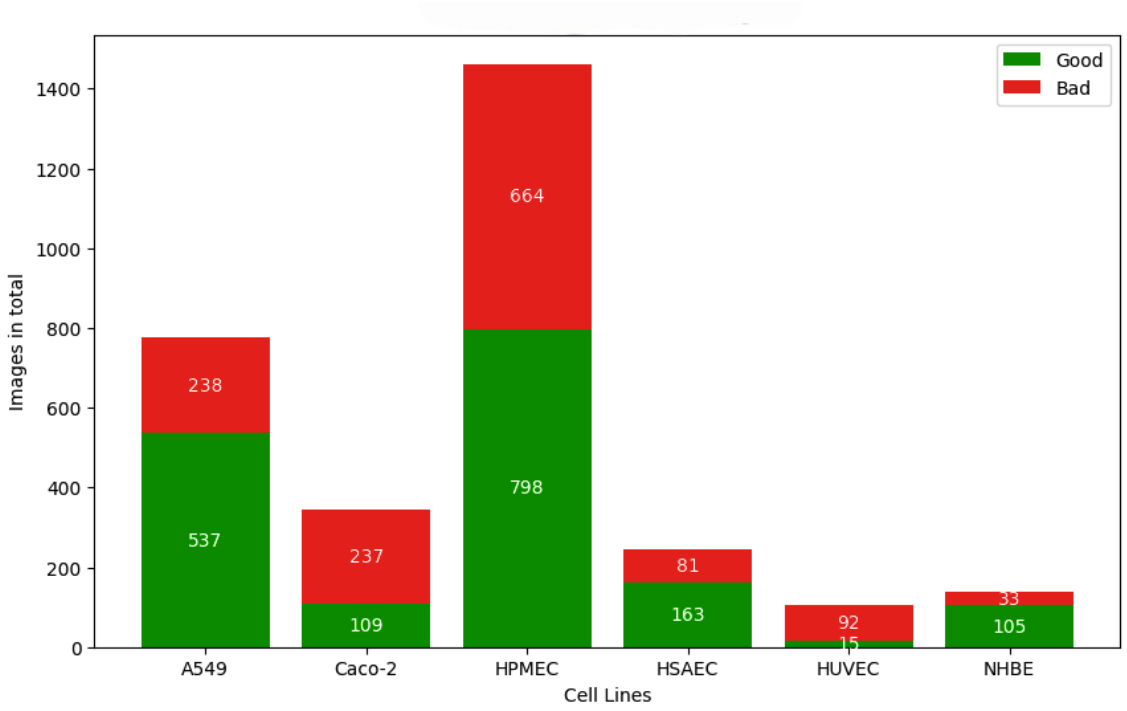
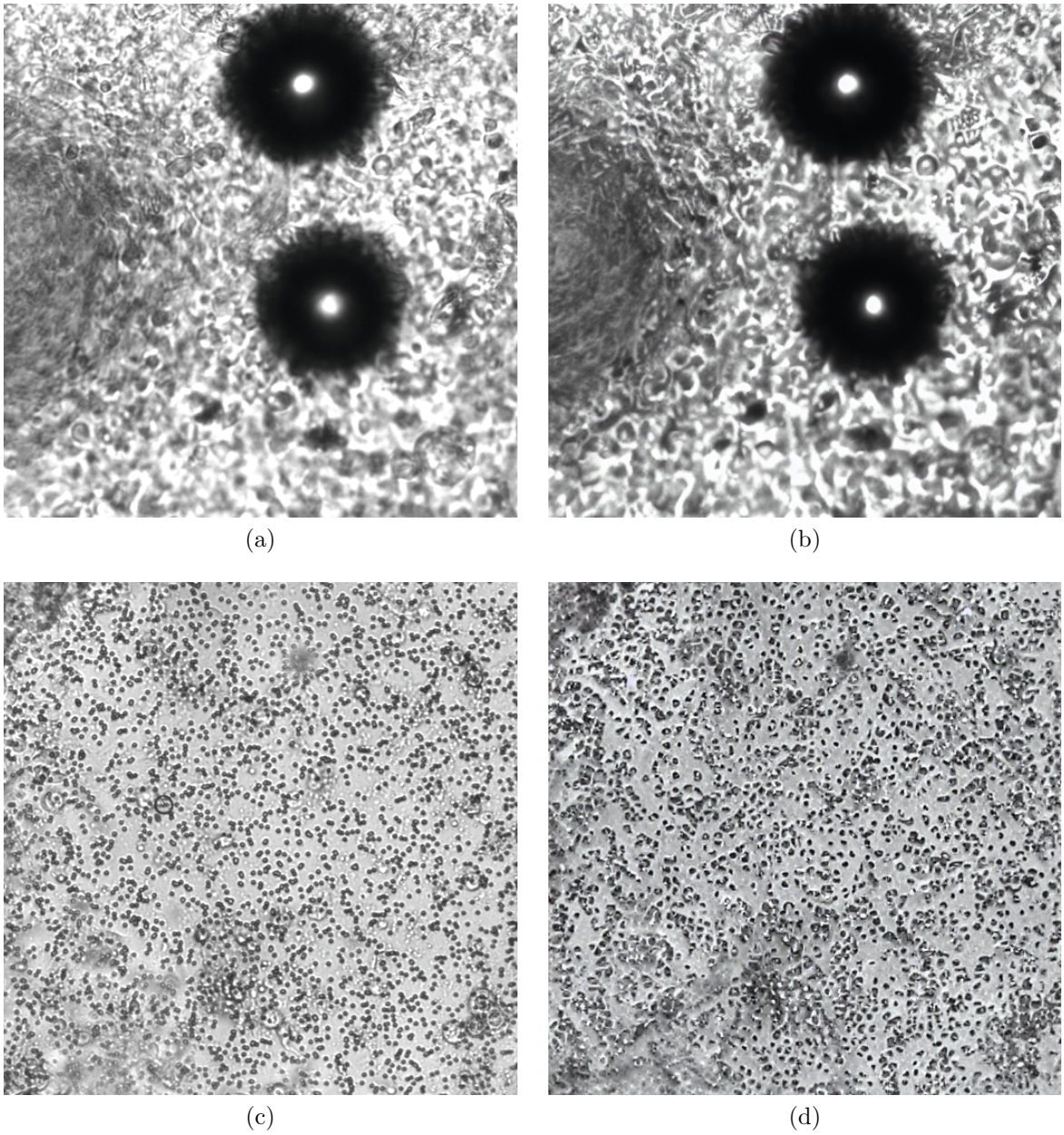


Figure 5.5: Distribution of images by cell lines and classes in the final OOC dataset.

5.4.2 Synthetic data for augmenting the final OOC image dataset

To augment the final OOC image dataset with synthetic data, the research team that I led employed several methods: image-to-image translation, inpainting with masks, image interpolation, and fine-tuning with the LoRA. I detail these methods below.

Image-to-image translation is such a transformation of an input image into an output image that the latter both retains some features of the former and acquires some new features. Using a latent diffusion model, the input image is first encoded into a latent representation that captures its essential features. This representation is then modified to some extent and decoded back into the image space, thus producing an output image. Leveraging the Stable Diffusion model for image-to-image translation, the original images were processed for 30 sampling steps using the Diffusion Probabilistic Model Second-Order Multistep Improved (DPM++ 2M) Karras sampler [316], a CFG of 7, and a denoising strength of 0.3. As a result, a counterpart was generated for each original image. Sample images obtained through this method are shown in Figure 5.6.



(a) (b) (c) (d)

Figure 5.6: Examples of image-to-image translation with Stable Diffusion: (a) a real-world input image, class 'good'; (b) an output image, class 'good'; (c) a real-world input image, class 'bad'; (d) an output image, class 'bad'.

Inpainting with masks involves selectively modifying an input image by applying a mask to designate which parts of the image should be altered. In this study, the masks consisted of black and white stripes, with white stripes designating the areas to be inpainted (i.e., replaced) with synthetic data, and black stripes designating the areas to remain unchanged. As a result, a partially new image was produced. To implement inpaining with masks using the Stable Diffusion model, the input images were processed for 50 sampling steps with the DPM++ 2M Karras sampler, a CFG of 7, and a denoising strength of 0.3. The resulting sample images are shown in Figure 5.7.

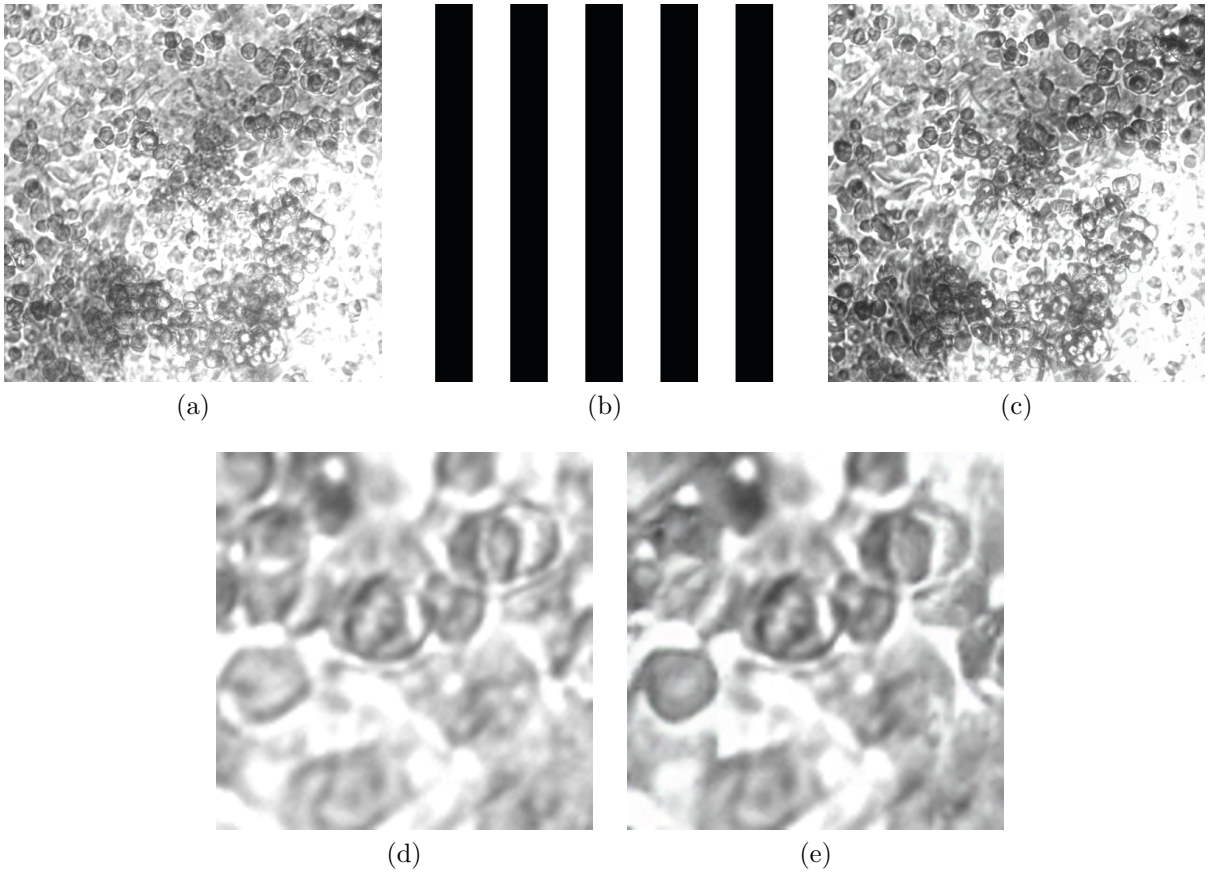
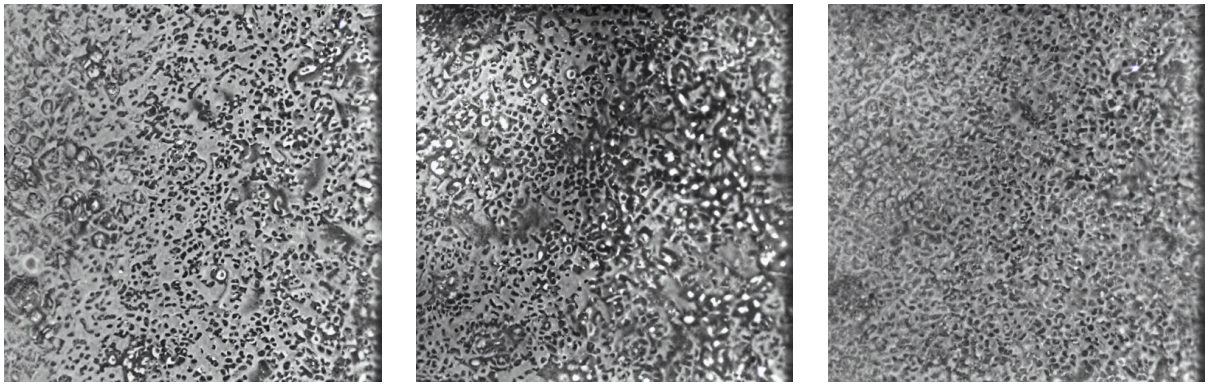


Figure 5.7: Example of inpainting with masks with the Stable Diffusion model. Top row: (a) a real-world input image; (b) a vertical mask; (c) an output image. Bottom row: close-ups of crops (100×100 pixels) from the top left corners of the same (d) input image and (e) output image, illustrating the difference between them due to inpainting.

The goal of **image interpolation** is to create an intermediate image between two or more input images. In the case of Stable Diffusion, interpolation is performed in the latent space; in particular, since the latent space learned by Stable Diffusion is a continuous manifold, it is possible to move along the path connecting two or more images while remaining on the manifold, and each intermediate step on that path will therefore be a valid image as well [317]. For the implementation of image interpolation in this study, Stable Diffusion was extended with a publicly available script[5], and the model processed the input images for 20 steps using the DPM++ 2M Karras sampler with the denoising strength set to 0.25, allowing the output images to preserve salient details without introducing noise artefacts. The CFG scale was fixed at the default value of 7, and no prompt was used. Sample images obtained through interpolation are shown in Figure 5.8.

[5] https://github.com/DiceOwl/StableDiffusionStuff/blob/main/interpolate.py; accessed 10 August 2024.



(a) (b) (c)

Figure 5.8: Example of interpolation with Stable Diffusion: (a) the first input image; (b) the second input image; (c) the output image.

For **image generation after fine-tuning Stable Diffusion with LoRA**, two LoRA models were created: one for the 'bad' class, and one for the 'good' class. The parameters for creating the LoRA models were as follows: two repeats for each image, training for 10 epochs, a training batch size of two, U-Net learning rate of 0.0005, and text encoder learning rate of 0.0001. The LoRA models were then used as fine-tuning extensions of the web UI implementation of Stable Diffusion to generate synthetic images with the following parameters: the DPM++ 2M Karras sampler, 20 sampling steps, a CFG scale of 7, a random seed, and a LoRA weight set to 0.7. Overall, the above parameters were similar to those used in the study on the initial dataset reported in Section 5.3.2, with the main difference being that in the study on the final dataset, the more advanced DPM++ 2M Karras sampler was used instead of the default Euler A sampler. Sample images generated by means of this approach are shown in Figure 5.9.

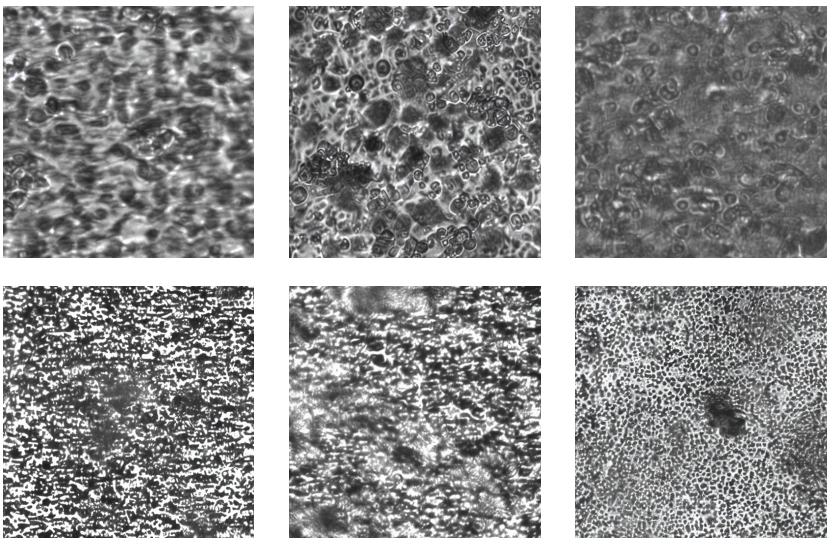


Figure 5.9: Examples of images generated after fine-tuning Stable Diffusion with LoRA: the top row - class 'good', the bottom row - class 'bad'.

5.4.3 Experiments with CNNs on the final dataset

Prior to training CNNs on the dataset, all images were standardised by centre-cropping from their original size of either 2056 × 1542 or 2048 × 1536 pixels to a uniform size of 512 × 512 pixels. This resizing met the input requirements of the pretrained CNN models; furthermore, focusing on the central areas of the images eliminated redundant edge regions, which often contained repetitive patterns and peripheral anomalies such as black lines and blurriness that did not contribute to model training. After resizing, the dataset was split into training, validation, and test subsets with a ratio of 70/10/20, ensuring that all classes, cell lines, and time points after seeding (0-1 days, 2-3 days, 4 days, and 4+ days) were proportionally represented in each split.

After preprocessing the images and splitting the dataset, the team that I lead conducted experiments with two CNN models provided in the Keras framework [78]: EfficientNet-B7 [107] and MobileNetV3Large [106]. Both models were pretrained on the ImageNet [19] dataset. The architecture of the models was as follows:

- input layer with a resolution of 600 × 600 pixels;
- data augmentation layer with random rotations (factor=0.25), random translations (height factor=0.1, width factor=0.1), random flips, and random contrast (factor=0.1);
- baseline model: either EfficientNet-B7, or MobileNetV3Large. The weights of the foundational model were initially frozen;
- `GlobalAveragePooling2D` layer;
- `BatchNormalization` layer with dropout (rate=0.2);
- `Dense` layer with a single neuron with a sigmoid activation function for binary classification.

Training of EfficientNet-B7 was divided into two stages. First, the model was trained for 30 epochs with the weights of the foundational EfficientNet-B7 model frozen to preserve the features learned during pretraining on ImageNet. Afterwards, the model was fine-tuned: the top 20 layers were unfrozen, and the model was trained for additional 30 epochs. During both stages, the Adam optimiser was set to a learning rate of 0.0001 with a decay rate of 0.0001.

Training of MobileNetV3Large began with 30 epochs with the weights of the foundational model frozen; the Adam optimiser was used with a learning rate of 0.0001 and a decay rate of 0.0001. After this initial phase, the fine-tuning involved unfreezing the last 15 layers and reducing the learning rate to 0.00001 to prevent overfitting. The model was then trained for additional 170 epochs with an early stopping callback, halting training if there was no improvement in validation accuracy over 30 consecutive epochs.

Results

To test Hypothesis 1 and obtain **baseline results** for further performance comparison, both models, EfficientNet-B7 and MobileNetV3Large, were first trained on the original dataset of real-world OOC microscopy images without synthetic augmentation. The results of the experiments are provided in Table 5.3. EfficientNet-B7 achieved an accuracy of 83%, while MobileNetV3Large achieved an accuracy of 81%. Both results are better than the accuracy of a putative 'naive' classifier, which is equivalent to the size of the largest class – 56%. The

superior performance of EfficientNet-B7 compared to MobileNetV3Large can be attributed to the larger model's better ability to generalise.

Table 5.3: Classification results for the EfficientNet-B7 and MobileNetV3Large models trained on the real-world images.

Model	**Precision**	**Recall**	**Accuracy**
EfficientNet-B7	**83**	77	**83**
MobileNetV3Large	79	**78**	81

The results of experiments on the dataset augmented with synthetic images generated by **image-to-image translation** are shown in Table 5.4. As can be seen, the highest accuracy for EfficientNet-B7 was achieved by the model trained on the dataset augmented with 100% of synthetic data, whereas the highest accuracy for MobileNetV3Large was with the model trained on the dataset augmented with 25% of synthetic data.

Table 5.4: Classification results for the EfficientNet-B7 and MobileNetV3Large models trained on the dataset augmented with synthetic data generated by image-to-image translation.

Model	**Dataset**	**Precision**	**Recall**	**Accuracy**
EfficientNet-B7	Real-world & 100% synth	82	**84**	**85**
	Real-world & 75% synth	81	77	82
	Real-world & 50% synth	**84**	77	83
	Real-world & 25% synth	**84**	75	83
	Baseline (real-world only)	83	77	83
MobileNetV3Large	Real-world & 100% synth	79	78	81
	Real-world & 75% synth	78	79	81
	Real-world & 50% synth	76	**83**	81
	Real-world & 25% synth	80	79	**82**
	Baseline (real-world only)	79	78	81

The results of experiments on the dataset augmented with synthetic images generated by **inpainting** are shown in Table 5.5. The highest accuracy for EfficientNet-B7 was achieved by the model trained on the dataset augmented with 100% of synthetic data, whereas the highest accuracy for MobileNetV3Large was achieved by the models trained on the datasets augmented with 100% and 75% of synthetic data.

Table 5.5: Classification results for the EfficientNet-B7 and MobileNetV3Large models trained on the dataset augmented with synthetic data generated by inpainting.

Model	Dataset	Precision	Recall	Accuracy
EfficientNet-B7	Real-world & 100% synth	84	79	**84**
	Real-world & 75% synth	82	**80**	83
	Real-world & 50% synth	**88**	68	82
	Real-world & 25% synth	84	68	80
	Baseline (real-world only)	83	77	83
MobileNetV3Large	Real-world & 100% synth	78	**83**	**82**
	Real-world & 75% synth	79	80	**82**
	Real-world & 50% synth	78	81	81
	Real-world & 25% synth	**80**	75	80
	Baseline (real-world only)	79	78	81

The results of the experiments on the dataset augmented with synthetic images generated by **interpolation** are provided in Table 5.6. The highest accuracy for EfficientNet-B7 was achieved by the model trained only on the real-world images, whereas the highest accuracy for MobileNetV3Large was achieved by the model trained on the dataset augmented with 25% of synthetic data.

Table 5.6: Classification results for the EfficientNet-B7 and MobileNetV3Large models trained on the dataset augmented with synthetic data generated by interpolation.

Model	Dataset	Precision	Recall	Accuracy
EfficientNet-B7	Real-world & 100% synth	**87**	66	80
	Real-world & 75% synth	**87**	68	81
	Real-world & 50% synth	85	67	80
	Real-world & 25% synth	83	71	81
	Baseline (real-world only)	83	**77**	**83**
MobileNetV3Large	Real-world & 100% synth	77	76	79
	Real-world & 75% synth	75	**81**	79
	Real-world & 50% synth	**80**	75	80
	Real-world & 25% synth	**80**	79	**82**
	Baseline (real-world only)	79	78	81

Finally, Table 5.7 summarises the results of experiments on the dataset augmented with synthetic images generated after **fine-tuning Stable Diffusion with LoRA**. The results indicate that augmentation with these synthetic images did not improve the accuracy of EfficientNet-B7 or MobileNetV3Large, as none of the models trained on the augmented data performed better than the baseline models.

Table 5.7: Classification results for the EfficientNet-B7 and MobileNetV3Large models trained on the dataset augmented with synthetic data generated after fine-tuning Stable Diffusion with LoRA.

Model	Dataset	Precision	Recall	Accuracy
EfficientNet-B7	Real-world & 100% synth	**86**	70	82
	Real-world & 75% synth	82	75	78
	Real-world & 50% synth	82	**79**	**83**
	Real-world & 25% synth	84	78	81
	Baseline (real-world only)	83	77	**83**
MobileNetV3Large	Real-world & 100% synth	75	76	76
	Real-world & 75% synth	78	78	77
	Real-world & 50% synth	78	82	80
	Real-world & 25% synth	77	**84**	80
	Baseline (real-world only)	**79**	78	**81**

5.5 Concluding remarks

One of the main practical obstacles to the further advancement of the promising OOC technology is the need for round-the-clock monitoring of tissue samples growing on chips, which is currently done by humans. The goal of the research reported in this chapter was to develop a CNN-based image classifier for assessing the quality of such samples, which would enable the automation of the monitoring in the future. To achieve this goal, I conducted two experimental studies: one on the initial OOC microscopy image dataset consisting of 822 images that represented five cell lines and were labelled as 'good', 'acceptable', or 'bad', and another on the final dataset consisting of 3 072 images that represented six cell lines and were labelled as 'good' or 'bad'. In both studies, the same two hypotheses were tested:

- **Hypothesis 1**: A CNN-based classifier achieves better accuracy on the real-world microscopy OOC image dataset than a putative 'naive' classifier.
- **Hypothesis 2**: The classification accuracy on the real-world microscopy OOC image dataset improves when a CNN-based classifier is trained on the dataset augmented with synthetic data generated with the Stable Diffusion model rather than solely on the real-world image dataset.

The first hypothesis was confirmed in both studies, in particular:

- In the first study, the EfficientNet-B7 model achieved an accuracy of 72.9%, whereas the accuracy of the putative 'naive' classifier on the initial dataset was estimated at 60.8%;
- in the second study, the EfficientNet-B7 model achieved an accuracy of 83%, and the MobileNetV3Large model achieved an accuracy of 81%, whereas the accuracy of the putative 'naive' classifier on the final dataset was estimated at 56%.

These results demonstrate the efficiency of the use of the selected CNN models for OOC image classification. The difference in results achieved by EfficientNet-B7 in the first study (on the initial dataset) compared to the second study (on the final dataset) is likely due to

two major factors: the increase in the size of the training dataset, and, perhaps even more importantly, the shift from the more challenging three-class classification task to the simpler binary classification task. Another noteworthy observation is that EfficientNet-B7 outperformed MobileNetV3Large on the final dataset, likely due to the higher learning capacity of the larger network (66.7 vs 5.4 million parameters[6]). However, the gap between the accuracy of the models is just 2%, which is substantially smaller than, e.g., the gap between the Top-1 accuracy of these models on the benchmark dataset ImageNet: 84.3% for EfficientNet-B7 vs 75.6% for MobileNetV3Large[7]. Therefore, for practical purposes, it may be more expedient to use MobileNetV3Large, as its smaller size allows it to run in inference mode on a mobile or edge device, which may be more suitable for a real-life OOC setup in lab conditions than using a PC.

The second hypothesis was not confirmed in the first study, as augmenting the real-world dataset with synthetic images led to a deterioration rather than an improvement in classifier performance for both synthetic datasets, that is, for datasets with LoRA weight = 1.0 and LoRA weight = 0.8. The negative impact of the synthetic data is particularly evident when the model is trained solely on synthetic data, as these models exhibit the worst accuracy. Remarkably, synthetic data generated with Stable Diffusion look similar to the original data (see Figure 5.4) and the model trained solely on it converges very well during training. However, due to the likely data distribution discrepancy between the real-world and synthetic data, augmenting the former with the latter did not yield any benefits. In the second study, the second hypothesis was confirmed for augmentation with synthetic data generated by means of image-to-image translation and inpainting, but not for augmentation with synthetic data generated using LoRA. In the case of synthetic data generated with interpolation, a minor improvement on the augmented dataset was achieved with the MobileNetV3Large model, but not with the EfficientNet-B7 model. These results indicate that the effectiveness of augmenting real-world biomedical image datasets with synthetic data may vary substantially depending on the specific method of data generation.

Finally, I would like to observe that similar to the study reported in Chapter 3, the improvement observed in the performance of models that had benefited from training on augmented datasets did not directly correlate with the amount of synthetic data used for augmentation. For instance, in case of the image-to-image approach to the synthetic data generation, the best-performing EfficientNet-B7 models were trained on the datasets augmented with 50% and 25% of the available synthetic data, while the best-performing MobileNetV3Large model was trained on the dataset augmented with 25% of the available synthetic data. For the inpainting approach, the best-performing EfficientNet-B7 model was trained on the dataset augmented with all available (100%) synthetic data, whereas the MobileNetV3Large models that demonstrated the best accuracy were trained on the datasets augmented with 100% and 75% of the available synthetic data. Therefore, consistent with the conclusions of Chapter 3, I note that the relationship between the amount of the data used for augmentation and the performance of the image understanding models trained on the augmented datasets is far from straightforward.

[6]Cf. https://keras.io/api/applications/ and https://keras.io/api/applications/mobilenet/#mobilenetv2-function; both accessed 2 September 2024.

[7]https://keras.io/api/applications/; accessed 2 September 2024.

Conclusion

This PhD thesis was concerned with the application of computer vision methods to three major image understanding tasks: image classification, object detection, and semantic segmentation. Specifically, I used computer vision methods for solving the following real-world problems:

- classification of hand-washing movements in a clinical setting to automate the monitoring of compliance of medical personnel with hand hygiene standards (Chapter 2);
- semantic segmentation of street views to enhance perception modules of self-driving cars (Chapter 3);
- detection of plastic bottles that can be picked up by a robotic arm to automate the production line in a manufacturing facility (Chapter 4);
- classification of microscopy images to automate the monitoring of the growth of organs-on-a-chip (Chapter 5).

As I outlined it in my overview of highlights of computer vision in Chapter 1, both methods and software tools in this field have developed tremendously since its emergence in the 1960s, and as a result of that growth, computers nowadays even outperform humans on some visual recognition tasks. However, since computers still do not understand visual environment in such a natural way as we, humans, do, each particular image understanding task still has to be solved on a case-by-case basis by choosing an appropriate approach, whether it be classical computer vision methods or the deep learning paradigm, deciding upon which specific model to use, finding a dataset or collecting and labelling it on one's own, preprocessing the data, and training and evaluating the model. When I followed these steps in the studies that laid the foundation of this thesis, my work was substantially facilitated by prior theoretical and practical advancements in the fields of computer vision and machine learning. In particular, instead of having to hand-engineer features for my models, as it was customary in the era of the predominance of classical methods, I adopted CNNs as a single approach for solving various image understanding tasks, from image classification to object detection to semantic segmentation. While the specifics of the model architectures varied, the fundamental advantage of CNNs, their ability to automatically learn hierarchical feature representations directly from data, motivated me to come up with **the central premise** that convolutional neural networks can successfully solve the image understanding tasks considered in this thesis. Rather than having to implement and train CNNs from zero, I had at my disposal publicly available CNN models implemented in deep learning frameworks: TensorFlow, Keras, and PyTorch. Moreover, the models were available not as bare architectures but with weights pretrained on large general-purpose datasets such as ImageNet and MS COCO, enabling transfer learning and fine-tuning the models on much smaller datasets that I was working with. For some of the tasks, I also was able to utilise suitable publicly available datasets: the

Cityscapes dataset was essential for my work on semantic segmentation of street views, while the Kaggle dataset allowed me to establish a baseline for the assessment of performance of CNN-based hand-washing movement classifiers. Furthermore, I leveraged open-source assets for generating synthetic data that I could then use for augmenting real-world datasets: thus, I generated street views with a high degree of photorealism, varying weather conditions, and a high number of salient objects such as cars and pedestrians with the CARLA simulator, and synthetic images of OOC cell tissue with complex textures using the Stable Diffusion generative model.

However, despite the availability of assets for implementing and training CNNs, I still had to overcome substantial challenges. One of these was the notoriously fluid[8] state of many state-of-the-art deep learning libraries: due to the very rapid pace and competitiveness of deep learning research, many researchers prefer to publish new studies and release new libraries rather than ensure the stability of the already released code and properly document it. As a result, when working with cutting-edge libraries rather than with the more stable mainstream deep learning frameworks per se, I often had to invest a lot of time in debugging the code and looking for answers on various forums rather than in documentation, which often was insufficiently detailed. Furthermore, since deep learning is an experimental field of science, the efficiency of debugging and solutions suggested in non-published sources could often be established only after conducting a time-consuming experiment with the model, and many times, the outcome was that the debugging or a search for solution had to be continued.

Another, more fundamental challenge was that in general, decisions regarding the architecture and hyperparameters of deep learning models that have to be made when applying CNNs to real-world problems remain largely heuristic. While many such choices are motivated – for instance, since it is known that a larger model such as EfficientNet-B7 is likely to generalise better than a smaller one such as EfficientNet-B0, one may choose the former over the latter when classifying a complex dataset – many other choices do reside in a gray area. Was unfreezing the top 20 layers of EfficientNet-B7 and the top 15 layers of MobileNetV3Large during the final training stage on the final OOC dataset (see Section 5.4.3) the optimal approach, or would it be better to unfreeze fewer layers, or, just the opposite, unfreeze the whole models!? What about having 128 neurons in the Dense layer of the baseline models trained on the METC and PSKUS datasets: was that number just right, or too small, or too high!? Could it be the case that arranging the same number of neurons in several layers rather than a single one would lead to a better accuracy? This thesis does not provide answers to these questions or a myriad of other possible similar questions, since, as it is usually the case with the applications of deep learning to complex real-world datasets, a comprehensive grid search for the best model parameters was not feasible due to the limited computational resources, long training times of models, and high number of hyperparameters and features of model architectures.

Finally, yet another major challenge was the issue of data acquisition and labelling. In particular, since there were not any publicly available datasets with labelled recordings of hand washing, it was necessary to collect and label such a dataset for developing a hand-washing movement classifier, and since an OOC image classifier that I needed to develop was supposed to process rather specific images generated with particular protocols on a particular OOC setup used in the AImOOC project, it was necessary to collect and process a dataset with such images as well. While I did not personally conduct data acquisition and labelling for either dataset, I had to translate the requirements of the deep learning pipelines to the

[8] I am afraid that some more orthodox Computer Science researchers would actually use the word 'sloppy' here.

experts – epidemiologists and biomedical researchers, respectively – and make sure that these requirements were eventually met. Some of the main challenges of the data acquisition for the work reported in this thesis were inter-annotator agreement (actually, often disagreement) when labelling hand-washing videos and a rather small size of the initial OOC cell image dataset. While the latter issue was largely resolved - first by employing cross-validation of models on several folds, and then by collecting and utilising a larger dataset - the labelling of the PSKUS dataset, the largest real-world dataset of hand-washing videos collected in the studies reported in Chapter 2, can still be considered rather noisy. In particular, since there was a $\approx 91\%$ agreement between annotators as to which frames should be labelled as `is_washing`, and a further $\approx 90\%$ agreement on the hand-washing movement code, it is obvious that training and evaluating classifiers on such noisy data is very likely to affect their performance negatively. In the retrospective, it appears to me that the data labelling process should have been monitored more closely so that it would be possible to identify the problem of a poor inter-annotator agreement earlier and mitigate it. Another possible solution was be to use additional lightning and film videos with cameras with higher resolution.

Despite the challenges that I outlined above, I consider the results of the studies that laid the foundation of this thesis to be rather successful, in particular:

- In the research on hand-washing movement classification, excellent results - an F_1 score of 96% - were achieved on the Kaggle dataset and satisfactory results - an F_1 score of 64% - were achieved on the METC dataset. While any of the models failed to achieve good performance on the most complex and noisy dataset, the PSKUS dataset, their failure highlighted the important yet often neglected problem of the translation of methodology from lab conditions to the complex real-world conditions. Furthermore, despite noise in the labelling of the PSKUS dataset, both it and METC, another dataset collected and published in open access in the course of research reported in Chapter 2, are valuable assets for further studies on hand-washing movement classification.

- In the research on improving the accuracy of semantic segmentation of street views, the mIoU of MobileNetV2 trained on the CCM-50 dataset (i.e., the dataset augmented with 50% of available synthetic data) was higher by 12% than that of the same model trained only on the real-world Cityscapes images, and the mIoU of Xception-65 trained on the CCM-25 dataset (i.e., the dataset augmented with 25% of available synthetic data) was higher by 7% than that that of the same model trained only on the real-world Cityscapes images. While it was not possible to predict what amount of synthetic data for augmentation would yield the best result, the fact that all models trained on the augmented datasets outperformed their counterparts trained solely on the real-world data demonstrated the usefulness of synthetic data for the enhancement of perceptual modules of self-driving cars. Furthermore, it is worth noting that the synthetic data was generated in a relatively simple and straightforward manner by running simulations in the open-source CARLA simulator, which demonstrated that very useful synthetic data do not have to be challenging to acquire.

- In the research on detecting graspable bottles, the state-of-the-art object detector YOLOv5 demonstrated that it can efficiently – the best model achieved a mAP of 77.7% with a threshold of 0.5 – detect bottles with the visibility above a certain threshold in a pile of similarly looking objects. This study also contributed to the topics of the use of synthetic data for training models and enhancing the photorealism of synthetic data, since the best-performing object detectors were trained on synthetic data enhanced with the CycleGAN model.

- In the research on OOC microscopy image classification, the best EfficientNet-B7 model achieved an accuracy of 85%, while the best MobileNetV3Large model achieved an accuracy of 82%. Since these best-performing models were trained on the dataset augmented with 100% and 25% of the available synthetic data generated by image-to-image translation with Stable Diffusion, these studies also contributed to the emerging research of generating biomedical imagery for training DNNs with large generative models.

As follows from the above recap, the results demonstrated that **the goal of the thesis – to provide efficient solutions for applied image understanding tasks – was achieved for all tasks except the classification experiments on the PSKUS dataset**. **The scientific novelty** of the studies reported in the thesis is, inter alia, due to the fact that in those studies, CNN models were successfully applied to the novel datasets: either real-world, or synthetic, or both. The findings of the studies considered jointly also make me to propose the following **thesis statements** for the defence:

- **Thesis statement one: In applications of CNNs to real-world image understanding tasks, data availability and quality present greater challenges than model selection and customisation.**
  This thesis statement is grounded in the studies that I presented in Chapters 2, 3, 4, and 5. Additionally, it is supported by observations in the literature (see Section 1.4.1), which highlight that while the AI community has heavily focused on model development and improvement, making DNN models more accessible, the availability and quality of data remain the primary enabling factors for effectively utilising DNNs.

- **Thesis statement two: CNNs that perform well when trained and evaluated on datasets acquired in laboratory conditions may struggle to achieve similar success when trained and evaluated on more complex real-world data.**
  This thesis statement is grounded in the research on hand-washing movement classification that I reported in Chapter 2. A major finding of the cross-dataset study [176] reported there was that CNNs that achieved high classification accuracy on the Kaggle dataset, which was acquired in simplified lab conditions, demonstrated substantially lower accuracy on the more complex METC dataset and failed to generalise on the even more challenging real-world PSKUS dataset[9]. There are compelling reasons to believe that the scope of this observation extends beyond this particular study, especially given the current focus in a substantial part of ML research on incremental improvements, often at the expense of generalisability (see Section 1.4.1).

- **Thesis statement three: While state-of-the-art CNN-based image classifiers and object detectors with a larger number of parameters typically demonstrate higher accuracy on benchmark datasets than their counterparts with a smaller number of parameters, this accuracy gap narrows or even vanishes when these models are trained and evaluated on smaller, more complex real-world datasets.**
  This thesis statement is grounded in the results of experiments in Chapter 4, in which I used the YOLOv5 object detector to detect graspable bottles, and in Chapter 5,

[9]Note that while the cross-dataset study I refer to here included experiments with training a model on one dataset and evaluating it on another dataset – e.g., a model was trained on the Kaggle dataset and evaluated on the METC dataset – my observation here is primarily about training and evaluating a model on **the same** dataset.

in which I used the EfficientNet-B7 and MobileNetV3Large models for classification on the final dataset of OOC microscopy images. Experiments in Chapter 4 were conducted with different sizes of the YOLOv5 architecture: Small (7.2 million parameters), Medium (21.2 million parameters), and Extra Large (86.7 million parameters). When the authors of these models benchmarked them on the MS COCO dataset, the accuracy of the object detection matched their size: the Small model achieved an mAP of 56.8%, the Medium - of 64.1%, and the Extra Large - of 68.9%[10]. However, when trained on the best dataset for bin-picking task, Augmented noise 1024 × 768, the Small model achieved an mAP of 77.7%, the Medium model - 75.3%, and the Extra Large model - 76.1%. These results indicate that in these experiments, the advantage of larger models vanished. As for experiments in Chapter 5, EfficientNet-B7 is substantially larger than MobileNetV3Large – 66.7 vs 5.4 million parameters, which is also reflected in the performance on the benchmarks dataset ImageNet: 84.3% for EfficientNet-B7 vs 75.6% for MobileNetV3Large[11]. However, that gap decreased to just 2% in favour of EfficientNet-B7 when these models were trained and evaluated on the OOC image dataset.

- **Thesis statement four: While augmenting real-world datasets with photorealistic synthetic images is an efficient way to improve the accuracy of CNNs trained on such data, increasing the amount of synthetic data does not directly correlate with improved accuracy on image understanding tasks.**
This thesis statement is grounded in the research in Chapters 3 and 5. In Chapter 3, I used various amounts of synthetic data – 25%, 50% and 100% of the available synthetic images – for augmenting Cityscapes, the real-world dataset of street views, to improve the accuracy of semantic segmentation. While the accuracy of the MobileNetV2 and Xception-65 models trained on augmented datasets improved compared to models trained only on the real-world images, the accuracy improvement did not correlate with the amount of synthetic data used for augmentation: the best-performing MobileNetV2 model was the one trained on the dataset augmented with 50% of synthetic data, whereas the best-performing Xception-65 model was the one trained on the dataset augmented with 25% of synthetic data. In Chapter 5, the most efficient approaches to generating synthetic data for augmenting real-world OOC image datasets for training the EfficientNet-B7 and MobileNetV3Large classifiers were image-to-image translation and inpainting. For image-to-image translation, the best-performing EfficientNet-B7 model was trained on the dataset augmented with 100% of synthetic data, while the best-performing MobileNetV3Large model was trained on the dataset augmented with 25% of synthetic data; for inpainting, the respective datasets were the ones augmented with 100% of synthetic data and with 75 and 100% of synthetic data. These results suggest that there is no straightforward correlation between the amount of the data used for augmentation and the accuracy of a model trained on the augmented dataset.

While the results of the studies were published in six scholarly articles indexed in Elsevier Scopus and/or Web of Science database (with at least two more publications forthcoming) and two scholarly publications not indexed in these databases as well as presented at four conferences, there is still ample room for further work. The primary objective for the near future is to apply the insights and knowledge gained during the work on this thesis to solve

[10] https://github.com/ultralytics/yolov5. Accessed 15 September 2024.

[11] Cf. https://keras.io/api/applications/ and https://keras.io/api/applications/mobilenet/#mobilenetv2-function. Both accessed 15 September 2024.

image understanding tasks in other ongoing research projects. Thus, the object detection approaches used in the study in Chapter 4 and the methods for generating synthetic data with large generative models to augment real-world image dataset developed and validated in the research reported in Chapter 5 are currently being applied and further improved in the project *Holographic microscopy- and artificial intelligence-based digital pathology for the next generation of cytology in veterinary medicine – VetCyto* to enhance the precision of microscopy-based veterinary diagnostics. The major goal for the more distant yet, I believe, still foreseeable future is to contribute to the development of models truly capable of image *understanding*.

Bibliography

[1] N. J. Nilsson, *The Quest for Artificial Intelligence*. Cambridge University Press, 2009.

[2] W. S. McCulloch and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," *The bulletin of mathematical biophysics*, vol. 5, pp. 115–133, 1943.

[3] J. Achiam *et al.*, "GPT-4 technical report," *arXiv:2303.08774*, 2023.

[4] D. Hassabis, "Artificial Intelligence: Chess match of the century," *Nature*, vol. 544, no. 7651, pp. 413–414, 2017.

[5] R. Szeliski, *Computer Vision: Algorithms and Applications*. Springer Nature, 2022.

[6] M. A. Boden, *Mind as Machine: A History of Cognitive Science*. Oxford University Press, 2008.

[7] Y. Guo, Y. Liu, T. Georgiou, and M. S. Lew, "A review of semantic segmentation using deep neural networks," *International journal of multimedia information retrieval*, vol. 7, pp. 87–93, 2018.

[8] Y. Amit, P. Felzenszwalb, and R. Girshick, "Object detection," in *Computer Vision: A Reference Guide*, pp. 1–9, Springer, 2020.

[9] S. Nikolenko, *Synthetic Data for Deep Learning*. Springer, 2021.

[10] W. Zhao, J. P. Queralta, and T. Westerlund, "Sim-to-real transfer in deep reinforcement learning for robotics: A survey," in *2020 IEEE symposium series on computational intelligence (SSCI)*, pp. 737–744, IEEE, 2020.

[11] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun, "CARLA: An open urban driving simulator," in *Conference on robot learning*, pp. 1–16, PMLR, 2017.

[12] I. Goodfellow *et al.*, "Generative adversarial nets," *Advances in neural information processing systems*, vol. 27, 2014.

[13] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, and B. Ommer, "High-resolution image synthesis with latent diffusion models," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10684–10695, 2022.

[14] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[15] N. O'Mahony *et al.*, "Deep learning vs. traditional computer vision," in *Advances in Computer Vision: Proceedings of the 2019 Computer Vision Conference (CVC), Volume 1*, pp. 128–144, Springer, 2020.

[16] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, pp. 273–297, 1995.

[17] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE transactions on information theory*, vol. 13, no. 1, pp. 21–27, 1967.

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

[19] J. Deng *et al.*, "ImageNet: A large-scale hierarchical image database," in *2009 IEEE conference on computer vision and pattern recognition*, pp. 248–255, IEEE, 2009.

[20] B. Jähne, H. Haussecker, and P. Geissler, *Handbook of Computer Vision and Applications*. San Diego: Academic Press, 1999.

[21] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2003.

[22] D. A. Forsyth and J. Ponce, *Computer Vision: A Modern Approach*. Prentice Hall Professional Technical Reference, 2002.

[23] X. Feng, Y. Jiang, X. Yang, M. Du, and X. Li, "Computer vision algorithms and hardware implementations: A survey," *Integration*, vol. 69, pp. 309–320, 2019.

[24] S. A. Papert, "The summer vision project," tech. rep., MIT, 1966.

[25] B. Jähne, ed., *Computer Vision and Applications: A Guide for Students and Practitioners*. Elsevier, 2000.

[26] The Encyclopedia Britannica Editors, "Computer vision." [Online], 2024. Available: https://www.britannica.com/technology/computer-vision. Accessed: 20 June 2024.

[27] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *IJCAI'81: 7th international joint conference on Artificial intelligence*, vol. 2, pp. 674–679, 1981.

[28] J.-Y. Bouguet, "Pyramidal implementation of the affine Lucas Kanade feature tracker description of the algorithm," tech. rep., Intel corporation, 2001.

[29] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE transactions on systems, man, and cybernetics*, vol. 9, no. 1, pp. 62–66, 1979.

[30] S. Beucher and F. Meyer, "The morphological approach to segmentation: The watershed transformation," in *Mathematical morphology in image processing*, pp. 433–481, CRC Press, 2018.

[31] J. Canny, "A computational approach to edge detection," *IEEE Transactions on pattern analysis and machine intelligence*, no. 6, pp. 679–698, 1986.

[32] I. Pitas, *Digital Image processing Algorithms and Applications*. John Wiley & Sons, 2000.

[33] S. Aditya, Y. Yang, and C. Baral, "Integrating knowledge and reasoning in image understanding," *arXiv:1906.09954*, 2019.

[34] D. Sarvamangala and R. V. Kulkarni, "Convolutional neural networks in medical image understanding: A survey," *Evolutionary intelligence*, vol. 15, no. 1, pp. 1–22, 2022.

[35] M. A. Al-Malla, A. Jafar, and N. Ghneim, "Image captioning model using attention and object features to mimic human image understanding," *Journal of Big Data*, vol. 9, no. 1, pp. 1–16, 2022.

[36] Y.-J. Zhang and Y.-J. Zhang, "Image engineering," in *Handbook of Image Engineering*, pp. 55–83, Springer, 2021.

[37] D. Ortego, J. C. SanMiguel, and J. M. Martinez, "Long-term stationary object detection based on spatio-temporal change detection," *IEEE Signal Processing Letters*, vol. 22, no. 12, pp. 2368–2372, 2015.

[38] S. Fazekas, B. K. Budai, R. Stollmayer, P. N. Kaposi, and V. Bérczi, "Artificial intelligence and neural networks in radiology–basics that all radiology residents should know," *Imaging*, vol. 14, no. 2, pp. 73–81, 2022.

[39] S. Z. Li, *Markov Random Field Modeling in Computer Vision*. Springer Science & Business Media, 2012.

[40] Y. LeCun, C. Cortes, and C. Burges, "The MNIST database of handwritten digits." [Online]. Available: http://yann.lecun.com/exdb/mnist/. Accessed: 20 June 2024.

[41] W. Rawat and Z. Wang, "Deep convolutional neural networks for image classification: A comprehensive review," *Neural computation*, vol. 29, no. 9, pp. 2352–2449, 2017.

[42] D. C. Cireşan, U. Meier, L. M. Gambardella, and J. Schmidhuber, "Deep, big, simple neural nets for handwritten digit recognition," *Neural computation*, vol. 22, no. 12, pp. 3207–3220, 2010.

[43] L. Liu *et al.*, "Deep learning for generic object detection: A survey," *International journal of computer vision*, vol. 128, pp. 261–318, 2020.

[44] Z. Zou, K. Chen, Z. Shi, Y. Guo, and J. Ye, "Object detection in 20 years: A survey," *Proceedings of the IEEE*, 2023.

[45] Y. Zheng, O. Andrienko, Y. Zhao, M. Park, and T. Pham, "DPPD: Deformable polar polygon object detection," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 78–87, 2023.

[46] R. Padilla, S. L. Netto, and E. A. Da Silva, "A survey on performance metrics for object-detection algorithms," in *2020 International conference on systems, signals and image processing (IWSSIP)*, pp. 237–242, IEEE, 2020.

[47] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

[48] R. Padilla, W. L. Passos, T. L. Dias, S. L. Netto, and E. A. Da Silva, "A comparative analysis of object detection metrics with a companion open-source toolkit," *Electronics*, vol. 10, no. 3, p. 279, 2021.

[49] Baeldung, "Intersection over union for object detection." [Online], 2024. Available: https://www.baeldung.com/cs/object-detection-intersection-vs-union. Accessed 23 June 2024.

[50] S. Minaee, Y. Boykov, F. Porikli, A. Plaza, N. Kehtarnavaz, and D. Terzopoulos, "Image segmentation using deep learning: A survey," *IEEE transactions on pattern analysis and machine intelligence*, vol. 44, no. 7, pp. 3523–3542, 2021.

[51] S. Hao, Y. Zhou, and Y. Guo, "A brief survey on semantic segmentation with deep learning," *Neurocomputing*, vol. 406, pp. 302–321, 2020.

[52] J. Wang, Y. Ma, L. Zhang, R. X. Gao, and D. Wu, "Deep learning for smart manufacturing: Methods and applications," *Journal of manufacturing systems*, vol. 48, pp. 144–156, 2018.

[53] G. Csurka, C. Dance, L. Fan, J. Willamowski, and C. Bray, "Visual categorization with bags of keypoints," in *Workshop on statistical learning in computer vision, ECCV*, vol. 1, Prague, 2004.

[54] G. Csurka, C. R. Dance, F. Perronnin, and J. Willamowski, "Generic visual categorization using weak geometry," in *Toward Category-Level Object Recognition*, pp. 207–224, Springer, 2006.

[55] L. Bai, Y. Li, M. Cen, and F. Hu, "3D instance segmentation and object detection framework based on the fusion of LIDAR remote sensing and optical image sensing," *Remote Sensing*, vol. 13, no. 16, p. 3288, 2021.

[56] D. A. Forsyth and J. Ponce, *Computer Vision: A Modern Approach*. Boston: Pearson, 2012.

[57] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes (VOC) challenge," *International journal of computer vision*, vol. 88, no. 2, pp. 303–338, 2010.

[58] C. H. Lampert, M. B. Blaschko, and T. Hofmann, "Beyond sliding windows: Object localization by efficient subwindow search," in *2008 IEEE conference on computer vision and pattern recognition*, pp. 1–8, IEEE, 2008.

[59] K. E. Van de Sande, J. R. Uijlings, T. Gevers, and A. W. Smeulders, "Segmentation as selective search for object recognition," in *2011 International conference on computer vision*, pp. 1879–1886, IEEE, 2011.

[60] P. Viola and M. J. Jones, "Robust real-time face detection," *International journal of computer vision*, vol. 57, pp. 137–154, 2004.

[61] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*, vol. 1, pp. 886–893, IEEE, 2005.

[62] D. Mumford and J. Shah, "Boundary detection by minimizing functionals," in *IEEE Conference on computer vision and pattern recognition*, vol. 17, pp. 137–154, San Francisco, 1985.

[63] J. du Buf, M. Kardan, and M. Spann, "Texture feature performance of image segmentation," *Pattern Recognition*, vol. 23, pp. 291–309, 1990.

[64] A. Treméau and N. Borel, "A region growing and merging algorithm to color segmentation," *Pattern Recognition*, vol. 30, pp. 1191–1203, Jul 1997.

[65] E. Borenstein and S. Ullman, "Class-specific, top-down segmentation," in *Computer Vision—ECCV 2002: 7th European Conference on Computer Vision Copenhagen, Denmark, May 28–31, 2002 Proceedings, Part II 7*, pp. 109–122, Springer, 2002.

[66] F. Schroff, A. Criminisi, and A. Zisserman, "Single-histogram class models for image segmentation," in *Computer Vision, Graphics and Image Processing: 5th Indian Conference, ICVGIP 2006, Madurai, India, December 13-16, 2006. Proceedings*, pp. 82–93, Springer, 2006.

[67] H. Yu *et al.*, "Methods and datasets on semantic segmentation: A review," *Neurocomputing*, vol. 304, pp. 82–103, 2018.

[68] J. Shotton, J. Winn, C. Rother, and A. Criminisi, "*TextonBoost*: Joint appearance, shape and context modeling for multi-class object recognition and segmentation," in *Computer Vision–ECCV 2006: 9th European Conference on Computer Vision, Graz, Austria, May 7-13, 2006. Proceedings, Part I 9*, pp. 1–15, Springer, 2006.

[69] J. Shotton, J. Winn, C. Rother, and A. Criminisi, "TextonBoost for image understanding: Multi-class object recognition and segmentation by jointly modeling texture, layout, and context," *International journal of computer vision*, vol. 81, pp. 2–23, 2009.

[70] J. Lafferty, A. McCallum, and F. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," in *International Conference on Machine Learning (ICML)*, 2001.

[71] P. Sturgess, K. Alahari, L. Ladicky, and P. H. Torr, "Combining appearance and structure from motion features for road scene understanding," in *BMVC-British Machine Vision Conference*, BMVA, 2009.

[72] O. Russakovsky *et al.*, "ImageNet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, pp. 211–252, 2015.

[73] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580–587, 2014.

[74] R. B. Girshick, P. F. Felzenszwalb, and D. McAllester, "Discriminatively trained deformable part models, release 5." [Online], 2012. Available: http://www.rossgirshick.info/latent/. Accessed 23 June 2024.

[75] A. Plaksyvyi, M. Skublewska-Paszkowska, and P. Powroznik, "A comparative analysis of image segmentation using classical and deep learning approach," *Advances in Science and Technology. Research Journal*, vol. 17, no. 6, 2023.

[76] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

[77] M. Abadi *et al.*, "TensorFlow: A system for large-scale machine learning," in *12th USENIX symposium on operating systems design and implementation*, pp. 265–283, 2016.

[78] F. Chollet, "Keras." [Online], 2015. Available: https://keras.io. Accessed 23 June 2024.

[79] A. Paszke *et al.*, "PyTorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[80] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[81] S. Pouyanfar *et al.*, "A survey on deep learning: Algorithms, techniques, and applications," *ACM Computing Surveys (CSUR)*, vol. 51, no. 5, pp. 1–36, 2018.

[82] M. Z. Alom *et al.*, "A state-of-the-art survey on deep learning theory and architectures," *Electronics*, vol. 8, no. 3, p. 292, 2019.

[83] J. Howard and S. Gugger, *Deep Learning for Coders with fastai and PyTorch*. O'Reilly Media, 2020.

[84] F. Chollet, *Deep learning with Python*. Simon and Schuster, 2021.

[85] A. Glassner, *Deep learning: A visual approach*. No Starch Press, 2021.

[86] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, *Dive into Deep Learning*. Cambridge University Press, 2023.

[87] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain.," *Psychological review*, vol. 65, no. 6, p. 386, 1958.

[88] J. A. Yacim and D. G. B. Boshoff, "Impact of artificial neural networks training algorithms on accurate prediction of property values," *Journal of Real Estate Research*, vol. 40, no. 3, pp. 375–418, 2018.

[89] K. Fukushima, "Visual feature extraction by a multilayered network of analog threshold elements," *IEEE Transactions on Systems Science and Cybernetics*, vol. 5, no. 4, pp. 322–333, 1969.

[90] V. Nair and G. E. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.

[91] G. S. Bhumbra, "Deep learning improved by biological activation functions," *arXiv:1804.11237*, 2018.

[92] D. E. Rumelhart, G. E. Hinton, and R. Williams, "Learning internal representations by error propagation," tech. rep., Institute for Cognitive Science, University of California, San Diego La Jolla, 1985.

[93] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors," *Nature*, vol. 323, pp. 533–536, 1986.

[94] N. Nedjah, I. Santos, and L. de Macedo Mourelle, "Sentiment analysis using convolutional neural network via word embeddings," *Evolutionary Intelligence*, vol. 15, no. 4, pp. 2295–2319, 2022.

[95] M. Nielsen, "Neural networks and deep learning." [Online], 2015. Available: http://neuralnetworksanddeeplearning.com/. Accessed 23 June 2024.

[96] H. Xiao, K. Rasul, and R. Vollgraf, "Fashion-MNIST: A novel image dataset for benchmarking machine learning algorithms," *arXiv:1708.07747*, 2017.

[97] S. Booth, Y. Zhou, A. Shah, and J. Shah, "Bayes-probe: Distribution-guided sampling for prediction level sets," *arXiv:2002.10248*, 2020.

[98] Y. Guo, Y. Liu, A. Oerlemans, S. Lao, S. Wu, and M. S. Lew, "Deep learning for visual understanding: A review," *Neurocomputing*, vol. 187, pp. 27–48, 2016.

[99] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *The Journal of physiology*, vol. 160, no. 1, p. 106, 1962.

[100] K. Fukushima, "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position," *Biological cybernetics*, vol. 36, no. 4, pp. 193–202, 1980.

[101] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[102] Y. LeCun *et al.*, "Backpropagation applied to handwritten zip code recognition," *Neural computation*, vol. 1, no. 4, pp. 541–551, 1989.

[103] NVIDIA Developer website, "Convolution." [Online], 2024. Available: https://developer.nvidia.com/discover/convolution. Accessed 5 January 2024.

[104] M. Yani, B. Irawan, and C. Setiningsih, "Application of transfer learning using convolutional neural network method for early detection of Terry's nail," in *Journal of Physics: Conference Series*, vol. 1201, IOP Publishing, 2019.

[105] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4510–4520, 2018.

[106] A. Howard *et al.*, "Searching for MobileNetV3," in *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 1314–1324, 2019.

[107] M. Tan and Q. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *International conference on machine learning*, pp. 6105–6114, PMLR, 2019.

[108] A. G. Howard *et al.*, "MobileNets: Efficient convolutional neural networks for mobile vision applications," *arXiv:1704.04861*, 2017.

[109] L. Sifre, *Rigid-motion Scattering for Image Classification*. PhD thesis, École Polytechnique, 2014.

[110] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*, pp. 448–456, PMLR, 2015.

[111] H. Cai, J. Lin, and S. Han, "Efficient methods for deep learning," in *Advanced Methods and Deep Learning in Computer Vision*, pp. 159–190, Elsevier, 2022.

[112] M. Tan *et al.*, "MnasNet: Platform-aware neural architecture search for mobile," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 2820–2828, 2019.

[113] J. Hu, L. Shen, and G. Sun, "Squeeze-and-excitation networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7132–7141, 2018.

[114] T.-J. Yang *et al.*, "Netadapt: Platform-aware neural network adaptation for mobile applications," in *Proceedings of the European conference on computer vision (ECCV)*, pp. 285–300, 2018.

[115] P. Ramachandran, B. Zoph, and Q. V. Le, "Searching for activation functions," *arXiv:1710.05941*, 2017.

[116] Y. Huang *et al.*, "GPipe: Efficient training of giant neural networks using pipeline parallelism," *Advances in neural information processing systems*, vol. 32, 2019.

[117] F. Chollet, "Xception: Deep learning with depthwise separable convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1251–1258, 2017.

[118] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation," *arXiv:1706.05587*, 2017.

[119] C. Szegedy *et al.*, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[120] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, "Rethinking the Inception architecture for computer vision," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2818–2826, 2016.

[121] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. Alemi, "Inception-v4, Inception-ResNet and the impact of residual connections on learning," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 31, 2017.

[122] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Semantic image segmentation with deep convolutional nets and fully connected CRFs," *arXiv:1412.7062*, 2014.

[123] L.-C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "DeepLab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected CRFs," *IEEE transactions on pattern analysis and machine intelligence*, vol. 40, no. 4, pp. 834–848, 2017.

[124] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam, "Encoder-decoder with atrous separable convolution for semantic image segmentation," in *Proceedings of the European conference on computer vision (ECCV)*, pp. 801–818, 2018.

[125] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv:1409.1556*, 2014.

[126] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3431–3440, 2015.

[127] M. Holschneider, R. Kronland-Martinet, J. Morlet, and P. Tchamitchian, "A real-time algorithm for signal analysis with the help of the wavelet transform," in *Wavelets: Time-Frequency Methods and Phase Space Proceedings of the International Conference, Marseille, France, December 14–18, 1987*, pp. 286–297, Springer, 1990.

[128] R. Mottaghi *et al.*, "The role of context for object detection and semantic segmentation in the wild," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 891–898, 2014.

[129] X. Chen *et al.*, "Detect what you can: Detecting and representing objects using holistic models and body parts," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1971–1978, 2014.

[130] M. Cordts *et al.*, "The Cityscapes dataset for semantic urban scene understanding," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3213–3223, 2016.

[131] V. Singh, "DeepLabv3 & DeepLabv3+ the ultimate PyTorch guide." [Online], 2022. Available: https://learnopencv.com/deeplabv3-ultimate-guide/. Accessed 7 March 2024.

[132] R. Girshick, "Fast R-CNN," in *Proceedings of the IEEE international conference on computer vision*, pp. 1440–1448, 2015.

[133] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, vol. 28, 2015.

[134] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask R-CNN," in *Proceedings of the IEEE international conference on computer vision*, pp. 2961–2969, 2017.

[135] G. Jocher, A. Chaurasia, and J. Qiu, "Ultralytics YOLOv8." [Online], 2023. Available: https://github.com/ultralytics. Accessed 7 August 2024.

[136] S. Aharon *et al.*, "Super-Gradients." [Online], 2021. Available: https://github.com/Deci-AI/super-gradients. Accessed 29 June 2024.

[137] M. Hussain, "YOLO-v1 to YOLO-v8, the rise of YOLO and its complementary nature toward digital manufacturing and industrial defect detection," *Machines*, vol. 11, no. 7, p. 677, 2023.

[138] A. L. Maas, A. Y. Hannun, and A. Y. Ng, "Rectifier nonlinearities improve neural network acoustic models," in *Proc. ICML*, vol. 30, p. 3, Atlanta, GA, 2013.

[139] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 7263–7271, 2017.

[140] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," *arXiv:1804.02767*, 2018.

[141] T.-Y. Lin *et al.*, "Microsoft COCO: Common objects in context," in *Computer Vision–ECCV 2014. ECCV 2014. Lecture Notes in Computer Science*, vol. 8693, pp. 740–755, Springer, 2014.

[142] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, pp. 2980–2988, 2017.

[143] W. Liu *et al.*, "SSD: Single shot MultiBox detector," in *Computer Vision–ECCV 2016. ECCV 2016. Lecture Notes in Computer Science.*, vol. 9905, pp. 21–37, Springer, 2016.

[144] C.-Y. Fu, W. Liu, A. Ranga, A. Tyagi, and A. C. Berg, "DSSD: Deconvolutional single shot detector," *arXiv:1701.06659*, 2017.

[145] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: Optimal speed and accuracy of object detection," *arXiv:2004.10934*, 2020.

[146] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1492–1500, 2017.

[147] C.-Y. Wang *et al.*, "CSPNet: A new backbone that can enhance learning capability of CNN," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pp. 390–391, 2020.

[148] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2117–2125, 2017.

[149] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia, "Path aggregation network for instance segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 8759–8768, 2018.

[150] S. Yun *et al.*, "CutMix: Regularization strategy to train strong classifiers with localizable features," in *Proceedings of the IEEE/CVF international conference on computer vision*, pp. 6023–6032, 2019.

[151] Z. Zheng *et al.*, "Distance-IoU loss: Faster and better learning for bounding box regression," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, pp. 12993–13000, 2020.

[152] D. Misra, "Mish: A self regularized non-monotonic activation function," *arXiv:1908.08681*, 2019.

[153] G. Jocher, "YOLOv5 by Ultralytics." [Online], 2020. Available: https://github.com/ultralytics/yolov5. Accessed 7 August 2024.

[154] D. Hendrycks and K. Gimpel, "Gaussian error linear units (GELUs)," *arXiv:1606.08415*, 2016.

[155] A. Buslaev *et al.*, "Albumentations: Fast and flexible image augmentations," *Information*, vol. 11, no. 2, p. 125, 2020.

[156] J. Terven, D.-M. Córdova-Esparza, and J.-A. Romero-González, "A comprehensive review of YOLO architectures in computer vision: From YOLOv1 to YOLOv8 and YOLO-NAS," *Machine Learning and Knowledge Extraction*, vol. 5, no. 4, pp. 1680–1716, 2023.

[157] Midjourney research lab, "Midjourney." [Online], 2022. Available: https://www.midjourney.com. Accessed 13 March 2024.

[158] A. Ramesh *et al.*, "Zero-shot text-to-image generation," in *International Conference on Machine Learning*, pp. 8821–8831, PMLR, 2021.

[159] E. Denton, A. Hanna, R. Amironesei, A. Smart, and H. Nicole, "On the genealogy of machine learning datasets: A critical history of ImageNet," *Big Data & Society*, vol. 8, no. 2, pp. 1–14, 2021.

[160] A. Krizhevsky, "Learning multiple layers of features from tiny images," Master's thesis, University of Toronto, 2009.

[161] S. Priya and R. A. Uthra, "Deep learning framework for handling concept drift and class imbalanced complex decision-making on streaming data," *Complex & Intelligent Systems*, vol. 9, no. 4, pp. 3499–3515, 2023.

[162] J. M. Johnson and T. M. Khoshgoftaar, "Survey on deep learning with class imbalance," *Journal of Big Data*, vol. 6, no. 1, pp. 1–54, 2019.

[163] F. Zhuang *et al.*, "A comprehensive survey on transfer learning," *Proceedings of the IEEE*, vol. 109, no. 1, pp. 43–76, 2020.

[164] S. Niu, Y. Liu, J. Wang, and H. Song, "A decade survey of transfer learning (2010–2020)," *IEEE Transactions on Artificial Intelligence*, vol. 1, no. 2, pp. 151–166, 2020.

[165] K. Weiss, T. M. Khoshgoftaar, and D. Wang, "A survey of transfer learning," *Journal of Big data*, vol. 3, pp. 1–40, 2016.

[166] Chollet, François, "Complete guide to transfer learning & fine-tuning in Keras." [Online], 2020. Available: https://keras.io/guides/transfer_learning/. Accessed 1 May 2024.

[167] B.-X. Wu, C.-G. Yang, and J.-P. Zhong, "Research on transfer learning of vision-based gesture recognition," *International Journal of Automation and Computing*, vol. 18, no. 3, pp. 422–431, 2021.

[168] S. Chiba and H. Sasaoka, "Basic study for transfer learning for autonomous driving in car race of model car," in *2021 6th International Conference on Business and Industrial Research (ICBIR)*, pp. 138–141, IEEE, 2021.

[169] J. Hua, L. Zeng, G. Li, and Z. Ju, "Learning for a robot: Deep reinforcement learning, imitation learning, transfer learning," *Sensors*, vol. 21, no. 4, p. 1278, 2021.

[170] H. E. Kim, A. Cosa-Linan, N. Santhanam, M. Jannesari, M. E. Maros, and T. Ganslandt, "Transfer learning for medical image classification: A literature review," *BMC Medical Imaging*, vol. 22, no. 1, p. 69, 2022.

[171] C. Shorten and T. M. Khoshgoftaar, "A survey on image data augmentation for deep learning," *Journal of Big Data*, vol. 6, no. 1, pp. 1–48, 2019.

[172] L. Alzubaidi *et al.*, "Review of deep learning: Concepts, CNN architectures, challenges, applications, future directions," *Journal of Big Data*, vol. 8, pp. 1–74, 2021.

[173] M. Ivanovs, R. Kadikis, M. Lulla, A. Rutkovskis, and A. Elsts, "Automated quality assessment of hand washing using deep learning," *arXiv:2011.11383*, 2020.

[174] M. Lulla *et al.*, "Hand-washing video dataset annotated according to the World Health Organization's hand-washing guidelines," *Data*, vol. 6, no. 4, p. 38, 2021.

[175] O. Zemlanuhina *et al.*, "Influence of different types of real-time feedback on hand washing quality assessed with neural networks/simulated neural networks," in *SHS Web of Conferences*, vol. 131, pp. 1–13, EDP Sciences, 2022.

[176] A. Elsts, M. Ivanovs, R. Kadikis, and O. Sabelnikovs, "CNN for hand washing movement classification: What matters more–the approach or the dataset?," in *2022 Eleventh International Conference on Image Processing Theory, Tools and Applications (IPTA)*, pp. 1–6, IEEE, 2022.

[177] S. L. Barnes, D. J. Morgan, A. D. Harris, P. C. Carling, and K. A. Thom, "Preventing the transmission of multidrug-resistant organisms: Modeling the relative importance of hand hygiene and environmental cleaning interventions," *Infection Control & Hospital Epidemiology*, vol. 35, no. 9, pp. 1156–1162, 2014.

[178] European Centre for Disease Prevention and Control, "Assessing the health burden of infections with antibiotic-resistant bacteria in the EU/EEA, 2016–2020," tech. rep., ECDC, Stockholm, 2022.

[179] C. J. Murray *et al.*, "Global burden of bacterial antimicrobial resistance in 2019: A systematic analysis," *The Lancet*, vol. 399, no. 10325, pp. 629–655, 2022.

[180] B. Allegranzi and D. Pittet, "Role of hand hygiene in healthcare-associated infection prevention," *Journal of Hospital Infection*, vol. 73, no. 4, pp. 305–315, 2009.

[181] C. Suetens *et al.*, "Prevalence of healthcare-associated infections, estimated incidence and composite antimicrobial resistance index in acute care hospitals and long-term care facilities: Results from two European point prevalence surveys, 2016 to 2017," *Eurosurveillance*, vol. 23, no. 46, pp. 1–17, 2018.

[182] E. Goldman, "Exaggerated risk of transmission of COVID-19 by fomites," *The Lancet Infectious Diseases*, vol. 20, no. 8, pp. 892–893, 2020.

[183] K. J. Mckay, R. Z. Shaban, and P. Ferguson, "Hand hygiene compliance monitoring: Do video-based technologies offer opportunities for the future?," *Infection, Disease & Health*, vol. 25, no. 2, pp. 92–100, 2020.

[184] World Health Organization, *WHO guidelines on hand hygiene in health care*. World Health Organization, 2009.

[185] N. Luangasanatip *et al.*, "Comparative efficacy of interventions to promote hand hygiene in hospital: Systematic review and network meta-analysis," *BMJ*, vol. 351, 2015.

[186] D. J. Gould, D. Moralejo, N. Drey, J. H. Chudleigh, and M. Taljaard, "Interventions to improve hand hygiene compliance in patient care," *Cochrane database of systematic reviews*, no. 9, 2017.

[187] N. Masroor, M. Doll, M. Stevens, and G. Bearman, "Approaches to hand hygiene monitoring: From low to high technology approaches," *International Journal of Infectious Diseases*, vol. 65, pp. 101–104, 2017.

[188] B. C. Knepper, A. M. Miller, and H. L. Young, "Impact of an automated hand hygiene monitoring system combined with a performance improvement intervention on hospital-acquired infections," *Infection Control & Hospital Epidemiology*, vol. 41, no. 8, pp. 931–937, 2020.

[189] The Joint Commission, "Measuring hand hygiene adherence: Overcoming the challenges," 2009.

[190] M. Willmott *et al.*, "Effectiveness of hand hygiene interventions in reducing illness absence among children in educational settings: A systematic review and meta-analysis," *Archives of Disease in Childhood*, vol. 101, no. 1, pp. 42–50, 2016.

[191] S. J. S. Aghdassi *et al.*, "A multimodal intervention to improve hand hygiene compliance in peripheral wards of a tertiary care university centre: A cluster randomised controlled trial," *Antimicrobial Resistance & Infection Control*, vol. 9, no. 1, pp. 1–9, 2020.

[192] M. Biswal *et al.*, "Evaluation of the short-term and long-term effect of a short series of hand hygiene campaigns on improving adherence in a tertiary care hospital in India," *American Journal of Infection Control*, vol. 42, no. 9, pp. 1009–1010, 2014.

[193] Y. Suzuki, M. Morino, I. Morita, and S. Yamamoto, "The effect of a 5-year hand hygiene initiative based on the who multimodal hand hygiene improvement strategy: An interrupted time-series study," *Antimicrobial Resistance & Infection Control*, vol. 9, pp. 1–12, 2020.

[194] R. T. Ellison III, C. M. Barysauskas, E. A. Rundensteiner, D. Wang, and B. Barton, "A prospective controlled trial of an electronic hand hygiene reminder system," in *Open Forum Infectious Diseases*, vol. 2, p. ofv121, Oxford University Press, 2015.

[195] M. McGuckin, R. Waterman, and J. Govednik, "Hand hygiene compliance rates in the united states—a one-year multicenter collaboration using product/volume usage measurement and feedback," *American Journal of Medical Quality*, vol. 24, no. 3, pp. 205–213, 2009.

[196] J. M. Boyce *et al.*, "Impact of an automated hand hygiene monitoring system and additional promotional activities on hand hygiene performance rates and healthcare-associated infections," *Infection Control & Hospital Epidemiology*, vol. 40, no. 7, pp. 741–747, 2019.

[197] J. A. Srigley, C. D. Furness, G. R. Baker, and M. Gardam, "Quantification of the hawthorne effect in hand hygiene compliance monitoring using an electronic monitoring system: A retrospective cohort study," *BMJ Quality & Safety*, vol. 23, no. 12, pp. 974–980, 2014.

[198] M. Oudah, A. Al-Naji, and J. Chahl, "Hand gesture recognition based on computer vision: A review of techniques," *Journal of Imaging*, vol. 6, no. 8:73, 2020.

[199] "Sample: Kaggle Hand Wash Dataset." [Online], 2019. Available: https://www.kaggle.com/realtimear/hand-wash-dataset. Accessed 18 February 2024.

[200] M. A. Ward *et al.*, "Automated and electronically assisted hand hygiene monitoring systems: A systematic review," *American Journal of Infection Control*, vol. 42, no. 5, pp. 472–478, 2014.

[201] C. Wang *et al.*, "Electronic monitoring systems for hand hygiene: Systematic review of technology," *Journal of Medical Internet Research*, vol. 23, no. 11, p. e27880, 2021.

[202] J. Srigley *et al.*, "Hand hygiene monitoring technology: A systematic review of efficacy," *Journal of Hospital Infection*, vol. 89, no. 1, pp. 51–60, 2015.

[203] C. Wang *et al.*, "Accurate measurement of handwash quality using sensor armbands: Instrument validation study," *JMIR MHealth Uhealth*, vol. 8, no. 3, p. e17001, 2020.

[204] V. Galluzzi, *Automatic recognition of healthcare worker hand hygiene*. PhD thesis, University of Iowa, 2015.

[205] V. Galluzzi, T. Herman, and P. Polgreen, "Hand hygiene duration and technique recognition using wrist-worn sensors," in *Proceedings of the 14th International Conference on Information Processing in Sensor Networks*, pp. 106–117, 2015.

[206] J. Hoey, A. Von Bertoldi, P. Poupart, and A. Mihailidis, "Assisting persons with dementia during handwashing using a partially observable Markov decision process.," in *Proceedings of the 5th International Conference on Computer Vision Systems (ICVS 2007)*, 2007.

[207] D. F. Llorca, I. Parra, M. Á. Sotelo, and G. Lacey, "A vision-based system for automatic hand washing quality assessment," *Machine Vision and Applications*, vol. 22, no. 2, pp. 219–234, 2011.

[208] S. Yeung *et al.*, "Vision-based hand hygiene monitoring in hospitals," *AMIA*, 2016.

[209] G. Li *et al.*, "Hand gesture recognition based on convolution neural network," *Cluster Computing*, vol. 22, no. 2, pp. 2719–2729, 2019.

[210] E. Prakasa and B. Sugiarto, "Video analysis on handwashing movement for the completeness evaluation," in *2020 International Conference on Radar, Antenna, Microwave, Electronics, and Telecommunications (ICRAMET)*, pp. 296–301, IEEE, 2020.

[211] A. Nagaraj, M. Sood, C. Sureka, and G. Srinivasa, "Real-time action recognition for fine-grained actions and the hand wash dataset," *arXiv:2210.07400*, 2022.

[212] C. Feichtenhofer, A. Pinz, and A. Zisserman, "Convolutional two-stream network fusion for video action recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1933–1941, 2016.

[213] K. Simonyan and A. Zisserman, "Two-stream convolutional networks for action recognition in videos," *arXiv:1406.2199*, 2014.

[214] K. Cikel, M. Arzamendia, D. Gregor, D. Gutiérrez, and S. Toral, "Evaluation of a CNN+LSTM system for the classification of hand-washing steps," in *XIX Conference of the Spanish Association for Artificial Intelligence (CAEPIA)*, 2021.

[215] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[216] K. Yamamoto, M. Yoshii, F. Kinoshita, and H. Touyama, "Classification vs regression by CNN for handwashing skills evaluations in nursing education," in *2020 International Conference on Artificial Intelligence in Information and Communication (ICAIIC)*, pp. 590–593, IEEE, 2020.

[217] W. Kay *et al.*, "The Kinetics human action video dataset," *arXiv:1705.06950*, 2017.

[218] Y. Yoshikawa, J. Lin, and A. Takeuchi, "STAIR actions: A video dataset of everyday home actions," *arXiv:1804.04326*, 2018.

[219] W. E. Trick *et al.*, "Impact of ring wearing on hand contamination and comparison of hand hygiene agents in a hospital," *Clinical infectious diseases*, vol. 36, no. 11, pp. 1383–1390, 2003.

[220] A. Hautemaniere *et al.*, "Factors determining poor practice in alcoholic gel hand rub technique in hospital workers," *Journal of infection and Public Health*, vol. 3, no. 1, pp. 25–34, 2010.

[221] World Health Organization, *WHO guidelines on hand hygiene in health care*. 2009.

[222] K. Cho, B. Van Merriënboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," *arXiv:1409.1259*, 2014.

[223] M. Ivanovs, K. Ozols, A. Dobrajs, and R. Kadikis, "Improving semantic segmentation of urban scenes for self-driving cars with synthetic images," *Sensors*, vol. 22, no. 6, p. 2252, 2022.

[224] A. Jiwani, S. Ganguly, C. Ding, N. Zhou, and D. M. Chan, "A semantic segmentation network for urban-scale building footprint extraction using RGB satellite imagery," *arXiv:2104.01263*, 2021.

[225] R. E. Huerta *et al.*, "Mapping urban green spaces at the metropolitan level using very high resolution satellite imagery and deep learning techniques for semantic segmentation," *Remote Sensing*, vol. 13, no. 11, p. 2031, 2021.

[226] K. Yuan, X. Zhuang, G. Schaefer, J. Feng, L. Guan, and H. Fang, "Deep-learning-based multispectral satellite image segmentation for water body detection," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 14, pp. 7422–7434, 2021.

[227] M. Wurm, T. Stark, X. X. Zhu, M. Weigand, and H. Taubenböck, "Semantic segmentation of slums in satellite images using transfer learning on fully convolutional neural networks," *ISPRS journal of photogrammetry and remote sensing*, vol. 150, pp. 59–69, 2019.

[228] S. Thrun, "Toward robotic cars," *Communications of the ACM*, vol. 53, no. 4, pp. 99–106, 2010.

[229] T. Litman, "Autonomous vehicle implementation predictions," tech. rep., Victoria Transport Policy Institute Victoria, Canada, 2017.

[230] J. Van Brummelen, M. O'Brien, D. Gruyer, and H. Najjaran, "Autonomous vehicle perception: The technology of today and tomorrow," *Transportation research part C: Emerging technologies*, vol. 89, pp. 384–406, 2018.

[231] F. Arena, G. Pau, and M. Collotta, "A survey on driverless vehicles: From their diffusion to security," *Journal of Internet Services and Information Security (JISIS)*, vol. 8, pp. 1–19, 2018.

[232] C. Badue *et al.*, "Self-driving cars: A survey," *Expert Systems with Applications*, vol. 165:113816, 2020.

[233] G. Biggi and J. Stilgoe, "Artificial intelligence in self-driving cars research and innovation: A scientometric and bibliometric analysis." [Online], 2021. Available: https://ssrn.com/abstract=3829897. Accessed 29 June 2024.

[234] SAE International, "Taxonomy and definitions for terms related to driving automation systems for on-road motor vehicles." [Online], 2018. Version J3016_201806. Available: https://www.sae.org/standards/content/j3016_201806/. Accessed 29 June 2024.

[235] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Transactions on intelligent vehicles*, vol. 1, no. 1, pp. 33–55, 2016.

[236] M. Treml *et al.*, "Speeding up semantic segmentation for autonomous driving," in *Neural Information Processing Systems (NIPS) Workshop 'Machine Learning for Intelligent Transportation Systems' (MLITS)*, 2016.

[237] Q. Sellat, S. K. Bisoy, and R. Priyadarshini, "Semantic segmentation for self-driving cars using deep learning: A survey," in *Cognitive Big Data Intelligence with a Metaheuristic Approach*, pp. 211–238, Elsevier, 2022.

[238] G. J. Brostow, J. Fauqueur, and R. Cipolla, "Semantic object classes in video: A high-definition ground truth database," *Pattern Recognition Letters*, vol. 30, no. 2, pp. 88–97, 2009.

[239] T. Scharwächter, M. Enzweiler, U. Franke, and S. Roth, "Efficient multi-cue scene segmentation," in *German Conference on Pattern Recognition*, pp. 435–445, Springer, 2013.

[240] H. Abu Alhaija, S. K. Mustikovela, L. Mescheder, A. Geiger, and C. Rother, "Augmented reality meets computer vision: Efficient data generation for urban driving scenes," *International Journal of Computer Vision*, vol. 126, pp. 961–972, 2018.

[241] J. Xie, M. Kiefel, M.-T. Sun, and A. Geiger, "Semantic instance annotation of street scenes by 3D to 2D label transfer," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3688–3697, 2016.

[242] G. Ros, L. Sellart, J. Materzynska, D. Vazquez, and A. M. Lopez, "The SYNTHIA dataset: A large collection of synthetic images for semantic segmentation of urban scenes," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3234–3243, 2016.

[243] S. R. Richter, V. Vineet, S. Roth, and V. Koltun, "Playing for data: Ground truth from computer games," in *European conference on computer vision*, pp. 102–118, Springer, 2016.

[244] M. Hahner, D. Dai, C. Sakaridis, J.-N. Zaech, and L. Van Gool, "Semantic understanding of foggy scenes with purely synthetic data," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pp. 3675–3681, IEEE, 2019.

[245] B. Wymann *et al.*, "TORCS: The open racing car simulator." [Online], 2015. Available: https://www.cse.chalmers.se/~chrdimi/papers/torcs.pdf. Accessed 29 June 2024.

[246] L. Berlincioni, F. Becattini, L. Galteri, L. Seidenari, and A. Del Bimbo, "Road layout understanding by generative adversarial inpainting," in *Inpainting and Denoising Challenges*, pp. 111–128, Springer, 2019.

[247] M. Bertalmio, G. Sapiro, V. Caselles, and C. Ballester, "Image inpainting," in *Proceedings of the 27th annual conference on computer graphics and interactive techniques*, pp. 417–424, 2000.

[248] G. Liu *et al.*, "Image inpainting for irregular holes using partial convolutions," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 85–100, 2018.

[249] F. S. Saleh, M. S. Aliakbarian, M. Salzmann, L. Petersson, and J. M. Alvarez, "Effective use of synthetic data for urban scene semantic segmentation," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 84–100, 2018.

[250] A. Cordeiro, L. F. Rocha, C. Costa, P. Costa, and M. F. Silva, "Bin picking approaches based on deep learning techniques: A state-of-the-art survey," in *2022 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pp. 110–117, IEEE, 2022.

[251] D. Duplevska, M. Ivanovs, J. Arents, and R. Kadikis, "Sim2real image translation to improve a synthetic dataset for a bin picking task," in *2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)*, pp. 1–7, IEEE, 2022.

[252] A. Dzedzickis, J. Subačiūtė-Žemaitienė, E. Šutinys, U. Samukaitė-Bubnienė, and V. Bučinskas, "Advanced applications of industrial robotics: New trends and possibilities," *Applied Sciences*, vol. 12, no. 1:135, 2021.

[253] J. Arents and M. Greitans, "Smart industrial robot control trends, challenges and opportunities within manufacturing," *Applied Sciences*, vol. 12, no. 2:937, 2022.

[254] The MathWorks, Inc., "Gazebo simulation of semi-structured intelligent bin picking for UR5e using YOLO and PCA-based object detection." [Online]. Available: https://www.mathworks.com/help/robotics/urseries/ug/gazebo-simulation-ur5e-semistructured-intelligent-bin-picking-example.html. Accessed 27 February 2024.

[255] Q. Bai, S. Li, J. Yang, Q. Song, Z. Li, and X. Zhang, "Object detection recognition and robot grasping based on machine learning: A survey," *IEEE Access*, vol. 8, pp. 181855–181879, 2020.

[256] H. Choi *et al.*, "On the use of simulation in robotics: Opportunities, challenges, and suggestions for moving forward," *Proceedings of the National Academy of Sciences*, vol. 118, no. 1, p. e1907856118, 2021.

[257] J. Anderson, *Methods and Applications of Synthetic Data Generation*. PhD thesis, Clemson University, 2021.

[258] N. Jakobi, P. Husbands, and I. Harvey, "Noise and the reality gap: The use of simulation in evolutionary robotics," in *Advances in Artificial Life: Third European Conference on Artificial Life*, pp. 704–720, Springer, 1995.

[259] E. Buls, R. Kadikis, R. Cacurs, and J. Arents, "Generation of synthetic training data for object detection in piles," in *Eleventh International Conference on Machine Vision (ICMV 2018)*, vol. 11041, pp. 533–540, SPIE, 2019.

[260] J. Arents *et al.*, "Synthetic data of randomly piled, similar objects for deep learning-based object detection," in *International Conference on Image Analysis and Processing*, pp. 706–717, Springer, 2022.

[261] V. Feščenko, J. Arents, and R. Kadikis, "Synthetic data generation for visual detection of flattened PET bottles," *Machine Learning and Knowledge Extraction*, vol. 5, no. 1, pp. 14–28, 2023.

[262] N. Sünderhauf *et al.*, "The limits and potentials of deep learning for robotics," *The International journal of robotics research*, vol. 37, no. 4-5, pp. 405–420, 2018.

[263] M. Carranza-García, J. Torres-Mateo, P. Lara-Benítez, and J. García-Gutiérrez, "On the performance of one-stage and two-stage object detectors in autonomous vehicles using camera data," *Remote Sensing*, vol. 13, no. 1:89, 2020.

[264] L. Tian, N. M. Thalmann, D. Thalmann, Z. Fang, and J. Zheng, "Object grasping of humanoid robot based on YOLO," in *Advances in Computer Graphics: 36th Computer Graphics International Conference*, pp. 476–482, Springer, 2019.

[265] Z. Cao, T. Liao, W. Song, Z. Chen, and C. Li, "Detecting the shuttlecock for a badminton robot: A YOLO based approach," *Expert Systems with Applications*, vol. 164:113833, 2021.

[266] G. Zhaoxin, L. Han, Z. Zhijiang, and P. Libo, "Design a robot system for tomato picking based on YOLO v5," *IFAC-PapersOnLine*, vol. 55, no. 3, pp. 166–171, 2022.

[267] A. S. Olesen, B. B. Gergaly, E. A. Ryberg, M. R. Thomsen, and D. Chrysostomou, "A collaborative robot cell for random bin-picking based on deep learning policies and a multi-gripper switching strategy," *Procedia Manufacturing*, vol. 51, pp. 3–10, 2020.

[268] S. Lee and Y. Lee, "Real-time industrial bin-picking with a hybrid deep learning-engineering approach," in *2020 IEEE International Conference on Big Data and Smart Computing (BigComp)*, pp. 584–588, IEEE, 2020.

[269] P. Torres, J. Arents, H. Marques, and P. Marques, "Bin-picking solution for randomly placed automotive connectors based on machine learning techniques," *Electronics*, vol. 11, no. 3:476, 2022.

[270] J. Tremblay *et al.*, "Training deep networks with synthetic data: Bridging the reality gap by domain randomization," in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pp. 969–977, 2018.

[271] B. Sun, J. Feng, and K. Saenko, "Return of frustratingly easy domain adaptation," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 30, 2016.

[272] K. Bousmalis, N. Silberman, D. Dohan, D. Erhan, and D. Krishnan, "Unsupervised pixel-level domain adaptation with generative adversarial networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3722–3731, 2017.

[273] A. Shrivastava and other, "Learning from simulated and unsupervised images through adversarial training," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2107–2116, 2017.

[274] Bousmalis *et al.*, "Using simulation and domain adaptation to improve efficiency of deep robotic grasping," in *2018 IEEE international conference on robotics and automation (ICRA)*, pp. 4243–4250, IEEE, 2018.

[275] K. Rao *et al.*, "RL-CycleGAN: Reinforcement learning aware simulation-to-real," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 11157–11166, 2020.

[276] D. Ho *et al.*, "RetinaGAN: An object-aware approach to sim-to-real transfer," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 10920–10926, IEEE, 2021.

[277] Blender Foundation, "Blender 3D computer graphics software." [Online], 2022. Available: https://www.blender.org/. Accessed 19 August 2024.

[278] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros, "Unpaired image-to-image translation using cycle-consistent adversarial networks," in *Proceedings of the IEEE international conference on computer vision*, pp. 2223–2232, 2017.

[279] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional networks for biomedical image segmentation," in *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*, pp. 234–241, Springer, 2015.

[280] A. Odena, V. Dumoulin, and C. Olah, "Deconvolution and checkerboard artifacts," *Distill*, vol. 1, no. 10, 2016.

[281] W. Shi *et al.*, "Is the deconvolution layer the same as a convolutional layer?," *arXiv:1609.07009*, 2016.

[282] M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter, "GANs trained by a two time-scale update rule converge to a local Nash equilibrium," *Advances in neural information processing systems*, vol. 30, 2017.

[283] K. Kleeberger, R. Bormann, W. Kraus, and M. F. Huber, "A survey on learning-based robotic grasping," *Current Robotics Reports*, vol. 1, pp. 239–249, 2020.

[284] C. M. Leung *et al.*, "A guide to the organ-on-a-chip," *Nature Reviews Methods Primers*, vol. 2, no. 1, p. 33, 2022.

[285] M. Ivanovs *et al.*, "Synthetic image generation with a fine-tuned latent diffusion model for organ on chip cell image classification," in *Proceedings of 2023 Signal Processing: Algorithms, Architectures, Arrangements, and Applications (SPA)*, pp. 148–153, IEEE, 2023.

[286] V. Movčana *et al.*, "Organ-on-a-chip (OOC) image dataset for machine learning and tissue model evaluation," *Data*, vol. 9, no. 2, p. 28, 2024.

[287] S. A. Ajagbe, K. A. Amuda, M. A. Oladipupo, F. A. Oluwaseyi, and K. I. Okesola, "Multi-classification of Alzheimer disease on magnetic resonance images (MRI) using deep convolutional neural network (DCNN) approaches," *International Journal of Advanced Computer Research*, vol. 11, no. 53, p. 51, 2021.

[288] A. Iqbal, M. Sharif, M. A. Khan, W. Nisar, and M. Alhaisoni, "FF-UNet: a U-shaped deep convolutional neural network for multimodal biomedical image segmentation," *Cognitive Computation*, vol. 14, no. 4, pp. 1287–1302, 2022.

[289] Villa-Pulgarin *et al.*, "Optimized convolutional neural network models for skin lesion classification.," *Computers, Materials & Continua*, vol. 70, no. 2, 2022.

[290] S. Sharma *et al.*, "Performance evaluation of the deep learning based convolutional neural network approach for the recognition of chest X-ray images," *Frontiers in oncology*, vol. 12, 2022.

[291] Rodriguez-Ruiz *et al.*, "Stand-alone artificial intelligence for breast cancer detection in mammography: Comparison with 101 radiologists," *JNCI: Journal of the National Cancer Institute*, vol. 111, no. 9, pp. 916–922, 2019.

[292] H. Ali, S. Murad, and Z. Shah, "Spot the fake lungs: Generating synthetic medical images using neural diffusion models," in *Irish Conference on Artificial Intelligence and Cognitive Science*, pp. 32–39, Springer, 2022.

[293] E. J. Hu *et al.*, "LoRA: Low-rank adaptation of large language models," *arXiv:2106.09685*, 2021.

[294] S. R. Richter, Z. Hayder, and V. Koltun, "Playing for benchmarks," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2213–2222, 2017.

[295] D. P. Kingma and M. Welling, "Auto-encoding variational Bayes," *arXiv:1312.6114*, 2013.

[296] A. Volokitin *et al.*, "Modelling the distribution of 3D brain MRI using a 2D slice VAE," in *Proceedings of 23rd International Conference on Medical Image Computing and Computer-Assisted Intervention–MICCAI 2020*, pp. 657–666, Springer, 2020.

[297] D. E. Diamantis, P. Gatoula, and D. K. Iakovidis, "EndoVAE: Generating endoscopic images with a variational autoencoder," in *2022 IEEE 14th Image, Video, and Multidimensional Signal Processing Workshop (IVMSP)*, pp. 1–5, IEEE, 2022.

[298] Y. Kim, S. Wiseman, A. Miller, D. Sontag, and A. Rush, "Semi-amortized variational autoencoders," in *International Conference on Machine Learning*, pp. 2678–2687, PMLR, 2018.

[299] J. Tomczak and M. Welling, "VAE with a VampPrior," in *International Conference on Artificial Intelligence and Statistics*, pp. 1214–1223, PMLR, 2018.

[300] L. Ma, R. Shuai, X. Ran, W. Liu, and C. Ye, "Combining DC-GAN with ResNet for blood cell image classification," *Medical & biological engineering & computing*, vol. 58, pp. 1251–1264, 2020.

[301] T. Iqbal and H. Ali, "Generative adversarial network for medical images (MI-GAN)," *Journal of medical systems*, vol. 42, pp. 1–11, 2018.

[302] H. Chen, "Challenges and corresponding solutions of generative adversarial networks (GANs): A survey study," in *Journal of Physics: Conference Series*, vol. 1827, p. 012066, IOP Publishing, 2021.

[303] M. Akrout *et al.*, "Diffusion-based data augmentation for skin disease classification: Impact across original medical datasets to fully synthetic images," *arXiv:2301.04802*, 2023.

[304] P. Chambon, C. Bluethgen, C. P. Langlotz, and A. Chaudhari, "Adapting pretrained vision-language foundational models to medical imaging domains," *arXiv:2210.04133*, 2022.

[305] J. Sohl-Dickstein, E. Weiss, N. Maheswaranathan, and S. Ganguli, "Deep unsupervised learning using nonequilibrium thermodynamics," in *International Conference on Machine Learning*, pp. 2256–2265, PMLR, 2015.

[306] F.-A. Croitoru, V. Hondru, R. T. Ionescu, and M. Shah, "Diffusion models in vision: A survey," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 45, no. 9, pp. 10850–10869, 2023.

[307] X. Li, M. Sakevych, G. Atkinson, and V. Metsis, "Biodiffusion: A versatile diffusion model for biomedical signal synthesis," *Bioengineering*, vol. 11, no. 4, p. 299, 2024.

[308] J. Ho, A. Jain, and P. Abbeel, "Denoising diffusion probabilistic models," *Advances in Neural Information Processing Systems*, vol. 33, pp. 6840–6851, 2020.

[309] P. Esser *et al.*, "Scaling rectified flow transformers for high-resolution image synthesis," in *Forty-first International Conference on Machine Learning*, 2024.

[310] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv:1810.04805*, 2018.

[311] A. Vaswani *et al.*, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[312] R. Rombach *et al.*, "Stable Diffusion." [Online], 2022. Available: https://github.com/CompVis/stable-diffusion. Accessed 6 August 2024.

[313] R. Rombach, A. Blattmann, K. Crowson, A. Khaliq, and P. Esser, "Latent diffusion models." [Online], 2022. Available: https://github.com/CompVis/latent-diffusion. Accessed 6 August 2024.

[314] A. Radford *et al.*, "Learning transferable visual models from natural language supervision," in *International Conference on Machine Learning*, pp. 8748–8763, PMLR, 2021.

[315] "Stable Diffusion web UI." [Online], 2022. Available: https://github.com/AUTOMATIC1111/stable-diffusion-webui. Accessed 6 August 2024.

[316] C. Lu, Y. Zhou, F. Bao, J. Chen, C. Li, and J. Zhu, "DPM-Solver: A fast ODE solver for diffusion probabilistic model sampling in around 10 steps," *Advances in Neural Information Processing Systems*, vol. 35, pp. 5775–5787, 2022.

[317] I. Stenbit, F. Chollet, and L. Wood, "A walk through latent space with Stable Diffusion." [Online], 2022. Available: https://keras.io/examples/generative/random_walks_with_stable_diffusion/. Accessed 7 August 2024.