# Development of a Fully-Convolutional-Network Architecture for the Detection of Defective LED Chips in Photoluminescence Images

Entwicklung einer Fully-Convolutional-Netzwerkarchitektur
für die Detektion von defekten LED-Chips
in Photolumineszenzbildern

Der technischen Fakultät
der Friedrich-Alexander-Universität Erlangen-Nürnberg
zur Erlangung des Doktorgrads Dr.-Ing.
vorgelegt von

Maike Lorena Stern

Als Dissertation genehmigt von der
Technischen Fakultät
der Friedrich-Alexander-Universität Erlangen-Nürnberg.
Tag der mündlichen Prüfung: 8. Mai 2020

Vorsitzender des Promotionsorgans: Prof. Dr.-Ing. habil. Andreas Paul Fröba
Gutachter: Prof. Dr. Klaus Meyer-Wegener
Prof. Dr. Björn Eskofier

**Abstract**

Nowadays, light-emitting diodes (LEDs) can be found in a large variety of applications, from standard LEDs in domestic lighting solutions to advanced chip designs in automobiles, smart watches and video walls. The advances in chip design also affect the test processes, where the execution of certain contact measurements is exacerbated by ever decreasing chip dimensions or even rendered impossible due to the chip design. As an instance, wafer probing determines the electrical and optical properties of all LED chips on a wafer by contacting each and every chip with a prober needle. Chip designs without a contact pad on the surface, however, elude wafer probing and while electrical and optical properties can be determined by sample measurements, defective LED chips are distributed randomly over the wafer. Here, advanced data analysis methods provide a new approach to gather defect information from already available non-contact measurements. Photoluminescence measurements, for example, record a brightness image of an LED wafer, where conspicuous brightness values indicate defective chips. To extract these defect information from photoluminescence images, a computer-vision algorithm is required that transforms photoluminescence images into defect maps. In other words, each and every pixel of a photoluminescence image must be classified into a class category via semantic segmentation, where so-called fully-convolutional-network algorithms represent the state-of-the-art method. However, the aforementioned task poses several challenges: on the one hand, each pixel in a photoluminescence image represents an LED chip and thus, pixel-fine output resolution is required. On the other hand, photoluminescence images show a variety of brightness values from wafer to wafer in addition to local areas of differing brightness. Additionally, clusters of defective chips assume various shapes, sizes and brightness gradients and thus, the algorithm must reliably recognise objects at multiple scales. Finally, not all salient brightness values correspond to defective LED chips, requiring the algorithm to distinguish salient brightness values corresponding to measurement artefacts, non-defect structures and defects, respectively.

In this dissertation, a novel fully-convolutional-network architecture was developed that allows the accurate segmentation of defective LED chips in highly variable photoluminescence wafer images. For this purpose, the basic fully-convolutional-network architecture was modified with regard to the given application and advanced architectural concepts were incorporated so as to enable a pixel-fine output resolution and a reliable segmentation of multiple scaled defect structures. Altogether, the developed *dense ASPP Vaughan* architecture achieved a pixel accuracy of 97.5 %, mean pixel accuracy of 96.2 % and defect-class accuracy of 92.0 %, trained on a dataset of 136 input-label pairs and hereby showed that fully-convolutional-network algorithms can be a valuable contribution to data analysis in industrial manufacturing.

**Zusammenfassung**

Leuchtdioden (LEDs) werden heutzutage in einer Vielzahl von Anwendungen verbaut, angefangen bei Standard-LEDs in der Hausbeleuchtung bis hin zu technisch fortgeschrittenen Chip-Designs in Automobilen, Smartwatches und Videowänden. Die Weiterentwicklungen im Chip-Design beeinflussen auch die Testprozesse: Hierbei wird die Durchführung bestimmter Kontaktmessungen durch zunehmend verringerte Chip-Dimensionen entweder erschwert oder ist aufgrund des Chip-Designs unmöglich. Die sogenannte Wafer-Prober-Messung beispielsweise ermittelt die elektrischen und optischen Eigenschaften aller LED-Chips auf einem Wafer, indem jeder einzelne Chip mit einer Messnadel kontaktiert und vermessen wird; Chip-Designs ohne Kontaktpad auf der Oberfläche können daher nicht durch die Wafer-Prober-Messung charakterisiert werden. Während die elektrischen und optischen Chip-Eigenschaften auch mittels Stichprobenmessungen bestimmt werden können, verteilen sich defekte LED-Chips zufällig über die Waferfläche. Fortgeschrittene Datenanalysemethoden ermöglichen hierbei einen neuen Ansatz, Defektinformationen aus bereits vorhandenen, berührungslosen Messungen zu gewinnen. Photolumineszenzmessungen, beispielsweise, erfassen ein Helligkeitsbild des LED-Wafers, in dem auffällige Helligkeitswerte auf defekte LED-Chips hinweisen. Ein Bildverarbeitungsalgorithmus, der diese Defektinformationen aus Photolumineszenzbildern extrahiert und ein Defektabbild erstellt, muss hierzu jeden einzelnen Bildpunkt mittels semantischer Segmentation klassifizieren, eine Technik bei der sogenannte Fully-Convolutional-Netzwerke den Stand der Technik darstellen. Die beschriebene Aufgabe wird jedoch durch mehrere Faktoren erschwert: Einerseits entspricht jeder Bildpunkt eines Photolumineszenzbildes einem LED-Chip, so dass eine bildpunktfeine Auflösung der Netzwerkausgabe notwendig ist. Andererseits weisen Photolumineszenzbilder sowohl stark variierende Helligkeitswerte von Wafer zu Wafer als auch lokal begrenzte Helligkeitsabweichungen auf. Zusätzlich nehmen Defektanhäufungen unterschiedliche Formen, Größen und Helligkeitsgradienten an, weswegen der Algorithmus Objekte verschiedener Abmessungen zuverlässig erkennen können muss. Schlussendlich weisen nicht alle auffälligen Helligkeitswerte auf defekte LED-Chips hin, so dass der Algorithmus in der Lage sein muss zu unterscheiden, ob auffällige Helligkeitswerte mit Messartefakten, defekten LED-Chips oder defektfreien Strukturen korrelieren.

In dieser Dissertation wurde eine neuartige Fully-Convolutional-Netzwerkarchitektur entwickelt, die die akkurate Segmentierung defekter LED-Chips in stark variierenden Photolumineszenzbildern von LED-Wafern ermöglicht. Zu diesem Zweck wurde die klassische Fully-Convolutional-Netzwerkarchitektur hinsichtlich der beschriebenen Anwendung angepasst und fortgeschrittene architektonische Konzepte eingearbeitet, um eine bildpunktfeine Ausgabeauflösung und eine zuverlässige Sementierung verschieden großer

Defektstrukturen umzusetzen. Insgesamt erzielt die entwickelte *dense-ASPP-Vaughan*-Architektur eine Pixelgenauigkeit von 97,5 %, durchschnittliche Pixelgenauigkeit von 96,2 % und eine Defektklassengenauigkeit von 92,0 %, trainiert mit einem Datensatz von 136 Bildern. Hiermit konnte gezeigt werden, dass Fully-Convolutional-Netzwerke eine wertvolle Erweiterung der Datenanalysemethoden sein können, die in der industriellen Fertigung eingesetzt werden.

**Acknowledgements**

More than ten years ago, when I was just about to finish my apprenticeship as a legal secretary and made the plan to catch up on my high-school diploma, I would never have imagined pursuing a PhD one day. Many accompanied me on my long and hilly way and I was lucky enough to meet knowledgeable and supportive people who believed in me and taught me the skills that led me to this most unlikely moment in my life, where I sit at my desk to write the acknowledgements of my PhD thesis. First of all, I would like to thank my supervisor Prof. Dr. Klaus Meyer-Wegener, who thankfully assumed the supervisor role from my late supervisor Prof. Dr. Lothar Frey. Not only did Prof. Meyer-Wegener support me with helpful answers to my many questions and a whole lot of precious suggestions that formed this work, but he also gave me the space to finish this thesis at my own pace and place. Surprised, we noticed each others enthusiasm about language, which resulted in lively discussions about English grammar and unexpected new knowledge on my side. Secondly, I would like to thank my second assessor, Prof. Dr. Björn Eskofier, for his valuable suggestions as well as for his time and effort. My actual way to this thesis started at the Fraunhofer Institute for Integrated Systems and Device Technology in Erlangen, where my then-supervisor Dr. Martin Schellenberger gave me the chance to dive into the world of science, accompanied by wonderful, supporting and jolly colleagues who carried me over many hills. At the institute, I walked my first project-management steps, discovered the world of machine learning and how rewarding a scientific discourse with colleagues can be. What I am most grateful for is the freedom and support Martin gave me to try out new methods and areas, even though he was not always certain they would pan out—as was I. My time at the institute would have surely evolved differently without two prior steps: my work at schodo GmbH and my time at the BioMEMS lab at the university of applied sciences Aschaffenburg, led by Prof. Dr. Christiane Thielemann, who took the time to give me valuable feedback about how to put results down—advise that I still benefit from. Working at schodo GmbH not only financed my studies in the best possible way but also introduced me to my dear friend Dominic Schott, who taught me how to wire an electric cabinet, supported me in uncountable ways and, most notably, showed me that programming is fun! Following all these steps, I am now working at OSRAM Opto Semiconductors, the company I was cooperating with during my time at the Fraunhofer IISB. The decision to join OS was mostly driven by my awesome colleague Dr. Hans Lindberg, who supported me not only with data, explanations and patience but also with his immense knowledge about computer vision and programming. Joining OS in these not so easy times was not always a fun ride, but then again showed me how amazing the people I am working with are—I am really looking forward to the things we will create together! Eventually, I would like to thank all the people in my private sphere, starting with my family: my mother, who taught me perseverance, my father, from whom I inherited my interest in technical stuff and most of all my brother Timo Hoppen, who

is my rock in the stormy sea—thank you, for countless conversations, for listening and talking and giving. During the making of this thesis, I was a horrible friend and I am most grateful for my wonderful friends Eva Seitz, Agnes Knörzer, Carmen Fleckenstein, Melanie Wullaert and Miriam Drescher, who graciously connived my shortcomings and always welcome me back home as if I was never gone. I miss you! Writing in a foreign language tends to be an adventure that might go horribly wrong but thanks to my Scottish language advisor Pete Agnew and his effort in proofreading this thesis I am somewhat assured it was not a completely bad idea—dear Pete, believe me, by now you know more about neural networks than I will ever know about music! Finally, I would like to thank my extraordinary husband Sebastian Stern, who supported me without measure, not only with his strength, cleverness and kindness but also by enduring all the lonely times I put on him just to follow my dream in a distant city or by spending endless hours at my desk. You are the one who always believed in me, stood by me and paved the way—thank you!

# Contents

# List of Figures

# List of Tables

# 1. Introduction & Motivation

In the manufacturing of light-emitting diodes, measurements constitute an inevitable but simultaneously unwanted process step: after all, measurements add no value to the product but enable the monitoring of product and process [84]. Here, the manufacturing of light-emitting diodes (LEDs) is a complex semiconductor-manufacturing process that includes a variety of different measurements, employed for process monitoring, the determination of LED chip properties and the detection of conspicuous or defective LED chips. Based on these measurements, process deviations can be identified early and defective LED chips can be rejected instead of further processed. Among the available measurement methods photoluminescence imaging has several advantages, namely being fast, cost-efficient, non-contact measurements. By irradiating the optical surface of an LED wafer photo-excitation is provoked, ultimately causing the emission of photons. Because only the upper optical layers are excited, the measured brightness values are dissimilar from the brightness an electrically excited LED chip would emit. Additionally, photoluminescence measurements generate images with varying brightness values from wafer to wafer as well as local areas of differing brightness. Hence, photoluminescence measurements are commonly employed for the detection of separation damages rather than a thorough defect detection. For this purpose, the electrical and optical properties of each LED chip are determined by wafer probing, an accurate but cost-intensive contact measurement. The ongoing decrease of LED chip dimensions and the consequential increase of chips per wafer, however, distinctively increase wafer-probing cycle times: with chip-edge lengths less than 100 µm and more than a million chips per wafer, contacting each LED chip for a measurement becomes increasingly time-consuming as well as prone to failure. Furthermore, chip designs without contact pad elude wafer probing completely. Albeit, comparing a wafer-probing-based defect map with photoluminescence images reveals that single defective LED chips as well as clusters of defective chips can be located in photoluminescence images as well.

Therefore, this work examines whether photoluminescence images can be used to detect defective LED chips. Starting from the premise that each pixel in a photoluminescence image corresponds to an LED chip, a computer vision algorithm for defect detection requires the reliable recognition of differing brightness values corresponding to conspicuous LED chips but must ignore varying brightness values caused by measurement artefacts. Additionally, the algorithm must be able to output a classification for each and every pixel in the input image. In recent years, deep-learning algorithms for computer vision are driving major advances in a variety of computer vision tasks such as image classifi-

cation [42, 64, 68, 115, 124, 136], object detection [36, 98, 99, 109] and semantic image segmentation [16, 57, 58, 72, 77, 101, 134, 137, 141]. The main advantage of deep-learning algorithms for computer vision, in comparison to many other data analysis techniques, is their ability to learn shared parameters in form of convolutional filters. These filters slide across the input image and are hence capable to detect data-specific structures that may occur in arbitrary image positions. Because shared parameters are computationally efficient, thousands of task-specific filters can be learned, enabling the distinction between defective LED chips and measurement artefacts. Furthermore, these algorithms obviate the need for hand-crafted features as they learn the suitable filter parameters for the task through backpropagation by themselves, hence covering a wide range of possible features. Fully convolutional networks, introduced by Long et al. [77], represent a specialised deep learning architecture for semantic segmentation. Here, semantic segmentation refers to the task of allocating each image pixel to a fixed set of class categories [28, 113, 126], which can be interpreted as classifying each LED chip as defective or not. Among all methods for pixel-wise prediction, fully convolutional networks were the first to enable end-to-end training as well as the transfer of pre-trained weights and have since been used for a variety of applications [2, 57, 69, 70].

In this dissertation, a novel fully-convolutional-network architecture for the detection of defective LED chips in photoluminescence images is developed, where wafer-probing-derived defect maps are used as pixel-wise labels. Standard network architectures are designed to achieve high accuracies on research datasets with hundreds of thousands of images, which usually depict pattern-rich everyday-life scenes with varying image objects [28, 104]. Wafer images exhibit, in contrast, only one image object with little variation in the defect pattern, compared to street scenes for example, but then again require the accurate classification of each and every LED chip (pixel). Moreover, the ratio between in-spec LED chips and defective LED chips is highly unbalanced and the rare occurrence of certain defect types, such as defect clusters, results in a very small dataset. Finally, obtaining pixel-wise labels constitutes a known hindrance for the compilation of datasets, especially with regard to large images consisting of over hundred thousand pixels. Wafer-probing-based defect maps can be used to generate pixel-wise labels but due to differences in the measurement techniques, image-label discrepancies occur.

In order to address the described challenges, first the application's technical background is described in chapter 2, starting with a sketch of LED manufacturing and a more thorough description of photoluminescence measurements, wafer probing and the composition of the training dataset. Then, chapter 3 expounds the theoretical aspects of fully convolutional networks, starting with the concepts of supervised learning and backpropagation, using the example of neural networks. On this basis, the building blocks of fully convolutional networks are described, namely the downsampling and upsampling path as well as skip connections. Afterwards, techniques to ease network training, such as parameter update optimisers and transfer learning are studied. Based on the theoretical background, chapter 4 studies state-of-the-art fully-convolutional-network-based archi-

tectures developed with respect to common research datasets as well as their application to related topics, such as medical images and industrial data. Due to the special scope of this dissertation, no preliminary research with regard to the analysis of photoluminescence wafer images is known to the author. Based on the acquired insights, a novel fully-convolutional-network architecture and possible extensions are illustrated in chapter 5, where the architecture is designed with regard to the given dataset. Afterwards, chapter 6 analyses the proposed network design and the corresponding hyperparameters by conducting comprehensive experiments: proceeding from the basic design idea, architectural design choices, such as the number of layers, skip connections and residual shortcuts [42, 44], were examined using performance metrics as well as feature visualisation [86, 136]. Studying hyperparameter tuning revealed the interconnection of learning rate, L2-regularisation strength and parameter update methods. Experiments with transfer learning exposed the benefit of a network partly initialised with transferred, pre-trained parameter values, despite the inherently different dataset the parameter-giving network was trained on [127]. Highly unbalanced datasets, such as the dataset at hand, are accompanied by the accuracy paradox [145], where a high prediction accuracy can be achieved by always predicting the label of the majority class. Here, weighted loss calculation was studied so as to equalise the class categories. Analysing prediction images provided conclusions about how image-label mismatches influence network learning and how to further improve prediction accuracy. The resulting network architecture achieved accurate segmentation results for single defective LED chips as well as repeatedly occurring defect structures. The segmentation of rarely occurring, large defect clusters, however, appeared flawed. Therefore, advanced architectural concepts were examined, namely densely connected convolutional blocks and atrous-spatial-pyramid-pooling modules. Here, the implementation of dense blocks enabled the construction of a more condensed network architecture with less feature maps and thus distinctively reduced overfitting. Atrous spatial pyramid pooling layers, on the other hand, increase the network's segmentation accuracy by capturing objects at multiple scales at once. Finally, comparing a manual training and validation dataset split with random cross-validation revealed the benefit of manually splitting small, highly variable datasets and allowed insights into how dataset compilation affects network training. By combining all methods, prediction accuracy could be distinctively increased, showing that fully convolutional networks can be used for the chip-wise detection of defective LED chips in photoluminescence images. The final network architecture and hyperparameters are then illustrated in chapter 7, including network implementation and evaluation. Note that partial results presented in this thesis regarding network design and hyperparameter tuning have been submitted in a paper co-authored with Martin Schellenberger [120] and co-authored with Hans Lindberg and Klaus Meyer-Wegener [119], respectively. Finally, chapter 8 concludes the acquired insights and discusses possible applications in LED manufacturing as well as further research possibilities.

# 2. Data Generation & Preparation

The manufacturing of light-emitting diodes (LEDs) is a complex semiconductor process with many, partly repeating process steps. Throughout the process chain various measurements are employed to monitor the process, determine optical and electrical properties and detect failures. Due to advanced LED concepts, contact measurements, which are commonly used for accurate electrical and optical measurements, get increasingly time-consuming, if feasible at all. Therefore, novel analysis methods must be developed, which are based on non-contact measurements: on the one hand, each LED chip's electrical and optical properties must be determined and on the other hand, defective LED chips must be detected. While LED properties can be derived from sample measurements, defects may stem from several causes and are randomly spread across the wafer. A possible solution provides the analysis of brightness wafer images, generated by a non-contact photoluminescence measurement: when comparing photoluminescence images with wafer-probing-derived defect maps, it becomes apparent that prober defects correlate with conspicuous brightness values. However, photoluminescence images also feature uneven brightness distributions from wafer to wafer as well as local areas of varying brightness next to salient brightness values unrelated to defect structures. Therefore, a possible analysis algorithm must reliably distinguish non-defect structures, functional wafer structures and measurement artefacts from actual defect structures, in addition to accurately segment multiple scaled defect structures, which altogether proves creating a hand-coded defect-detection algorithm difficult. Thus, this work studies fully-convolutional-network algorithms for the detection of defective LED chips in photoluminescence images. Because the training of supervised-learning algorithms requires input-label pairs (see also chapter 3), defect maps derived from wafer probing are used as label images, despite expectable discrepancies. In order to describe the used data, first the basic structure of LEDs and the manufacturing process are elaborated, including photoluminescence measurements and wafer probing. Then, the composition of the training dataset is described.

## 2.1. Light-Emitting-Diode Manufacturing

Light-emitting diodes, commonly abbreviated as LEDs, are semiconductor light sources that radiate light by means of electroluminescence, where electrons in the semiconductor's conduct band recombine with holes in the valence band and as a consequence emit

photons [108]. First discovered by accident in 1907, modern LED chips come in very different designs and dimensions, from micro-LEDs with edge lengths under 100 micrometers up to applications with edge lengths of several millimetres. The basic setup of an LED chip before packaging is sketched in figure 2.1. Here, the dark blue area is composed of an n-doped semiconductor layer that is placed on top of an optically active layer, followed by a p-doped semiconductor layer. The active layer itself is composed of several highly specialised semiconductor layers, where so-called quantum-well layers and barrier layers alternate in order to increase the recombination rate of electrons and holes. The composition of the quantum wells in the active layer determines the emitted wavelength: as an instance, quantum wells consisting of indium gallium aluminium phosphide (InGaAlP) emit photons corresponding to the red and yellow spectral region, where with increasing indium content in the quantum-well layers the emitted wavelength rises. Additional functional layers conduct the electrical current to the n-type and p-type contacts, reflect photons back to the surface or provide isolation. Finally, a comparably thick silicon substrate layer stabilises the LED structure.

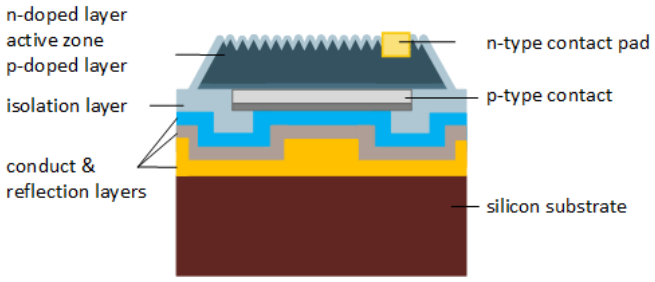



Figure 2.1.: Schematic construction of a light-emitting-diode chip, which is composed of more than a hundred highly specialised layers, including an active zone of light-emitting quantum wells. The rough chip surface further increases light decoupling, while functional layers serve as electrical conductor or reflect photons back to the top.

As depicted in figure 2.2, the manufacturing of LEDs starts with metal organic vapour phase epitaxy (MOVPE), where the different semiconductor layers are deposited on a sapphire wafer substrate [35, 96]. Hereby, the resulting optical properties depend on the settings of the highly complex MOVPE process. Furthermore, parasitic chemical processes reduce the material growth rate and deposit material at the reactor walls, causing dropped down particles on the wafer surface and subsequently failed LED chips [78]. After epitaxy, the resulting properties of the semiconductor layers are determined by various measurements, including particle measurements, photoluminescence spectroscopy and early electrical measurements. After growing the semiconductor layers, additional functional layers are applied using chip technologies, including metallisation, photolithography, chemical etching and grinding, where process steps may be repeated several times [32]. Throughout these process steps, particles and process errors may introduce flaws, such as poor current conductivity, in addition to widening the parameter value distribu-

tion. Eventually, the LED chip's surface is structured so as to optimise light decoupling and finally n-type contact pads are added. Before fully separating all layers by dicing, the optical and electrical properties of each LED chip are determined via wafer probing. Here, each and every chip's n-type contact pad is connected with a prober needle and a thorough test cycle is performed. Moreover, wafer probing incorporates previous measurements so as to refrain from testing chips already known to be defective. As an instance, one process step involves the bonding of two substrates, where particles might cause voids in the bond. The void area, determined by an earlier ultrasonic measurement, is forwarded to the wafer-probing step and affected chips are marked correspondingly. Then, all other chips are measured, based on an electrical and optical test cycle and with regard to a predefined tolerance range. If a reading exceeds a limit value, the test cycle will be terminated and the chip will be marked as failure along with the according defect cause. However, at which test step limits are exceeded does not necessarily trace back to the original defect cause. Hence, the wafer-prober test map delivers an approximate defect cause for flawed chips and exhaustive electrical and optical properties of the other chips.

After wafer probing, the chips are fully separated but stay in formation on a foil substrate to be evaluated for possible separation damages. For this purpose, a fast, non-contact photoluminescence measurement is conducted, which determines the wafer's brightness by measuring the photoluminescence intensity. The measurement process, as depicted in figure 2.3, can be sketched as follows [96]: a light source, whose photon energy is larger than the band gap of the wafer's quantum well's alloy, create electron-hole pairs in the illuminated area. After residing in the conduction band, the electrons recombine with the holes in the valence band to a lower energy state by emitting mostly photons. Due to the discrete energy levels, the emitted photon's energy and thus wavelength theoretically equals the band gap energy of the active zone's material. In practice, however, the wavelength of the emitted photons may be altered by different influences, including properties of the optical layer and package properties. The emitted photoluminescent photons are then captured by a high-resolution camera, where reflected photons of the excitation wavelength have been filtered out before. The resolution of the photoluminescence measurement employed in this work is 50 µm × 50 µm, with a chip size of 250 µm × 250 µm. Thus, each chip's brightness value is an average over 25 photoluminescence measurements, where each LED chip is displayed as one pixel in a photoluminescence image. Depending on the wafer probing and photoluminescence-measurement results, defective LED chips are rejected and chips within the specifications (in-spec) are binned with respect to brightness, wavelength and forward voltage. Depending on the final product, one to several LED chips are packaged and integrated into a module, where a module may contain several hundreds of LED chips. Thus, one undetected defective LED chip may cause the rejection of a multitude of in-spec LED chips.

As described, photoluminescence measurements take place after wafer probing and are already used to detect defective LED chips—however, only defects caused by separation

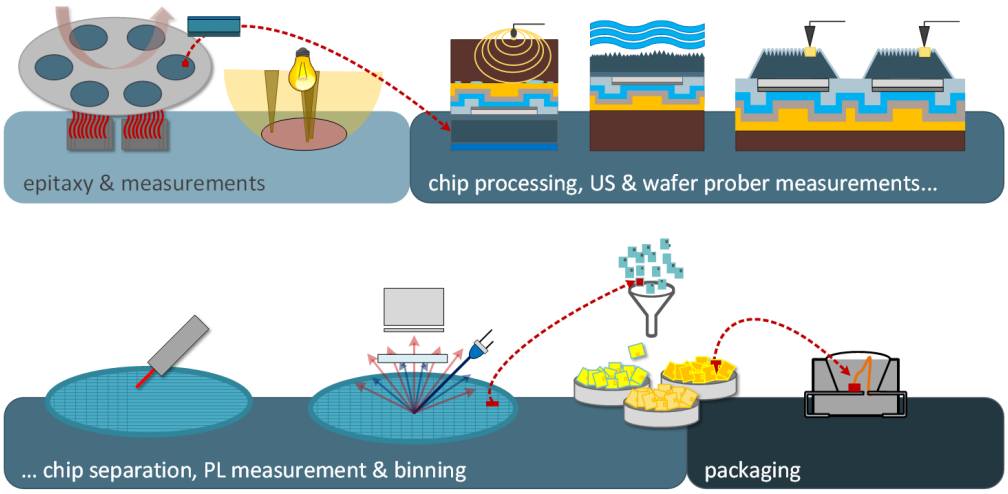



Figure 2.2.: Overview over the LED-manufacturing process, starting with epitaxy. Here, the optical layers are deposited on a sapphire substrate, where epitaxy reactors commonly process several wafers at once. Then, functional layers are applied in multiple process steps, where measurements monitor the process quality. Ultrasonic measurements, for example, determine voids caused by particles in a wafer-bonding-process step. Eventually, the optical layer's surface is structured and separated. In the subsequent wafer-probing measurement, each LED chip's electrical and optical properties are determined, also revealing defective LED chips. After completely separating the chips, a photoluminescence measurement is performed to detect separation damages. Finally, the chips are removed from the wafer structure, binned into their corresponding parameter bin, if not rejected, and forwarded to the packaging process.

damages and only in combination with wafer probing. In this work, the employment of photoluminescence images for the detection of all kinds of defects in a stand-alone application is studied, using measurement data from the running LED-manufacturing process. Preparing real-world measurement data for network training involves several pre-processing steps, including data selection and transformation to a network-readable format, which are elaborated in the following section.

## 2.2. Input and Label-Image Preprocessing

This section describes the data preparation of both, photoluminescence images as well as wafer-probing-derived defect maps. Note that the measurement results are saved as equipment-specific text files, which are pre-processed using the Python programming

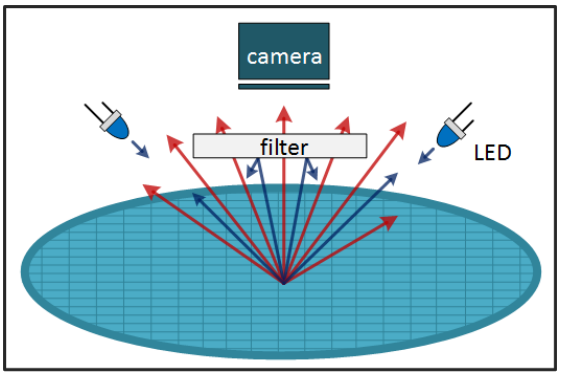



Figure 2.3.: Schematic of a photoluminescence measurement, used to determined photoluminescence intensities, that is an LED chip's brightness. LEDs of the measurement equipment excite the electrons in the quantum wells of the LED wafer's active zone and hereby provoke the emission of photons. Then, a high-resolution camera captures the emitted photons, where a filter is used to filter out reflected photons of the excitation wavelength.

language. Two datasets were sampled from the manufacturing process, covering 145 and 136 InGaAlP wafers, respectively, each with 133,717 chips of size 250 µm × 250 µm. Figure 2.4 displays a photoluminescence image with the corresponding wafer-probing-based label image. As depicted, all defect types and structures are subsumed in one defect class and all remaining chips are assigned to an in-spec class. Areas not corresponding to the wafer as well as alignment markers and optical character recognition (OCR) chips are assigned to a miscellaneous class. Assessing the class distribution of all pixels reveals that the class categories are highly unbalanced, that is the number of defective chips is far less than the number of in-spec chips as well as miscellaneous pixels. When categorising the defect structures into single defective chips and salient defect structures, such as defect clusters and cracks, another imbalance occurs: wafers with salient defect structures, especially large defect clusters, occur rarely in the dataset compared to wafers with only single defective chips and voids. Moreover, while single defects, voids and cracks have a consistent appearance, defect clusters assume very different shapes and sizes, in addition to varying brightness values, and are thus the most difficult to segment accurately (see also figures 2.5 and 8.1). Therefore, only wafers displaying at least one salient defect structure were added to the dataset, where the minority of wafers showed a large defect cluster.

### 2.2.1. Input Images

Photoluminescence-measurement results are not saved as an image, but as a list in a text file, where a brightness value is reported for each LED chip. Every line in the mea-

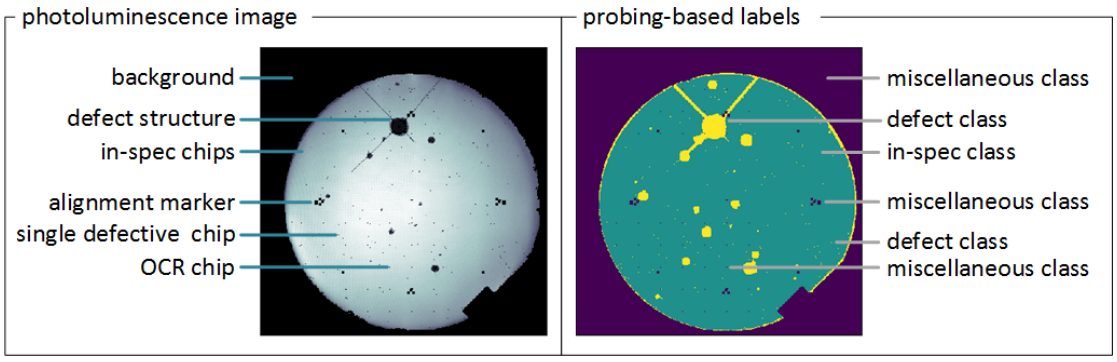



Figure 2.4.: Photoluminescence image and corresponding wafer-probing-based label image. The objective of the network algorithm is to segment the photoluminescence image into three class categories, namely in-spec chips (turquoise), defective chips (yellow) and miscellaneous pixels (blue). Note, that all defect causes are subsumed in one defect class, but varying defect structures (single defective chips, cracks, voids and defect clusters) are differently hard to segment accurately.

surement file contains the chip's x- and y-coordinates in the wafer matrix, the measured photoluminescence intensity over an 8 bit greyscale, additional post-processed brightness values as well as an evaluation mark. In order to create a photoluminescence image, a zero matrix of size $442 \times 440$ is filled with chip values, where applicable. Limit violations are not addressed any further, instead the corresponding brightness values are used as is, so as to provide the network with the unaltered photoluminescence-measurement results. Restricted wafer edge areas, alignment markers and OCR chips are set to zero, in accordance with the wafer-prober-based defect map. Note that the aforementioned voids, which are caused by particles in a bonding-process step, are also visible in photoluminescence images. However, the affected area in photoluminescence images is smaller than the area determined by the ultrasonic measurement, as visible in figure 2.4 and figure 6.17. That is, even though the ultrasonic measurement detects a gap at the void edge, the damage does not affect the photoluminescence property of the LED chip. Experiments in chapter 6 reveal that the network adopts these repeated input-label mismatches: if a salient defect structure resembles a void, then an area larger than the visible defect area is classified as defective. Other defects, such as single defective chips and defect clusters, are segmented within their visible borders. Because the network's interpretation of the actual void area is necessarily flawed, embedding the ultrasonic measurement into the photoluminescence image distinctively increases network accuracy, as shown in chapter 6. For this purpose, two datasets were compiled and used in chapter 6, one dataset with 145 unaltered photoluminescence images and one dataset with 136 photoluminescence images with embedded ultrasonic-defect information. Note that additional input-label mismatches occur due to the fact that the photoluminescence measurement takes place after wafer probing and the subsequent chip-separation step. Separation damages are

therefore not marked as defects in wafer-prober-based defect maps, which may cause wrongful misclassifications. This effect occurs especially at the wafer edge, as shown in chapter 6.

Figure 2.5 depicts four photoluminescence images, which visualise the varying brightness distributions. While some measurements result in evenly distributed brightness values for in-spec chips and darker pixels for defective chips, other wafers display local areas of differing brightness, that is measurement artefacts. As a result, the remaining areas appear less well resolved and the brightness of in-spec chips and defective chips converges. On the other hand, the various functional and defect structures are shown: next to the aforementioned voids, single defective chips, cracks and defect clusters are displayed. While cracks correspond to defective LED chips, film tears, as shown in image b), correspond to less bright but not to defective chips and thus belong to the in-spec class. Note that OCR chips appear as single dark pixels, just as defective LED chips do. However, the results in chapter 6 reveal that the network learns to reliably distinguish OCR chips from single defective chips. Incidentally, OCR chips are not always visible to the naked eye on the images, due to the image resolution. Finally, observing the three depicted defect clusters (images b, c and d) illustrates the range of shapes, sizes and brightness values the defect clusters may assume.

### 2.2.2. Pixel-Wise Labels

In order to train supervised-learning algorithms, input-label pairs must be provided, where the label image is used to calculate the loss between the network's prediction and the corresponding true labels. However, it is not feasible to manually label hundreds of images on a pixel-wise level. As elaborated in the first section, wafer probing provides accurate defect information and thus the measurement aimed to be supplemented is repurposed for labelling. Wafer-probing defect information covers electrical and optical failures in addition to the results of previous measurements, such as the ultrasonic measurement that provides void information. Like the photoluminescence measurement, wafer probing results and limit violations are listed in a text file, where each chip is denoted by its coordinates. To create a defect map, all chips that exceed predefined property limits are classified as defective and all other chips are classified as in-spec. Following the photoluminescence images, non-wafer areas, restricted areas, alignment markers and OCR chips are assigned to a miscellaneous class. Now, each pixel of the label image has a value in the range of $[0, 1, 2]$ (misc./in-spec/defect), as shown in figure 2.4. However, as described in chapter 3, the network algorithm outputs the result of a softmax function, where a prediction vector $\boldsymbol{p}$ is returned for every pixel, consisting of three normalised values that sum to 1. As an instance, consider a pixel that is labelled as defect (2), whereas the network outputs a prediction vector $\boldsymbol{p} = [0.1, 0.3, 0.6]$, that is a probability is calculated for each class category. To adjust the label image correspondingly and enable the loss calculation, one-hot encoding is applied. Here, one-hot

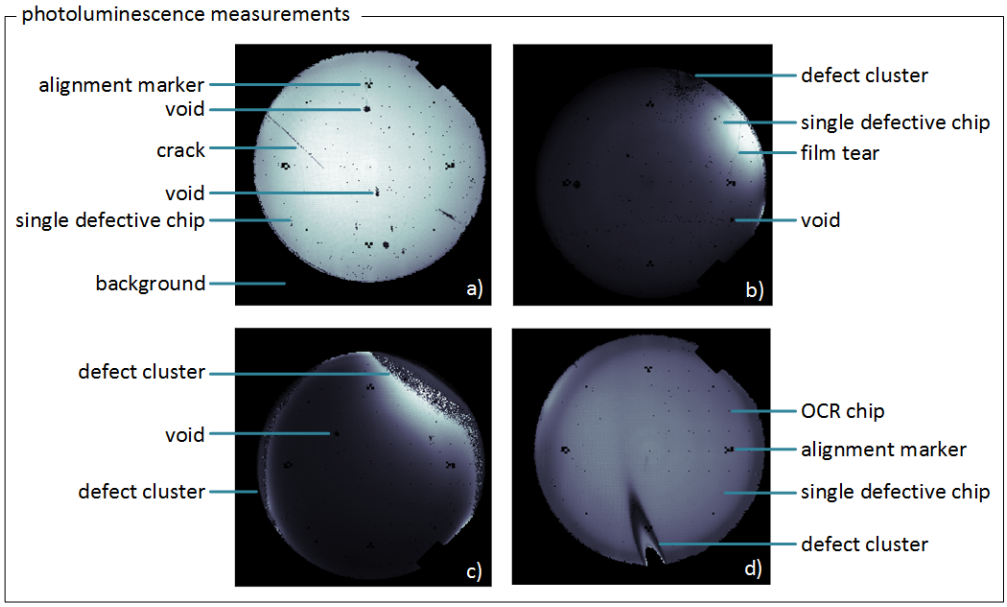



Figure 2.5.: Examples of photoluminescence images, which visualise the varying measurement results. On the one hand, brightness values may vary from measurement to measurement. On the other hand, local areas of differing brightness outshine other areas and as a result the brightness of in-spec chips and defective chips converges in those areas. Furthermore, defect clusters may assume different shapes and sizes in addition to varying brightness gradients. The denoted defect types, namely single defective chip, void, crack and defect cluster, are all subsumed in one defect class but are differently hard for the algorithm to segment accurately. Film tears, as shown in picture b), correspond to less bright but not to defective chips and are thus labelled as in-spec chips, contrary to cracks, which are labelled as defective. Functional structures, such as alignment markers and OCR chips, as well as non-wafer areas are subsumed in a miscellaneous class.

encoding refers to the encoding of categorical integer features using a one-hot encoding scheme [94], where a binary column is created for every category and a sparse matrix is returned. Now, the aforementioned defect class pixel is labelled $\boldsymbol{y} = [0, 0, 1]$, rather than 2. Figure 2.6 illustrates the transformation: a photoluminescence image and the three according label maps are displayed, where the pixels of the class category equal 1 and all other pixels equal 0. The three maps are then stacked, yielding a $442 \times 440 \times 3$ label image. Eventually, the network returns three probability maps and based on the calculated loss between probability maps and label maps the network's parameters are optimised via backpropagation.

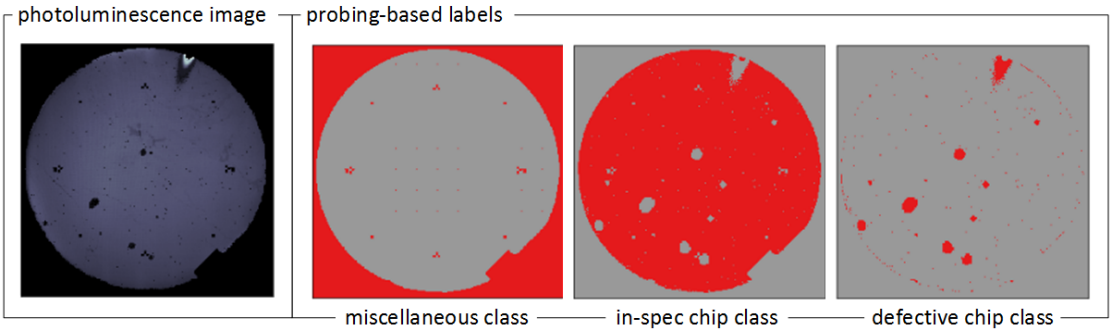



Figure 2.6.: Photoluminescence image and label maps, where each map corresponds to one class category. As an instance, the miscellaneous-class map sets all non-wafer pixels, alignment markers and OCR chips to 1, while all other pixels are set to 0. The three class-category maps are then stacked as a $410 \times 410 \times 3$ label image.

### 2.2.3. Data Selection

As described, for the training and study of the developed fully-convolutional-network architecture two datasets were compiled, one with 145 unaltered photoluminescence images and one with 136 photoluminescence images with embedded defect information from ultrasonic measurements. Compiling small, industrial datasets of measurement images with known input-label mismatches involves the risk of hindering network training through inconclusive training examples. Therefore, wafer images with extensive input-label mismatches, as an instance caused by measurement post-processing or broken wafers, were removed from the datasets. Additionally, only wafers with salient defect structures were selected: even though using all measurements from the running production would distinctively increase the number of training examples in the dataset, experiments in chapter 6 reveal that the given dataset sizes of 145 and 136 images, respectively, are sufficient to achieve an accurate segmentation of common wafer and defect structures. Only the segmentation of previously unknown, large defect cluster shapes in validation images shows flaws, which indicates that the examples in the dataset are not sufficient for the network to generalise from. Therefore, adding inconspicuous wafers to the dataset would not result in an increase in network performance, whereas the experiments indicate that additional examples of large defect clusters will further increase network performance and segmentation accuracy.

### 2.2.4. Data Augmentation

It is generally agreed upon that the success of deep-learning methods can be attributed to large-scale labelled datasets, together with high-capacity models and increased computational power [121]. The ImageNet dataset, for example, covers 1 million images [104], and even smaller datasets for semantic segmentation, like the PASCAL VOC dataset, are composed of over 10,000 images [28]. Using large datasets, where the different objects are depicted from various angles or under various environmental aspects, teaches the network invariance against common variations and thus increases classification accuracy. However, as presented in chapter 4, medical and industrial datasets usually consist of only up to a few hundred images, due to the difficulty of image acquisition. In order to teach the network robustness against variations, despite the small number of training examples, data augmentation is used. Here, the available dataset is increased by introducing little variations to the image, where manipulations that imitate possible real variations cause the most distinct increase in network performance. In case of everyday-life scenes several augmentation methods may be employed, including cropping, filtering, adding noise or pixel dropout. Photoluminescence images, however, are always taken under the same setup and with a steady image quality. Thus, data augmentation was reduced to 45°, 90° and 135° rotations of the images, which tripled the number of training examples and increased network performance, as shown in chapter 6. Additional augmentation techniques, such as brightness manipulations and random elastic deformations [101] did not contribute to a higher network performance.

## 2.3. Summary

In this chapter, the technical background and preparation of the procured dataset were presented, starting with the basic structure of LED chips and a sketch of the manufacturing process, with an emphasis on photoluminescence measurements as well as wafer probing. Then, the transformation of both measurement results into network training data was described, including which measurement information was used and how. Furthermore, the occurrence of input-label mismatches as well as the dataset compilation with regard to data selection and data augmentation were studied. Following this description of data generation and preparation, the next chapter presents the theoretical background of fully-convolutional-network design and hyperparameters, followed by state of the art and related work in chapter 4.

# 3. Fully-Convolutional-Network Theory

In this chapter, the theoretical background of fully convolutional networks is presented, starting with the basic idea of supervised-learning algorithms. Here, the algorithm learns a mapping function by being trained on an input-label dataset, where a regularised loss function is used to evaluate the performance [81]. The second section specifies the aforementioned methods with the example of a neural-network algorithm for multi-class classification and introduces the backpropagation algorithm, used to update the randomly initialised network parameters. Subsequently, methods to ease network training are presented, such as parameter-update optimisation and batch normalisation. Following the general concepts of neural-network algorithms, the specific building blocks of fully convolutional networks are introduced, namely downsampling path, upsampling path and skip connections: first, the downsampling path extracts high-level semantic information from the input image via convolutional pooling blocks, whose elements convolution, activation function and pooling operation are described. Then, the original image dimensions are restored by the upsampling path. For this purpose, different upsampling methods have been proposed, of which transpose convolution and interpolation are presented. In order to increase the output resolution, skip connections fuse the downsampling and the upsampling path. Additionally, residual shortcuts are studied, which further widen the network architecture and thereby ease network training. Furthermore, the concept of transfer learning is described, where the network is initialised with transferred, pre-trained parameters instead of small random values. The implementation of fully convolutional networks in various applications is then illustrated in the following chapter.

## 3.1. Supervised Learning

Wafer brightness images, generated by photoluminescence measurements, contain information which exceed the measurement's intended scope. Therefore, this work studies whether a computer vision algorithm can be employed to output a wafer-defect map based on photoluminescence images. In other words, we want to create an algorithm that performs a mapping $f : X \to Y$, where $X$ represents the space of measurement images and $Y$ is the space of label images [81, 110]. Here, the label set is restricted to $\{0, 1\}$, where 1 corresponds to a pixel that depicts a defective LED-chip and 0 corresponds to all the other pixels. The algorithm is then to predict each pixel's defect

probability $\hat{y} = P(y_i = 1 \mid x_i)$. However, highly variable brightness values and local measurement artefacts, which superimpose the actual measurement, exacerbate the manual specification of a mapping algorithm.

Supervised learning algorithms, on the other hand, exploit the fact that we can easily provide examples for the algorithm to learn the underlying mapping function from. Concretely, we can extract a dataset $\mathcal{D}$ of $m$ independent and identically distributed (i.i.d.) examples, $\mathcal{D} = \{(x_1, y_1), ..., (x_m, y_m)\}$, from the data-generating distribution $P^*$. Here, the samples follow an empirical distribution of the unknown distribution of $P^*$ and with an increasing number of training examples $\mathcal{D}$ approaches the true distribution. Given the dataset, we can train the algorithm by searching the hypothesis space $\mathcal{F}$ of candidate mapping functions $y = f(x)$. Hereby, the class of functions $\mathcal{F}$ we consider is restricted by our choice of learning algorithm and the actual functions $f \in \mathcal{F}$ are given by different values of learned function parameters [60]. The function's performance can then be evaluated by measuring the accordance between the predicted label $\hat{y} = f(x)$ and the true label $y$ with a scalar-valued loss function $L(\hat{y}, y)$. For classification problems, a common choice is cross-entropy loss with

$$L(\hat{y}, y) = \mathbb{E}_{(x,y)\sim P^*}[log\, \tilde{P}(x)], \tag{3.1}$$

where $\tilde{P}$ is the learned distribution. Intuitively, the closer the learned distribution $\tilde{P}$ is to the true distribution $P^*$, the smaller the expected cross-entropy loss. Ideally, we aim to find the mapping function $f^*$ that precisely captures the data-generating distribution $P^*$ and minimises the expected loss:

$$f^* = \arg\min_{f\in\mathcal{F}} \mathbb{E}_{(x,y)\sim P^*} L(f(x), y). \tag{3.2}$$

In practice, however, the empirical distribution $\mathcal{D}$ is rarely comprehensive enough to accurately represent $P^*$. Therefore, we average the loss over all available training data and thereby approximate the expected loss with

$$\tilde{f} \approx \arg\min_{f\in\mathcal{F}} \frac{1}{m} \sum_{i=1}^{m} L(f(x_i), y_i). \tag{3.3}$$

In summary, we procure an exemplary dataset and train the learning algorithm by searching the hypothesis space for a mapping function $\tilde{f}$ that minimises the empirical loss over the available training examples, assuming that $\tilde{f}$ is a well enough proxy for $f^*$. Unfortunately, one way to achieve a training loss of zero is to learn a function that maps

the correct label $y_i$ to each $x_i$ in the training dataset and returns zero for all other $(x, y) \sim P^*$. As a result, the chosen hypothesis overfits to the training data and thus generalises poorly to previously unseen data. Moreover, different dataset samples $\mathcal{D}$ of the true distribution $P^*$ will cause highly variable results. This estimation error, denoted as variance, decreases with increasing training-set size $m$ and increases with the complexity of the hypothesis space $\mathcal{F}$ [60, 66, 68, 110]. We can limit $\mathcal{F}$ by introducing a restriction term to the training objective in equation 3.3:

$$\tilde{f} \approx \arg\min_{f \in \mathcal{F}} \frac{1}{m} \sum_{i=1}^{m} L(f(x_i), y_i) + R(f) \tag{3.4}$$

where $R$ is a scalar-valued function that imposes a preference for simpler mapping functions. Thereby, we take into account that a more complex hypothesis $f \in \mathcal{F}$ has to fit a larger sample size in order to guarantee a small true loss $\mathbb{E}_{(x,y)\sim P^*} L(f(x), y)$ [110]. Neural-network algorithms, which are the focus of this work, are especially prone to overfitting: as an instance, Zhang et al. [138] have shown that neural networks with sufficient capacity can fit randomly labelled images of random pixels with zero training error, demonstrating the memorisation capabilities of network algorithms.

## 3.2. Neural-Network Algorithms

We can specify our previous considerations on supervised-learning algorithms by introducing the example of a classification network. For this purpose, we extend our task to multi-class classification, and design the network to assign one of three discrete class categories $C$ to a wafer image. To do so, we set up a hypothesis class $\mathcal{F}$ of chained, non-linear functions $f(x) = \text{softmax}(\boldsymbol{W}_2 \tanh(\boldsymbol{W}_1^\intercal \boldsymbol{x} + \boldsymbol{b}_1) + \boldsymbol{b}_2)$, which input the vectorised image $\boldsymbol{x} \in \mathbb{R}^n$ and output a vector $\hat{\boldsymbol{y}} \in \mathbb{R}^3$ of class probabilities. Now, the hypothesis space is spanned over the four parameters $(\boldsymbol{W}_1, \boldsymbol{W}_2, \boldsymbol{b}_1, \boldsymbol{b}_2)$, where $\boldsymbol{W}_1$ is a matrix of size $n \times H$, $\boldsymbol{b}_1$ is a vector of size $H$, $\boldsymbol{W}_2$ is a matrix of size $H \times C$ and $\boldsymbol{b}_2$ is a vector of size $C$. Here, $H$ denotes the number of neurons in a network layer; because $H$ is predetermined, it is a so-called hyperparameter and constitutes a possible design choice to impose a hard constraint on the algorithm's capacity. In other words, by keeping the number of neurons in the network small, overfitting is diminished.

The mapping function $f(x)$ first multiplies the image vector $\boldsymbol{x}$ with the weight matrix $\boldsymbol{W}_1$ before adding an offset $\boldsymbol{b}_1$, denoted as bias. Here, the weights allow the network to adjust each pixel's contribution to the result and every neuron learns a different set of weights. Then, the hyperbolic tangent function is applied elementwise, squashing the $H$ values to the interval $[-1, 1]$ and thereby introducing non-linearity. Because we want the algorithm to predict one of three class categories, the second set of parameters, $\boldsymbol{W}_2$ and $\boldsymbol{b}_2$,

covers three neurons. The result of multiplying $\boldsymbol{W}_2$ with the previous layer's output and adding $\boldsymbol{b}_2$ is commonly interpreted as a vector of logits. Hence, by applying the softmax function to each logit $z_i$, with $p_i = e^{z_i} / \sum_{c=1}^{C} e^{z_c}$, we obtain a prediction vector $\boldsymbol{p}$ of normalised values between 0 and 1, which sum to 1. Now, we can compare the inferred probability with the ground-truth, the one-hot encoded label: for example, if the first class is the correct one, then $\boldsymbol{y} = [1, 0, 0]$ and the prediction result may be $\hat{\boldsymbol{y}} = [0.6, 0.3, 0.1]$. Using the aforementioned cross-entropy loss with $L(\hat{y}, y) = -\sum_{c=1}^{C} y_c \log \hat{y}_c$, only the true class contributes to the loss value and incorrect predictions are penalised higher than predictions close to one. In full, we obtain

$$\tilde{f} = \underset{\boldsymbol{W}_1, \boldsymbol{W}_2, \boldsymbol{b}_1, \boldsymbol{b}_2}{\arg\min} \; -\frac{1}{m} \sum_{j=1}^{c} \sum_{i=1}^{m} \boldsymbol{y}_{i,c} \log \Big( \boldsymbol{W}_2 \tanh\big(\boldsymbol{W}_1^{\intercal} \boldsymbol{x}_{i,c} + \boldsymbol{b}_1\big) + \boldsymbol{b}_2 \Big) + \lambda \|w\|_2^2 \tag{3.5}$$

for our exemplary network algorithm, where $\|\boldsymbol{w}\|_2^2 = w_1^2 + w_2^2 ... + w_n^2$ is an L2-regularisation term that penalises high weight values, which can be interpreted as adding a measure of complexity to the loss function. The strength of the regularisation term is adjusted with $\lambda$, which is a hyperparameter itself.

### 3.2.1. Backpropagation

After specifying the supervised-learning architecture, the next step is to determine the candidate function $\tilde{f} \in \mathcal{F}$ that minimises the expected loss. We can formulate this objective as an optimisation problem of the general form $\tilde{\boldsymbol{\theta}} = \arg\min_{\theta} g(\boldsymbol{\theta})$, where $g(\boldsymbol{\theta}) = \frac{1}{m} \sum_{i=1}^{m} L(f_\theta(x_i), y_i) + R(\boldsymbol{\theta})$ and $\boldsymbol{\theta}$ is the parameter vector. Even though theoretically any function can be approximated with a one-layer network, in practice multi-layer networks have been observed to often generalise better, resulting in thousands to millions of network parameters [20, 48, 68]. Hence, determining $\tilde{f}$ by randomly searching the parameter space with stochastic optimisation methods such as hill-climbing would be computationally inefficient. The backpropagation algorithm, on the other hand, is capable of handling large search spaces efficiently by iteratively changing the parameters into the direction of the loss function's steepest descent, as shown in figure 3.1. Here, backpropagation exploits the fact that the partial derivatives of $g$, with respect to $\boldsymbol{\theta}$, are a measure of the loss function's steepness. In other words, the gradient $\nabla_\theta g$, which is a vector of the partial derivatives, gives us the slope of the cost function along the parameters' dimensions. Thus, by iteratively determining $\nabla_\theta g$ and updating $\boldsymbol{\theta}$ we can search for $\tilde{\boldsymbol{\theta}}$.

In order to train the network, we set up a cycle of forward and backward propagation: first, we initialise the parameter vector $\boldsymbol{\theta}$ with small random values, drawn from a Gaussian distribution with a standard deviation of $\sqrt{2/N}$, where $N$ denotes the number of

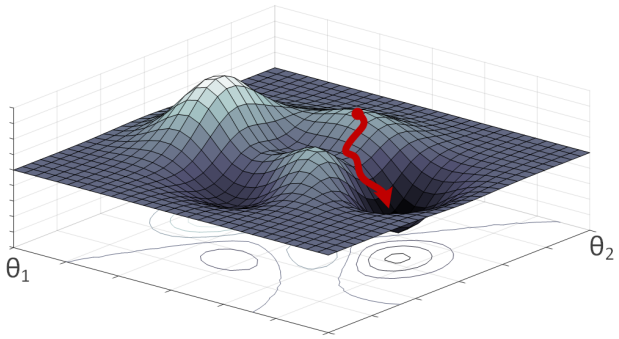



Figure 3.1.: Exemplary loss function $L(\boldsymbol{\theta})$ with respect to the parameters $\theta_0$ and $\theta_1$. The red arrow starts at a possible initial function value, given by randomly initialised parameters. Updating the parameters with backpropagation, that is by following the direction of steepest descent, iteratively minimises the loss-function value until the algorithm converges to a minimum.

incoming nodes [43]. Then, we propagate a batch of data $\{(x_i, y_i)\}_{i=1}^{m}$ through the algorithm (using the randomly initialised parameters) to yield a prediction and determine the corresponding loss $g$. Based on the loss function we can now calculate the gradient with respect to the parameters, using the chain rule of derivation. Consider, as an example, the simplified inner network function $z = \tanh(\theta^\intercal x)$, which can be split into an outer and an inner part, namely: $v = \tanh(u)$ and $u = \theta^\intercal x$. In order to calculate the partial derivative $\frac{\partial z}{\partial \theta}$ we first determine the partial derivatives of the intermediate terms, starting with the outer term, $v' = \frac{\partial z}{\partial v} = 1 - tanh^2(u)$ and followed by the inner term $u' = \frac{\partial v}{\partial u} = x$. Then, we can calculate the partial derivative of $z$ with respect to $\theta$ by multiplying the intermediate terms: $\frac{\partial z}{\partial \theta} = \frac{\partial z}{\partial v}\frac{\partial v}{\partial u}$. Hence, proceeding from the loss function and propagating the partial derivatives of the intermediate terms back through the network, as shown in figure 3.2, allows us to determine the gradient $\nabla_\theta g$ and update the parameters, with $\boldsymbol{\theta'} = \boldsymbol{\theta} - \eta \nabla g(\boldsymbol{\theta})$. Here, $\eta$ denotes the learning rate, a hyperparameter used to adjust the step size of the parameter update. Because the loss function depends on the unknown distribution of the training examples, the learning rate is an empirically determined value that can range from 0.1 to $1 * 10^{-14}$ and has a distinctive influence on the training result [6]. Using a learning rate that is too high yields parameter updates that overleap the function's minimum instead of converging to it, whereas a very small learning rate causes unreasonably long training durations. A common way to reduce training time is to start with a higher learning rate, which is then gradually decayed with ongoing training duration, with $\eta = \eta * k^{\frac{s}{ks}}$, where $k$ is the decay rate, $s$ is the current training iteration (step) and $ks$ is the decay step. The influence of different learning-rate values and decay rates is studied in chapter 6, including the mutual interference of learning rate and the aforementioned L2 regularisation.

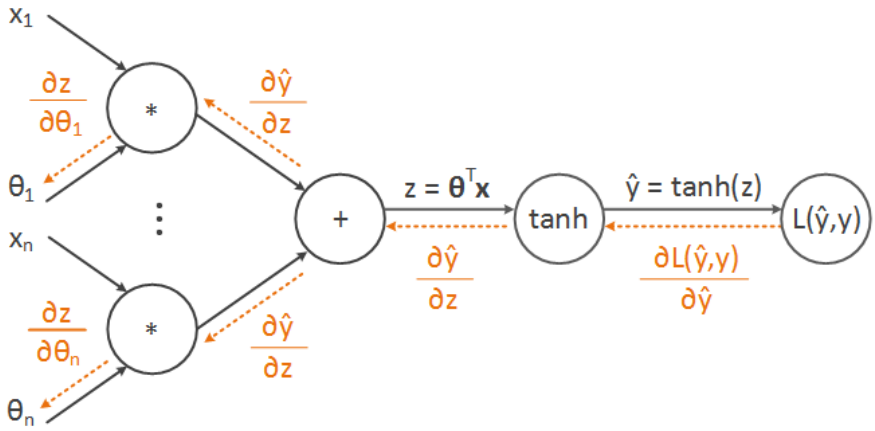



Figure 3.2.: Neural network forward (black) and backward (orange) propagation. The algorithm first calculates the solution function $\hat{y} = \tanh(\boldsymbol{\theta}^\intercal \boldsymbol{w})$ and the according loss $L(\hat{y}, y)$ with respect to each datapoint. Then, the partial derivative of the loss with respect to each parameter is calculated and propagated backwards. Adapted from Fei-Fei et al. [30]

Writing the parameter update equation in its component form, with $\theta_i' = \theta_i - \eta \frac{\partial g(\theta)}{\partial \theta_i}$, elucidates that gradient descent alters each parameter separately. Hence, each parameter learns its own value depending on the training data, making backpropagation an efficient way to optimise the network. In practice, training datasets can cover thousands to millions of images [28, 73, 104], which is why the gradient is commonly not determined based on the whole dataset but is estimated on a subset (minibatch), a method denoted as stochastic gradient descent [8]. Smith and Le [116] have shown that the optimal minibatch size $m'$ is proportional to the size of the training set as well as the learning rate, with $m' \propto \eta m$. That is, when increasing the minibatch size the learning rate should be increased accordingly, where small, homogeneous training sets benefit more from smaller minibatch sizes [63]. Furthermore, by using various noisy estimates of the gradient stationary points in the loss function, such as saddle points and local minima, can be avoided and the parameters are driven to broader, better generalising minima [81, 93].

### 3.2.2. Optimisation

Commonly, neural-network algorithms are multi-variable optimisation problems with various network layers and millions of parameters [12]. Due to the nature of the backpropagation algorithm, layers deep in the network will receive larger parameter updates than shallow layers, a problem denoted as vanishing gradients [3]. In addition, the direction of steepest descent is orthogonal to the contour lines of the loss function $g(\boldsymbol{\theta})$;

therefore, large parameter updates may cause oscillations across the loss function's high curvature areas, reversing previous updates. Momentum-based learning aims to address these shortcomings by adding exponential smoothing to the update vector, with $\boldsymbol{v}_t = \beta \boldsymbol{v}_{t-1} + \eta \nabla_\theta g(\boldsymbol{\theta})$, where $\beta \in (0, 1)$ is the so-called friction or momentum parameter, which adjusts the contribution of previous parameter updates and is usually set to 0.9 [102, 122, 123]. The parameter update rule is then changed to $\boldsymbol{\theta} = \boldsymbol{\theta} - \boldsymbol{v}_t$. By using momentum, oscillations are cancelled out to the benefit of small updates in a consistent direction, accelerating network training comparable to a ball gaining momentum as it rolls downhill. Just like a ball, momentum-based learning tends to slightly overshoot but nevertheless achieves better results than momentum-free update rules [3]. Another way to optimise the learning process is to address the layer-specific partial-derivative magnitudes directly by using individual adaptive learning rates for each parameter. To do so, the AdaGrad (Adaptive Gradient) algorithm separately accumulates the squared partial derivatives to then adjust each parameter's learning rate accordingly [25], yielding

$$\begin{aligned} v_{t,i} &= v_{t-1,i} + (\Delta\theta_i)^2 \\ \theta_i &= \theta_i - \frac{\eta}{\sqrt{v_{t,i} + \epsilon}} \, \Delta\theta_i, \end{aligned} \tag{3.6}$$

where $\epsilon$ is added to avoid division by zero and is usually in the order of $10^{-8}$. Individually adapted learning rates increase the robustness of stochastic gradient descent by updating frequent parameters with smaller steps and infrequent parameters with larger steps [23]. However, by dividing the learning rate $\eta$ by the accumulated partial derivatives $v_{t-1,i}$ over the course of network training may prematurely slow down learning and prevent convergence. This drawback can be dissolved by using an exponentially decaying average of the squared gradients instead, as does the RMSProp algorithm:

$$\begin{aligned} v_{t,i} &= \rho \, v_{t-1,i} + (1 - \rho) \, (\Delta\theta_i)^2 \\ \theta_i &= \theta_i - \frac{\eta}{\sqrt{v_{t,i} + \epsilon}} \, \Delta\theta_i, \end{aligned} \tag{3.7}$$

where $\rho$ is a decay factor, which is usually set to 0.9 [46]. Now, the learning rate is divided by the exponentially decaying average before calculating the parameter update. As a result, the influence of older gradients decays exponentially with time, preventing a premature stagnating of the learning process. But because the running estimate of $v_{t-1,i}$ is initialised to zeros, the RMSProp algorithm is biased during the initial steps. Therefore, the Adam (Adaptive Moment Estimation) algorithm calculates the exponentially decaying average based on bias-corrected estimates [59]. Additionally, Adam also incorporates a momentum term to the update rule, yielding $m_{t,i} = \beta_1 \, m_{t-1,i} + (1 - \beta_1) \Delta\theta_i$ as

an estimate of the first-order moment and $v_{t,i} = \beta_2\, v_{t-1,i} + (1-\beta_2)\,(\Delta\theta_i)^2$ as an estimate of the uncentered variance, the second-order moment. Note, that both terms use different decay parameters, where Kingma and Ba [59] suggest $\beta_1 = 0.9$ and $\beta_2 = 0.999$. In order to counteract the biases, introduced by initialising $m_{t,i}$ and $v_{t,i}$ with zero vectors, Adam employs bias-corrected moments, yielding $\hat{m}_{t,i} = \frac{m_{t,i}}{(1-\beta_1^t)}$ and $\hat{v}_{t,i} = \frac{v_{t,i}}{(1-\beta_2^t)}$. The parameter update is then given by

$$\theta_i = \theta_i - \frac{\eta\beta}{\sqrt{\hat{v}_{t,i} + \epsilon}}\,\hat{m}_{t,i}. \tag{3.8}$$

In summary, the Adam update rule combines the improved gradient calculation of the Momentum method with the individually adapted learning rate of RMSProp and is therefore a common first choice as optimisation method [39]. Ultimately, the choice of optimisation algorithm depends on the distribution of the dataset as well as the other hyperparameters, as demonstrated in chapter 6. Here, the aforementioned optimisation methods have been studied in combination with two different learning rates, revealing that for the employed dataset RMSProp and Adam perform comparably well, whereas other methods underperform.

### 3.2.3. Batch Normalisation

Since deep neural networks are notoriously difficult to train [37, 122] several leverage points have been developed to ease network training. Next to regularisation methods and optimised parameter updates it is common to employ so-called batch-normalisation layers, introduced by Ioffe and Szegedy [53]. The underlying assumption here is to reduce the internal covariate shift of the data while passing through the network: optimising the parameter values of a layer during network training may effect the output statistics of this layer and hence result in a changed distribution of the subsequent layer's input. Batch normalisation addresses this effect by adding a learned normalisation calculation prior to the activation function of each hidden layer, with $f(\boldsymbol{x}) = \phi(\mathrm{BN}(\boldsymbol{W}^\intercal \boldsymbol{x}))$, where BN is batch normalisation and $\phi$ denotes an activation function. Note, that the bias vector is omitted since its effect would be cancelled by the subsequent mean subtraction of the batch-normalisation layer. First, the mean and variance of the current minibatch $\mathcal{B} = \{x_i, ..., x_m\}$ are determined with $\mu_\mathcal{B} = \frac{1}{m}\sum_{i=1}^{m} x_i$ and $\sigma_\mathcal{B}^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_\mathcal{B})^2$, before normalising each datapoint, with

$$\hat{x}_i = \frac{x_i - \mu_\mathcal{B}}{\sqrt{\sigma_\mathcal{B}^2 + \epsilon}}. \tag{3.9}$$

However, normalising each input of a sigmoid activation function, as an instance, would constrain them to the function's linear regime and thus restrict the network's representation power. Therefore, an additional scale and shift operation is introduced with learned parameters: $y_i = \gamma\hat{x}_i + \beta$, where $\gamma$ and $\beta$ enable the network to recover the original activations if suitable. Now, a layer's input distribution is no longer affected by parameter changes of previous layers but is determined by the batch normalisation parameters $\gamma$ and $\beta$. Therefore, when using batch normalisation, the network performance depends less on careful initialisation and hyperparameter tuning, hence easing network training. While inference, inputs are not normalised based on the minibatch statistics; instead, the moving average of the entire population is used. For this purpose, mean and variance in the batch normalisation equation are replaced by their constant estimates $\mathrm{E}[x]$ and $\mathrm{V}[x]$, respectively, turning the normalisation into a linear transformation.

To sum up, supervised learning describes a group of algorithms that are trained on a dataset of input-label examples to learn the underlying mapping function. For this purpose, a hypothesis space of candidate mapping functions, given by the network architecture, is searched by an optimisation algorithm for the candidate function of minimum loss. Here, the common choice of optimisation algorithm is backpropagation, a method that determines the partial derivatives of the network's loss with respect to each parameter. While training, each parameter is iteratively adjusted into the direction of steepest descent, based on the partial derivatives propagated backwards through the network by means of the chain rule of derivation. In order to prevent overfitting, a regularisation term is added to the loss function, which penalises high weight values and as a result imposes a preference for simpler mapping functions. Enhanced optimisation methods, such as the Adam algorithm, improve the basic stochastic-gradient-descent implementation of backpropagation by adding momentum to the parameter update as well as individually adapted learning rates for each parameter. Finally, batch-normalisation layers remove the statistical dependency between layers by enabling the network to scale and shift each layer's input as suitable. The combination of the aforementioned methods reduces training time by enabling a faster convergence to the loss function's minimum and improves the network's performance by mitigating its dependence on careful parameter initialisation and hyperparameter tuning.

## 3.3. Fully Convolutional Networks

The previous section introduced the theoretical background of supervised-learning algorithms—with an emphasis on neural networks—and several methods to optimise network training. Convolutional and fully convolutional networks, respectively, are network algorithms with an architecture specialised for computer vision [33, 34, 64, 67, 103]. Here, convolutional layers are employed for image analysis, which input not the vectorised image but slide a small kernel across the image in its original shape. Thereby,

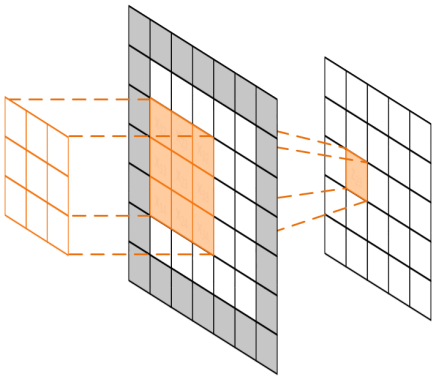


Figure 3.3.: Convolution of an image (middle) with a $3 \times 3$ kernel (left). The kernel slides across the image and at each position multiplies the kernel weight with the corresponding pixel value. The sum of the products plus bias is then the neuron's output at the position of the centre pixel, creating a new image which is denoted as feature map. Because convolution decreases image dimensions, the image is padded with zeros (depicted in grey) so as to keep image dimensions stable.

the implicit structure of images is exploited: on the one hand, images are composed of repeated patterns, changes in illumination or geometrical patterns for example, and on the other hand, pixels in close proximity to each other are highly correlated [7]. By means of the learned kernel parameters local image features are extracted to create a new representation of the image, called feature map. As shown in figure 3.3, a feature map is computed by calculating the weighted sum of the covered pixels at each kernel position, with

$$z_{i',j',f'} = \phi\left(b_{f'} + \sum_{i=1}^{H_f}\sum_{j=1}^{W_f} x_{i'+i-1,j'+j-1,f} w_{ijff'}\right), \tag{3.10}$$

where $z_{i',j',f'}$ is the convolution operation's output located at the kernel centre, $x_{i'+i-1,j'+j-1,f}$ denotes the input image pixel, $w_{ijff'}$ is the kernel weight and $b_{f'}$ is the bias. With regard to convolutional layers, the bias is a means to adjust the feature map's brightness. The kernel's height and width are given by $H$ and $W$, respectively, where typical kernel dimensions range between $1 \times 1$ and $11 \times 11$ and hence, the number of neurons per kernel is reduced to $H \times W$. As will be expanded upon shortly, each convolutional layer consists of several kernels and may input feature maps of a previous layer. Thus, we denote the input map index with $f$ and the newly created feature map index with $f'$.

Due to consistent and efficiently to compute gradients, hidden convolutional layers typically employ rectified linear units (ReLU) as activation function, with $\phi(z) = \max(0, z)$ [56, 83]. Because the ReL unit derives to $f'(z) = 0$ if $z < 0$ it may cause so-called dead neurons [3, 30]. As an instance, using a high learning rate to update the parameters may result in parameter values that cause zero gradients irrespective of the input, whereupon the neuron no longer contributes in distinguishing between different input instances. However, Glorot et al. [38] have shown that sparse representations, caused by the ReL unit's hard threshold, can be beneficial by leading to information disentanglement and increased linear separability. Alternative activation functions address the ReL unit's hard threshold: as an instance, the leaky ReLU adds a small slope to the negative part, with $\phi(z) = \max(\alpha z, z)$, where typically $\alpha \approx 0.2$ [79], whereas the softplus unit smoothly approximates the ReLU function, with $\phi(z) = \log(\mathrm{e}^z + 1)$. The experiments in chapter 6 reveal that for the given dataset and network architecture ReL units yield the highest performance.

To illustrate the advantage of convolutional layers, consider a fully-connected layer. Here, each pixel of the vectorised image is connected to every neuron of the first hidden layer. As an instance, a network that inputs a $28 \times 28$ greyscale image into a first hidden layer with 100 neurons employs 78,500 parameters for just one layer, in comparison to 10 parameters for a $3 \times 3$ kernel plus bias. Furthermore, vectorising the image discards spatial information and thus slight translations of the image object would require the network to learn a new set of parameters to recognise the same object. By sliding a kernel across the image and thus sharing kernel parameters between pixels, equivariance to translations is implemented and one kernel can be used to extract the same feature from all image positions. Because images are a combination of various patterns, convolutional layers consist of a variety of convolutional operations. Each operation learns its own kernel parameters and thus creates an individual feature map. Additionally, a cascade of convolutional layers is set up, that is the created feature maps are fed to another set of convolutional operations. Thereby, layers deep in the network input highly preprocessed feature maps. However, even though convolutional layers distinctively decrease the number of parameters, storing the feature maps consumes computational resources. To reduce memory consumption and increase the number of feature maps, so-called pooling layers are introduced, which calculate the summary statistics of subregions and hence decrease image dimensions [9, 144]. A common pooling operation is maxpooling, where a kernel slides across the image and outputs the highest value of each window, yielding an output size $o$, with

$$o = \lfloor \frac{i_H - k_H}{s} + 1 \rfloor \times \lfloor \frac{i_W - k_W}{s} + 1 \rfloor. \tag{3.11}$$

Here, $i$ denotes the image dimensions, $k$ the kernel dimensions and $s$ is the stride, that is the number of pixels the kernel is moved along in one step. Max pooling implements invariance to small translations of the input: even if the input is translated by a small

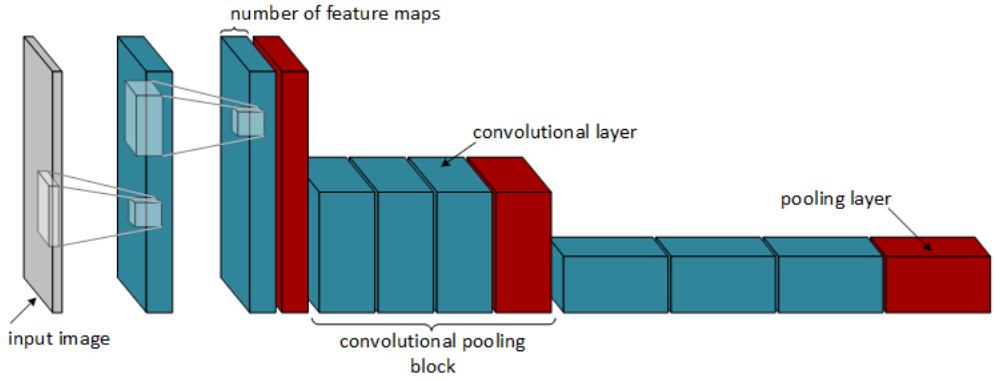



Figure 3.4.: Bi-pyramid of convolutional pooling blocks. A convolutional pooling block usually consist of two to three convolutional layers, followed by a pooling layer that downsamples feature-map dimensions. On the one hand, subsampling operations reduce the memory consumption and thus enable a larger number of subsequent kernels and feature maps, respectively. On the other hand, with every downsampling operation each neuron's receptive field is increased such that neurons deep in the network can be indirectly connected to most of the input image, despite distinctively smaller kernel sizes.

amount, such as a slightly rotated wafer, most of the inputs do not change their value. Jarrett et al. [56] and Saxe et al. [107] have shown that randomly initialised convolutional pooling architectures with ReLU activation functions can yield surprisingly good classification results because pooling functions are inherently translation invariant and hence frequency selective. Figure 3.4 visualises the described bi-pyriamid of convolutional layers, where the spatial resolution of the feature maps is progressively decreased while the number of feature maps—and accordingly the richness of representation—is increased. The specific architectural setup of a network depends on the data, but as depicted, it is common to combine two to three convolutional layers before pooling, where the number of kernels per layer ranges between 64 and 1024 [42, 64, 68, 115, 124, 136].

In sum, a typical convolutional layer inputs either the input image or the output of a previous layer on which it applies several parallel convolutions to create a set of feature maps. This setup is repeated two to three times until feature-map dimensions are reduced with a pooling operation and fed to the next convolutional pooling block. The cascade of convolutional layers enables the detection of high-order features: while the first layer extracts elementary visual features of each neuron's local receptive field in the input image, the subsequent layers combine the extracted features of previous representations. Here, the receptive field denotes the image area a neuron "sees", which corresponds to the kernel dimension in the first layer. Due to the layer connections and enhanced by downsampling operations, a neuron's receptive field increases with increasing network

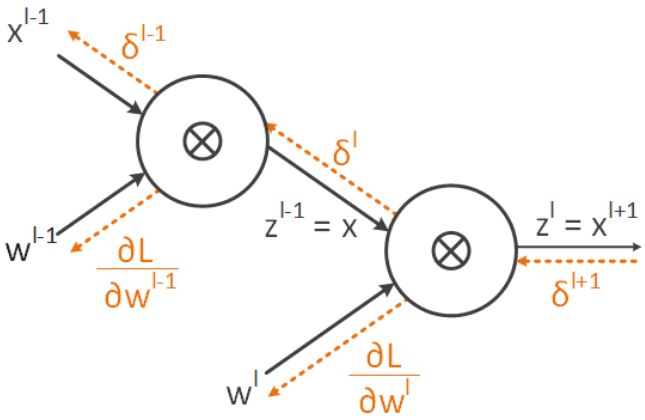



Figure 3.5.: Snippet of the computational graph with two convolutions. The gradient of the loss $L$ with respect to the weights $w$ is used to update the kernel weights, whereas the loss gradient with respect to the input feature-map pixels $z$ is used to propagate the gradient back through the network through each feature map. Adapted from Fei-Fei et al. [30]

depth [39]. Therefore, neurons very deep in a network can be indirectly connected to most of the input image.

### 3.3.1. Backpropagation

Just like fully-connected layers, (fully) convolutional networks are trained with backpropagation, where the kernel parameters are initialised randomly before updating them based on the loss gradient. Figure 3.5 shows a snippet of the computational backpropagation graph depicting two succeeding convolutional operations, where $\boldsymbol{X}$ is the input feature map and $\boldsymbol{Z}$ is the resulting feature map as well as input to the subsequent operation [22, 30]. As illustrated, the backward pass cleaves in two: one path updates the kernel weights and a second path propagates the gradients backwards through the network. The backpropagation is implemented by calculating the loss gradient with respect to the feature-map pixels $\frac{\partial L}{\partial z}$, which is denoted as $\boldsymbol{\delta}$. Using the delta map, we can calculate the weight update employing the chain rule with

$$\begin{aligned}\frac{\partial L}{\partial w_{ijff'}} &= \sum_{i'j'f'} \delta^{l+1}_{i'j'f'} \frac{\partial z_{i'j'f'}}{\partial w_{ijff'}} \\ &= \sum_{i'j'f'} \delta^{l+1}_{i'j'f'} x_{i'+i-1,j'+j-1,f}. \end{aligned} \tag{3.12}$$

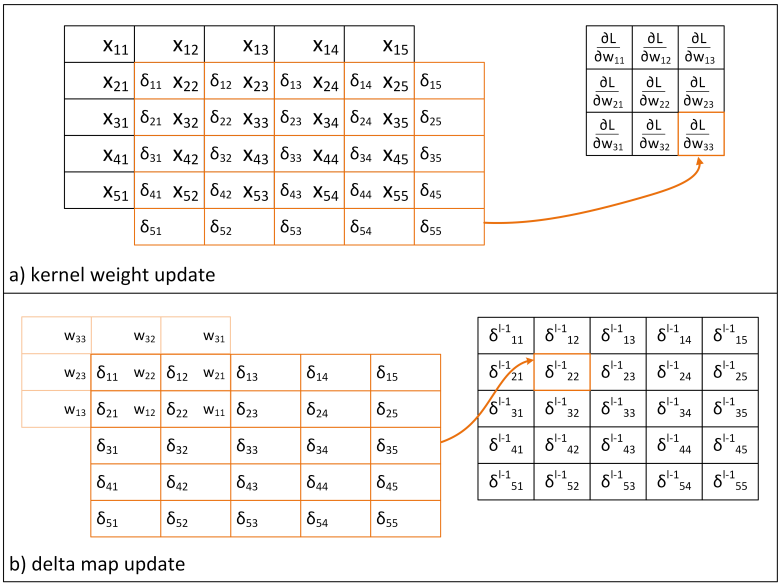



Figure 3.6.: Calculation of the kernel weight and delta-map updates. (a) To update the kernel weights with gradient descent, the loss gradient with respect to the parameters is calculated by cross correlating the input map with the map of the backpropagated loss gradients. The larger the influence of a weight on the resulting loss the larger the update. (b) To backpropagate the loss gradient through the network, the loss gradient with respect to the output $\boldsymbol{Z}$, denoted as $\boldsymbol{\delta}^{l-1}$, is calculated by convolving the weight kernel with the delta map of backpropagated loss gradients. The resulting $\boldsymbol{\delta}^{l-1}$ values are then passed backwards through each element or pixel of $\boldsymbol{Z}^{l-1}$.

Note, that the weight update is determined by means of cross-correlation between input feature map and delta map, as shown in figure 3.6 a). The delta map $\boldsymbol{\delta}^{l-1}$, which propagates the gradient backwards to the previous layer $l-1$, is calculated employing convolution, with

$$\delta^{l-1}_{ijff'} = \sum_{i'j'f'} \delta^{l}_{i'j'f'} \frac{\partial z_{i'j'f'}}{\partial x_{ijf}} \tag{3.13}$$

$$= \sum_{i'j'f'} \delta^{l}_{i'j'f'} w_{i-i'+1,j-j'+1,f}, \tag{3.14}$$

as shown in figure 3.6 b). As described, after convolution of the input map a rectifying non-linearity is applied, with $f(z) = \max(0, z)$, which derives to $f'(z) = 0$ if $z < 0$ and $f'(z) = 1$ if $z > 0$. Because the derivative of the ReLU operation is not defined for $z = 0$,

built-in functions of neural-network libraries commonly set $f(z) = 0$ for $z = 0$. Reducing feature-map dimensions with pooling layers does not include learnable parameters. Thus, the gradients are passed backwards through the layer unchanged. In case of maxpooling, a gradient is passed through the unit with the maximum argument only and set zero for the remaining ones.

Altogether, a convolutional pooling block commonly covers two to three convolutional layers that apply the consecutive operations convolution, batch normalisation and activation function to the incoming feature maps, followed by a subsampling operation that reduces the image dimensions. In convolutional neural networks for image classification [42, 68, 115, 124, 136] sequences of convolutional pooling blocks are employed to extract semantic features from input images: through downsampling, the spatial information are compressed and eventually fed into a sequence of fully-connected layers, which serve as multi-class classifier. Fully convolutional networks for semantic segmentation, as introduced by Long et al. [77], are based on convolutional networks but omit all fully-connected layers. Instead, the image is passed through a cascade of downsampling and upsampling layers, thereby enabling pixel-wise classifications. For this purpose, it would seem sensible to omit downsampling and thus not reduce image dimensions. However, downsampling serves several causes, such as implementing invariance to small translations, enabling an increase of representations by setting up a bi-pyramid and increasing the neuron's receptive field. Therefore, fully convolutional networks first downsample the image before subsequently upsampling the feature maps to retrieve spatial information. The following section will introduce the upsampling path of a fully convolutional network, as counterpart to the downsampling part which was described in this section.

### 3.3.2. Transpose Convolution

While convolutional neural networks are employed to detect what an image shows, fully convolutional networks must additionally retrieve where the information is located. For this purpose, the network is set up of two parts: a downsampling part of convolutional pooling blocks, which compute coarse, semantic information and an upsampling path, which retrieves fine appearance information [77]. In order to restore the spatial dimensions of the input image, Long et al. [77] employ transposed convolution (also denoted as fractionally strided convolution or deconvolution). For the sake of argument, first consider the 1D transposed convolution of a vector, as shown in figure 3.7, a) [26, 112]. Here, one input pixel of the vector $\boldsymbol{x}$ is mapped to an enlarged output area in vector $\boldsymbol{y}$ using a $1 \times 4$ kernel. The transposed convolution is represented as sparse matrix, where each column equals the according value of $\boldsymbol{x}$ and non-zero elements equal the kernel weights $\boldsymbol{w}$. As depicted, the output size can be increased by a stride of 2 (or more) and adjusted with cropping. Accordingly, grey pixels in the matrix represent multiplication with zero and grey pixels in $\boldsymbol{y}$ are cropped areas. Note that transposing the sparse matrix would result in convolution. Figure 3.7, b) shows a visualisation of a computationally

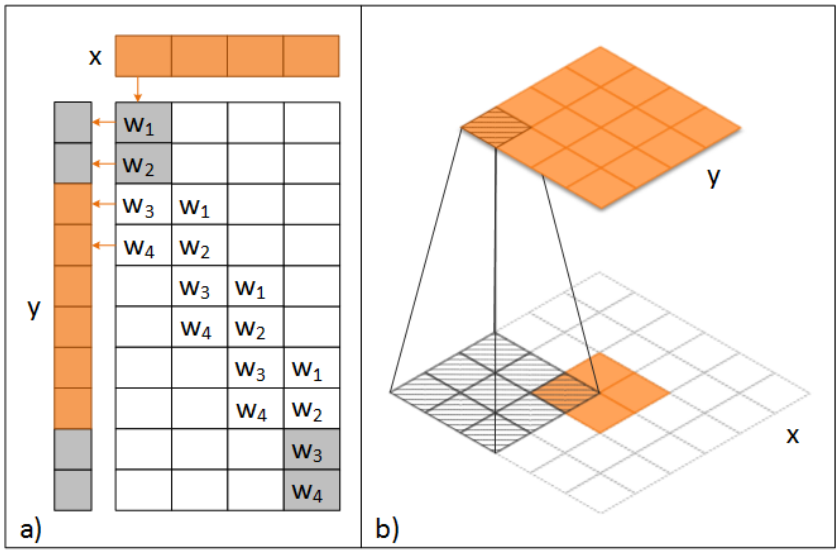



Figure 3.7.: Visualisation of transpose convolution operations. (a) 1D cropped transposed convolution with stride 2 of a signal $\boldsymbol{x}$ by a $4 \times 1$ filter to obtain a signal $\boldsymbol{y}$. The transpose convolution is represented as sparse matrix, where non-zero elements equal the kernel weights $\boldsymbol{w}$, grey pixels in the matrix represent multiplication with zero and grey pixels in $\boldsymbol{y}$ represent cropping. Each input pixel is multiplied with all kernel weights, upsampling the input vector dimensions. Adapted from Shi et al. [112]. (b) 2D transposed convolution: the kernel $\boldsymbol{W}$ is slid across the input matrix $\boldsymbol{X}$ to create a new, enlarged representation $\boldsymbol{Y}$. Adapted from Dumoulin and Visin [26].

less efficient, direct 2D transposed convolution of a $2 \times 2$ matrix $\boldsymbol{X}$, where a kernel $\boldsymbol{W}$ is slid across the input image to create a new, enlarged representation $\boldsymbol{Y}$. Contrary to convolution, a single image pixel is associated to several output pixels, implementing a one-to-many relationship and thus increasing image dimensions.

Transposed convolution allows the network to restore the spatial dimensions of low resolution, high-level representations by means of learnable parameters. However, transposed convolution is prone to cause checkerboard artefacts in the upsampled image, restricting filter options and thus sacrificing model capacity. An artefacts avoiding alternative is resizing the image by interpolation before applying convolution and an activation function [24, 86]. Common interpolation methods are nearest neighbour interpolation, bilinear interpolation and bicubic interpolation, as shown in figure 3.8. The choice of interpolation technique depends on the image properties of the dataset: bilinear and bicubic interpolations, as an instance, blur high-frequency features whereas nearest neighbour interpolation yields a piecewise-constant result without any blurring.

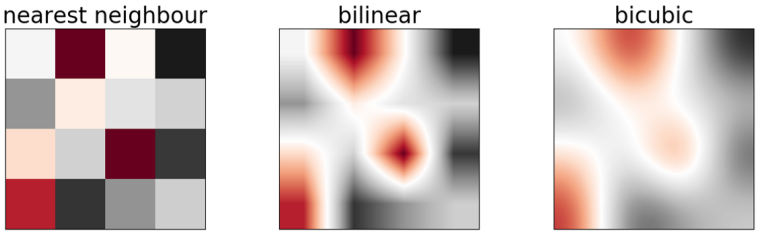



Figure 3.8.: Interpolation methods. Nearest neighbour interpolation (left) assigns the value of the nearest pixel to the interpolated pixel. Bilinear interpolation (middle) assigns the average of four neighbourhood pixels, weighted with respect to their distance. Bicubic interpolation (right) calculates a weighted average as well, but takes 16 neighbourhood pixels into account.

Using either one of the aforementioned methods, the downsampling path can now be complemented with an upsampling path, which restores the original image dimensions. However, as one would assume, the spatial resolution of upsampled images after several downsampling steps is not sufficient to accurately classify single pixels. Therefore, so-called skip connections are introduced to the network architecture, which combine fine-grain local information of shallow layers with coarse semantic information of deeper layers [77].

### 3.3.3. Skip Connections & Residual Modules

Next to a poor upsampling resolution, network structures where each layer inputs the feature maps of the previous layer embody a strictly sequential pipeline. Thus, all image features are being abstracted to the same level—although some representations may be better learned by shallow networks. This results in the so-called degradation problem, where with increasing network depth training accuracy decreases rather than increases. In theory, the network could learn parameters which transform dispensable layers into identity mappings by merely copying shallower layers. In practice, however, equipping the network with skip connections eases identity mappings and as a result increases training accuracy [42, 44]. Residual modules skip intermediate layers, as depicted in figure 3.9: the output of a layer, $x_{i-1}$, is bypassed one or more subsequent layers and eventually added to the last layer's convolutional output $f(x_{i-1})$, yielding $y_i = ReLU(f(x_{i-1}) + x_{i-1})$ as output of the $i_{\text{th}}$ block.

Note that skip connections within residual modules are also denoted as residual shortcuts. Using residual shortcuts, an identity mapping can easily be performed by pushing the residual to zero. In addition, gradients can be propagated backwards more efficiently, improving network training by alleviating the vanishing-gradient problem. Unravelling

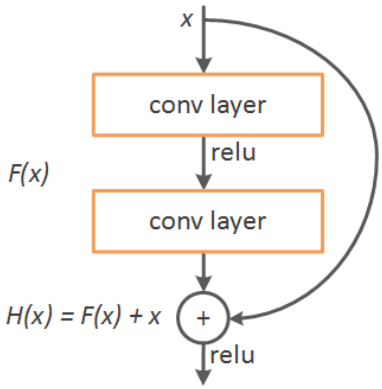



Figure 3.9.: Residual modules implement skip connections to bypass intermediate layers with an identity function. As a result, multiple new paths widen the network structure, ease the flow of information and thus network training. Depiction adapted from He et al. [42].

residual networks [42], which implement residual modules throughout the architecture, reveals that they can be interpreted as ensembles of relatively short networks [128]. The reason behind this is that residual modules introduce $O(2^n)$ implicit paths to the network, where $n$ is the number of modules and hence adding a module doubles the number of paths. Consequently, removing layers of a trained network by chance results in only minimal performance impacts, contrary to strictly sequential network architectures, where the only viable path would be corrupted. Hence, residual shortcuts increase the network's width by enabling multiple paths of variable length [135]. Fully convolutional networks, as described by Long et al. [77], do not use residual modules but implement skip connections between the downsampling and upsampling path. Thereby, fine-grain local information of shallow layers is fused with coarse semantic information of deeper layers and thus the segmentation resolution is refined, as is verified in chapter 6.

In sum, the three building blocks of fully convolutional networks are downsampling layers, which extract compressed high-level semantic information, upsampling layers, which retrieve the spatial information and skip connections, which create shortcuts between both parts so as to increase the spatial resolution. Chapter 6 studies the impact of skip connections on network performance and also demonstrates how residual modules ease network training. Note, that even though the optimal network architecture depends on the data, it is common to imitate successful convolutional network architectures. One reason is the increase in network performance and decrease in training time due to the transfer of pre-trained parameters, a technique denoted as transfer learning [91, 127, 132].

### 3.3.4. Transfer Learning

The idea of initialising the network with parameters pre-trained on a large research dataset, such as ImageNet [51, 104], seems only suitable for similar image recognition tasks. However, visualising networks trained on different datasets shows the same transition from simple to specific filters with increasing network depth, where the first layer learns basic features, such as colours and Gabor-like filters, which occur in most image compositions [132, 133, 136]. Therefore, the performance of networks used for photoluminescence image analysis can be increased with transfer learning, despite the differences of the training datasets. However, the experiments presented in chapter 6 also show that transferring incongruous parameters of deep layers decreases the network's performance compared to a network where only applicable parameters are transferred. It becomes apparent that the number of transferred layers is a hyperparameter, which must be carefully tuned.

Nowadays, parameters pre-trained with standard convolutional networks on research datasets are available for all common network libraries. The decision which architecture to imitate determines the network's downsampling path and should take into account the network capacity required by the data. In comparison to the images of standard datasets, which depict everyday scenes, photoluminescence images are composed of only one image object with, in comparison, little variation in the defect patterns. Thus, to keep the number of parameters small, the 16-layer VGG 16 network was used for parameter transfer. The VGG 16 network was introduced in 2014 by Simonyan and Zisserman [115] at the ImageNet Large Scale Visual Recognition Challenge and is used for transfer learning until today, due to its consistent architecture [58, 77, 85, 134, 146]. More sophisticated architectures with more layers and a higher performance on the ImageNet dataset, such as VGG 19, ResNet [42] and GoogLeNet [124], have since been introduced but provide too much complexity for photoluminescence images. Note, that the VGG 16 network is a multi-class classification network, consisting of 13 convolutional and three fully-connected layers. It is, however, possible to transform the parameters of fully-connected layers into convolutional layers. As an instance, the first fully-connected layer with 4,096 neurons follows the last convolutional layer with 512 feature maps of size $7 \times 7$. Usually, the convolutional layer is flattened before connecting its neurons in full to the subsequent fully-connected layer. In order to obtain convolutional layers only, we omit the flattening and instead apply convolution with $7 \times 7$ kernels. Thereby, the fully-connected layer is transformed into a convolutional layer of size $1 \times 1 \times 4096$. In other words, we obtain 4,096 feature maps with only one feature each. As a result, all parameters of a convolutional network can be transferred to the downsampling part of a fully convolutional network. The upsampling part is then initialised randomly.

In sum, transfer learning increases network performance by re-using the parameters of a network trained on a large research dataset. After initialising the network's downsampling path with the transferred parameters and the remaining layers with small random

numbers, the network is trained end-to-end. Hereby, the transferred parameters are fine-tuned to the actual dataset while the randomly initialised parameters learn a new set of filters from scratch. With regard to the special composition of photoluminescence images, sophisticated network architectures with high complexity were discarded; instead, the VGG 16 network was used for parameter transfer, due to its consistent architecture and small number of layers.

## 3.4. Summary

This chapter provided the theoretical background of fully convolutional networks, starting with the concept of supervised-learning algorithms and the concrete example of a vanilla neural network for image classification. Here, the algorithm learns the underlying mapping function of the dataset's input-label pairs by training and is optimised via backpropagation. Because neural-network algorithms are prone to overfit as well as difficult to train, several optimisation methods were described, covering L2 regularisation, parameter-update optimisation and batch normalisation. Altogether, the presented methods are common practice to ease network training and increase network performance of all kinds of network architectures. Afterwards, fully convolutional networks were introduced, which represent a specialised network architecture for semantic segmentation and advance the concept of supervised-learning algorithms to enable pixel-wise classifications. Based on the idea of convolutional neural networks, the network architecture consists of three parts, namely a downsampling path, an upsampling path and skip connections. Next to the classic fully-convolutional-network architecture, additional methods were illustrated, including residual shortcuts and transfer learning. As described in the following chapter, the classic fully-convolutional-network architecture by Long et al. [77] was developed with regard to a large research dataset of everyday-scene images [28]. However, detecting defective LED chips in photoluminescence images poses a different challenge to the network design. Chapter 5 therefore introduces a modified fully-convolutional-network architecture, designed with regard to a small dataset of simple composed photoluminescence images. Afterwards, advanced architectural concepts are described, which further improve network performance. The corresponding empirical analysis of the developed network architecture is illustrated in chapter 6.

# 4. State of the Art & Related Work

Neural network architectures derive their final mapping function from the data they are trained on and despite being self-learning algorithms—how well the architecture is designed with regard to the data distinctively influences network performance [3, 39, 60, 81]. For network research, there are several large and neatly labelled datasets available, covering a wide range of applications from everyday and urban street scenes to 3D CAD meshes [10, 11, 19, 28, 31, 73, 80, 97, 104]. Since different kinds of datasets impose different requirements on the network, the following overview over state-of-the-art architectures focusses on networks developed for the pixel-wise classification of greyscale and RGB images. Here, first the original fully convolutional network for semantic segmentation is introduced, followed by several advancements to the network architecture. The theoretical background of the presented ideas was elaborated in the previous chapter. The second section presents work loosely related to the segmentation of photoluminescence images, where the focus lies not on the network architecture itself but on the network in relation to the employed dataset. At the beginning, the two biggest research areas are covered, namely computer vision for autonomous vehicles and medical imaging. The analysis of medical images shares several similarities with photoluminescence wafer images, concretely small datasets and a more resembling image composition. Eventually, networks designed for surface defect detection are presented, where the authors work with industrial data. However, because there are no publicly available datasets, the number of publications in this area is distinctively smaller and until now no fully-convolutional-network applications for surface defect detection have been published. Therefore, related algorithmic applications are presented, covering convolutional neural networks and handcrafted segmentation algorithms.

## 4.1. State of the Art

In 2014, Long et al. [77] introduced the first end-to-end trainable network architecture for semantic pixel-wise labelling, which obviated the need for additional pre- and post-processing parts. The architecture of fully convolutional networks (see figure 4.1) is composed of three building blocks, which were introduced in chapter 3: downsampling path, upsampling path and skip connections.

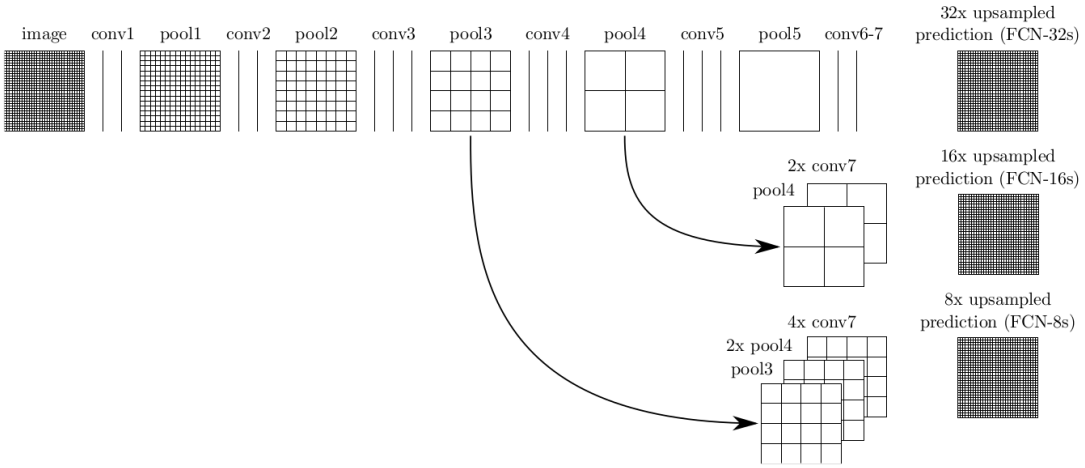



Figure 4.1.: Fully-convolutional-network architecture by Long et al. [77], designed with regard to everyday and street-scene images [19, 28, 73]. Vertical lines represent convolutional layers, where the number of kernels per layer in the downsampling path follows the VGG 16 network [115] so as to enable transfer learning. Squares represent the current image's resolution, where input and output image have the same resolution, while pooling layers decrease the image resolution. The curved lines indicate skip connections, which fuse fine-grain local information of shallow layers with coarse semantic information of deeper layers and thus refine the output resolution. Note that FCN-8 denotes a network architecture with two upsampling stages and skip connections, respectively, where the output image is upsampled 8 times after the last upsampling stage. Reproduced with kind permission of Evan Shelhamer.

In their work, Long et al. [77] showed that by designing the downsampling path as an imitation of a standard convolutional-neural-network architecture, pre-trained network parameters can be transferred so as to increase network performance. A common choice for this technique, denoted as transfer learning, is the VGG 16 network by Simonyan and Zisserman [115], which outperformed transferred parameters of other architectures in Long et al. [77]. The VGG 16 network and accordingly the fully convolutional network's downsampling path consist of 13 convolutional layers and three fully-connected layers, which are in case of fully convolutional networks either transformed to convolutional layers or omitted. Here, the downsampling path retrieves coarse, semantic information from the input and outputs highly condensed feature maps, which are then upsampled using transposed convolution, so as to restore local appearance information. As Long et al. [77] have shown, upsampling the downsampled features maps in one step yields a coarse segmentation result. Therefore, two additional upsampling stages were introduced, which in addition are fused with their downsampling counterpart via skip connections. In total, this architecture, denoted as FCN-8, covers 15 downsampling layers and three upsampling layers with corresponding skip connections, where the final softmax layer

outputs the probability distribution over the predicted class categories. Incidentally, fully convolutional networks were designed for the segmentation of everyday-scene images, such as given by the PASCAL VOC dataset [28], which covers 11,530 images with 20 class categories (persons, animals, vehicles, furniture).

Meanwhile, several working groups have advanced the network architecture: as an instance, both Kendall et al. [58] and Noh et al. [85] proposed encoder-decoder architectures: here, the downsampling path (encoder), which consists of the 13 convolutional layers of the VGG 16 network, is mirrored by the upsampling path (decoder). With this distinctive increase in upsampling layers, both architectures achieve higher classification accuracies on the PASCAL VOC dataset but provide too much complexity for the analysis of photoluminescence images. Note that the network of Kendall et al. [58], denoted as SegNet, additionally introduced a different upsampling technique. Here, the decoder stage upsamples the feature maps not with transposed convolution but based on the maxpooling indices. Dense feature maps are then obtained by subsequent convolution with trainable kernels. Altogether, SegNet consists of 26 convolutional layers, 13 downsampling and 13 upsampling layers.

In contrast, Yu and Koltun [134] have developed a different approach regarding the downsampling path: instead of consistently subsampling the image dimensions with maxpooling operations, a subset of interior convolutional pooling blocks is replaced by dilated convolutions. Thereby, multi-scale contextual information is aggregated without loosing spatial resolution. Subsampling operations, like maxpooling, not only condense the information but also increase the neuron's receptive field in deeper layers, enabling the network to "see" most of the input image. Dilated convolutions, on the other hand, increase the receptive field by modifying the convolution operator instead of downsampling the feature map. Because the spatial consistency between neighbouring pixels grows weaker as the dilation factor increases, Yu and Koltun [134] replace only the last VGG 16 layers with dilated convolution and as a result achieve a higher accuracy on the PASCAL VOC dataset. Dilated convolutions are used in various contexts [16, 41, 75, 89] and are also examined in this work.

Images often contain objects of varying size, depending on how close by or far away the object was photographed as an instance, causing coarse representations of small objects and fragmented segmentations of large objects. Chen et al. [15] approached this challenge by implementing a spatial pyramid of dilated convolutions at the end of the downsampling path, where the feature maps are probed with multiple sampling rates at once. Thus, multiple effective receptive fields are obtained, which allow the capturing of objects at multiple scales [45]. Furthermore, conditional random fields are added as post-processing step [62]: here, the network's coarse prediction scores are combined with low-level information captured by local pixel interactions, thus refining the segmentation result. In order to omit the resulting two-step training process, Zheng et al. [142] proposed to redefine conditional random fields as recurrent neural network so as to fully integrate it

into the convolutional neural network. In their latest work, denoted as DeepLab3, Chen et al. [16] introduced a network architecture without conditional random fields. Instead, they optimise the spatial pyramid module by adding batch normalisation [53] and include feature-level information via global average pooling [75, 141]. The additional global context clarifies local confusions, which can result in fragmented segmentations, and smooth the segmentation. Incidentally, the work of Liu et al. [75] indicates that adding image features has a similar effect as adding conditional random fields, next to simplifying the training process. Accordingly, the implementation of dilated convolutions and image-level features by means of a spatial pyramid module is examined in chapter 6.

Another approach to increase segmentation accuracy is the consistent implementation of skip connections, based on the success of very deep convolutional networks build with residual modules, such as ResNet [42]. Huang et al. [50] advanced the concept of skip connections even further in their densely connected convolutional network (DenseNet). Here, each layer inputs all preceding feature maps, which strengthens feature propagation and alleviates the vanishing gradient problem. Jégou et al. [57] as well as Zhu and Newsam [146] extended DenseNet to fully convolutional networks and implemented dense connections throughout the downsampling and upsampling path as well as skip connections that fuse both paths. The usage of residual shortcuts and dense blocks is further investigated in chapter 6. When analysing the architectures of Jégou et al. [57] and Zhu and Newsam [146], differences occur in the number of layers and dense blocks as well as the employed hyperparameters. These differences stem from the datasets the architectures were developed for: Jégou et al. [57] aim to segment urban scenes and evaluated their architecture on the CamVid [10] and the Gatech [97] datasets. Zhu and Newsam [146], on the other hand, address unsupervised motion estimation, where the optical flow of objects is estimated and evaluated their work on the Flying Chairs [31], the MPI Sintel [11] as well as the KITTI Optical Flow [80] dataset. It becomes apparent that every application and dataset, respectively, requires a specifically designed architecture in order to achieve state-of-the-art performance.

## 4.2. Related Work

Next to the presented methods in the previous section that focussed on the network architecture, this section presents different network designs with respect to their application and the used dataset. Here, computer vision for autonomous vehicles and medical-image analysis are arguably the largest research areas. As an instance, most of the aforementioned network architectures (FCN-8 [77], SegNet [58], DeepLab [15]) are listed in the leaderboard of the Cityscapes dataset benchmark [19, 54]. Compared to photoluminescence images of LED wafers, the images of urban street scenes differ most obviously in the number of depicted objects as well as the object variations, as shown in figure 4.2. Consider, as an instance the image of a street, showing houses, cars, pedestrians, traffic

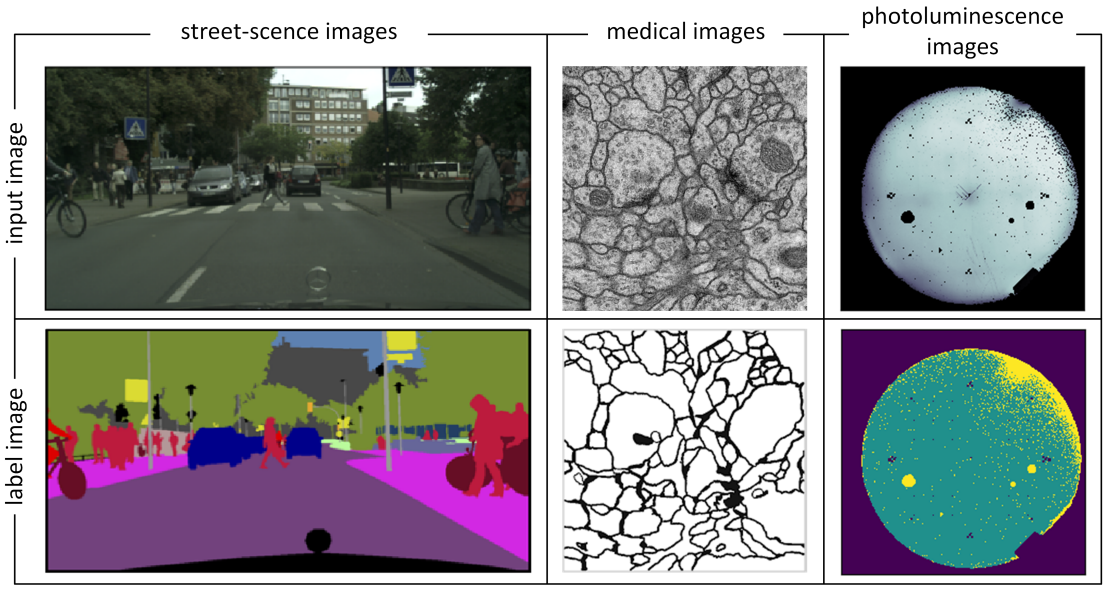



Figure 4.2.: Comparison of differently composed images, namely street-scene images of the Cityscapes dataset [19] used in the development of a variety of network architectures [15, 58, 77], transmission electron microscopy images of brain microcircuitry [13] used by the U-Net architecture [101], and photoluminescence images of LED wafers, used in this work.

lights and so on. On the one hand, the number of objects and class categories per image is usually more than one and can differ significantly from recording to recording. On the other hand, houses, cars and traffic lights are inherently differently sized and assume various scales, depending on the distance to the camera for example. Photoluminescence wafer images, in contrast, always depict one wafer recorded in the same setting—here, the challenge stems from the differing brightness values of the measurement in addition to varying defect sizes and shapes, where single defective LED chips correspond to one pixel (see also chapter 2).

### 4.2.1. Network Architectures for Everyday-Scene Understanding

As a result of the complex scenery, networks for (street) scene understanding are typically very deep, counting up to 100 layers [16, 57], and while they aim for a high segmentation accuracy, pixel-level accuracy is not their focus. Lin et al. [72], as an instance, have introduced a multi-path refinement network (RefineNet) for the segmentation of street scenes, which exploits image features at multiple levels of abstraction through a cascaded architecture. In addition, chained residual pooling operations capture image background context: contrary to wafer images with a steady background, in street-scene images every-

thing except the given class categories counts as background, including complex scenery. Dai et al. [21] also introduced several paths to their network structure, but use them as multi-task outputs for instance-aware semantic segmentation: in the first stage, object instances are marked with boxes (bounding boxes), where two persons correspond to two instances of the same class category, for example. In the second stage, each instance is masked (semantic segmentation) before categorising them (instance segmentation) in the third stage. Distinguishing the various instances of an object obviously advances the objective of scene understanding. However, neither photoluminescence images nor wafer-prober-based labels allow consistent conclusions about the defect cause. In contrast, the network architecture of Pohlen et al. [95] also couples two processing streams, but in a simplified way: here, skip connections and residual shortcuts are rearranged to one residual stream, which retains the full image resolution throughout the process. The second stream follows the typical sequence of downsampling and upsampling, where the result at each stage is additionally upsampled and added to the residual stream. Analysing segmentation results of both networks reveals two observations: on the one hand, objects are well recognisable but appear blurred. And on the other hand, slim objects such as lamp posts are often portrayed holey and coarse. It is obvious, however, that pixel-level accuracy might be desirable for scene-understanding tasks, but contrary to wafer images where every pixel corresponds to an LED chip, it may not be a requirement.

### 4.2.2. Network Architectures for Medical-Imaging Datasets

A different kind of data is covered by medical imaging, where the wide range of applications includes measurement methods such as (functional) magnetic resonance imaging (fMRI / MRI) [125], transmission electron microscopy (TEM) [101] and computed tomography (CT) [17, 111]. Since 2016, medical image analysis is dominated by deep learning methods [74]. Contrary to images of everyday and street scenes, medical images depict just one object with a fairly steady background: consider, as an instance, a CT image of the abdomen that is recorded to examine the liver with regard to lesions [5, 17]. Here, challenges are a low contrast between liver and lesion as well as a varying number of differently sized and shaped lesions, among others. Hence, medical imaging is in general more closely related to photoluminescence measurements than urban street scenes, even though medical images may display more variations. On the task of liver segmentation and lesion detection, Ben-Cohen et al. [5] were the first to employ fully convolutional networks, with which they outperformed previous state-of-the-art methods. In addition to the architecture described by Long et al. [77], they experimented with a fourth skip connection (FCN-4), which they found to not improve accuracy in case of liver segmentation but to be beneficial for the segmentation of the smaller sized lesions. Another congruence to photoluminescence images are the small and special datasets: Litjens et al. [74] observed that the incorporation of expert knowledge into data pre-processing and augmentation is a key determinant, next to the network architecture and hyperparameter tuning. In line with this, Ben-Cohen et al. [5] increased their dataset by a factor

of 4 using differently scaled images. Furthermore, they augmented the input liver slices by adding two neighbourhood slices and equalised the class-category balance using class weights.

In a later work, Christ et al. [17] also attended to automatic liver and lesion segmentation, using a cascaded fully convolutional network. Their data preparation process involved several preprocessing steps, which increased the differentiation of abnormal liver tissue in CT/MRI volumes. Moreover, they performed several data augmentations, namely rotation, translation, elastic deformation as well as adding Gaussian noise to the images. The network itself consists of two parts: first, a fully convolutional network segments the liver from the abdomen and creates a new image, which depicts the region of interest. The segmented liver is then the input for a second fully convolutional network, which segments the lesions. Finally, a 3D conditional random field refines the predicted lesion areas and outputs the final segmented CT/MRI volume. While achieving state-of-the-art results, Christ et al. [17] noted that in case of highly heterogeneous structures finding well generalising 3D conditional random field hyperparameters proved difficult and referenced to the aforementioned attempts of including conditional random fields into the network [16, 142]. The basic network architecture used by Christ et al. [17] derives from the so-called U-Net, developed by Ronneberger et al. [101] for the segmentation of brain microcircuitry (see figure 4.2). Here, the concept of encoder-decoder networks is combined with skip connections, that is, the upsampling path mirrors the downsampling path and all stages are connected via skip connections, as shown in figure 4.3. In order to teach the network invariance against image shifts and rotations as well as robustness against deformations and grey value variations, despite a very small dataset of only 30 TEM images, Ronneberger et al. [101] used excessive data augmentation in form of random elastic deformations [114] and dropout layers [118]. In their work, Ronneberger et al. [101] point out the importance of elastic deformation augmentation and attribute the associated performance increase to the fact that deformations are a common biomedical tissue variation, which can be simulated efficiently. To improve the border segmentation quality of close-by or touching cells, they also use weighted loss calculation.

Since their introduction in 2015, U-Nets were widely adopted: Çiçek et al. [18] extended the U-Net architecture to the application of biomedical volumetric image segmentation by replacing 2D operations with 3D. Li et al. [69] employed U-Net for pixel-level sea-land segmentation and introduced so-called DownBlocks, which combine two convolutional layers of 64 and 32 kernels, in order to increase the receptive field while decreasing the number of parameters. Furthermore, each DownBlock implements a residual shortcut between block input and output and forwards the result to the subsequent block as well as the corresponding UpBlock. Here, too, data augmentation is used to increase network robustness via random cropping, where only images are kept which display both, sea and land objects. Other network architectures, however, are also still in use: Sharma et al. [111] applied a 10 layer encoder-decoder network to the automatic segmentation of kidneys, whereas Tai et al. [125] employed an FCN-8 architecture for the segmentation

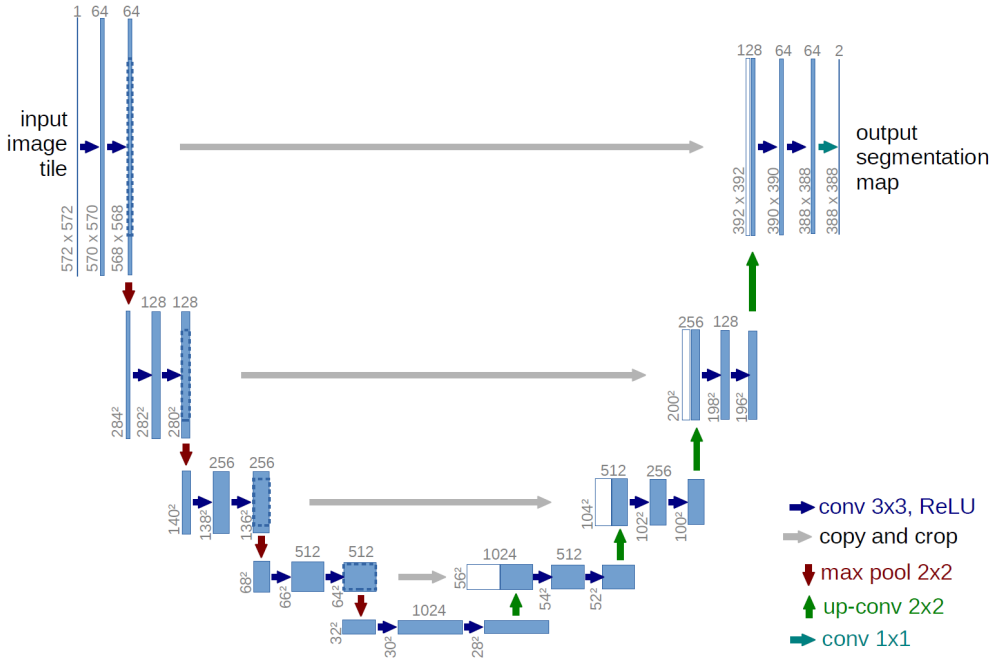



Figure 4.3.: U-Net architecture by Ronneberger et al. [101], designed with regard to transmission electron microscopy images of brain microcircuitry [13]. Here, each stage covers two consecutive layers and the upsampling path of the U-Net architecture mirrors the downsampling path, incorporating a so-called encoder-decoder setup, where all stages are connected via skip connections. Reproduced with kind permission of Olaf Ronneberger.

of multi-channel fMRI images. Here, each of the 32 fMRI channels corresponds to the different intensities of various tissues in the vertebral area of human beings. In order to decrease the dimensionality of the input volume, Tai et al. [125] employed principal component analysis and calculated a set of three principal-component images, which represent 99.9 % of the original information. Altogether, the dataset consists of only six images, of which one was spared for testing. The network was initialised with VGG 16 weights and trained with a very small initial learning rate of $10^{-14}$ for 5.000 epochs. Tai et al. [125] assume that only limited changes in the transferred weights are necessary for the network to converge. With a mean intersection over union metric of 60.7, the network implementation outperformed previous state-of-the-art methods such as k-nearest neighbours with a 13.1 mean intersection over union.

In sum, analysing the literature about fully convolutional networks for medical imaging reveals that due to the small datasets, more effort is expended on pre-processing and data augmentation. Furthermore, the network architectures are inherently simpler, that is spatial pyramids or multi-path networks are not employed. Rather, skip connections are implemented consistently in combination with weighted loss calculation, so as to

improve pixel-level accuracy. Accordingly, Litjens et al. [74] observed in their review study that given the same dataset and network architecture, distinctive performance differences could be achieved depending on how the data is presented to the network.

### 4.2.3. Network Architectures for Industrial Datasets

Using industrial datasets to implement a surface-defect-detection algorithm arguably shares the most similarities with photoluminescence-image analysis. Here, an object's surface is analysed based on the image of an optical measurement. Contrary to the aforementioned tasks, there are no publicly available research datasets of industrial data and as a result the number of publications is distinctively smaller. Furthermore, the majority of the publications describes the implementation of convolution neural networks for defect classification. As an instance, Park et al. [92] studied a five layer convolutional neural network for several material surfaces, including wafers, stone and wood, where the network predicts whether a defect is depicted or not. Both, Soukup and Huber-Mörk [117] and Faghih-Roohi et al. [29] developed convolutional neural networks for the detection of rail surface defects. Here, Soukup and Huber-Mörk [117] implemented a three-layer network and outperformed a model-based approach with the assistance of data augmentation, whereas Faghih-Roohi et al. [29] designed a six layer network which they trained with a dataset of raw images. In order to address the varying environmental impact on images of civil infrastructure, such as bridges, dams and skyscrapers, Cha et al. [14] developed an eight layer convolutional neural network. To keep their $5,888 \times 3,584$ pixel images manageable they implemented a sliding window scanning plan, where every image is cropped into pieces of $256 \times 256$ pixels in two slightly shifted scans. Thereby, missed cracks at the image edge were prevented. After classifying each image, a crack map of affected images was composed—thus indicating that a fully-convolutional-network architecture might be beneficial in further studies. They also studied the usage of classical edge-detection methods, namely Sobel and Canny filters, but noticed that due to the noisy images no useful crack information was obtained. Their convolutional neural network implementation, on the other hand, yielded a classification accuracy of 97.95 % while proving robust against varying lighting situations and civil structures. Closer to semantic segmentation is the work of Li et al. [71], who implemented a so-called bounding-box algorithm. Here, a convolutional neural network predicts not only a set of class categories but also the coordinates of a rectangle, which bounds a detected defect. While bounding-box networks provide only a coarse localisation of the defect, they are also computationally efficient and allow real-time classification [49, 76], thus enabling the intended employment in a filling line production environment.

Implementations of semantic-segmentation algorithms are presented by Zhang et al. [139], Wen et al. [131] and Amirul Anwar and Zaid Abdullah [4]. However, none of them utilises fully convolutional networks: Zhang et al. [139] describe the segmentation of defects in X-ray images of aluminium alloy wheels, using a six step segmentation process, where an

adaptive threshold-segmentation algorithm is combined with a morphological reconstruction operation. As noted by the authors, the algorithm's performance depends crucially on the setting of five different parameters, and as a result the algorithm does not inherently generalise well to varying brightness values. Wen et al. [131], on the other hand, implement a three step surface-inspection system for mechanical components, namely bearing rollers. First, a contour-detection module determines whether the bearing roller's contour is in fact a circle. Then, a convolutional neural network detects possible defects and marks them with bounding boxes as region of interest. The localised regions are subsequently extracted and forwarded to a third module, which segments the defective area. The segmentation algorithm is composed of median filtering, Otsu thresholding [90] and morphological processing. Afterwards, the segmented area of each defect image is multiplied with a defect-type-dependant coefficient and the resulting values are summed, so as to determine whether the component failed the quality control. The third study, conducted by Amirul Anwar and Zaid Abdullah [4], examined the detection of multi-cracks in electroluminescence images of multi-crystalline solar cells. Here, the objective is to distinguish micro-crack pixels from intrinsic cell structures, such as dislocation clusters and grain boundaries as well as other defect structures, such as broken fingers. The proposed segmentation algorithm consists of five steps, starting with image pre-processing, where the image is filtered in the frequency domain before normalisation. Then, an anisotropic diffusion filter is applied, followed by post-processing, which includes double thresholding and intensity tracing and thresholding. Afterwards, the defect shape is analysed and classified by a support vector machine. Hereby, the authors outperform classical methods such as Canny, Otsu and Sobel and achieve an accuracy of over 88 %, with a comparably long image processing time of 4.1 seconds. Moreover, the smallest micro-cracks detectable by the algorithm cover 47 pixels—thus the algorithm does not operate on pixel-level. In all three applications, the utilisation of multi-step algorithms, which include the setting of multiple parameters, poses the question whether fully convolutional networks could have been implemented with the same or even better performance—however, the authors do not mention any experiments in this direction.

## 4.3. Summary

In sum, fully convolutional networks have outperformed other computer-vision methods in a variety of research areas in the recent years, such as scene understanding and medical imaging. The given overview over network architectures for different applications has shown that each application requires its own network design. Moreover, studying publications with regard to very small medical-imaging datasets emphasised the importance of a carefully compiled dataset. Due to the lack of public datasets, only few publications describe the implementation of algorithms for the detection of surface defects, with a focus on convolutional neural networks and sophisticated, hand-crafted segmentation algorithms. The success of fully convolutional networks on small medical-

imaging datasets, however, indicates that the application of fully convolutional networks on industrial datasets may be beneficial as well. Therefore, in the following chapter a modified fully-convolutional-network architecture is presented, designed with regard to the detection of defective LED chips in photoluminescence images. Moreover, advanced architectural concepts that further increase network performance are introduced, namely densely connected convolutional layers and atrous-spatial-pyramid-pooling modules. Next to dataset preparation, the performance of neural-network algorithms depends heavily on hyperparameter tuning. Thus, chapter 6 analyses the influence of different architectural concepts, various hyperparameters as well as how data preparation may be used to increase segmentation accuracy. Afterwards, chapter 7 concludes the final network architecture and states implementation details.

# 5. Network-Design Approach for the Given Application

In the previous chapter, an overview of fully-convolutional-network architectures for a variety of applications was presented, demonstrating that each application requires its own network architecture design, in addition to a carefully prepared dataset. Based on these insights, the dataset description in chapter 2 and the theoretical background in chapter 3, network-design ideas were developed, which address the differences between datasets containing images of everyday scenes and medical images, respectively, and the given application's small dataset of photoluminescence images. Contrary to the segmentation of the aforementioned images, photoluminescence images of LED wafers are simply composed but feature highly variable brightness values and defect structures. Additionally, each and every pixel corresponds to an LED chip and thus must be classified accurately. The subsequently presented network architecture approaches this challenge by setting up a novel upsampling path, where the basic network architecture is inspired by Long et al. [77] and the design of the downsampling path follows Simonyan and Zisserman [115], so as to enable the transfer of pre-trained parameters. Afterwards, advanced architectural concepts are introduced, namely densely connected convolutional blocks [50] and atrous-spatial-pyramid-pooling modules [16], so as to further refine the network architecture for the given application with respect to the recognition of multiple scaled image objects. The influence of these architectural concepts on network performance, in addition to hyperparameter tuning and data preparation, is examined in the following chapter, while chapter 7 presents the final network architecture, implementation and evaluation.

## 5.1. Basic Fully-Convolutional-Network Design

Fully-convolutional-network algorithms apply thousands of self-learned filters on an input image in order to determine each pixel's class category. When designing the network architecture, the composition of the dataset's images and the number of class categories influence the optimal number and interconnection of layers, where networks for everyday-scene image segmentation may be composed of over 100 layers [10, 42, 57]. Networks for medical images, on the other hand, are commonly shallower but still cover a large number of kernels (see also table 6.1). In contrast, photoluminescence images are simply com-

posed, greyscale images that cover only three class categories, which indicates a shallow network architecture with a limited number of kernels. Because the downsampling path follows the VGG 16 network by Simonyan and Zisserman [115], so as to enable transfer learning, the downsampling path was basically preserved. Instead, a novel upsampling path was developed, which provides a pixel-wise output resolution and reliably distinguishes salient brightness values corresponding to defective LED chips from measurement artefacts and non-defective structures.

Figure 5.1 illustrates the basic design idea of the developed network architecture. The depicted network inputs a $442 \times 440 \times 1$ greyscale photoluminescence image, where $442 \times 440$ are the image dimensions and 1 represents the number of colour channels. First, the input image is run through two convolutional layers with 64 kernels each, where the second layer processes the feature maps generated by the first layer. Note that each convolutional layer covers the consecutive operations convolution, batch normalisation and ReLU activation function. Then, image dimensions are reduced using a maxpooling operation. The subsequent convolutional layers consist of 128 kernels each. Hereby, a bi-pyramid of decreased dimensions and an increased number of representations throughout the downsampling path is initiated. Consequently, feature maps in the last layer in the downsampling path are reduced to a size of $13 \times 13$, where the two layers cover 512 and 64 feature maps, respectively. In total, the downsampling path consists of 15 layers in six stages, where the first 13 layers imitate the VGG 16 architecture and the last two layers feature a reduced number of kernels so as to reduce network complexity. Moreover, additional residual shortcuts enable the network to bypass redundant layers in every three-layer block and thus allow the network to adjust the abstraction of image features by itself.

Following the same logic of reduced network complexity, the number of layers in each stage of the upsampling path is restricted to one, with only 64 kernels per layer. This novel approach takes the simple composition of photoluminescence images into account, proceeding on the assumption that sufficient feature extraction is performed in the downsampling path, while the upsampling path is focussed on providing a pixel-level output resolution. For this purpose, the upsampling path mirrors the downsampling path and thus implements gradual upsampling steps using bilinear interpolation, where the subsequent convolution operation refines the result. Furthermore, by connecting each upsampling stage with its downsampling counterpart via skip connections, coarse semantic information of deep layers is fused with fine-grain local information of shallower layers, which further increases the output resolution. To employ skip connections between layers with a varying number of feature maps, first the depth of the shallow layer is reduced, where $1 \times 1$ kernels are a common choice due to their small number of parameters. Then, both feature maps are summed before applying a ReLU activation function. The resulting feature maps are then again resized and enhanced with spatial information from shallower layers, until image dimensions are eventually restored. At last, the layer depth is reduced so as to match the number of class categories before calculating output

probabilities using a softmax activation function. Altogether, the illustrated architecture features a shallow network design, which enables pixel-fine and thus chip-fine output images due to the consistent implementation of skip connections and gradual upsampling steps. The effects of architectural and hyperparameter choices are further examined in chapter 6, including the number of layers in the downsampling path, the number of skip connections and residual shortcuts as well as the upsampling operation. Note that partial results of the architecture design and hyperparameter tuning have also been discussed in Stern and Schellenberger [120].

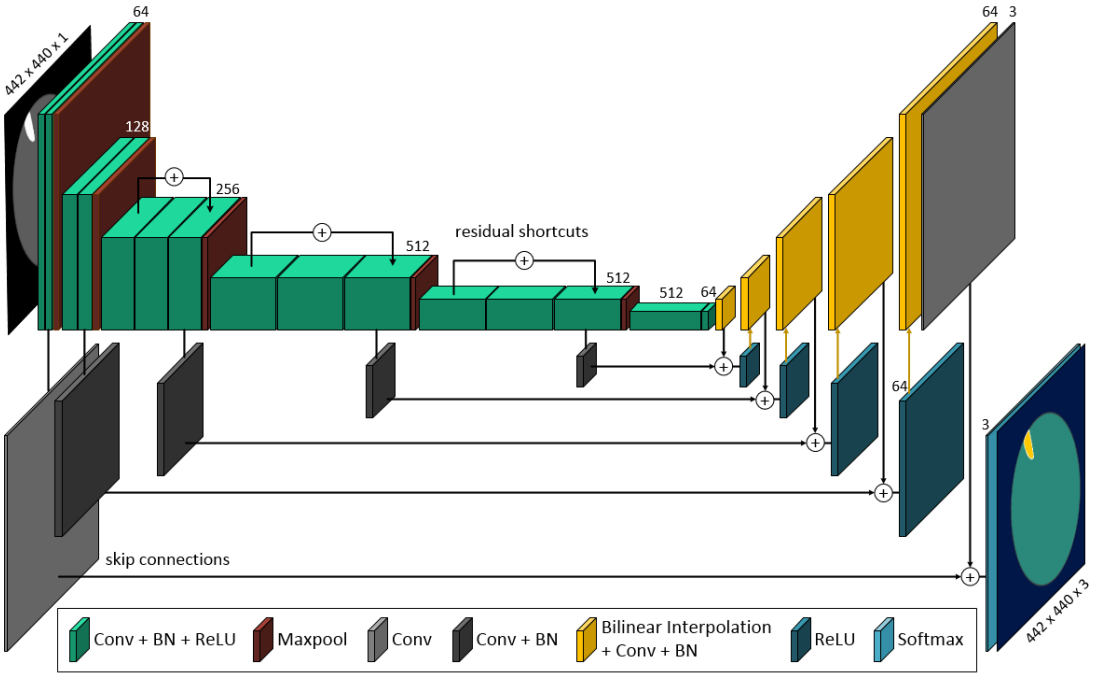



Figure 5.1.: Example of a fully-convolutional-network design developed in this work [120]. The network first downsamples the input image using several convolutional layers (green), where each convolutional pooling block consists of two to three convolutional layers followed by a maxpooling operation. Afterwards, spatial dimensions are retrieved by the consecutive operations bilinear interpolation, convolution and activation function (yellow). To adjust the network design to the special composition of photoluminescence images, the number of feature maps in each upsampling layer is restricted to 64. Finally, skip connections fuse fine-grain spatial information of shallower layers with semantic information of deeper layer and thus improve the output resolution. Additional residual shortcuts enable the network to bypass redundant convolutional layers and thus diminish the degradation problem. Depiction adapted from Tai et al. [125], where *Conv* denotes $3 \times 3$ convolution and *BN* denotes batch normalisation.

## 5.2. Advanced Architectural Methods

The previous section presented a modified fully-convolutional-network architecture, designed with regard to the given dataset. The corresponding analyses in chapter 6 reveal that the proposed architecture yields reliable segmentation results for single defective LED chips as well as repeatedly occurring defect structures. Defect clusters with a rare shape and brightness pattern, however, may result in a flawed segmentation. Therefore, the possible extension of the developed network architecture with advanced architectural methods was studied, namely densely connected convolutional blocks and atrous-spatial-pyramid-pooling modules. On the one hand, densely connected convolutional blocks, also denoted as dense blocks, advance the idea of residual shortcuts by connecting each and every layer in the network so as to mitigate the vanishing-gradient problem [50]. On the other hand, atrous-spatial-pyramid-pooling modules address the challenge of multiple scaled image objects by probing incoming feature maps with different receptive fields at once, using dilated convolutions and global average pooling [16].

### 5.2.1. Densely Connected Layers

Based on the idea of residual modules (see figure 3.9), where shortcuts between remote layers enable the network to individually adjust the degree of feature processing, Huang et al. [50] developed the concept of dense blocks. Here, all preceding feature maps serve as layer input, with $x_l = H_l([x_0, x_1, ..., x_{l-1}])$, where $x_l$ is the current layer, $H_l$ is a composite function of the three consecutive operations convolution, batch normalisation and ReLU activation function and $[x_0, x_1, ..., x_{l-1}]$ refers to the concatenation of the previous feature maps. Note that residual modules combine both layers by summation, whereas dense blocks employ concatenation to further ease the information flow through the network. Figure 5.2 illustrates the idea of densely connected convolutional networks (DenseNets), with multiple additional paths: a DenseNet architecture with $L$ layers consists of $\frac{L(L+1)}{2}$ direct connections between its layers, whereas a vanilla network consists of only $L$ connections.

Jégou et al. [57] as well as Zhu and Newsam [146] extended DenseNets to fully convolutional networks by implementing dense connections throughout the downsampling and upsampling paths, in addition to skip connections. However, in each dense block in the upsampling path, both, the number of feature maps as well as the feature-map resolution, increase. Therefore, only feature maps of the preceding dense block are upsampled, yielding $L_{db} \times k$ output feature maps, where $k$ is the number of feature maps each layer produces, denoted as growth rate, and $L_{db}$ denotes the number of layers in the corresponding dense block. In comparison, the $l_{th}$ layer in the downsampling path outputs $k_0 + k \times (l-1)$ feature maps, where $k_0$ is the number of channels in the input layer. The direct access of each layer in a DenseNet to both, the input as well as the loss gradients,

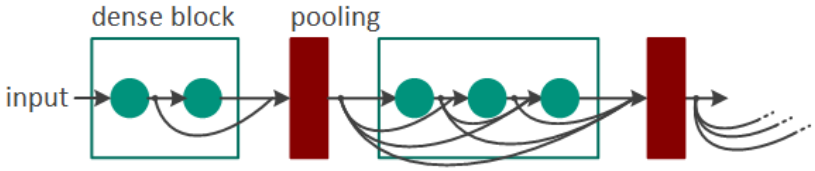



Figure 5.2.: Densely connected convolutional blocks, denoted as dense blocks. Here, each dense block consists of two to three convolutional layers, followed by a downsampling pooling operation. Dense connections are created by feeding each layer with all preceding, concatenated feature maps. As a result, feature map re-use is encouraged, yielding condensed, easy to train models. Depiction adapted from Huang et al. [50].

eases network training and leads to implicit supervision. As a result, the re-use of feature maps is encouraged throughout the network, which is favourable due to the simple composition of photoluminescence images and thus enables condensed architectures. As shown in chapter 6, reducing the number of kernels per layer in the downsampling path while incorporating dense connections throughout the aforementioned network architecture reduces overfitting and thus increases network accuracy.

### 5.2.2. Atrous Spatial Pyramid Pooling

The segmentation of defective LED chips in photoluminescence images is challenging for two reasons: on the one hand, each image pixel correlates to a chip and thus pixel-wise output resolution is necessary. And on the other hand, defective LED chips occur as single defect as well as clustered in large defect structures. In other words, defect objects exist at multiple scales, which may result in flawed segmentation results. To address this challenge, Chen et al. [16] have introduced a so-called atrous-spatial-pyramid-pooling (ASPP) module, which probes an incoming feature map at different scales in parallel by means of dilated (also called atrous) convolutions.

Even though fully convolutional networks output an image of the same resolution as the input image, consecutive pooling steps reduce the feature map's dimensions in the downsampling path. Hereby, the ongoing compression of high-level semantic information along with the increasing receptive field compensate for the loss of image resolution and allow the network to learn progressively abstract feature representations. Dilated convolutions, on the other hand, increase the network's receptive field without decreasing feature-map dimensions [16, 41, 47, 109, 134]. Instead of applying the kernel values to a corresponding area of neighbouring pixels, as shown in 3.3 and 5.3 (left), the kernel values are dilated by sampling the input image with a dilation rate $r$, yielding

$$z_{i',j',f'} = \sum_{i=1}^{H_f} \sum_{j=1}^{W_f} x_{i'+ir-1,j'+jr-1,f} w_{ijff'}, \tag{5.1}$$

where $z_{i',j',f'}$ is a pixel in an output feature map $\boldsymbol{f'}$ and $\boldsymbol{w}$ is a $H \times W$ sized filter that is slid across the sparsely sampled input feature map $\boldsymbol{x}$. Assuming a dilation rate of 2 and a $3 \times 3$ kernel, the covered area increases to $5 \times 5$ pixels, where only every other pixel is used, as shown in figure 5.3 (right). Therefore, dilated convolution is also denoted as atrous convolution, from the French *à trous*, meaning *with holes*. Note that a dilation rate of 1 corresponds to standard convolution. By replacing convolutional layers by dilated-convolution layers, the receptive field grows exponentially with the layer depth, without loss of image resolution. However, the spatial consistency between neighbouring pixels grows weaker with an increasing dilation factor, which is unfavourable for small image objects. Furthermore, keeping feature-map dimensions consistent increases the network's memory consumption. It is thus common to replace only a subset of interior subsampling layers, as an instance the last two convolutional blocks of the downsampling path.

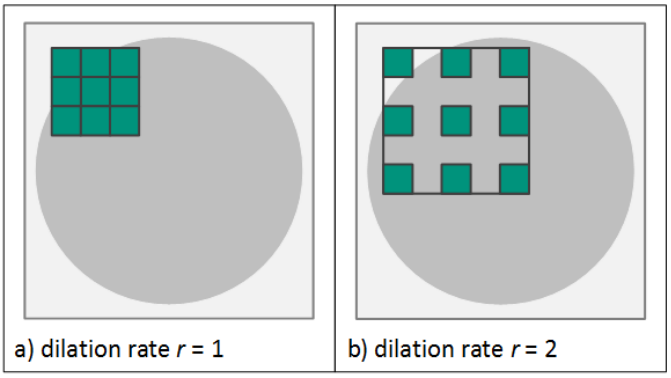



Figure 5.3.: Dilated convolution with a $3 \times 3$ kernel. The left side (a) shows dilated convolution with a dilation rate of 1, which corresponds to standard convolution. The right side (b) depicts a dilation rate of 2, where the neuron's receptive field is increased without the loss of spatial resolution.

By implementing multiple dilated-convolution layers with different dilation rates in parallel, ASPP modules analyse the incoming feature maps with various receptive fields at once, without changing feature-map dimensions [15, 16, 40, 65, 141]. This parallel processing approach enables the network to better recognise image objects at different scales, such as single defective LED chips and defect clusters. Next to dilated-convolution layers, ASPP modules incorporate image-level features by means of global average pooling, so as to prevent fragmented segmentation results: Zhou et al. [143] and Liu et al. [75] have shown that the effective receptive field of deep convolutional layers is smaller than theoretically anticipated, and thus does not capture the global image context. Moreover,

Chen et al. [16] have shown that with increasing dilation rates the effective filter size is reduced to $1 \times 1$, because non-centre weights are applied to zero padded and thus invalid image regions. As a result, the receptive field achieved by dilated convolutions is not large enough to capture image-level context. Therefore, global average pooling is applied to the incoming feature maps, followed by the consecutive operations $1 \times 1$ convolution, batch normalisation and bilinear interpolation to retrieve feature-map dimensions [75, 141]. The resulting image-level features are then added to the dilated convolution results via concatenation and forwarded to the subsequent layer. In sum, ASPP modules apply different dilated-convolution layers with various receptive fields to the incoming feature maps in parallel, in addition to the incorporation of image-level features, and thus improve the recognition of multiple scaled image objects, such as single defective LED chips and large defect clusters. The analysis of ASPP modules in chapter 6 reveals that following Chen et al. [16] by replacing the last convolutional block in the downsampling path with an ASPP module does not distinctively increase segmentation accuracy. However, replacing the penultimate upsampling layer with an ASPP module as well, does in fact increase segmentation accuracy, especially with regard to uncommon defect patterns [119].

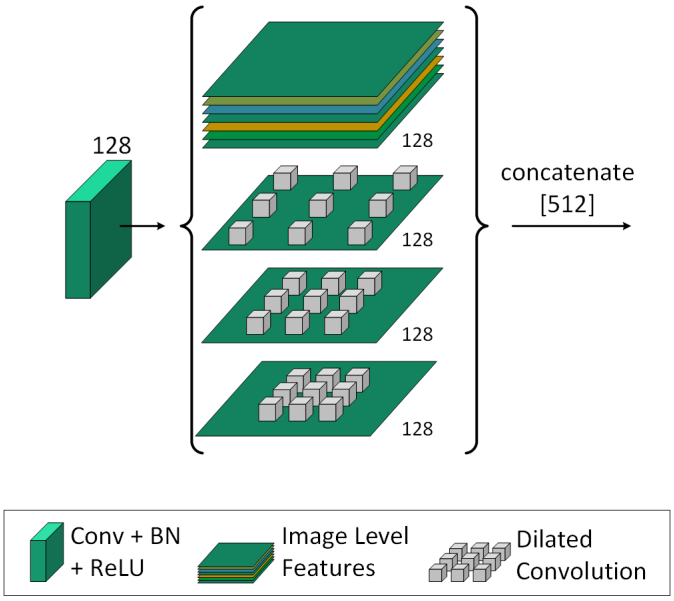



Figure 5.4.: Atrous-spatial-pyramid-pooling (ASPP) module, which applies four layers to the incoming feature maps in parallel. On the one hand, dilated-convolution layers with different dilation rates process the incoming feature maps with differently sized receptive fields and thus recognise multiple scaled image objects, such as single defective LED chips and defect clusters. Image-level features, on the other hand, are calculated via global average pooling and incorporate global image context so as to prevent fragmentary segmentation results. Depiction adapted from Chen et al. [16].

## 5.3. Summary

In this chapter, design ideas for a fully-convolutional-network architecture for the detection of defective LED chips in photoluminescence images were proposed. First, a modified version of the original fully-convolutional-network architecture by Long et al. [77] was described, where the number of layers in the downsampling path, the design of the upsampling path and the number of skip connections were chosen with regard to the given application. Then, advanced architectural concepts were introduced in order to further increase the network's segmentation accuracy, namely densely connected convolutional blocks and atrous-spatial-pyramid-pooling modules. Setting up a fully-convolutional-network architecture and examining the proposed design ideas involves a large number of hyperparameters. First, the basic architecture must be tuned, including the number of layers, the number of skip connections and residual shortcuts as well as the upsampling operation. Next to design choices, the training process must be adjusted to the data by means of hyperparameters, such as learning rate and regularisation strength. Using the tuned, basic network architecture, the influence of advanced architectural methods may be analysed. Finally, the preparation and compilation of the employed dataset distinctively influences network performance, as well. Therefore, in the following chapter the effects of architecture design and hyperparameter tuning on network performance, in addition to the influence of dataset compilation, are studied. The final network architecture, implementation and evaluation are presented in chapter 7.

# 6. Analysis of Network Design, Hyperparameter Tuning & Data Preparation

Developing a neural-network algorithm can be divided in three parts, namely preparing the dataset, designing the network architecture and tuning the network's hyperparameters. Here, the network's architecture determines the hypothesis space of candidate input-label mapping functions, in which the learning algorithm searches for a mapping function that minimises the loss over the available training examples. In other words, the network design defines the algorithm's underlying function, whose parameter values are learned during network training. Tuning network hyperparameters then constrains how the algorithm updates the parameters at each training step. Therefore, network design, hyperparameter tuning and dataset compilation distinctively influence the network's performance and only if attuned to each other the network can learn to generalise well to previously unseen data. As elaborated in chapter 4, diverse architectures have been developed by different working groups to match various types of datasets and the more complex the image composition, the deeper and more sophisticated the network architecture. However, photoluminescence wafer images are, in comparison to standard research datasets, rather simply composed: the images always display one image object, an LED wafer, recorded under the same settings. Thus, the challenge stems not from the complex image composition but from varying wafer-to-wafer brightness values, local areas of differing brightness, defect clusters that assume various shapes and multiple sizes as well as salient structures, such as film tears, that do not correlate to defective chips, contrary to similar looking cracks. Furthermore, because each pixel of the photoluminescence images equals an LED chip, pixel-level prediction resolution is necessary. Finally, input-label mismatches occur, caused by deriving the label images from defect information generated by wafer probing. Therefore, this chapter studies how the architecture design ideas proposed in the previous chapter influence network performance, in addition to hyperparameter tuning and data compilation. First, architectural aspects of the proposed basic network architecture are examined, such as the number of layers in the downsampling path, the number of skip connections and the effect of residual shortcuts. Then, hyperparameter tuning is studied, including learning rate and regularisation strength, network initialisation and transfer learning, upsampling and optimisation methods. Because learning rate, L2 regularisation and optimiser method influence each other, all methods are examined on their own as well as in combination. Afterwards, data-specific optimisation methods are investigated, namely the employment of a weighted loss calculation to equalise the unbalanced class categories and the embedding of ultrasonic measurements into photoluminescence images. Eventually, the extension

of the developed architecture design with advanced architectural methods is studied, namely densely connected convolutional blocks as well as atrous-spatial-pyramid-pooling modules. Finally, the influence of dataset composition and the network's generalisation ability are analysed using $k$-fold cross validation.

## 6.1. Network Architecture

In this section, the influences of network depth and residual shortcuts are studied. For this purpose, only the examined network part is changed so as to observe the consequential effects, using the basic network architecture presented in the previous chapter and the subsequently described hyperparameters. As an instance, when conducting experiments regarding network depth, the number of layers is changed, whereas all other network aspects remain unchanged. Figure 5.1 illustrates the architecture used for the experiments: the downsampling path follows the VGG 16 network [115], as described by Long et al. [77], and sets up a bi-pyramid of convolutional pooling blocks, where with decreasing feature map size the number of filters rises. Imitating the architecture of a standard research network enables the transfer of pre-trained parameters, which increases network performance, and thus the first four layers are initialised with VGG 16 weights while the remaining ones are initialised randomly, following He et al. [43]. The upsampling path is specifically designed to meet the challenges of the data: since photoluminescence images are composed comparably simple, each upsampling stage covers only one convolutional layer with 64 kernels, so as to keep the number of parameters small and prevent overfitting. Moreover, in order to achieve pixel-level resolution, the upsampling path mirrors the downsampling path and both parts are consistently connected via skip connections. Upsampling is performed by means of bilinear interpolation followed by a convolutional operation. Then, the upsampled, coarse information is fused by skip connections with fine-grain local information from the corresponding shallow layer in the downsampling path. For this purpose, the number of feature maps in the last layer of a downsampling block is reduced to 64 with $1 \times 1$ kernels. Subsequently, the feature maps of both paths are added and a ReLU activation function is applied. Residual shortcuts, which are implemented in every three-layer convolutional pooling block, minimise the degradation problem by widening the network. Furthermore, batch-normalisation layers ease network training by dissolving a layer's dependence on the data distribution of the previous layer's output. When the original image dimensions are restored, the number of feature maps is reduced to three, according to the number of class categories. Finally, a softmax function is applied so as to obtain a network output of three probability maps, which may be subsumed into one image by keeping only the maximum argument of each pixel.

Using the described architecture, first the number of layers in the downsampling path was examined. Table 6.1 lists four different versions of the aforementioned architecture, where the first version (*Standard*) incorporates the full VGG 16 architecture into the

downsampling path. Note that the two penultimate layers, each with 4,096 filters, are originally fully-connected layers that have been transformed into 4096×1×1 convolutional layers. The last layer in the downsampling path serves as link between the downsampling and the upsampling part. All architectures have been trained for 200 epochs with a learning rate of $8 \cdot 10^{-4}$ and an L2-regularisation strength of $5 \cdot 10^{-4}$. The *Standard* architecture achieved a validation accuracy of 98.7 % and a training accuracy of 99.0 %. Thus, the network started to overfit to the training data, which indicates that the network complexity is sufficient for the dataset. Proceeding from the VGG 16 architecture, three additional versions with a reduced number of layers were tested, where the number of layers deep in the downsampling path was reduced gradually. Architecture 2 (*Fleming*), employs only two layers in the sixth stage, with 512 and 64 filters, respectively. The last two architectures 3 (*Vaughan*) and 4 (*Broomstick*) completely omit the sixth stage and accordingly the first upsampling stage. For the sake of completeness, the analysis also covers a *U-Net* architecture, which was developed by Ronneberger et al. [101] for the segmentation of medical images but is meanwhile used for a wide range of applications [18, 55, 69, 140]. As illustrated in figure 4.3 and table 6.1, *U-net* features a different network architecture, with a shallower downsampling path that is mirrored by the upsampling path in full and thus covers a larger number of kernels in total.

Table 6.1.: Overview over the four tested architecture variants. The *Standard* architecture (network 1, see figure 5.1) follows the VGG 16 setup in its downsampling path, while network 2 (*Fleming*) is equipped with an abbreviated last stage of only 512 and 64 feature maps, respectively. Network 3 (*Vaughan*) and network 4 (*Broomstick*) completely omit the last downsampling stage and accordingly the first upsampling stage, where network *Broomstick* employs only 512 and 64 feature maps in the last two downsampling layers, respectively. The last layer outputs three probability maps, one for each class category.

| layers | architecture | | | | |
|---|---|---|---|---|---|
| | 1: Standard | 2: Fleming | 3: Vaughan | 4: Broomstick | 5: U-Net [101] |
| conv1 | 64, 64 | 64, 64 | 64, 64 | 64, 64 | 64, 64 |
| conv2 | 128, 128 | 128, 128 | 128, 128 | 128, 128 | 128, 128 |
| conv3 | 256, 256, 256 | 256, 256, 256 | 256, 256, 256 | 256, 256, 256 | 256, 256 |
| conv4 | 512, 512, 512 | 512, 512, 512 | 512, 512, 512 | 512, 512, 512 | 512, 512 |
| conv5 | 512, 512, 512 | 512, 512, 512 | 512, 512, 64 | 512, 64 | 1024, 1024 |
| conv6 | 4096, 4096, 64 | 512, 64 | - | - | - |
| up1 | 64 | 64 | - | - | - |
| up2 | 64 | 64 | 64 | 64 | 512, 512 |
| up3 | 64 | 64 | 64 | 64 | 256, 256 |
| up4 | 64 | 64 | 64 | 64 | 128, 128 |
| up5 | 64, 3 | 64, 3 | 3 | 3 | 64, 64, 3 |

The training results of the five architectures have been summarized in table 6.2. Here, the network performance is evaluated with regard to the validation dataset, using pixel accuracy, the averaged class-wise pixel accuracy and the accuracy of the defect class, with:

- pixel accuracy: $\sum_i p_{ii} \,/\, \sum_i t_i$
- mean pixel accuracy: $(1/n_c) \sum_i p_{ii} \,/\, t_i$
- defect-class accuracy: $p_{dd} \,/\, t_d$

where $p_{ji}$ is the amount of pixels of class $j$ predicted to belong to class $i$, $t_i = \sum_j p_{ij}$ is the total number of pixels in class $i$, $n_c$ is the number of classes, $p_{dd}$ are the true positives of the defect class and $t_d = \sum_j p_{dj}$ is the total number of pixels in the defect class $d$.

Table 6.2.: Pixel accuracy (PA), mean pixel accuracy (MPA) and defect-class accuracy (DCA) of the five network architectures *Standard*, *Fleming*, *Vaughan*, *Broomstick* and *U-net* [101] as described in table 6.1.

| | PA | MPA | DCA |
|---|---|---|---|
| 1 Standard | 98.7 | 84.1 | 53.7 |
| 2 Fleming | 98.7 | 84.1 | 53.8 |
| 3 Vaughan | 98.7 | **84.3** | **54.2** |
| 4 Broomstick | 98.7 | 83.8 | 52.9 |
| 5 U-net | 97.8 | 72.2 | 17.6 |

Analysing the network performances in table 6.2 reveals that regardless of the number of layers in the downsampling path, all variants of the proposed architecture design achieve the same validation pixel accuracy. However, with regard to mean pixel accuracy and defect-class accuracy the *Vaughan* network slightly outperforms the other architectures, indicating that the most suitable number of layers lies in the middle of the tested variations. The *U-Net* architecture, in contrast, yields the lowest performance, indicating that the deeper upsampling path is less beneficial with regard to photoluminescence images. Incidentally, the consistent implementation of residual shortcuts in every three-layer block of the other architectures enables them to adjust the number of employed kernels themselves—hence, the other networks may approximate *U-Net's* shallower downsampling path by skipping the middle layer of each three-layer block.

Figure 6.1 visualises the accumulated validation metrics over the training progress of the best performing architecture (*Vaughan*) in addition to the deepest network (*Standard*) and the most shallow network (*Broomstick*). Note, that the additional subplots depict each metric with a logarithmically scaled x-axis, in order to highlight the early training phase. Moreover, the plots illustrate validation accuracy as solid line and the difference

between training and validation accuracy as area over the line, where a value was recorded every 50 steps. In the left panel, the accumulated pixel accuracy with respect to the training steps is displayed. At first appearance, it seems as if all networks perform equally well and no more progress is made after the first, drastic increase. However, plotting the pixel accuracy over a logarithmic scale reveals differences in the initial training phase: even though all three architectures were initialised equally, the initial pixel accuracy varies from about 30 % (*Broomstick*) to nearly 80 % (*Vaughan*), whereas the training accuracies start much closer together, in the range between 83 % and 85 %. With ongoing network training, all architectures converge to 98.7 % validation accuracy.

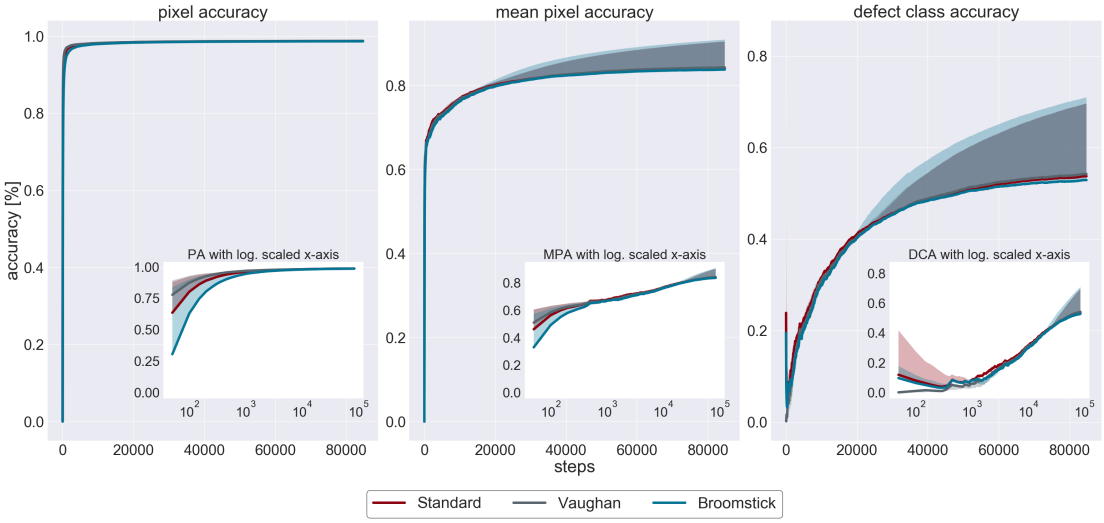



Figure 6.1.: Visualisation of validation (solid line) and training (area over the line) metrics of the differently deep network architectures *Broomstick* (shallow), *Vaughan* (medium) and *Standard* (deep). Left: pixel accuracy, middle: mean pixel accuracy, right: defect-class accuracy. The subplots show the same metrics with respect to a logarithmically scaled x-axis, in order to highlight the initial training phase. The visualisation illustrates how different the learning progresses even though all architectures achieve similar accuracies, eventually. The *Broomstick* architecture, which yields the worst performance, also shows the highest difference between training and validation accuracies.

Analysing the two remaining metrics, mean pixel accuracy and defect-class accuracy, shows that all networks do in fact proceed to learn, even though pixel accuracy stagnates. Note that the distinctive difference between pixel accuracy and the other two metrics is caused by the accuracy paradox: due to the unbalanced class categories, a high prediction accuracy can be achieved by always predicting the labels of the two majority classes, in-spec and miscellaneous. The logarithmically scaled subplots illustrate the variances in the learning process, depending on the architecture. While the mean-pixel-accuracy plot resembles the aforementioned pixel-accuracy plot, the architectures

behave differently with regard to defect-class accuracy. Here, the eventually best performing architecture, *Vaughan*, starts with the lowest accuracy. Contrarily, the other two architectures start with a higher defect-class accuracy that distinctively drops after the initial steps. This behaviour can be explained by the random initialisation of the deepest layers. As a result, some of the randomly initialised output pixels correlate to the true class category. However, the network first learns to distinguish non-wafer and wafer areas before recognising more fine-grained structures and thus the initially high defect-class accuracies drop before increasing again. In sum, the *Vaughan* network shows the most robust learning progress and finally yields the highest performance of all evaluated network architectures. Still, with just slightly deviating validation metrics it appears as if the number of layers barley influences the resulting network performance. Therefore, the impact of skip connections and residual shortcuts is examined, which create additional pathways and thus enable the network to bypass redundant layers.

### 6.1.1. Residual Shortcuts and Skip Connections

Fully convolutional networks employ cascades of filters, which analyse the image features and eventually segment the image into the given class categories. However, not every image feature requires the same amount of processing and thus residual shortcuts are implemented, which enable the network to individually choose the degree of processing for each feature. Skip connections, on the other hand, fuse shallow layers of the downsampling path with deeper layers in the upsampling path and by doing so refine the output resolution. The influence of residual shortcuts and skip connections can be observed by removing both before training the network for 200 epochs, using the same hyperparameters as before. Now, the shallow *Vaughan* architecture's defect-class accuracy drops from 54.2 % to 30.7 %, whereas the deeper *Standard* network yields only 0.01 % defect-class accuracy, compared to 53.7 % before. This indicates that the last layers of the *Standard* network do not contribute to the classification and are mostly bypassed by the skip connections and the residual shortcuts. Figure 6.2 visualises pixel accuracy, mean pixel accuracy and defect-class accuracy of the *Standard* architecture without residual shortcuts and skip connections (blank), with residual shortcuts but without skip connections (no skips) and with residual shortcuts and gradually added skip connections (1 - 5 skips), starting by fusing the deepest layers. It becomes apparent that adding two skip connections enables the network to skip the deepest layers and distinctively increases network accuracy. The highest accuracy in all three metrics is achieved with four skip connections, where all but the input and output layer are fused, while adding a fifth skip connection results in a slight drop in accuracy.

Removing skip connections and residual shortcuts not only prevents the network from skipping redundant layers but also exacerbates parameter updates. During backpropagation, the gradients of a sequential network pass all layers from the network output back to the first layer, which may result in vanishing gradients caused by the chained calcula-

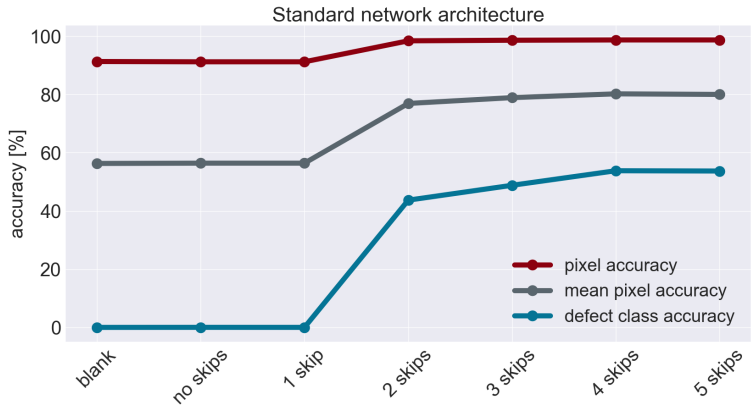



Figure 6.2.: Influence of residual shortcuts and a varying number of skip connections on pixel accuracy (red), mean pixel accuracy (grey) and defect-class accuracy (blue). While residual shortcuts alone (denoted as *no skips*) and one skip connection barely influence network performance, the implementation of two skip connections distinctively increases all three metrics, with a peak at four skip connections. Note that figure 6.3 further examines the impact of residual shortcuts. Because skip connections refine the spatial resolution of the network output they have the greatest influence on defect-class accuracy. The details and results of the plot have also been summarized in table B.1

tion. Skip connections, on the other hand, allow the gradients to bypass several layers, resulting in a more direct parameter update which eases network training. Figure 6.3 illustrates this effect, by plotting pixel accuracy, mean pixel accuracy and defect-class accuracy of a *Standard* architecture without residual shortcuts and skip connections (red line), with residual shortcuts but without skip connections (grey line) and with residual shortcuts and two skip connections (blue line). Note that the first two architectures have been trained for 400 instead of 200 epochs and with a higher learning rate. On the one hand, the distinctive increase in network performance due to the two skip connections becomes apparent, especially with respect to defect-class accuracy. Incidentally, the fact that all networks achieve over 90 % pixel accuracy despite the low defect-class accuracy is a consequence of the aforementioned accuracy paradox. On the other hand, the plots illustrate that networks without residual shortcuts and skip connections require more training steps and a higher learning rate. Moreover, even though the previous plot (6.2) showed no difference between a network with and without residual shortcuts, the longer training duration reveals that residual shortcuts indeed ease training and yield a faster increase in accuracy. In sum, residual shortcuts and skip connections widen the network with additional paths that allow the algorithm to bypass redundant layers and thus ease network training. As a result, the number of layers employed in the downsampling path only slightly influences the resulting accuracy, even though more suitable network architectures, such as the *Vaughan* network, demonstrate a more stable training behaviour.

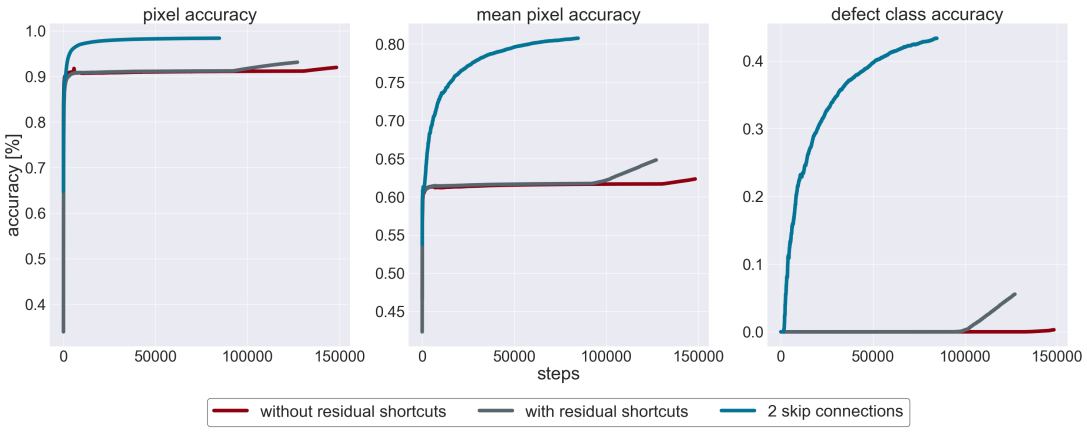



Figure 6.3.: Influence of residual shortcuts on pixel accuracy (left), mean pixel accuracy (middle) and defect-class accuracy (right). The plot compares networks trained without residual shortcuts (red), with residual shortcuts (grey) and with residual shortcuts and two skip connections (blue). Note that the latter was trained for 200 epochs while the first two networks were trained for 400 epochs and with a higher learning rate. The plots reveal how residual shortcuts and skip connections increase network performance, where two skip connections distinctively ease network training.

### 6.1.2. Filter and Feature-Map Visualisation

Neural-network algorithms are composed of millions of self-learned, interacting parameters and non-linearities and have thus long been considered as black boxes, where it is not clear what the intermediate layers compute. Meanwhile, various methods for the visualisation of convolutional layers have been developed [27, 82, 87, 133]. These methods allow an interpretation of what kind of filters the network has learned and thus a more founded assessment of the architecture. In order to visualise network filters, the trained network is turned upside down, so to speak, with a random noise image as input. Then, the derivatives of the chosen filter are used to enhance the image in such a way that the activations of the filter's neurons are maximised. In other words, an image is created that shows which kinds of patterns the filter has learned to recognise. Based on the aforementioned results, all following analyses were conducted on the *Vaughan* network. Figure 6.4 depicts a selection of filters of the first two convolutional layers (layers 1_1 and 1_2), each of which covers 64 filters in total.

As is typical for convolutional neural networks [133], filters in the first layer recognise simple geometrically patterns as well as colours and shades of grey, respectively. However, some filters displays noisy patterns, such as the third filter of layer 1_1, which result in all-zero feature maps. The regular occurrence of these *dead* filters can be an indicator

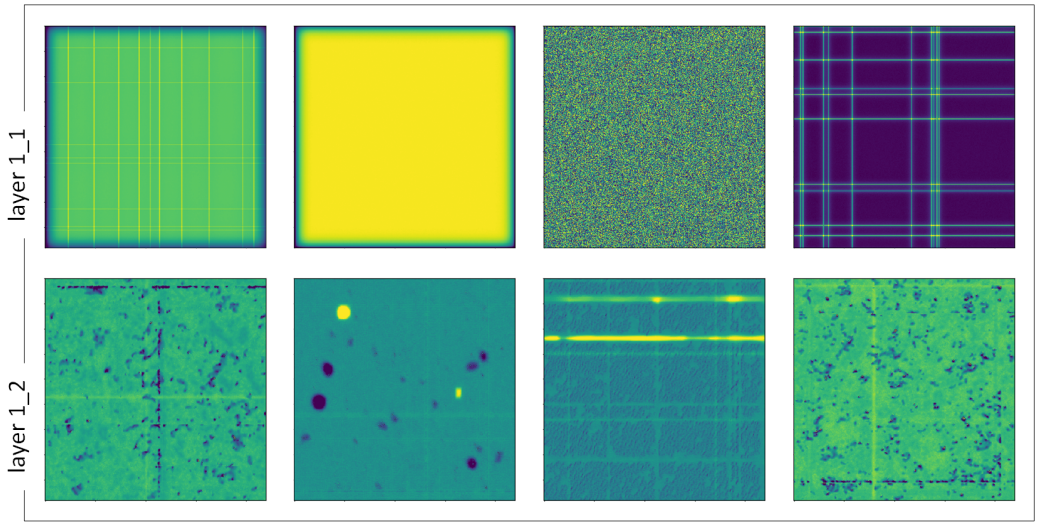



Figure 6.4.: Visualisation of patterns recognised by the network's learned filters, where the first row corresponds to the first network layer (layer 1_1) and the second row to the second layer (layer 1_2) of the first convolutional block. Filters of the first layer commonly recognise simple geometrical patterns, colours and shades of grey, respectively. The third filter shows a noisy pattern, indicating a so-called *dead* filter, which generates all-zero feature maps. The second layer inputs the pre-processed feature maps of the first layer and recognises more complex patterns, which combine various features such as blobs and lines.

of badly tuned hyperparameters [30] or unused filters in a residual module. Accordingly, about half the filters of the bypassed layers 3_2 and 4_2 show noisy patterns, contrary to a minority of filters in the other layers. The second row in figure 6.4 depicts a sample of filters of the second layer 1_2, which inputs the feature maps of the first layer. As elaborated in chapter 3, with increasing network depth each layer's receptive field increases and the filters recognise more complex patterns. These effects can be clearly seen in networks that analyse everyday scenes, as shown by Olah et al. [87], where deep filters display dog snouts, eyes and buildings. Defect structures and wafer patterns, however, are less noticeable and thus the filters are less interpretable. Still, figure 6.4 shows that the second layer's filters are less clean than those of the first layer and seem to combine various patterns, such as lines and blobs. Analysing the corresponding feature maps in figure 6.6 illustrates the effects of both layers: the depicted output map of layer 1_1 emphasises changes in brightness, whereas the output of layer 1_2 elevates in-spec areas by highlighting ascending and descending edges, causing a 3D like appearance.

Figure 6.5 displays a filter example for each of the nine remaining layers. Here, the filters show varying structured patterns, partly interspersed with blobs that might correspond to void structures as well as granular structures, which might relate to single defective

chips or OCR chips. While an increasing complexity with increasing network depth is not noticeable, the patterns show a higher frequency. As mentioned before, layer 3 and 4 are residual modules, where the middle layers 3_2 and 4_2 are bypassed by a residual shortcut. As a result, half of the filters in layer 3_2 and over 3/4 of the filters of layer 4_2 show noisy patterns. Interestingly, only a minority of the subsequent layer's filters are dead, even though these filters could be bypassed by the skip connections. This indicates, in addition with the drop in network accuracy of the shallow *Broomstick* network, that the network benefits from a self chosen number of deep layers and that residual shortcuts and skip connections complement each other. Analysing the sixth layer of the *Standard* network architecture supports this observation. Here, layers 6_2 and 6_3 display mostly dead filters, even though layer 6 is not a residual module, which indicates that the layers are nonetheless bypassed by the skip connections.

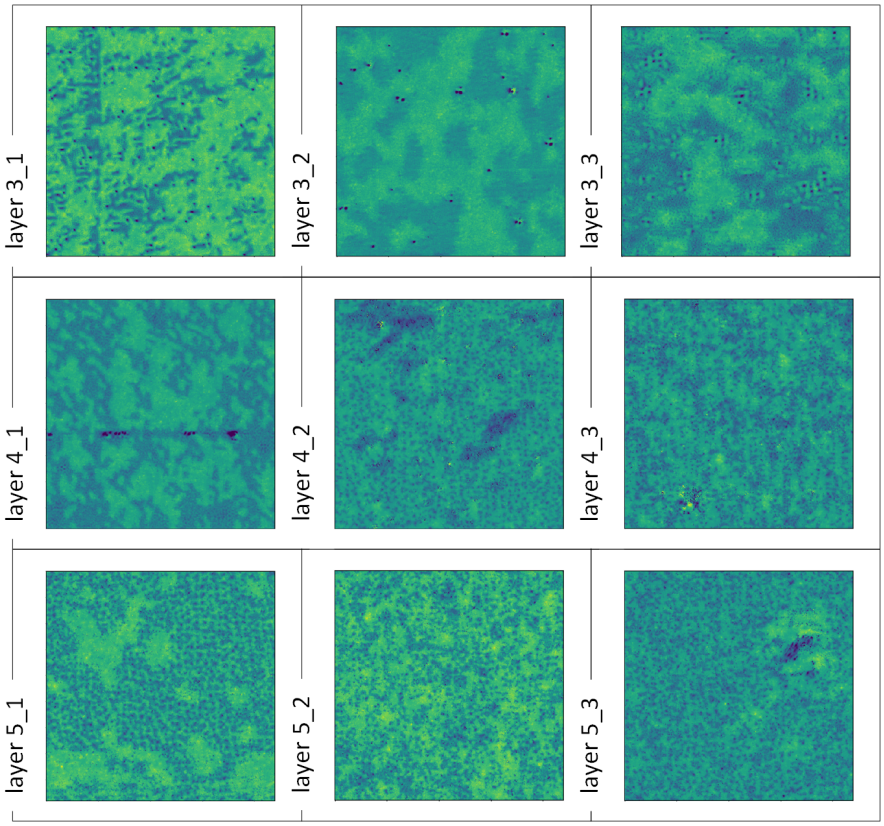



Figure 6.5.: Visualisation of patterns recognised by the network's deep layers. Contrary to filters in the first two layers, deep filters depict overall less variability and the different textures cannot easily be attributed to wafer structures.

Figure 6.6 illustrates how the input image is processed by the network; the corresponding input and label images are shown in figure 2.4. Note that the feature-map dimensions are reduced after each convolutional block via maxpooling, which manifests in the loss of

resolution. The input and output images are of size $442 \times 440$ and the smallest feature maps in the fifth convolutional block are of size $28 \times 28$. Analysing the selected feature maps reveals how the network extracts features from the input image: in the first few layers, the in-spec wafer area is emphasised in contrast to background, functional and defective areas. With increasing network depth, the feature maps show more diversity and maps that highlight the miscellaneous class (3_3), in-spec chips (3_1) or the negative of the in-spec class (3_2) appear. Moreover, defect and functional structures are emphasised (3_1). While similar feature maps can be observed in the fourth layer, albeit with less resolution, feature maps of the fifth layer are less interpretable because the information is too highly condensed. As elaborated in chapter 3, the neuron's effective receptive field, that is the area in the input image the neuron is connected to through the previous layers, increases with every layer and pooling operation. As a result, each pixel in the last layer's feature maps represents an evaluation over a $196 \times 196$ area in the input image. After downsampling, the original image resolution is restored via upsampling, illustrated by the four feature map images *up_1* to *up_4*. Finally, the output layer is reduced from 64 to 3 maps, where each map outputs the probabilities of one class category. The presented output map depicts the defect class and combines the evaluations of several previous feature maps. Interestingly, while many feature maps can be associated with one class category alone, others are a combination of class categories. As an instance, the feature map sample *up_4* covers the background as well as salient defect structures. Nonetheless, the defect-class output map displays a clear segmentation of the defective areas alone.

In sum, this section examined how design choices influence network performance, starting with network depth. Here, four differently deep network architectures were compared, where the second shallowest *Vaughan* architecture slightly outperformed the remaining networks. The only slight differences in network performance were then traced back to the thorough implementation of residual shortcuts and skip connections. Studying the effects of skip connections on the *Standard* architecture revealed that two skip connections already ease network training distinctively and thus increase network performance, where an architecture with four skip connections outperformed the remaining variants. Even though residual shortcuts did not have an obvious influence on network performance, the analysis of training progress with and without residual shortcuts showed that residual shortcuts in fact ease network training. Moreover, investigating the network's learned filters exposed a high number of so-called dead filters in bypassed layers in contrast to a minority of dead filters in the remaining layers, indicating that the network does use residual shortcuts to adjust the number of utilised filters. Finally, studying visualised network filters and feature maps of intermediate layers allowed an interpretation of what kind of filters the network has learned and how the network processes the input images. The following section studies the influence of hyperparameter tuning on network performance, using the aforementioned *Vaughan* network with residual shortcuts implemented in every three-layer convolutional block as well as three skip connections, which connect the corresponding inner layers of downsampling and upsampling path.

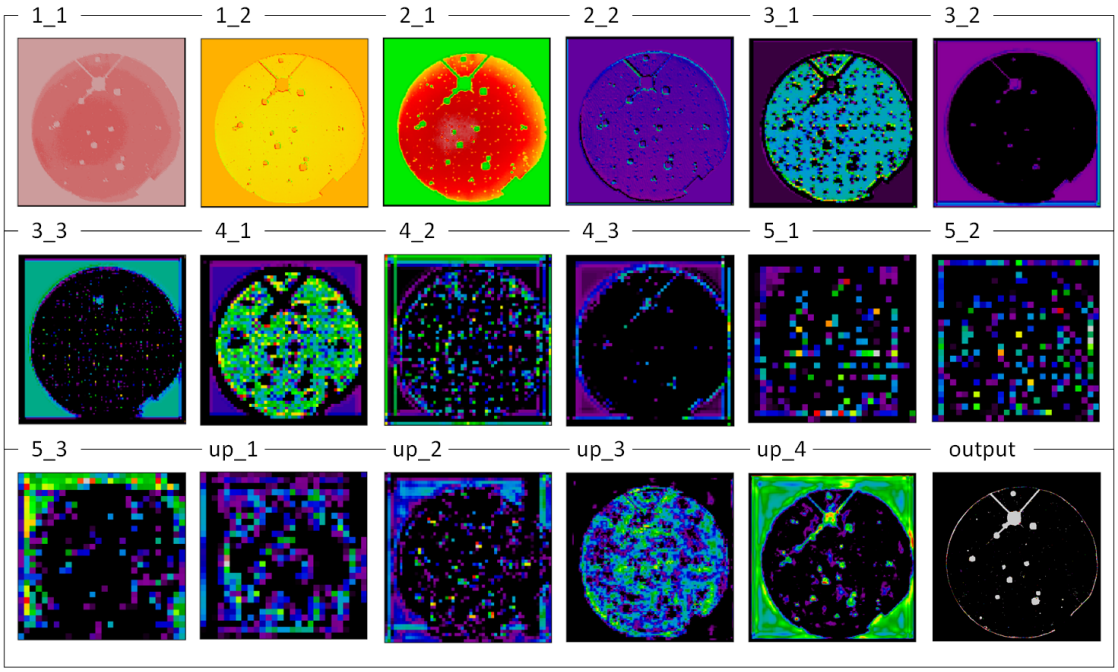



Figure 6.6.: Visualisation of the network's feature maps. The depicted images represent the transformation of the input image while being processed by the network. Note that the image resolution decreases with every downsampling layer block (2-5) and increases again with every upsampling layer (up 1-4). It becomes apparent that with increasing network depth the information gets increasingly compressed up to the point where the feature maps are no longer interpretable. Furthermore, the feature maps illustrate how the network collects evidence for each class category.

## 6.2. Hyperparameter Tuning

The previous section explored the effects of architectural design choices, more precisely how the number of layers influences network performance and how residual shortcuts and skip connections ease network training. Subsequently, filter and feature map visualisation were used to investigate which patterns the network has learned to recognise and how feature maps develop throughout the network. Next to the network architecture, the network's performance depends strongly on hyperparameters, where due to the high amount of hyperparameters and their mutual influence onto each other it is common to rely on best practices, such as employing the ReLU activation function or updating the parameters using the Adam optimiser [30, 39]. Other hyperparameters, including the learning rate or model complexity, depend strongly on the dataset and therefore must be determined individually. This section studies various significant hyperparameters, starting with the resizing operation in the upsampling layer. Then, parameter initialisation

using transfer learning, the influence of the learning rate as well as the regularisation of the model complexity are investigated. Finally, the impact of different parameter update methods is examined, followed by weighted loss calculation and the usage of ultrasonic-embedded photoluminescence images to mitigate input-label mismatches.

### 6.2.1. Resizing Operation

Fully convolutional networks filter the images so as to extract their core features and determine each pixel's class category. While the downsampling path is used to extract semantic meaning, the upsampling path serves as a means to restore the spatial information. Chapter 3 introduced two upsampling methods: on the one hand, the feature maps can be upsampled using a predefined interpolation function, such as nearest neighbour, bilinear or bicubic interpolation, before applying convolution and a ReLU activation function. By doing so, upsampling is initialised by the interpolation method and the network is trained to further adjust the reconstructed signal with learnable filters. On the other hand, the convolutional operation can be reversed via transposed convolutions, where the network learns the upsampling from scratch. Table 6.3 compares pixel accuracy, mean pixel accuracy and defect-class accuracy of the aforementioned *Vaughan* network, with varying upsampling operations. As before, all network variations achieve similar pixel accuracies, while mean pixel accuracy and defect-class accuracy show more diversity. Interestingly, the combination of a fixed interpolation method with a subsequent adjustment outperforms the purely self-learned transposed convolution. Of all tested methods, bilinear interpolation achieves the highest mean pixel accuracy and defect-class accuracy, followed by nearest neighbour interpolation. As shown in figure 3.8, bilinear interpolation represents a trade-off between nearest neighbour interpolation, which yields a piecewise signal reconstruction, and bicubic interpolation, with a rather blurry result. Thus, bilinear interpolation is used as upsampling method in the following network studies.

Table 6.3.: Performance metrics of network architectures with different upsampling operations. The table compares performance by pixel accuracy (PA), mean pixel accuracy (MPA) and defect-class accuracy (DCA) on the validation dataset. While all upsampling operations achieve similar pixel accuracies, bilinear interpolation slightly exceeds the other interpolation methods with regard to mean pixel accuracy and defect-class accuracy. The networks were initialised randomly and trained for 200 epochs.

| upsampling operation | PA | MPA | DCA |
|---|---|---|---|
| transposed convolution | 98.5 | 82.7 | 49.8 |
| bilinear | 98.6 | **83.4** | **51.4** |
| nearest neighbour | 98.6 | 83.3 | 51.3 |
| bicubic | 98.6 | 82.9 | 50.0 |

### 6.2.2. Activation Function

From a mathematically standpoint, neural-network algorithms are chained non-linear functions, where the activation function introduces the non-linearity to the calculation. Currently, the ReLU activation function counts as best practise, due to its efficiently to compute and consistent gradients, which diminish the vanishing gradient problem. Furthermore, the ReL unit's hard threshold causes sparse representations, whose advantageousness depend on the network setup and dataset. Figure 6.7 illustrates the influence of different activation functions on the validation defect-class accuracy, namely ReLU, leaky ReLU, softplus and tanh. While the classical tanh activation function initially achieves the highest performance, it starts to level off after about 10,000 training steps, presumably due to vanishing gradients, yielding a final defect-class accuracy of 41.4 %. Furthermore, enabling dense representations by using softplus or leaky ReL units does not benefit network performance, where softplus achieves the lowest accuracy with 40.3 % followed by leaky ReLU with 41.3 % defect-class accuracy. Even though ReL units initially lag behind tanh activation functions, they eventually yield the highest performance with 43.9 %.

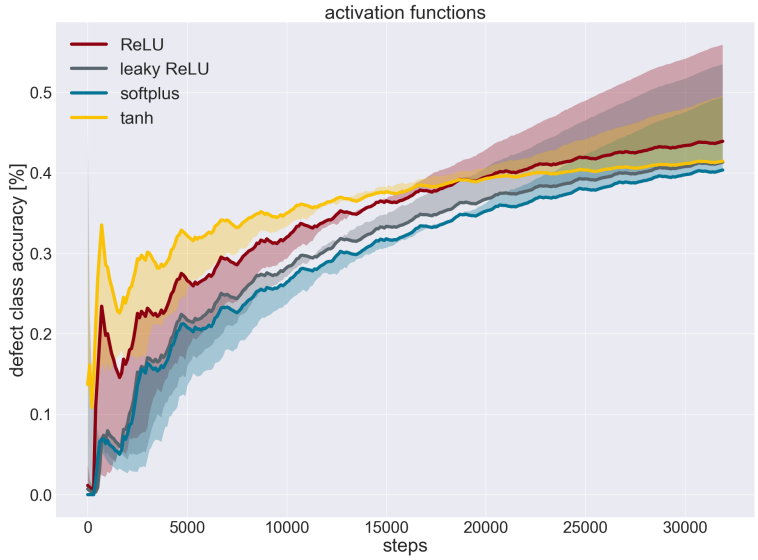



Figure 6.7.: Comparison of different activation functions, namely ReLU (red), leaky ReLU (grey), softplus (blue) and tanh (yellow), where ReL units eventually yield the highest defect-class accuracy. The details and results of the plot have also been summarized in table B.2.

### 6.2.3. Network Initialisation

The strength of neural-network algorithms lies in the trainable weights and biases, which are adjusted to the training dataset via backpropagation. Various studies have examined the influence of parameter initialisation on the training progress, due to the necessity of varying initial parameter values [37, 39, 43]: initialising all parameters with the same value would result in equal parameter updates and thus inhibit any learning. The common practice for convolutional and fully convolutional networks is the so-called He initialisation. Here, the parameters are initialised with random values, drawn from a Gaussian distribution with a standard deviation of $\sqrt{2/N}$, where $N$ is the number of incoming nodes [43]. Assume, as an instance, a previous layer of 64 feature maps and $3 \times 3$ convolutional kernels, then the number of incoming nodes is calculated as $N = 9 \cdot 64 = 576$.

Another method of parameter initialisation is transfer learning, as described in chapter 3. Here, the network is initialised with parameter values obtained by previous training on a very large research dataset. During network training, the parameter values are merely finetuned to the actual dataset, using a comparably small learning rate. The idea behind transfer learning is to exploit the fact that all images are inherently composed of similar patterns, such as light-dark transitions and basic shapes. However, with increasing network depth the complexity and differentiation of the learned filters increases and transferability decreases. Therefore, the number of layers initialised with transferred weights must be adjusted to the dataset. Table 6.4 summarises the conducted experiments: first, all network layers were initialised randomly, following He et al. [43]. Then, parameter values trained on the ImageNet dataset with a VGG 16 network were transferred into the first two, four, seven and ten layers, respectively [104, 115]. The table lists similar results for all initialisation methods, except defect-class accuracy, where He initialisation yields the lowest accuracy with 51.4 %, whereas all VGG initialisations achieve over 53 %. This result indicates that network performance benefits from pre-trained weights. Moreover, the performance peaks when transferring pre-trained parameters to the first four layers only, suggesting that the transfer of unsuitable parameters slightly hinders the learning process in comparison to He initialised parameters. Thus, in the following experiments the first four network layers are initialised with pre-trained weights and the remaining layers are initialised randomly.

### 6.2.4. Learning Rate

Network training updates the parameter values via backpropagation into the direction of the loss function's minimum. The exact update value is determined by each parameter's gradient in combination with the learning rate, with $\theta' = \theta - \eta \nabla \theta$, where $\theta$ is the parameter and $\eta$ is the learning rate. Tuning the learning rate to a suitable value distinc-

Table 6.4.: Performance metrics of differently initialised network architectures. Here, He denotes the random initialisation of all parameter values, following He et al. [43], whereas VGG_x stands for the transfer of pre-trained values into the $x$ first network layers and He initialisation of the remaining layers. The table compares performance by pixel accuracy (PA), mean pixel accuracy (MPA) and defect-class accuracy (DCA) on the validation dataset. While similar values are reached in pixel accuracy, defect-class accuracy can be increased by about 3 % with VGG 4 initialisation compared to He initialisation.

| initialisation method | PA | MPA | DCA |
|---|---|---|---|
| He | 98.6 | 83.4 | 51.4 |
| VGG 2 | 98.7 | 84.0 | 53.1 |
| VGG 4 | 98.7 | **84.3** | **54.2** |
| VGG 7 | 98.7 | 84.1 | 53.8 |
| VGG 10 | 98.7 | 84.2 | 53.8 |

tively influences network performance: if the learning rate value is too high, then the loss function's minima may be overstepped and the algorithm may fail to converge or even eventually diverge, whereas a very small learning rate requires unreasonably long training durations [30, 46]. It is therefore common to choose a higher initial learning rate, which is reduced during network training. One way to do so is with an exponentially decaying learning rate, as described in chapter 3. Figure 6.8 visualises pixel accuracy, mean pixel accuracy and defect-class accuracy for various initial learning-rate values, which were exponentially decayed with a decay rate of 0.96 over 200 epochs. To equally cover the full range from $1 \cdot 10^{-1}$ to $1 \cdot 10^{-5}$, random values were determined on a logarithmic scale, with two additional values on the range limits. Note that initial learning-rate values up to $1 \cdot 10^{-15}$ are not unusual, especially with regard to transfer learning and finetuning [125]. However, the results in figure 6.8 reveal that for the given network and dataset learning-rate values around $1 \cdot 10^{-3}$ achieve the best performance, while values at the range limit show a distinctive drop in accuracy from over 50 % defect-class accuracy to about 30 %.

Figure 6.9 visualises how mean pixel accuracy and defect-class accuracy develop during network training for three learning-rate values, namely the lower limit $1 \cdot 10^{-5}$, the best performing value $1 \cdot 10^{-2.62}$ and the upper limit $1 \cdot 10^{-1}$. Both plots reveal that the upper limit learning rate (red line) performs comparably weak in the early training but improves eventually, due to the decaying learning rate. The lower limit learning rate (blue line) shows the opposite development, where the initial increase in defect-class accuracy declines rapidly in conjunction with the decaying learning rate, whereas the intermediate learning rate (grey) converges more slowly and distinctively outperforms the other two learning-rate values. In sum, combining a tendentially too high initial learning with learning-rate decay allows the network to make rapid progress at the early

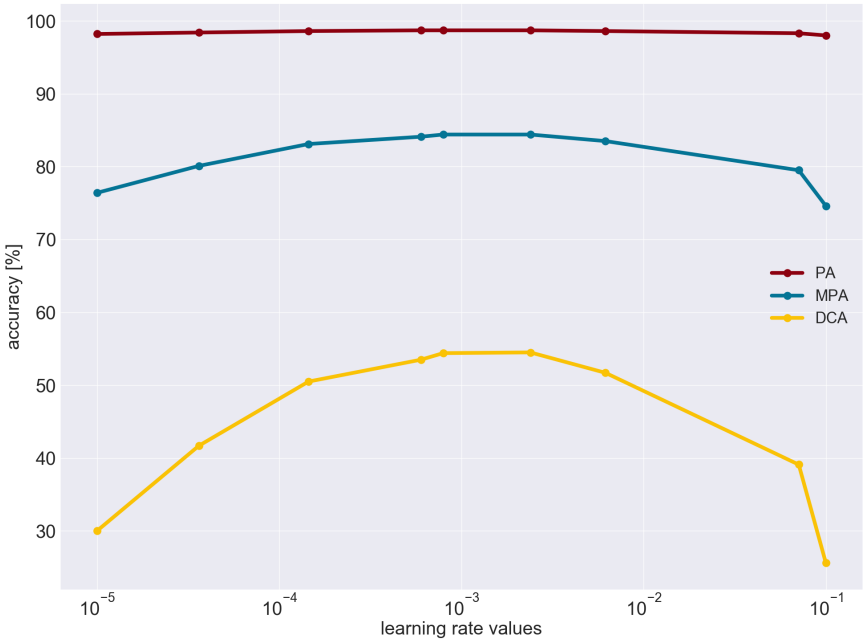



Figure 6.8.: The influence of different initial learning-rate values on network performance, namely pixel accuracy (PA), mean pixel accuracy (MPA) and defect-class accuracy (DCA). The studied values range from $1 \cdot 10^{-1}$ to $1 \cdot 10^{-5}$ and were drawn randomly from a logarithmic scale. It becomes apparent that learning rates around $1 \cdot 10^{-3}$ achieve distinctively higher defect-class accuracies than values at the range limit. The highest accuracy yields an initial learning rate of $1 \cdot 10^{-2.62}$, with 54.5 % defect-class accuracy, in comparison to 25.6 % with a learning rate of $1 \cdot 10^{-1}$. The details and results of the plot have also been summarised in table B.3.

training and eventually converge to the loss function's minimum employing only small parameter updates.

### 6.2.5. L2 regularisation

Neural-network algorithms cover thousands to millions of parameters and are thus prone to overfit, where the algorithm learns a decision boundary that fits perfectly to the training data but does not generalise well to previously unseen data. As a result, validation accuracy levels off, and may eventually diverge, while training accuracy still rises, as shown in figures 6.1 and 6.12. Adding a regularisation term to the loss function that penalises high weight values imposes a preference for simpler network functions on the optimisation process. In other words, the network model's complexity is constrained,

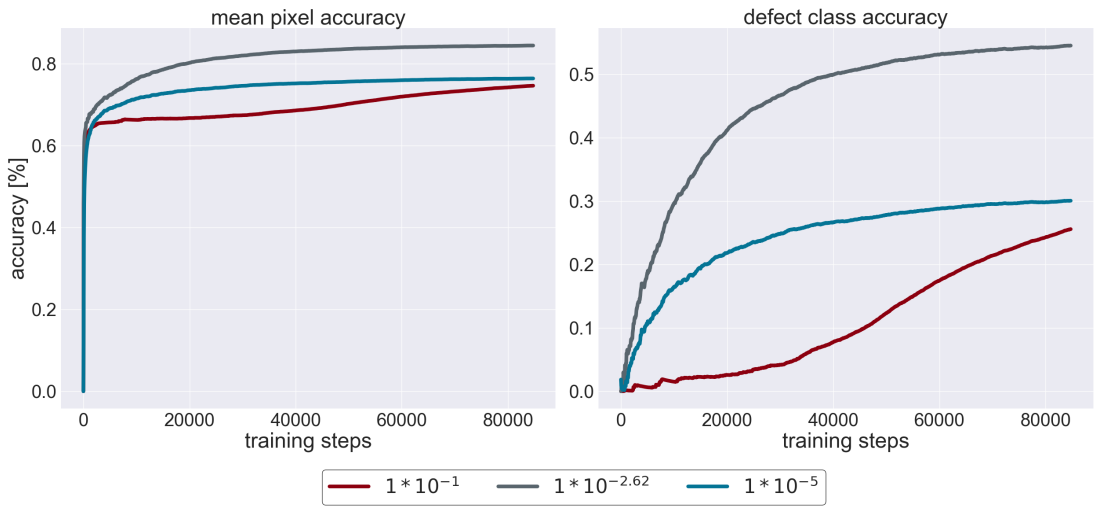



Figure 6.9.: Training progress of a network trained with three different initial learning rates, namely $10^{-1}$ (red), $10^{-2.62}$ (grey) and $10^{-5}$ (blue), which were exponentially decayed during training. The highest learning rate (red) initially shows a slow learning progress but gains performance due to learning-rate decay, whereas the lowest learning rate (blue) starts off better but eventually levels off. The intermediate learning rate (grey) yields fast initial progress with a comparably high learning rate but then converges to the loss function's minimum due to smaller parameter updates caused by the decaying learning rate.

which diminishes overfitting. The left panel in figure 6.10 shows the same pixel accuracy, mean pixel accuracy and defect-class accuracy of a network trained with different learning-rate values as figure 6.8, but in addition visualises the difference between validation and training accuracies. It becomes apparent that the peak in training defect-class accuracy does not correlate with the peak in validation accuracy, indicating that overfitting hinders generalisation. The right panel in figure 6.8 shows the influence of different L2-regularisation strengths in the range from $1 \cdot 10^{-1}$ to $1 \cdot 10^{-5}$ on training and validation metrics on a network trained with a learning rate of $1 \cdot 10^{-2.62}$. The dotted line represents training and validation defect-class accuracy without regularisation and illustrates the decrease of validation and training metrics with increasing regularisation strength. While validation accuracy drops rather slowly in the beginning, L2 regularisation distinctively decreases training accuracy to the point where the training accuracy falls below validation accuracy.

Figure 6.11 illustrates another aspect of L2 regularisation, namely the mutual influence of learning rate and L2 strength on each other. Here, the red and grey plots equal the defect-class accuracies of the previous figure (6.10), where red represents a network trained with

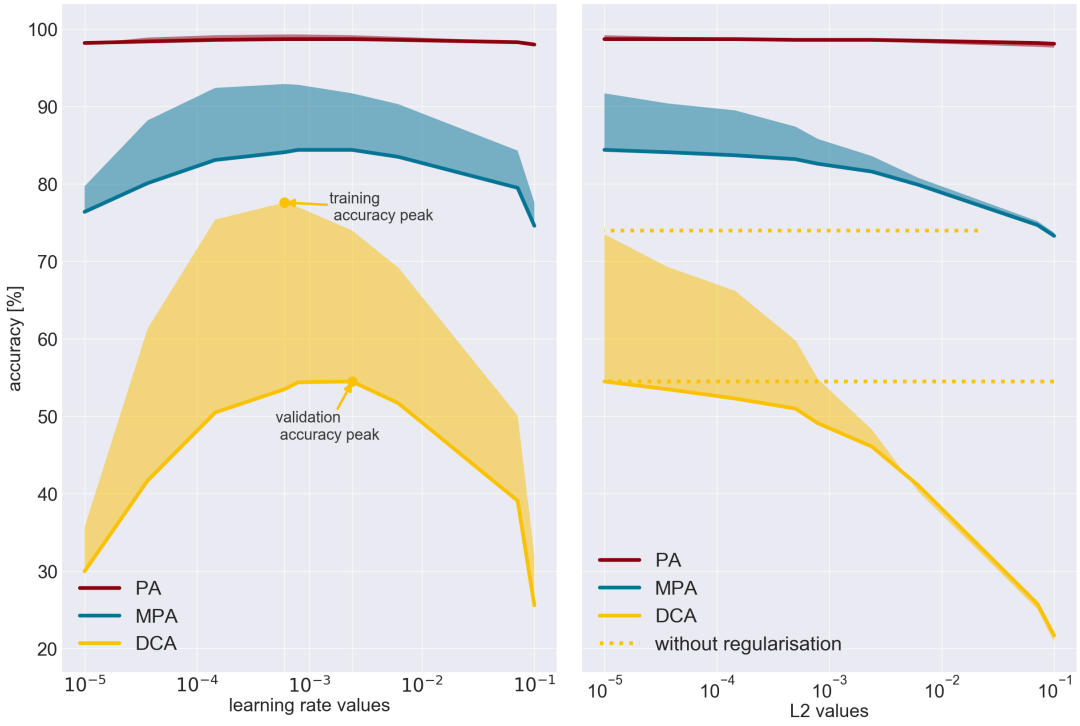



Figure 6.10.: Visualisation of overfit (left) and the influence of L2 regularisation (right). The left panel illustrates the difference between validation accuracy (solid line) and training accuracy (area over the line) for pixel accuracy (PA), mean pixel accuracy (MPA) and defect-class accuracy (DCA) of a network trained with various learning-rate values in the range from $1{\cdot}10^{-1}$ to $1{\cdot}10^{-5}$. It becomes apparent that the peak in training accuracy does not correlate to the peak in validation accuracy, which indicates that overfitting hinders network generalisation. The right panel illustrates the influence of different L2-regularisation strengths in the range from $1 \cdot 10^{-1}$ to $1 \cdot 10^{-5}$ on a network trained with a learning rate of $1 \cdot 10^{-2.62}$. The dotted line marks the unregularised training and validation defect-class accuracy, respectively. It becomes apparent that L2 regularisation mitigates overfitting but restricting the network's complexity exceedingly declines overall network performance. The details and results of the plot have also been summarised in table B.4.

different learning-rate values without regularisation and grey represents the influence of various L2-regularisation strengths on a network trained with a learning rate of $1{\cdot}10^{-2.62}$. The additional blue plot represents the influence of different L2-regularisation strengths on a network trained with a smaller learning rate of $1 \cdot 10^{-3.1}$. In other words, figure 6.11 compares the effect of L2 regularisation on a network trained with two different

learning rates (grey and blue line) in combination with an unregularised network trained with different learning rates (red line). Note that the x-axis corresponds to learning-rate values with respect to the red plot and L2 regularisation values with respect to the blue and grey plots. Thus, the dots in the red plot illustrate validation and training accuracy for both learning-rate values, $1 \cdot 10^{-2.62}$ on the right and $1 \cdot 10^{-3.1}$ on the left, where the latter overfits more strongly. Both networks achieve a comparable validation accuracy of 54.5 % and 54.4 %, respectively. The grey and blue dots correspond to an L2-regularisation strength of $5 \cdot 10^{-4}$ and hence visualise the change in network performance for both networks, when regularised. Here, the higher learning rate (grey) shows a more distinctive reaction, where training and validation accuracy decrease from 74 % to 59.8 % and from 54.5 % to 51.0 %, respectively. The training accuracy of the smaller learning rate (blue) decreases as well, albeit not as strongly, from 77.0 % to 70.1 %. Moreover, validation accuracy peaks at the chosen regularisation strength and achieves a defect-class accuracy of 54.4 %, which is comparable to the non-regularised accuracy of 54.0 %. These results visualise the mutual influence of learning rate and L2 values and the necessity to tune both values with respect to each other.

Figure 6.12 compares the training process of a network trained with low (red, L2 = $1 \cdot 10^{-5}$), medium (grey, L2 = $5 \cdot 10^{-4}$) and strong regularisation (blue, L2 = 0.1). As is to be expected, imposing a strong regularisation (blue line) on the network model's complexity decelerates learning, whereas defect-class accuracy grows steadily. Finally, validation and training defect-class accuracy converge and with ongoing training the network goes into overfitting, yielding a validation accuracy of 44.6 % and a training accuracy of 45.1 %. The red plot illustrates a weakly regularised network, where the highest initial performance is followed by strong overfitting. As a result, validation accuracy levels off with 51.0 % while training accuracy rises further to 74.9 %. Regularising the network with a medium strength (grey line) mitigates overfitting and even though the network learns more slowly at the beginning it catches up after about 10,000 steps and excels the weakly regularised network, yielding a validation accuracy of 53.3 % and a training accuracy of 72.4 %. In sum, adding an L2-regularisation term to the network's loss function diminishes overfitting by restricting the model complexity and thus improves the network's generalisation ability. However, the impact of L2 regularisation on network performance depends on the learning rate as well, which exacerbates the tuning process.

### 6.2.6. Optimiser

The previous two sections studied the influence of initial learning rate, learning-rate decay and regularisation strength on the network performance. However, setting one learning rate value for all network parameters may result in the so-called vanishing gradient problem, where parameters in the shallow layers receive small parameter updates compared to deep layer parameters. Moreover, large updates may oscillate across the loss

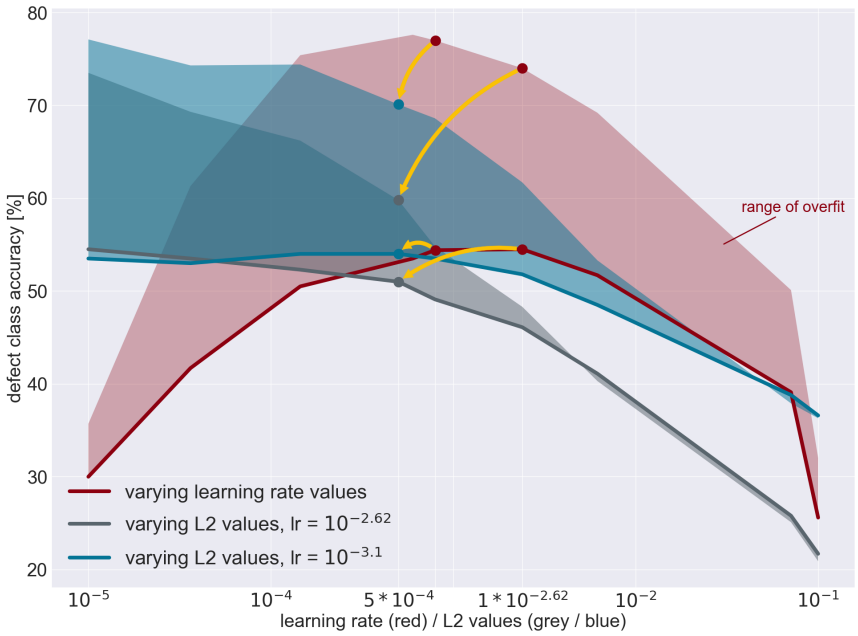



Figure 6.11.: Influence of L2 regularisation on different learning-rate values. The red plot shows validation (solid line) and training (area over the line) defect-class accuracy of a network trained with various learning-rate values and the red dots illustrate the performance of two selected networks trained with a learning rate of $1 \cdot 10^{-2.62}$ (right) and $1 \cdot 10^{-3.1}$ (left), respectively. The blue and grey plots show the influence of different L2-regularisation strengths on the aforementioned networks, where grey corresponds to a learning rate of $1 \cdot 10^{-2.62}$ and blue correlates to a learning rate of $1 \cdot 10^{-3.1}$. It becomes apparent that applying an L2-regularisation strength of $5 \cdot 10^{-4}$ on both networks achieves differing results, where validation and training accuracy of the higher learning rate (grey) are distinctively reduced while the lower learning rate (blue) yields a comparable validation accuracy in conjunction with a decreased overfitting. The details and results of the plot have also been summarized in table B.4 and table B.5.

function's high curvature areas, reversing previous updates and thus hindering the learning process. Chapter 3 introduced various optimisation methods, which aim to address these shortcomings, and figure 6.13 visualises pixel accuracy, mean pixel accuracy and defect-class accuracy of the described methods gradient descent, momentum, AdaGrad, Adam and RMSProp. The solid line illustrates the results of a network trained with an initial learning rate of $10^{-2.62}$, whereas the dotted line represents an initial learning rate of $8 \cdot 10^{-4}$. As could be observed for all previous experiments, pixel accuracy shows only slight variations, while mean pixel accuracy and defect-class accuracy vary distinctively, depending on optimiser method and learning rate. Here, the standard gradient-descent

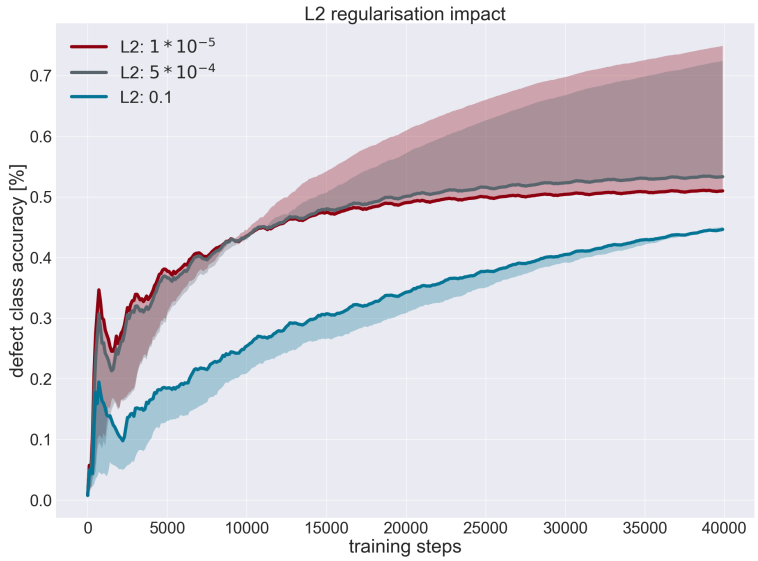



Figure 6.12.: Influence of varying L2 values on the training progress of defect-class accuracy, illustrated by a network trained with low (red, L2 = $1 \cdot 10^{-5}$), medium (grey, L2 = $5 \cdot 10^{-4}$) and strong regularisation (blue, L2 = 0.1). The area over the curve displays the difference between validation and training accuracy. It becomes apparent that implementing a strong regularisation on the network (blue) decelerates the learning progress, whereas a weakly regularised network learns rapidly at the beginning but levels off eventually, due to overfitting. Imposing a tuned regularisation on the network initially mitigates learning but due to reduced overfitting the network catches up and excels the weakly regularised network. The details and results of the plot have also been summarized in table B.6.

update method achieves comparable accuracies for both learning-rate values and underperforms compared to almost all other methods with 35.0 % and 31.0 % defect-class accuracy, respectively. Adding a momentum term to the gradient-descent calculation incorporates previous parameter updates and thus accelerates network training. However, momentum-based learning tends to overshoot the loss function's minimum, which presumably causes the low defect-class accuracy of 33.3 % of the higher learning rate (solid line). The smaller learning rate (dotted line) surpasses all aforementioned metrics with a defect-class accuracy of 40.0 % and thus clearly benefits from the additional momentum term.

Another way to optimise the learning process is by calculating individual learning rates for each parameter, using the AdaGrad algorithm, which achieves a defect-class accuracy of 38.7 % with respect to the higher learning rate. However, a disadvantage of the AdaGrad optimiser is a prematurely slowed down learning progress caused by dividing

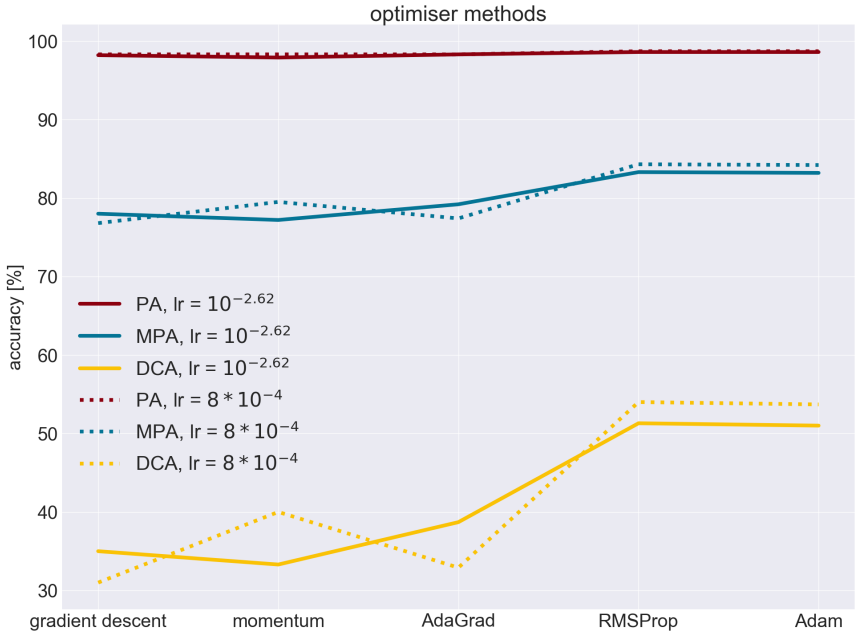



Figure 6.13.: Comparison of optimiser methods with respect to two different learning rates. The figure visualises pixel accuracy (PA, red), mean pixel accuracy (MPA, blue) and defect-class accuracy (DCA, yellow) of various parameter update methods, namely gradient descent, momentum, AdaGrad, RMSProp and Adam, where the solid line represents a higher learning rate of $1 \cdot 10^{-2.62}$ and the dotted line a lower learning rate of $8 \cdot 10^{-4}$. The results show that RMSProp and Adam outperform other parameter update methods. Moreover, the performance of all optimisers depends on the learning rate, where RMSProp and Adam benefit from a lower learning rate. The details and results of the plot have also been summarised in table B.7.

the learning rate by the accumulated squared gradients. This effect especially applies to already low learning rates, as is verified by a defect-class accuracy of only 32.9 %. The RMSProp algorithm dissolves this shortcoming by implementing an exponentially decaying average of the squared gradients, where the influence of stale gradients decays over time and thus a premature stagnation of the learning process is prevented. Figure 6.13 illustrates the distinctive increase in defect-class accuracy for both learning rates, where the lower learning rate yields an accuracy of 54.0 % and hence outperforms the higher learning rate with 51.3 % defect-class accuracy. The Adam algorithm advances the concept of RMSProp and additionally adds a momentum term to the parameter update and is thus the current best-practice method. However, for the given dataset the Adam optimiser achieves slightly lower defect-class accuracies, yielding 51.0 % with respect to the higher learning rate and 53.7 % with respect to the smaller learning rate. In sum, using advanced optimiser methods distinctively increases network performance, where

both, Adam and RMSProp, achieve comparable results and benefit from a lower learning rate.

### 6.2.7. Weighted Loss Calculation

When examining network validation metrics, the difference in sensitivity with regard to network tuning becomes apparent: while pixel accuracy varies around 98.5 % in most experiments, mean pixel accuracy and defect-class accuracy show a broader range of reaction to network changes. This behaviour points to the fundamental imbalance of the data's class categories, where pixels of the miscellaneous and in-spec classes, respectively, distinctively outnumber pixels of the defect class. As visualised in figure 6.2, a network without skip connections does not recognise defect structures but still achieves a pixel accuracy of about 91 %, and even well-tuned network architectures do not exceed a defect-class accuracy of 55 %, due to the accuracy paradox. As indicated by previous experiments, prolonging network training would lead to increased overfit instead of a further rise in validation defect-class accuracy. Thus, the class category's imbalance is equalised by weighting the loss calculation that is used to update the network parameters. For this purpose, a weight map is calculated based on the label image, where pixels corresponding to the defect class are attached a higher weight value than pixels of the remaining two class categories. Figure 6.14, left, visualises pixel accuracy, mean pixel accuracy and defect-class accuracy of a network trained with differently valued defect-class loss weights. In a first experiment, all losses were equally multiplied with 100, thus upscaling the backpropagated loss values. While pixel accuracy rises slightly from 98.5 % to 98.7 %, defect-class accuracy increases distinctively from 41.7 % to 54.2 %. This indicates that upscaling losses improves parameter updates with regard to underrepresented class categories, while the increase in pixel accuracy shows that dominant class categories benefit as well.

Next, higher weights from 200 to 30,000 were added to defect-class losses, while miscellaneous and in-spec-class losses, respectively, were multiplied with 100 again. The distinctive increases in defect-class accuracy and mean pixel accuracy reveal that adding a higher weight to an underrepresented class category changes parameter updates to its benefit. Here, mean pixel accuracy peaks at a defect class weight of 7,500 with an accuracy of 91.4 %, while defect-class accuracy rises continuously to 84.6 %. Pixel accuracy, on the other hand, drops to 92.5 %, thus revealing the increasing amount of misclassifications in the remaining two classes, as will be expounded later. Figure 6.14, right panel, depicts validation (solid line) and training (area over the line) defect-class accuracy for a selection of differently weighted loss calculations. Comparing an unweighted loss calculation (red plot) with an equally weighted loss calculation (grey plot) visualises the improved learning process, even though validation accuracy levels off after about 40,000 steps. Adding additional weight to the defect class alone further increases defect-class accuracy but also causes distinctive overfit, where validation accuracy drops as the network memorises

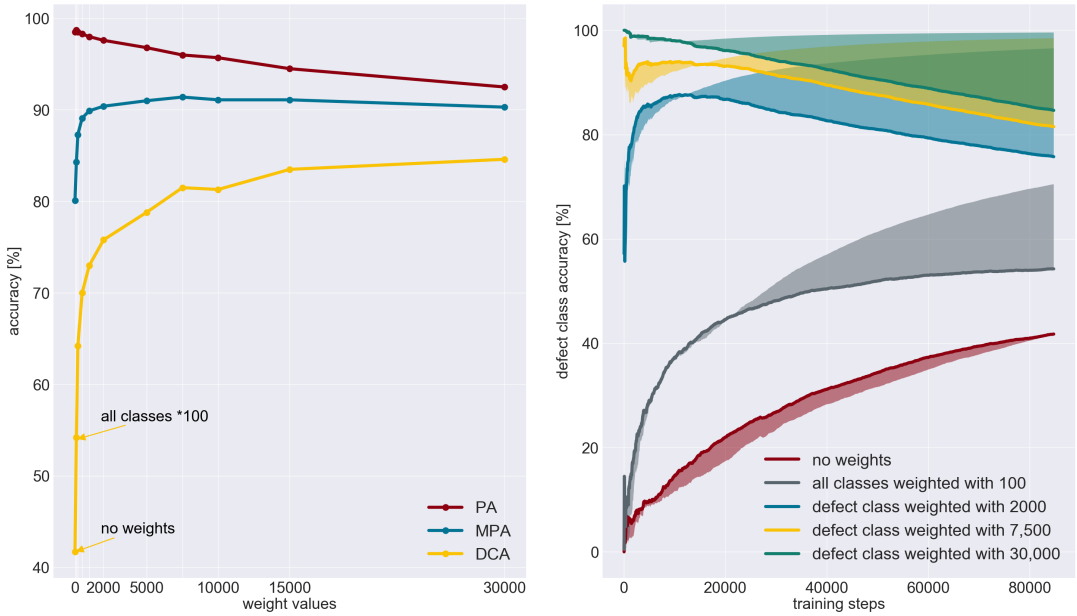
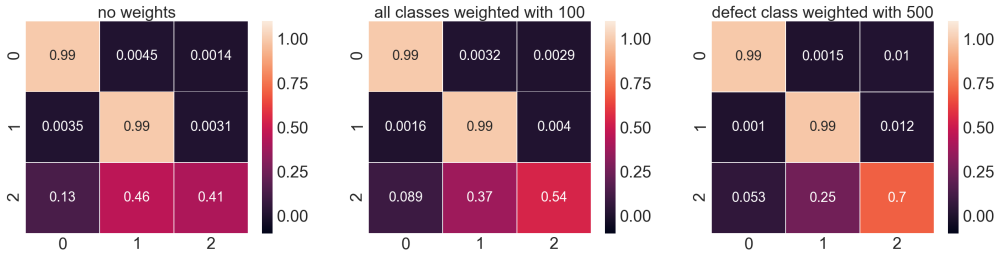



Figure 6.14.: Comparison of differently weighted loss values, implemented to equalise the imbalance between miscellaneous and in-spec-class pixels, respectively, and defect-class pixels. The left panel shows pixel accuracy (PA), mean pixel accuracy (MPA) and defect-class accuracy (DCA) of three loss weight scenarios: no loss weights, all class categories multiplied with 100, and finally additional, increasing defect-class-loss weight values, starting from 200 up to 30,000. It becomes apparent that equalising the class categories distinctively increases defect-class accuracy. The right panel shows validation defect-class accuracy (solid line) and training defect-class accuracy (area over the line) for various defect-class-loss weights. The plots reveal that with increasing loss weights defect-class accuracy increases as well but also leads to early overfitting and a decline in validation accuracy.

the training examples. When studying the plots it would seem reasonable to multiply defect-class losses with 30,000 and stop training after a few thousand steps. Empirically, however, the representation of salient defect structures in validation images improves with longer training durations, even though validation accuracy decreases. Additionally, functional structures such as alignment markers and OCR chips are initially ascribed to the defect class and only with ongoing training the network learns to correctly distinguish functional structures from defect structures. When examining the confusion matrices of differently weighted loss calculations (figures 6.15 and 6.16) it becomes apparent how miscellaneous and in-spec-class accuracies drop with increasing weight values, due to an increase in false negatives in favour of the defect class: up until a loss weight of 500, both miscellaneous and in-spec class yield an accuracy of about 99 %, while the number of misclassifications slightly increases in favour of the defect class and slightly decreases

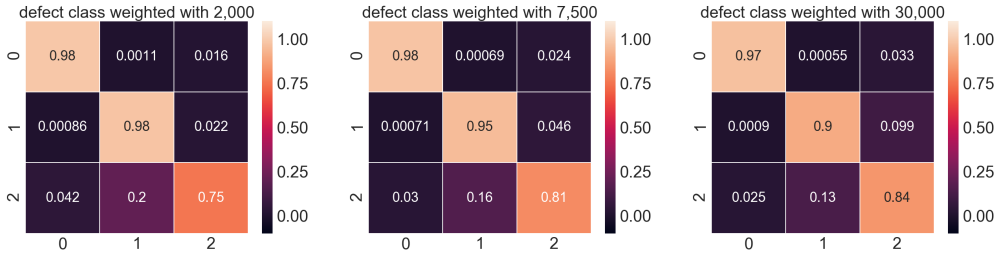
with regard to the respective other class category. This trend intensifies with increasing defect-loss weights, where a loss weight of 2,000 causes a drop in miscellaneous and in-spec-class accuracy to about 98 %, while defect-class accuracy increases to 75.8 %. Here, the majority of misclassifications can be ascribed to the in-spec class, with about 20 %. Incidentally, with regard to wafer images, misclassifications of the miscellaneous class may be discarded, given that non-wafer areas as well as alignment markers and OCR chips are known beforehand. The selection of a loss weight value finally depends on the application—based on the metrics and the decrease in miscellaneous and in-spec-class accuracies, a defect loss weight of 2,000 seems reasonable and is used in all following experiments.

In sum, weighting the loss of underrepresented class categories is a measure to equalise unbalanced class categories and thus increases mean pixel accuracy as well as defect-class accuracy. If possible, the weight value may be calculated depending on the class categories' ratio. In case of defective LED chips, however, the number of defects distinctively fluctuates from wafer to wafer, and thus a suitable weight value must be determined empirically.




(a) (b) (c)

Figure 6.15.: Confusion matrices of differently weighted loss calculations, where 0 equals the defect class, 1 equals the in-spec class and 2 equals the miscellaneous class. Each row corresponds to the true class category and each column corresponds to the predicted class category. Figure a) shows the labelling results of a network trained without weighted loss calculation, revealing that both, miscellaneous (0) and in-spec (1) class, achieve overall low misclassifications, whereas the defect class (2) yields only 41 % true positives and 46 % false negatives with respect to the in-spec class. Multiplying all losses with 100 (figure b) improves this ratio, where defect-class accuracy rises to 54 %. Figure c) visualises a network where an additional weight of 500 is applied to defect-class losses, resulting in an increase in defect-class accuracy, however with an adverse effect on the remaining two class categories, where the amount of false negatives with respect to the defect class slightly rises.




(a) (b) (c)

Figure 6.16.: Confusion matrices of differently weighted loss calculations, where 0 equals the defect class, 1 equals the in-spec class and 2 corresponds to the miscellaneous class. Each row corresponds to the true class category and each column corresponds to the predicted class category. Figure a) shows the labelling results of a network trained with additional defect-class losses of 2,000. Here, true positives with respect to the defect class have further increased in comparison to figure 6.15, albeit less strong from 70 % to 75 %. Like before, false negatives of the miscellaneous and in-spec class with respect to the defect class increase. This trend can also be observed for figure b) and c), where the defect loss weights have been further increased.

### 6.2.8. Ultrasonic-Measurement Embedding

Fully convolutional networks infer a prediction for each input pixel and thus output a prediction image of the same size as the input image. While performance metrics calculate how many pixels have been classified correctly, studying prediction images allows an evaluation of how well the network has learned to segment defect structures and whether the output resolution is fine-grain enough to depict single defective LED chips. Moreover, as expounded in chapter 2, mismatches between the input image and label image occur, caused by the different underlying measurement techniques: photoluminescence measurements determine the brightness of the wafer's optical surface, while ultrasonic measurements detect the actual voids and cracks within the wafer structure. To spare void and crack areas from electrical and optical measurements, the ultrasonic measurement results are forwarded to wafer probing as defect information. Therefore, prober-based defect maps incorporate ultrasonic defects in addition to electrical and optical defects, and as a result, voids and cracks occupy a larger area in label images than in photoluminescence images. Figure 6.17 shows validation input images (left), label images (middle) and prediction images (right) of a well-tuned fully convolutional network with a pixel accuracy of 96.5 %, mean pixel accuracy of 90.3 % and defect-class accuracy of 73.1 %. Note that for the purpose of this paragraph and following experiments, a new dataset of 136 input-label pairs was compiled, resulting in slightly decreased performance metrics compared to the previous studies.

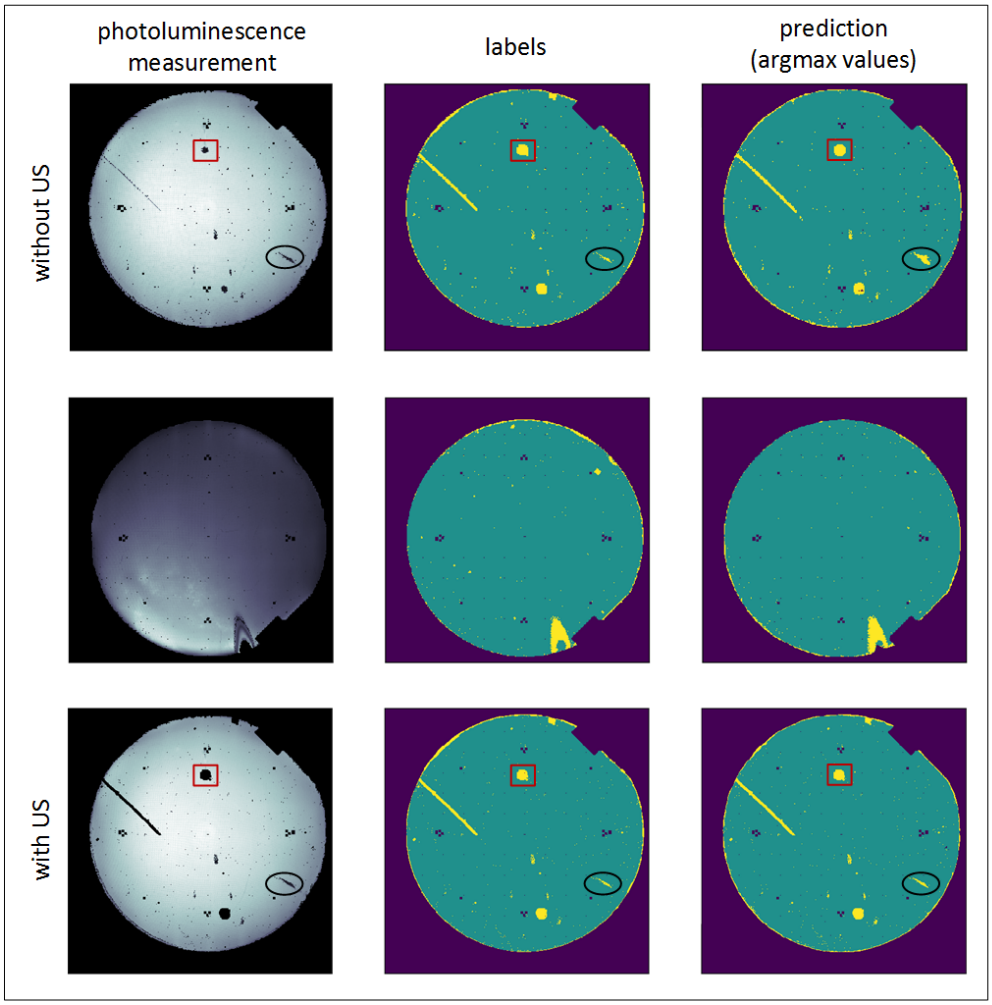



Figure 6.17.: Analysis of the influence of input-label mismatches on segmentation accuracy. Row 1 visualises multiple input-label mismatches: crack and void (red square) structures in photoluminescence images appear small compared to label images. Examining the prediction image reveals that the network adopts repeated misclassifications and enlarges the segmentation area accordingly. While defect clusters are correctly distinguished and depicted (row 2), structures similar to voids are misclassified (row 1, black circle). Embedding photoluminescence images with ultrasonic measurements (row 3) prevents misclassifications and increases segmentation accuracy as well as defect-class accuracy from 75.8 % to 86.6 %.

When analysing figure 6.17, it becomes apparent that the developed network's output resolution is fine-grain enough to depict single defective LED chips. Furthermore, the network has learned to distinguish miscellaneous structures, such as OCR chips and

alignment markers (see also figure 2.4), from defect structures. Eventually, comparing photoluminescence image and label image in figure 6.17 (row 1), visualises the mentioned input-label mismatches, where cracks and voids (red square) occupy a larger area in the label image than in the input image. Analysing the corresponding prediction image reveals that the network has adopted the repeated misclassifications and thus cracks as well as voids are segmented more extensively than single defective chips. Note that the network has also learned to distinguish defect clusters from voids and cracks and thus the segmented area in figure 6.17, row 2, corresponds to the area visible in the photoluminescence image. However, not all defect areas are recognised correctly, as the black circled defect structure in figure 6.17, row 1 is evidence of, which is mistakenly treated as a void. Additionally, even though the network has learned to enlarge void and crack segmentations, the prediction is expectably not accurate to a pixel. Therefore, the aforementioned new dataset was compiled, where photoluminescence images were embedded with ultrasonic-defect information. As visualised in figure 6.17, both, void (red square) and crack structures are evidently enlarged in the photoluminescence image, whereas the misclassified defect structure (black circle) remains in its shape. Examining the corresponding prediction image shows that, as a result, all defect structures are segmented accurately and hence defect-class accuracy increases over ten percentage points to 86.6 %, as presented in table 6.5.

Table 6.5.: Performance metrics of a network trained with the regular photoluminescence image dataset (without US) as well as with a dataset of photoluminescence images embedded with ultrasonic-defect information (with US), where the additional defect information distinctively increases network accuracy. The table lists validation pixel accuracy (PA), mean pixel accuracy (MPA) and defect-class accuracy (DCA).

| | PA | MPA | DCA |
|---|---|---|---|
| without US | 96.5 | 90.3 | 73.1 |
| with US | 98.0 | 94.3 | 86.6 |

### 6.2.9. Summary

After studying architectural design choices of fully convolutional networks in the last section, this section examined how to tune the network's hyperparameters so as to maximise network performance. For this purpose, the influence of various hyperparameters was studied as well as how different hyperparameters interact with each other. First, the impact of various resizing operations was examined, where the best results were achieved by initialising the upsampling operation with bilinear interpolation followed by convolution and a ReLU activation function. Then, experiments with different activation functions revealed that for the given dataset and network architecture ReL units

outperform other methods. The influence of various initialisation methods, namely random initialisation in comparison to the transfer of pre-trained parameter values, was studied afterwards, where the highest network performance was achieved for a randomly initialised network in combination with a pre-trained parameter transfer into the first four layers. Then, the impact of learning rate, L2 regularisation and parameter update method were examined independently as well as in combination with each other. Finally, data-specific tuning methods were studied: introducing individually weighted loss calculations equalised the class categories' imbalance and resolving input-label mismatches by embedding photoluminescence images with ultrasonic-measurement defect information increased segmentation accuracy. Altogether, the presented hyperparameter tuning methods increased mean pixel accuracy from about 84 % to 94.3 % and defect-class accuracy from about 50 % to 86.6 %.

## 6.3. Advanced Architectures

The previous two sections studied the influence of various architectural design choices as well as the impact of hyperparameter tuning on network performance. Basis of these examinations was a fully-convolutional-network architecture [77], modified with respect to the special composition of photoluminescence images and the necessity of a pixel-wise prediction resolution. The resulting, tuned *Vaughan* network architecture yielded a pixel accuracy of 98.0 %, mean pixel accuracy of 94.3 % and defect-class accuracy of 86.6 %. For a depiction of the architectural concept refer to figure 5.1, which visualises the similar design of a slightly deeper architecture. In order to further increase defect-class accuracy, this section studies the implementation of advanced architectural concepts into the *Vaughan* architecture, namely densely connected convolutional layers [50] as well as atrous-spatial-pyramid-pooling modules [16]. Note that partial results of the described advanced architecture design as well as aspects of the according hyperparameter tuning have also been discussed in Stern et al. [119].

### 6.3.1. Densely Connected Convolutional Blocks

As described in chapter 3, densely connected convolutional networks advance the concept of residual networks [42] by feeding all preceding, concatenated feature maps as input to the current layer (see also figure 5.2). Hereby, all layers obtain direct access to the input image as well as the loss gradients. However, expanding this concept to fully convolutional networks requires a special setup of the upsampling path, due to the increase in feature-map dimensions next to the continuously growing number of feature maps [57, 146]. Therefore, in this work's version of a densely connected fully convolutional network, downsampling and upsampling path are designed differently: in the downsampling path,

each layer inputs all preceding, concatenated feature maps before applying the consecutive operations $3 \times 3$ convolution, batch normalisation and ReLU activation function, as described by Huang et al. [50]. After two to three densely connected convolutional layers, a maxpooling operation reduces the feature-map dimensions of all preceding, concatenated feature maps and forwards them to the subsequent dense block. Because dense connections encourage the re-use of feature maps, the number of kernels in each layer in the downsampling path is reduced, as listed in table 6.6. Note that the number of intermediate output feature maps in the upsampling path increases from 64 to 128, due to the new structure: the first upsampling layer inputs all 1,152 preceding, concatenated feature maps of the downsampling path and applies the consecutive operations bilinear upsampling, $3 \times 3$ convolution, batch normalisation and ReLU activation function. The resulting 64 feature maps are concatenated with the 64 feature maps supplied by the skip connection. Here, the densely connected shallow couterpart is bypassed in full and reduced to 64 feature maps by applying the consecutive operations $3 \times 3$ convolution, batch normalisation and ReLU activation function. Even though it is common practise to employ $1 \times 1$ convolution to reduce the number of feature maps, empirically $3 \times 3$ convolution achieved a slightly better performance. The 128 concatenated feature maps are then forwarded to the subsequent upsampling layer, which repeats the described procedure until input image dimensions are restored and a probability map for each class category is generated. The main modifications of the upsampling path are, on the one hand, that the skip connections bypass densely connected layers and on the other hand, upsampled feature maps and bypassed feature maps are concatenated instead of added. Thus, even though the upsampling layers are not thoroughly densely connected, the information flow through the network is improved.

Figure 6.18 (left) visualises defect-class accuracies of the *Vaughan* network in addition to the three studied advanced architecture concepts, which were all trained on ultrasonic-embedded photoluminescence images. The red line corresponds to the *Vaughan* network architecture, whereas the grey line corresponds to the densely connected architecture. It becomes apparent that both networks initially achieve comparable results but after about 12,000 training steps the *Vaughan* architecture starts to overfit to the training data, resulting in a decrease in validation defect-class accuracy. Figure 6.18 (right) illustrates the different degrees of overfitting of both networks: with increasing training duration, the *Vaughan* architecture's validation and training defect-class accuracy diverge more strongly than those of the dense network. Thus, employing less but densely connected feature maps encourages the re-use of feature maps and thus yields a higher performance, with an increase in mean pixel accuracy from 94.3 % to 95.6 % and an increase in defect-class accuracy from 86.6 % to 90.6 %. Pixel accuracy, however, drops slightly from 98.0 % to 97.4 %, indicating an increase in misclassifications of the other two class categories.

The rise in misclassifications with regard to pixels classified as in-spec and miscellaneous, respectively, can also be observed when analysing prediction images, as shown in figure 6.19. Comparing the *Vaughan* network's prediction image to the label image

Table 6.6.: Number of feature maps of each layer of the *Vaughan* network compared to a densely connected version, which enables a reduced number of feature maps, due to the network's ability to re-use them. Note that the number of feature maps in the upsampling path increases because the feature maps of the skip connection are concatenated to the upsampled feature maps rather than added.

| layer | architecture | |
|---|---|---|
| | 1: Vaughan network | 2: dense Vaughan |
| conv1_x | 64, 64 | 32, 32 |
| conv2_x | 128, 128 | 64, 64 |
| conv3_x | 256, 256, 256 | 64, 64, 64 |
| conv4_x | 512, 512, 512 | 128, 128, 128 |
| conv5_x | 512, 512, 64 | 128, 128, 128 |
| up1 | 64 | 128 |
| up2 | 64 | 128 |
| up3 | 64 | 128 |
| up4 | 3 | 3 |

reveals that the defect structure is shown frayed, whereas the dense network's prediction covers the whole defect structure but over-segments the area. Note that the wafer edge shows a high number of misclassified pixels as well. However, when comparing photoluminescence image and label image, the wafer edge in the first image appears frayed, whereas the label image shows a clear border line. The difference in both images is caused by the chip-separation step that takes place after wafer probing (on which the label images are based) and before the photoluminescence measurement, as expounded in chapter 2. Therefore, the classification of the wafer edge may be discarded with regard to an evaluation of segmentation performance. The poor depiction of the defect structure can be ascribed to a lack of representation of similar structures in the training dataset: on the one hand, the depicted defect structure is unique within the dataset and on the other hand, differently sized and shaped defect structures in the training dataset are segmented accurately. Because it is not always feasible to procure a dataset from a running production that covers all possible defect structures, another approach is necessary to improve the depiction of uncommon structures.

### 6.3.2. Atrous Spatial Pyramid Pooling

Analysing flawed segmentation results reveals that mostly large defect structures are affected, due to their rare occurrence in the dataset. Here, the network's deficiency to generalise from smaller defect structures may be caused by the different scales defect

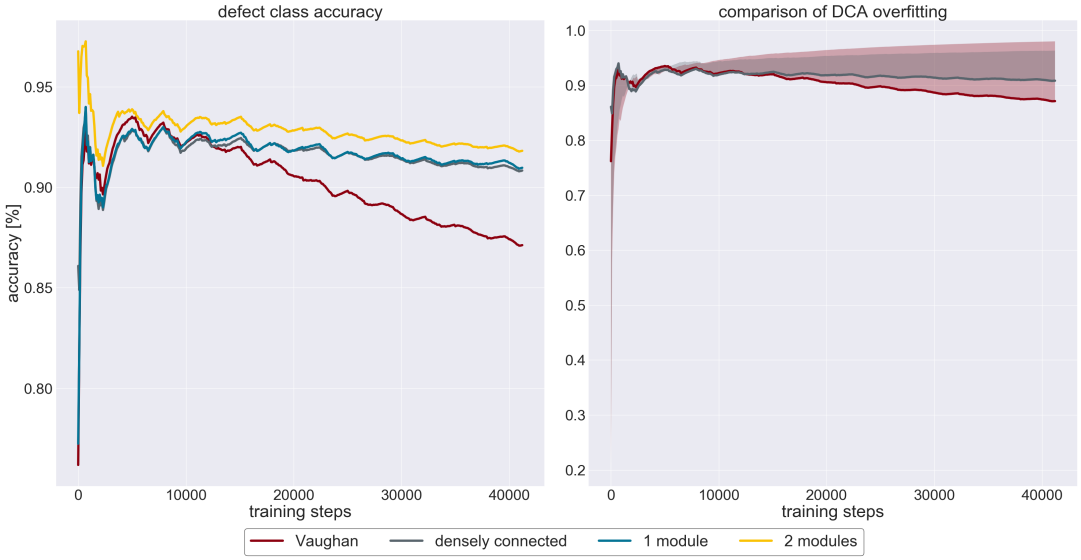



Figure 6.18.: Left: Defect class accuracy of the previously developed *Vaughan* architecture (red) compared to the additional implementation of advanced architectural concepts, namely densely connected layers (grey) as well as one (blue) and two atrous-spatial-pyramid-pooling modules (yellow). It becomes apparent that advanced architectural concepts further increase the accurate depiction of defect structures. One reason is the decrease of overfitting, caused by densely connected layers: the right panel visualises the varying divergence of validation (solid line) and training defect-class accuracy (area over the line) of the *Vaughan* network and the dense *Vaughan* network. The details and results of the plots have also been summarised in table B.9.

structures assume. To address this challenge, atrous-spatial-pyramid-pooling (ASPP) modules were studied. As expounded in chapter 3 and visualised in figure 5.4, ASPP modules implement a multiple-scales analysis by forwarding the incoming feature maps to several dilated-convolution layers in parallel. Additionally, global context is incorporated by extracting features on image-level via global average pooling. Following Chen et al. [16] and Zhao et al. [141], the last convolutional block of the aforementioned dense *Vaughan* network's downsampling path was replaced by an ASPP module. The new structure is as follows: the 768 feature maps of the densely connected fourth convolutional block are simultaneously forwarded to the ASPP module's five layers, without reducing the feature-map dimensions via subsampling. Then, the first four layers apply dilated convolutions at various dilation rates $r$, namely $r = 1, 2, 6$ and 12, followed by the consecutive operations $3 \times 3$ convolution, batch normalisation and ReLU activation function. Hereby, the incoming feature maps are sampled at different rates and thus with multiple receptive fields at once, without loosing spatial resolution by downsampling. In order to prevent fragmented segmentation results, the last module layer incorporates

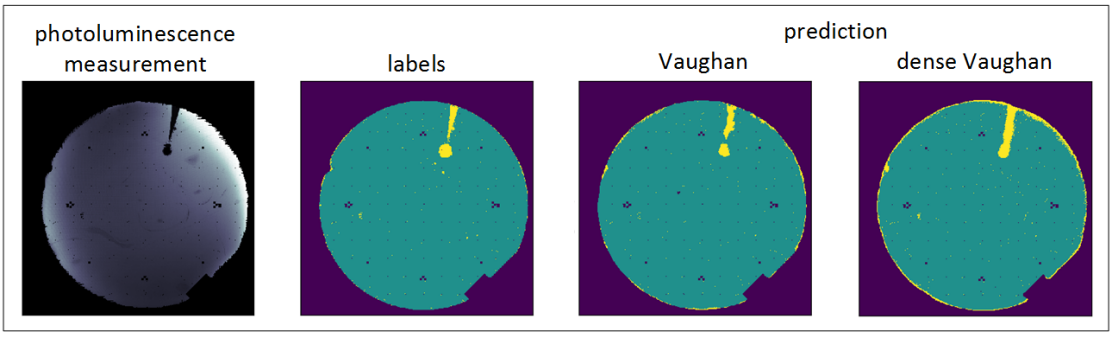



Figure 6.19.: Prediction images of the *Vaughan* network versus a densely connected version. The latter achieves a less frayed segmentation but over-segments the defect area, causing an increase in defect-class accuracy along with a drop in pixel accuracy.

global context: first, image-level features are captured via global average pooling, which reduces feature-map dimensions to $1 \times 1 \times 768$. In order to decrease the number of feature maps to 128 and retrieve the previous feature-map dimensions of $56 \times 55$, the consecutive operations $1 \times 1$ convolution, batch normalisation, bilinear upsampling and ReLU activation function are applied. Eventually, the output of all five module layers as well as the preceding fourth layer are concatenated, yielding 1,408 feature maps, which are then forwarded to the first upsampling layer. Even though the resulting network architecture empirically achieved a more accurate segmentation result, defect-class accuracy remained nearly unchanged with 90.7 % (dense *Vaughan*: 90.6 %), as shown in figure 6.18 (left panel, yellow line). Moreover, mean pixel accuracy dropped slightly to 95.5 % (dense *Vaughan*: 95.6 %), indicating that the special image composition of the dataset does not benefit much from a larger receptive field at this position in the processing chain.

To study whether an increased receptive field at a higher feature map resolution benefits network performance, a second ASPP module was implemented, which complements the third and thus penultimate layer of the upsampling path, yielding the following structure: first, the feature maps of the preceding layer are upsampled and concatenated with the bypassed feature maps of the skip connection, as described before. Then, the resulting 128 feature maps are forwarded to the ASPP module, which covers three-layers, two dilated-convolution layers with $r = 2$ and 4, respectively, as well as a global-average-pooling layer. Note that the number of module layers and kernels as well as the dilation rates are hyperparameters, where the used values were chosen based on experiments [119]. Eventually, module input and output are concatenated, resulting in a total of 224 feature maps, which are forwarded to the last layer. Contrary to the previous architecture, the last upsampling stage now consists of two layers, one upsampling layer and an additional output layer, which reduces the upsampling layer's 128 feature maps to three probability maps by applying the consecutive operations $1 \times 1$ convolution and softmax activation function. As shown in figure 6.18 (left panel, green line), a densely connected *Vaughan*

network with two ASPP modules outperforms other architectural designs with a pixel accuracy of 97.5 %, mean pixel accuracy of 96.1 % and defect-class accuracy of 91.8 %. Incidentally, omitting the first ASPP module diminishes network performance, as listed in table B.9.

Comparing prediction images of the *Vaughan* network with images of a densely connected *Vaughan* architecture with two ASPP modules (*dense ASPP Vaughan*) shows that both architectures achieve comparable segmentation results for common defect structures, as displayed in figure 6.20, row 1. It becomes apparent that both networks have learned to distinguish miscellaneous structures, film tears and measurement artefacts from defect structures and accurately depict single defective LED chips, voids and cracks. The remaining rows in figure 6.20 reveal, however, apparent differences in the segmentation of defect structures that occur rarely or are unique within the dataset, such as large defect clusters and densely clustered single defective LED chips. While the *Vaughan* network's segmentation appears patchy and frayed, respectively, the *dense ASPP Vaughan* architecture generally displays a more accurate segmentation result. As shown in row 3, a common misclassification of the latter network are overly dense depicted defect areas. Note, however, the corresponding label image with a similar depiction, where the wafer edge in the photoluminescence image displays a large number of single salient chips which overlie a darkened area, indicating low brightness values outside the specification range. As a result, the label image displays a continuous defect area that transitions into single defects. While the network accurately segments continuous defect areas as well as single defective LED chips, the transition area's depiction is insufficient. Apparently, the dataset does not provide enough examples of the variety of transition areas to accurately generalise from. The influence of dataset compilation on the segmentation accuracy is also studied in the following section.

In sum, this section examined the influence of advanced architectural concepts on network performance. First, densely connected convolutional layers were implemented, which encourage the re-use of feature maps and thus allow a more condensed network architecture. As a result, defect-class accuracy increased from 86.6 % to 90.6 % due to a decrease in overfitting. Then, atrous-spatial-pyramid-pooling (ASPP) modules were studied, revealing that the highest performance is achieved with two modules, placed at the end of the downsampling and upsampling path, respectively. While defect-class accuracy increased only slightly to 92.0 %, the empirical segmentation accuracy of large defect clusters as well as dense defect areas increased. The analysis of prediction images also revealed that the dataset's coverage of large defect structures is not sufficient enough to yield accurate segmentation results of all defect structures. Thus, the next section studies the generalisation ability of the previously introduced network models by determining their performance on a test dataset. Afterwards, the influence of dataset augmentation and partition are examined.

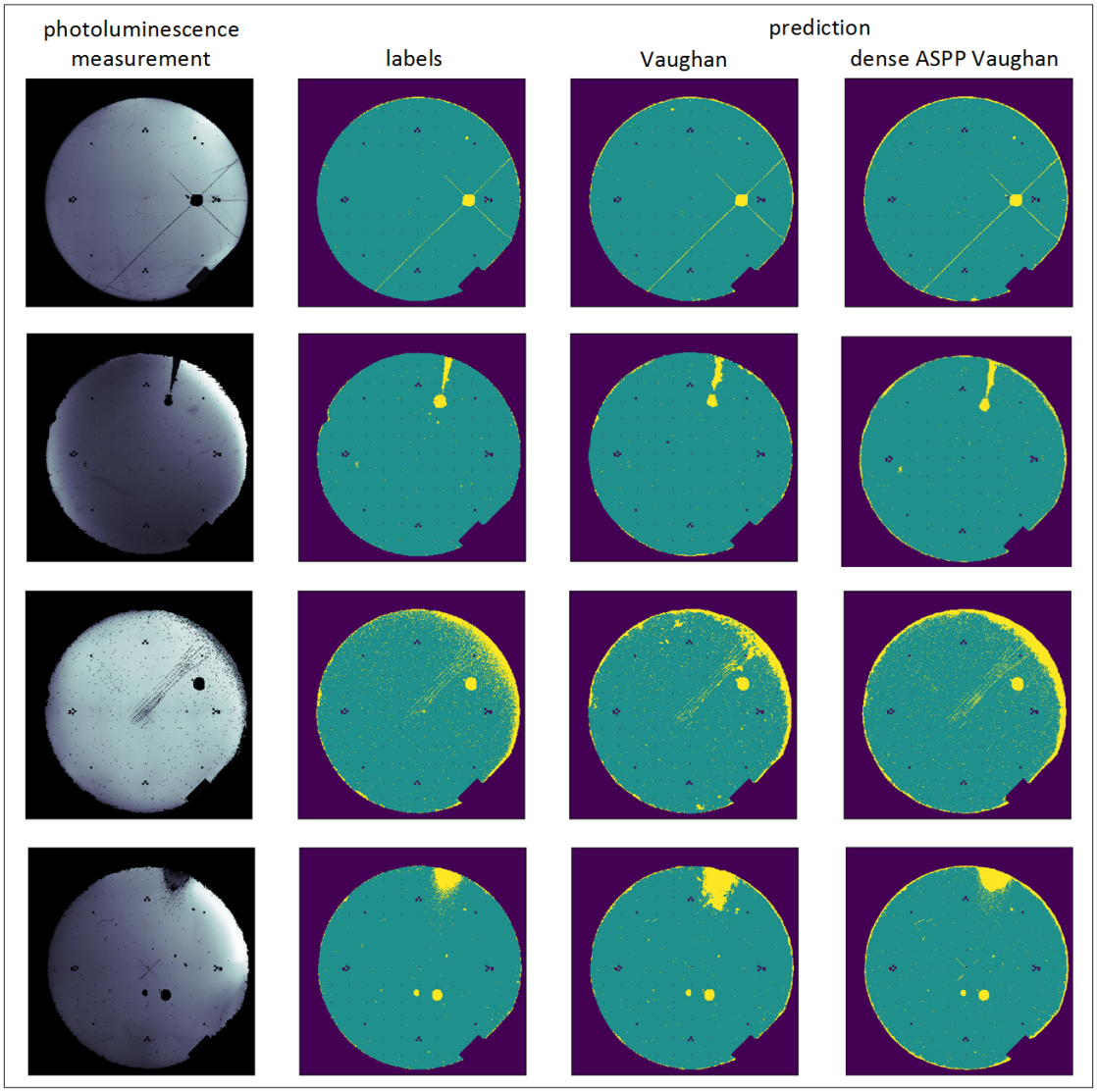



Figure 6.20.: Segmentation accuracy comparison of the *Vaughan* network versus a *dense ASPP Vaughan* network, where both networks achieve comparable results for common defect structures, such as single defects, voids and cracks, but the latter network achieves a more accurate segmentation of large defect structures. However, the transition area between dense defect areas and single defects is still displayed insufficiently (row 3), due to a lack of examples in the dataset.

### 6.3.3. Network Evaluation based on Test Data

To provide an unbiased estimation of the generalisation error, the previously introduced network models are evaluated with a test dataset [100], covering the architectures *Vaughan*, *dense Vaughan* and *dense ASPP Vaughan* with one ASPP module in the downsampling path, one ASPP module in the upsampling path as well as the combination of both ASPP modules. Here, the generalisation error measures a network's ability to accurately segment previously unknown images, that is images the network was not trained on. While this definition also applies to the validation dataset, the validation metrics are used to evaluate the different hypotheses in the process of model tuning and thus do not provide a truly unbiased estimation [105]. Hence, an additional dataset was compiled with 366 photoluminescence measurement images taken from the running production, where only samples with distinctive differences between input image and label image as well as fractured wafers were discarded. Note that in the following section a second test set is employed, which does cover photoluminescence images with severe input-label mismatches as well as fractured wafers. The results and details on the first test set have been summarised in table 6.7.

Table 6.7.: Validation and test pixel accuracy (PA), mean pixel accuracy (MPA) and defect-class accuracy (DCA) of the previously introduced network architectures. The results reveal a drop in test defect-class accuracy for all five models, where the final *dense ASPP Vaughan* architecture shows the slightest difference and the overall highest (test) defect-class accuracy. As observed before, a higher defect-class accuracy correlates with a lower pixel accuracy and mean pixel accuracy, indicating over-segmentation of defect structures and thus causing misclassifications regarding non-defect pixels.

| | PA | PA test1 | MPA | MPA test1 | DCA | DCA test1 |
|---|---|---|---|---|---|---|
| Vaughan | 98.0 | 98.7 | 94.3 | 94.1 | 86.6 | 83.5 |
| dense Vaughan | 97.4 | 98.4 | 95.6 | 95.7 | 90.6 | 88.7 |
| 1 module (downsampling) | 97.3 | 98.3 | 95.5 | 96.1 | 91.0 | 90.4 |
| 2 ASPP modules | 97.5 | 98.5 | 96.2 | 96.7 | 92.0 | 91.5 |
| 1 module (upsampling) | 97.5 | 98.3 | 95.4 | 95.8 | 91.2 | 88.5 |

Analysing the test results reveals that all five models yield a lower test defect-class accuracy, where the *dense ASPP Vaughan* architecture with two ASPP modules achieves the slightest difference between validation and test defect-class accuracy. Moreover, the results show that the combination of two ASPP modules best equips the network to detect and segment unknown defect structures, whereas the drop in test defect-class accuracy of a single ASPP module in the upsampling path indicates overfitting to the training and validation data, while still outperforming the basic *Vaughan* architecture. In addition, examining the test results of the *dense Vaughan* model as well as the model

with one ASPP module in the downsampling path also verifies that the investigated advanced architectural concepts increase the network's segmentation ability. Altogether, the previous sections demonstrated the distinct influence of network-architecture design and hyperparameter tuning on network performance. The following section illustrates the effect of data preparation, covering the increase of the training set size via data augmentation and with new examples, respectively, as well as dataset partition.

## 6.4. Data Preparation

While network architecture and hyperparameter tuning are highly influential design choices with regard to network performance, compiling a dataset determines the network's training experience. This is especially the case for the given data, which is procured from a running production that provides only occasionally wafer images with the kind of salient defect structures that are difficult to segment for the network algorithm. As mentioned before, two datasets were used in this work, one with 145 unaltered photoluminescence images (dataset 1) and one with 136 photoluminescence images with embedded ultrasonic-defect information (dataset 2). Even though all of the selected wafer images display salient defect structures, the majority of these structures are voids, which only vary in size, followed by cracks. Defect clusters, on the other hand, assume various shapes and sizes as well as brightness gradients, and occur only rarely in the datasets. To provide the network with a thorough coverage of available defect clusters as well as a representative validation set, network training was performed with a manual training-validation split of about 20 % validation images, resulting in 111 training images and 25 validation images (dataset 2). Furthermore, the number of training examples was increased using data augmentation, where each image was rotated by 45°, 90° and 135°, respectively. The effect of data augmentation, which teaches the network invariance against common variations, can be shown by training the *dense ASPP Vaughan* network without data augmentation, where the same pixel accuracy as before (97.5 %) was achieved, but mean pixel accuracy decreased from 96.2 % to 93.8 % and defect-class accuracy decreased from 92.0 % to 87.6 %. Note that for this and all following experiments the ultrasonic-embedded dataset (dataset 2) was used. Incidentally, it is apparent that rotated wafer images provide a useful extension of the dataset while other data augmentation methods, such as added noise, cropping or elastic deformations, represent image manipulations that do not occur naturally in the dataset. Consequently, experiments with brightness variations and elastic deformations did not increase the network's generalisation ability. Moreover, analysing prediction images reveals segmentation difficulties only with regard to uncommon defect clusters as well as the transition between dense and single defect areas, as shown in figure 6.22 for example. These observations indicate that increasing dataset size with valuable examples will increase segmentation accuracy, where the impact will be more distinctive for training examples that show rare defect structures. Consequently, doubling the dataset with 134 additional training ex-

amples resulted in an increased pixel accuracy of 97.6 %, mean pixel accuracy of 96.8 % as well as defect-class accuracy of 93.5 %, where the training examples are a subset of the aforementioned test dataset 1 and cover mainly salient and a small number of rare defect structures.

Next to dataset size, the influence of the network's training experience is studied, using different dataset splits. For this purpose, $k$-fold cross-validation is employed, where the details are listed in table B.10 and visualised in figure 6.21. Here, $k$-fold cross validation denotes a model validation technique, where the data is split into $k = 4$ subsets of equal size, called folds. Then, the network is trained with $k-1$ folds, while the remaining fold is used for validation. By iterating over all $k$ folds and averaging the results, the network's generalisation ability can be determined [94]. Note that in addition to validation metrics, test defect-class accuracy was determined using a second test dataset of unaltered photoluminescence images, images previously rejected due to severe input-label mismatches and fractured wafers. Hereby, the networks' generalisation ability regarding difficulty to segment data samples can be evaluated. Consequently, test defect-class accuracy is lower than validation defect-class accuracy and the previously determined test metrics, due to inevitable misclassifications. When analysing figure 6.21, the distinctive influence of how the dataset is split into training and validation set becomes apparent. Here, the worst performing data partition, fold 1, achieves a defect-class accuracy of 76.8 % while the highest performing partition, fold 3, yields a defect-class accuracy of 93.4 % and thus distinctively exceeds the cross validation average of 87.8 %. The legitimate presumption that the validation set of fold 1 contains only defect structures that are difficult to segment can be disproved by analysing the test defect-class accuracies (figure 6.21, blue line), where fold 1 performs worst as well, while fold 3 outperforms all other partitions again. These results verify the assumption that the contribution of common defect structures to network learning, such as voids and cracks, is small compared to the contribution of rare defect structures, such as large defect clusters.

Accordingly, manually splitting the dataset to ensure an even distribution of rare defect structures outperforms the averaged cross-validation metrics, especially with regard to defect-class accuracy (92.0 % to 87.8 %). Predefining a network's training experience by dividing small, highly unbalanced datasets by hand may therefore be a beneficial approach, given that cross-validation splits are performed randomly and might not always result in a well-performing split. Moreover, manually splitting the dataset allows a better control over what the network learns, as is verified by the segmentation results in figure 6.22. Here, prediction images of all five partitions are compared using examples from the test dataset, where it becomes apparent that all training experiences yield comparable segmentation results for common defect structures as well as functional structures. Rare or unknown defect structures, such as large defect clusters (column 1) and missing wafer parts (column 2) are segmented differently well, where empirically no training experience consistently outperforms or underperforms the others. As an instance, fold 1 achieves a comparably well segmentation of the defect cluster in column 1 but fails to

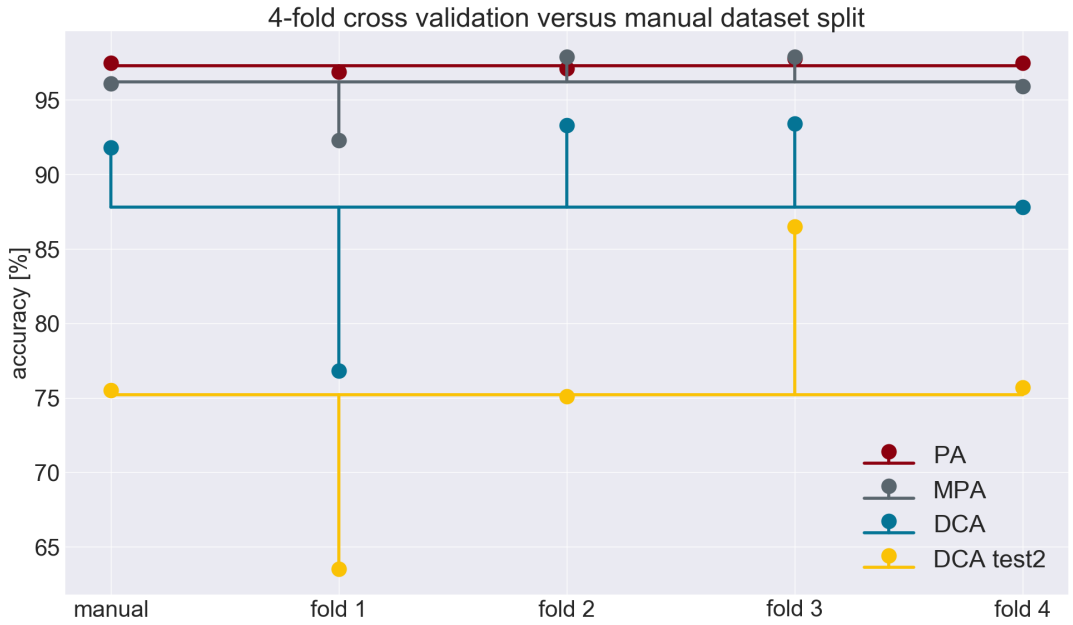



Figure 6.21.: Pixel accuracy (PA), mean pixel accuracy (MPA), defect-class accuracy (DCA) and test defect-class accuracy (DCA test2) of a *dense ASPP Vaughan* network trained with a manual train-validation split in comparison to 4-fold cross validation, where the straight line marks the cross validation average. The high variety of the dataset in combination with the small number of training examples causes distinctively differing cross-validation results, where the manual training-validation split slightly outperforms the cross-validation average, except for defect-class accuracy, where a distinctively better result is achieved. Details and results have also been summarised in table B.10

correctly recognise missing wafer parts in column 2, whereas the best performing fold 3 yields a patchy segmentation of the defect cluster but correctly classifies the missing wafer parts. Note, however, that for the most defect structures the empirically most convincing segmentation result does not necessarily correspond to the highest pixel accuracy. The distinction between cracks and film tears (column 3) has only been learned by a network trained with fold 4 or the manually split dataset, where the latter achieves sufficient segmentation results for all displayed test images. These experiments show that even though small datasets of highly variable data achieve convincing results, dataset compilation and splitting must be handled with care.

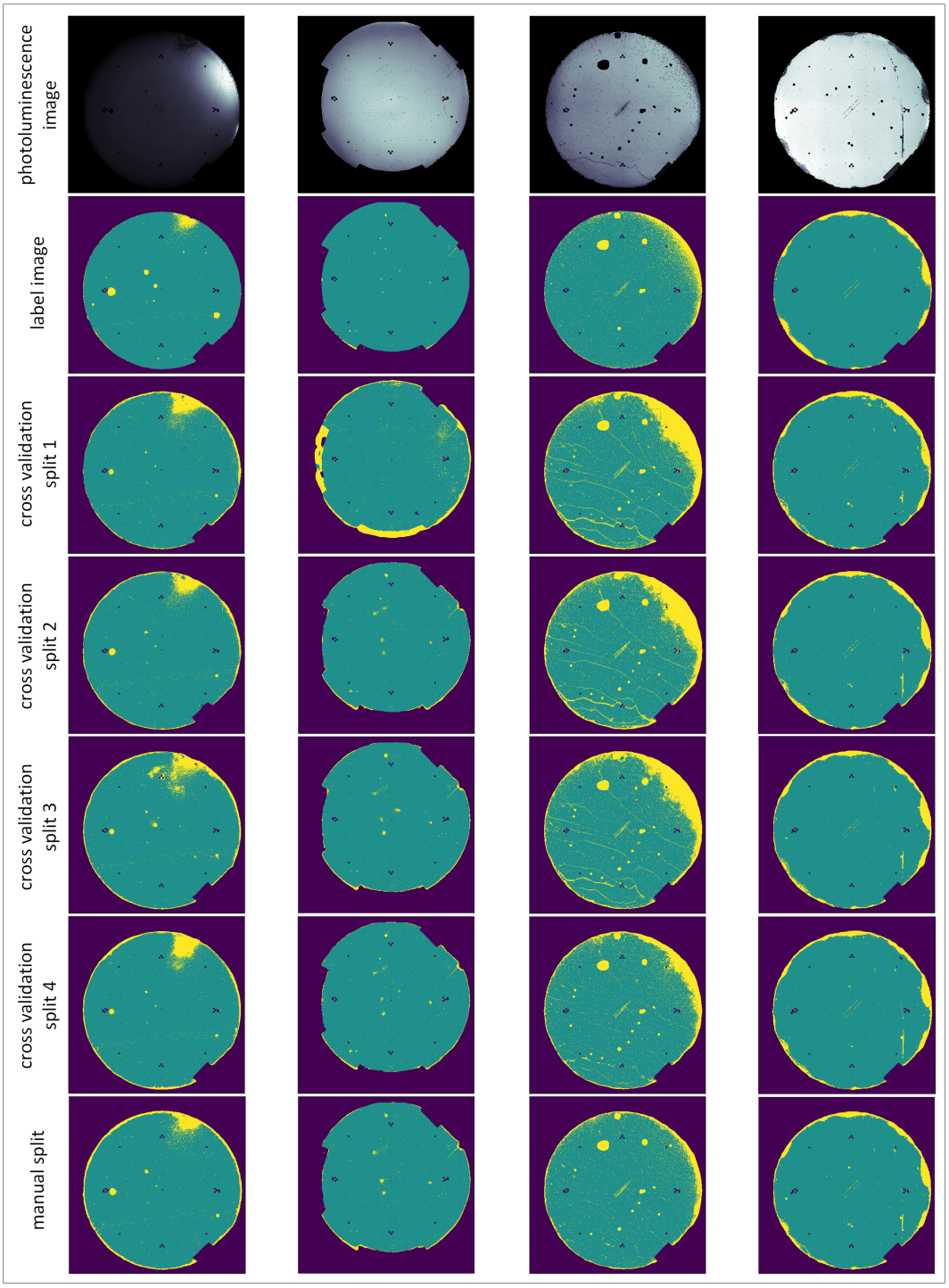



Figure 6.22.: Segmentation-accuracy comparison of test images output by a *dense ASPP Vaughan* network trained with a manually split dataset and 4-fold cross validation, respectively.

## 6.5. Summary

Detecting defective LED chips in photoluminescence wafer images requires a novel network architecture that implements the reliable classification of multiple scaled objects, while enabling a pixel-level output resolution. In this chapter, the main aspects of network design, hyperparameter tuning and dataset compilation were studied, revealing how the different building blocks influence network performance. First, experiments regarding the network architecture were conducted in order to analyse the interaction of network depth and skip connections. Visualising network filters and intermediate feature maps allowed an interpretation of what the network has learned to recognise and how the network processes the input images. Based on these results, different hyperparameters were studied, covering different upsampling operations, activation functions and network initialisation using transfer learning. Eventually, the effects of learning rate, L2 regularisation and parameter update method were studied independently as well as in combination with each other, revealing their interdependency. Next to classical hyperparameters, data-specific tuning methods were examined, including weighted loss calculations and the embedding of defect information from ultrasonic measurements into photoluminescence images.

Altogether, the presented hyperparameter tuning methods increased mean pixel accuracy from about 84 % to 94.3 % and defect-class accuracy from about 50 % to 86.6 %, using a modified fully-convolutional-network design. Further studying prediction images exposed that the developed network yields accurate segmentation results for single defects and defect structures that occur repeatedly in the dataset but infers flawed segmentations of large, unknown defect clusters. To improve segmentation accuracy, advanced architectural concepts were examined, namely densely connected convolutional blocks as well as atrous-spatial-pyramid-pooling (ASPP) modules. The resulting *dense ASPP Vaughan* network achieved a pixel accuracy of 97.5 %, mean pixel accuracy of 96.2 % and a defect-class accuracy of 92.0 % on a dataset of 136 image-label pairs as well as a test defect-class accuracy of 91.5 % on a test dataset of 366 images from the running production.

Finally, the influences of dataset compilation were studied by comparing a manual training-validation split with 4-fold cross validation. Analysing validation and test results (of, a second test set) of all dataset partitions revealed the benefit of manually dividing small datasets with highly variable image objects. Moreover, the conducted experiments, in combination with the test prediction-image analysis of all dataset partitions, indicated that adding valuable training examples of rare defect structures to the dataset will further increase network performance and segmentation accuracy. Consequently, doubling the dataset size increased network performance in general, where defect-class accuracy in particular increased from 92.0 % to 93.5 %. Altogether, the resulting performance metrics and segmentation results of the developed network architecture verify that a well designed and tuned fully convolutional network can be employed for the chip-wise

analysis of photoluminescence images. In the following chapter, the resulting network architecture is described in detail.

# 7. Implementation & Evaluation of the Resulting Network Architecture

In this work, the employment of fully-convolutional-network algorithms for the detection of defective LED chips in photoluminescence images is studied. Here, the challenge lies in the multiple-scaled defect objects, which range from single defective chips to large defect clusters, in addition to a small dataset of simply composed photoluminescence wafer images with highly variable brightness values and defect structures. The previous chapters first presented the employed dataset, followed by the theoretical background, state-of-the-art architectures as well as related work and finally illustrated network-design ideas for the given application. Afterwards, the effects of architecture design, hyperparameter tuning and dataset compilation were analysed. In this chapter, the final version of the developed network architecture is presented, which is based on the previous observations and addresses the aforementioned challenges, followed by details about implementation and evaluation.

## 7.1. Fully-Convolutional-Network Architecture

Based on the theoretical background presented in chapters 3 and 5 as well as the conducted experiments in chapter 6, a densely connected fully convolutional network with a novel upsampling path and two atrous-spatial-pyramid-pooling modules was developed, named *dense ASPP Vaughan*. As shown in figure 7.1, the network inputs photoluminescence images of size $442 \times 440 \times 1$, where the last tensor dimension denotes the number of colour channels. Because photoluminescence measurements provide 8bit greyscale values that correlate to a chip's brightness, the number of colour channels equals 1. Note that the photoluminescence images were not normalised or pre-processed in any other way than described in chapter 2. Instead, batch normalisation was employed as input layer (figure 7.1, light grey block), where a batch corresponds to the $N$ pixels of a single input image, which allows the network to apply a learned normalisation, if beneficial. As a result, when running the network algorithm in production, image pre-processing can be reduced to the necessary steps of transforming the measurement results into a three-dimensional tensor of the aforementioned dimensions. After batch normalisation, the input image is forwarded to the first convolutional pooling block, which consists of two convolutional layers with 32 kernels each (figure 7.1, green blocks) and a subsequent

maxpooling operation (red block). All convolutional layers in this network, if not mentioned otherwise, employ $3 \times 3$ kernels with a stride of 1, where the input feature maps are zero-padded so as to keep their dimensions stable. Moreover, convolutional layers in the downsampling path apply the consecutive operations convolution, batch normalisation and ReLU activation function. Due to the usage of dense connections, the maxpooling operation inputs the concatenated feature maps of both preceding convolutional layers and reduces the incoming tensor dimensions from $442 \times 440 \times 64$ to $221 \times 220 \times 64$ with a $2 \times 2$ kernel and a stride of 2, before forwarding them to the subsequent block.

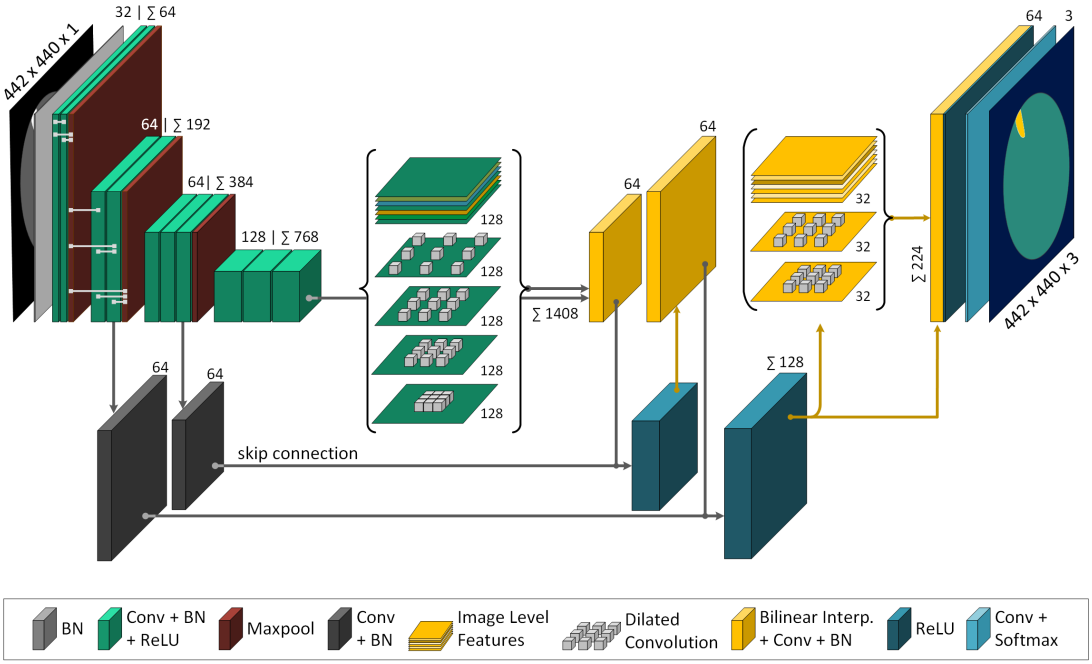



Figure 7.1.: Visualisation of the developed *dense ASPP Vaughan* architecture, where the legend states the layer operations and BN equals batch normalisation [119]. The numbers on top of each dense block and layer, respectively, denote the number of kernels per layer and the number of concatenated output feature maps per block (indicated by a sum sign), if applicable. The horizontal grey lines in the first two blocks illustrate the dense connections, exemplarily for all downsampling blocks. Atrous-spatial-pyramid-pooling (ASPP) modules are indicated by brackets, where each module layer processes the same input simultaneously. Layers below the main branch indicate skip connections, which concatenate feature maps of shallow and deep layers.

Figure 7.1 visualises the dense connections of the first two blocks (horizontal grey lines), where the number of output feature maps of each block is written on top of the corresponding block (marked with a sum sign), next to the number of kernels per convolutional layer. As an instance, the second block employs two convolutional layers with 64 kernels each and inputs the concatenated, dimension-reduced 64 feature maps of the previous

block. The second layer of the second block then inputs the concatenated feature maps of the previous block as well as the feature maps of the preceding layer, which sum to 128. Finally, a maxpooling layer downsamples the feature-map dimensions and forwards all 192 feature maps to the subsequent third block. Incidentally, the number of kernels in each convolutional layer was reduced in comparison to a non-dense network model, because densely connected layers encourage the re-use of feature maps and thus enable more condensed architectures with less feature maps.

The first four blocks of the downsampling path follow the typical network-model bi-pyramid, where every convolutional pooling block reduces the feature-map dimensions while increasing the number of kernels. Hereby, the network's receptive field widens and highly compressed, semantic information can be extracted by a variety of kernels. At the same time, the spatial resolution of feature maps deep in the network is distinctively diminished, as shown in figure 6.6. Moreover, differently sized image objects, such as single defective LED chips, voids and large defect clusters, are all analysed with the same receptive field. Therefore, the fifth convolutional block features an atrous spatial pyramid pooling (ASPP) module, which inputs the concatenated 768 feature maps of the fourth layer in their original dimensions, that is the fourth layer does not employ a pooling operation. The forwarded feature maps are then processed by all five module layers in parallel, where the module consists of four dilated-convolution layers and one global-average-pooling layer, which extracts image-level features. Here, each of the four dilated-convolution layers samples the input image with a different dilation rate $r$, with $r = 1, 2, 6$, and 12, so as to implement various receptive fields at once. Note that a dilation rate of 1 equals a standard convolutional operation and that all dilated-convolution layers apply the consecutive operations dilated convolution, batch normalisation and ReLU activation function. In order to prevent fragmented segmentation results, the fifth module layer incorporates image-level features by applying global average pooling, where the incoming $56 \times 55 \times 768$ feature maps are downsampled to $1 \times 1 \times 768$, that is one feature per feature map is extracted. To retrieve the previous feature-map dimensions of $56 \times 55$ and decrease the number of feature maps to 128, the consecutive operations $1 \times 1$ convolution, batch normalisation, bilinear upsampling and ReLU activation function are applied. Finally, the feature maps of the five module layers and the module input are concatenated, resulting in 1408 feature maps altogether, and forwarded to the first upsampling layer.

As visualised in figure 7.1, the upsampling path employs skip connections to combine the upsampled coarse semantic information with fine-grain local information from shallow layers, so as to increase the output resolution. For this purpose, the first upsampling layer inputs the aforementioned 1408 feature maps of the fourth and fifth downsampling layer and outputs 64 upsampled feature maps. Here, all upsampling layers apply the consecutive operations bilinear interpolation, $3 \times 3$ convolution and batch normalisation, where the upsampling stages mirror the downsampling stages. After upsampling, the resulting feature maps are concatenated with the bypassed feature maps of the skip connection, where in case of the first upsampling stage the 384 feature maps of the

densely connected third block are reduced to 64 feature maps via $3 \times 3$ convolution and batch normalisation. Finally, a ReLU activation function is applied to the concatenated 128 feature maps, which are then forwarded to the second upsampling layer. Note that the upsampling layers are not densely connected, due to consistently increasing feature-map dimensions which would require unreasonable computational resources.

The second upsampling layer features another ASPP module, which contrary to the first module does not input the feature maps of the preceding layer. Instead, the incoming feature maps are first upsampled, following the same procedure as before and the resulting, 128 concatenated feature maps are then forwarded to the ASPP module. Altogether, the module consists of three parallel layers, two dilated-convolution layers with $r = 2$ and 4 and a global-average-pooling layer, where each module layer generates 32 feature maps, which are eventually concatenated with the module input and forwarded to the subsequent layer. The final network layer then calculates the three output maps in two steps: first, the incoming feature maps are upsampled with the standard consecutive operations bilinear interpolation, $3 \times 3$ convolution, batch normalisation and ReLU activation function. Hereby, the network pre-processes the concatenated information from the preceding layer and restores the original dimensions of the input image. Then, the number of feature maps is reduced to three, using $1 \times 1$ convolution, before applying a softmax activation function, which calculates a vector of three normalised probability values for each output pixel. If the network output is used for classification rather than network training, a subsequent *argmax* function may be employed to determine the most probable class category for each image pixel and to generate a one-dimensional prediction image, as shown in figure 6.20 for example.

## 7.2. Implementation and Network Training

Setting up a network architecture determines the algorithm's underlying function, that is a hypothesis space of possible input-label mapping functions is created, in which the algorithm searches for a parameter combination that minimises the loss function. This parameter search is denoted as network training and, in this work, starts with network-parameter values that are initialised with a combination of transferred, pre-trained parameter values in the first four layers and randomly initialised parameter values in the remaining layers, as described in chapter 6. Note that the network at hand does not follow a standard research architecture and thus the transferred parameters are adjusted to the reduced number of kernels in the first four layers.

To train the network, a dataset of 136 photoluminescence images with embedded defect information from an ultrasonic measurement and corresponding wafer-probing-based label images was compiled and pre-processed, resulting in input tensors of size $442 \times 440 \times 1$ and label tensors of size $442 \times 440 \times 3$, where the last tensor dimension corresponds to

the number of class categories. To observe the network's training progress with regard to its generalisation-ability, about 20 % of the dataset's input-label pairs were manually selected and partitioned into a validation dataset, resulting in 111 training images and 25 validation images. Furthermore, the number of training examples was increased to 444 by rotating the images 45°, 90° and 135°. During network training, training and validation pixel accuracy, mean pixel accuracy and defect-class accuracy were determined after every 50 training steps. As shown in figure 7.2, the learning process is divided in two parts: first, the network propagates the input image forward, so as to infer a prediction and calculate the difference between prediction image and label image via cross-entropy loss. Then, the loss gradients are propagated backwards through the network so as to update the parameter values and thereby iteratively minimise the loss. The basic procedure of network training is sketched by the following pseudo code:

**Algorithmus 1 :** Network Training

```
Data : training and validation datasets
set hyperparameters: learning rate, L2-regularisation strength, loss weights;
set mini-batch size and number of epochs;
load data and partition it into mini-batches;
initialise weights and biases;
while number of epochs > 0 AND early stopping criterion not satisfied do
    randomly shuffle mini-batches;
    foreach mini-batch in the training dataset do
        Propagate the input forward through the network:
        process the mini-batch and infer a prediction;
        calculate the cross-entropy loss;
        if number of steps mod 50 == 0 then
            determine training and validation metrics;
        end
        Propagate the loss backward through the network:
        calculate parameter updates via RMSprop;
        update each network weight and bias;
    end
end
save the network's final parameter values;
```

Here, the following, empirically determined hyperparameter values are used for network training: the learning rate, which determines the magnitude of the parameter updates in combination with the optimiser method, is initially set to $5 \cdot 10^{-4}$ and then exponentially decayed every 250 steps with a decay rate of 0.96. In order to constrain the network's complexity and prevent overfitting, the L2 norm of the network's weights is added to the cross-entropy loss, where the L2-regularisation strength is set to $5 \cdot 10^{-4}$. Moreover, to equalise the unbalanced class categories, defect-class losses are multiplied with 2,000 and losses of the remaining two class categories are multiplied with 100. Parameter updates

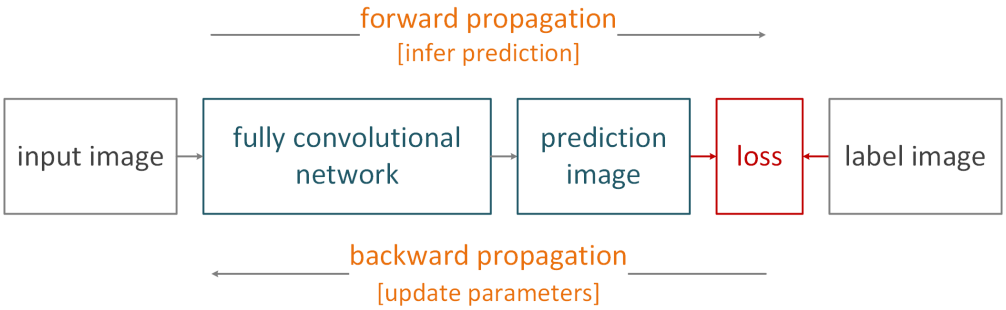



Figure 7.2.: Training a neural-network algorithm is divided in two parts: first, an input image is propagated forward through the network to infer a prediction image. Based on the prediction image and the corresponding label image, the cross-entropy loss of every output pixel is calculated. Then, the loss gradients are propagated backwards through the network so as to update the parameter values via gradient-descent optimisation. This cycle of forward and backward propagation is iterated until the optimisation process converges and a parameter combination is found that minimises the loss for the training data.

are calculated by an RMSprop optimiser, which individually adjusts the learning rate for every parameter. As is common for fully convolutional networks [77], the data is partitioned into mini-batch sizes of 1, that is one input-label pair is processed by the network at a time, denoted as (training) step. Overall, the network is trained for 80 epochs, where an epoch refers to a full iteration over all shuffled examples in the training dataset and thus one epoch covers as many steps as there are mini-batch partitions in the training dataset. Next to the number of epochs, the duration of network training can also be adjusted by an early stopping criterion: as an instance, training is prematurely stopped if validation accuracy did not increase for the last $n$ epochs.

Following hyperparameter setup, data loading and partitioning as well as network initialisation, network training is performed for the given number of epochs or until the early stopping criterion is satisfied. To speed up the training process, network training was performed on an NVIDIA Tesla P100 (16 GB) GPU in addition to an Intel Xeon 2.60 GHz CPU and took about three hours. Furthermore, all developed code was written in Python 3.6, where the Python libraries NumPy [88] and numpy-groupies were used for tensor operations, matplotlib was used for visualisations [52] and the network itself was written in the TensorFlow framework, version 1.8 [1]. During network training, the network architecture's graph together with the learned parameter values are saved in various, TensorFlow-specific checkpoint files. Based on these checkpoint files, the network can be easily restored and finetuned to new data. Here, the learned parameter values are used to initialise the network graph, before re-training the network for a small number of epochs and with a small learning rate on an extended dataset, which comprises old and

new training examples. Alternatively, a dataset of only new examples may be employed. To implement the network in production, however, it is convenient to save only the information needed for forward inference. Here, all network nodes for backward propagation are discarded and the parameter values are stored as constant, numerical values. The resulting, so-called frozen model can then be employed as part of an inference pipeline, which inputs photoluminescence measurements and outputs the inferred defect map.

## 7.3. Evaluation

As described in the previous chapter, training the developed *dense ASPP Vaughan* architecture on the aforementioned dataset of photoluminescence images with embedded defect information from an ultrasonic measurement yields a validation pixel accuracy of 97.5 %, mean pixel accuracy of 96.2 %, and defect-class accuracy of 92.0 %. Here, the dataset covers 111 training and 25 validation images, where the size of the training set was increased to 444 images via data augmentation. The achieved segmentation accuracy is a result of the specifically designed network architecture in combination with a comprehensive tuning process, which covered architectural design aspects and hyperparameter tuning. Moreover, data-specific adjustments, such as weighted loss calculation, photoluminescence images with ultrasonic-measurement embeddings and careful dataset compilation, distinctively increased network performance. The analysis of prediction images reveals that the developed network architecture reliably distinguishes salient brightness values corresponding to defect structures from functional structures, non-defect structures and measurement artefacts and thus yields accurate segmentation results for single defective LED chips as well as defect structures that occur repeatedly in the dataset or feature a comparatively common shape. Additionally, a test defect-class accuracy of 91.5 % on a test dataset of 366 measurements from the running production indicates that the *dense ASPP Vaughan* architecture generalises well to unknown wafer images. However, prediction-image analysis also shows that uncommonly shaped, large defect clusters still may result in flawed segmentations because this defect type occurs rarely in the dataset and features highly variable brightness patterns. By increasing dataset size over time with photoluminescence images of wafers that show salient defect structures the empirical distribution of the training dataset further approaches the true, data-generating distribution. Consequently, doubling the dataset size with 134 additional training examples of wafers with salient defect structures results in an increased pixel accuracy of 97.6 %, mean pixel accuracy of 96.8 % and defect-class accuracy of 93.5 %.

## 7.4. Summary

In this chapter, the insights acquired in the previous chapters about network theory, architecture design, hyperparameter tuning and dataset compilation were brought together and the final design, implementation and evaluation of the developed network architecture were presented. First, the resulting network architecture was described, which was specifically designed for the detection of defective LED chips in photoluminescence images. Here, the combination of a novel upsampling path, densely connected convolutional blocks and atrous-spatial-pyramid-pooling modules increases the output resolution and allows the network to accurately segment multiple scaled objects, such as single defective LED chips and defect structures. Then, the hyperparameter setup was specified, which results from the comprehensive tuning analysis in chapter 6. Details about network implementation, the used programming framework as well as network evaluation complete the chapter. The following chapter concludes this work by summarising the findings and limitations, evaluating the practical application and recommending further research possibilities.

# 8. Conclusion

The manufacturing of light-emitting diodes (LEDs) is a complex semiconductor process, which is interspersed with measurements. However, the employment of contact-measurement techniques, such as wafer probing, is getting increasingly difficult—if feasible at all—due to ever decreasing chip sizes and advanced chip designs that cannot be contacted by prober needles. While electrical and optical LED-chip properties can be determined by sample measurements, defective LED chips are distributed randomly over the wafer. One solution approach are advanced data-analysis methods, which gather additional information from already employed non-contact measurements, such as photoluminescence measurements: comparing brightness images generated by a photoluminescence measurement with wafer-probing-derived defect maps reveals that defective LED chips can be recognised in photoluminescence images as well, due to conspicuous brightness values. As shown in chapter 2 and figure 8.1, however, photoluminescence images feature varying brightness values from wafer to wafer in addition to local areas of differing brightness. Furthermore, not all salient structures visible on wafer images correlate to defective LED chips, which makes it infeasible to create a reliable computer vision algorithm by hand.

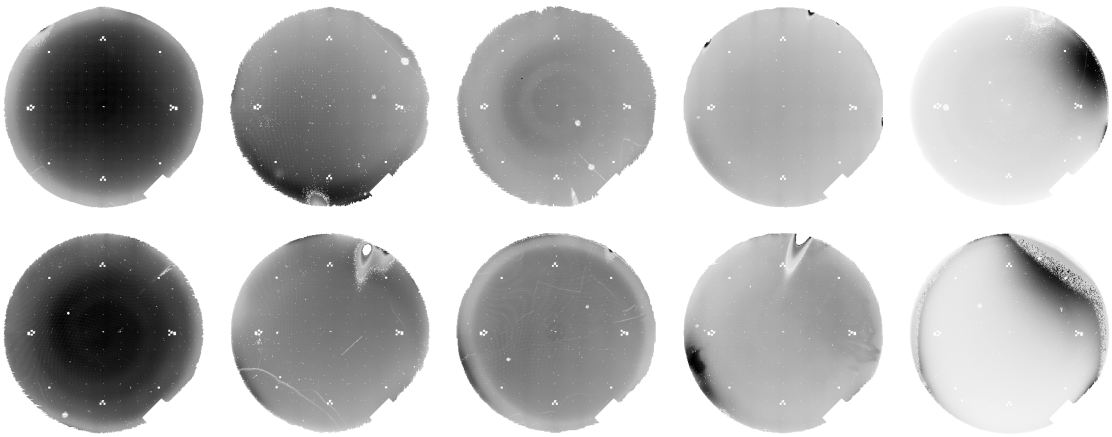


Figure 8.1.: Overview over the range of brightness variations in photoluminescence wafer images and the different shapes and sizes of defect structures.

Therefore, this work studied the employment of fully-convolutional-network algorithms for the detection of defective LED chips in photoluminescence images. Being self-learning algorithms, fully convolutional networks enable the pixel-wise classification of the input

image by learning thousands of task-specific convolutional filters, based on the training data [77]. Hence, fully-convolutional-network algorithms are the state-of-the-art method for semantic-segmentation tasks, covering a wide range of applications, as presented in chapter 4. With regard to the analysis of photoluminescence images, the network algorithm is trained to segment a wafer image into three class categories, namely defect class, in-spec class and a miscellaneous class, where the last category subsumes functional structures and non-wafer areas. In other words, the network algorithm is trained to infer a class category for every pixel of the photoluminescence image, where each pixel corresponds either to an LED chip or to non-wafer background pixels and functional structures, respectively. Hereby, the special composition of photoluminescence images poses a challenge for the network design: on the one hand, photoluminescence images depict less image objects than research-dataset images, which provokes overfitting and diminishes the network's generalisation ability. And on the other hand, the possible defect structures assume very different sizes and shapes, where defect scales range from single defective LED chips to large defect clusters. Moreover, the scarcest occurring defect structure, defect clusters, show the highest variation in size and shape as shown in figure 8.1, which exacerbates the compilation of a comprehensive dataset. Additionally, because only a minority of LED chips on a wafer are defective, the three class categories are highly unbalanced. Thus, only wafer images that exhibit salient defect structures were added to the dataset. The resulting dataset, which was studied in chapter 2, consists of 136 photoluminescence input images and corresponding wafer-probing-derived label images, of which only a fraction showed large defect clusters.

In order to develop a fully-convolutional-network architecture that addresses the aforementioned challenges, which were presented in chapter 2, the theoretical background was studied in chapter 3, followed by state-of-the-art network architectures and related work with regard to street scene datasets, medical imaging and industrial applications in chapter 4. Then, a modified fully-convolutional-network architecture was presented in chapter 5, specifically designed with regard to photoluminescence images. Additionally, the possible extension of the developed architecture with advanced architectural methods was introduced. Based on the acquired insights, comprehensive experiments were conducted and presented in chapter 6: first, the effects of different architectural design choices in combination with the given dataset were studied, resulting in a shallow architecture, where all intermediate downsampling and upsampling layers are connected via skip connections, which distinctively increase network performance. Then, the influence of hyperparameter tuning on network performance was examined, revealing the interaction of learning rate, L2-regularisation strength and optimiser method. Experiments with transfer learning exposed the benefit of a network partly initialised with transferred, pre-trained parameter values, despite the inherently different dataset the parameter-giving network was trained on. In order to equalise the aforementioned highly unbalanced class categories, individually weighted loss calculation was studied. Here, a combination of an overall loss weight of 100 with a defect-class-specific loss weight of 2,000 distinctively increased defect-class accuracy and mean pixel accuracy. Further

increasing defect-class-loss weights resulted in even higher defect-class accuracies but with an adverse effect on the overall pixel accuracy. Finally, analysing prediction images revealed that the network had adopted repeated input-label mismatches, which were resolved by embedding ultrasonic-defect information into the photoluminescence input images. The resulting network architecture, named *Vaughan*, achieved a pixel accuracy of 98.0 %, mean pixel accuracy of 94.3 % and defect-class accuracy of 86.6 % and the examined prediction images showed an accurate segmentation of single defective LED chips and common defect structures, such as voids and cracks. Furthermore, the network had learned to distinguish salient non-defect structures, such as film tears, functional structures and measurement artefacts from actual defect structures. Due to the insufficient segmentation of scarcely occurring, large defect clusters, however, the implementation of advanced architectural concepts into the developed architecture was studied. On the one hand, densely connected convolutional layers encourage the re-use of feature maps and thus enabled a more condensed, less overfitting network architecture. And on the other hand, two ASPP modules increased segmentation accuracy by analysing incoming feature maps at multiple scales at once. The resulting *dense ASPP Vaughan* network yields a pixel accuracy of 97.5 %, mean pixel accuracy of 96.2 % and defect-class accuracy of 92.0 % on the aforementioned dataset.

Next to the analysis of network design and hyperparameter tuning, chapter 6 also studied the influence of dataset compilation, using 4-fold cross validation. Here, the large difference in validation performance between the different folds revealed the great influence of single training examples on network performance, due to the high variability in defect structures. Incidentally, independent of the dataset partition, each network learned to accurately segment repeatedly occurring defect structures, single defective LED chips and functional structures. These results illustrate the importance of dataset compilation for small datasets with highly variable image objects and thus, providing the network with a manual training-validation split, which guarantees the thorough coverage of available defect clusters as well as a representative validation set, outperformed the cross-validation average. Overall, the analysis of prediction images (see also figure 8.2) and network performance verifies that fully convolutional networks can be used for the detection of defective LED chips in photoluminescence images. Moreover, it can be expected that adding valuable training examples to the dataset, whenever they occur in the running production, will further increase segmentation accuracy, especially with regard to large defect clusters and other rare defect structures. Consequently, doubling the dataset size with 134 additional training examples resulted in an increase in pixel accuracy of 97.6 %, mean pixel accuracy of 96.8 % as well as defect-class accuracy of 93.5 %, where the training examples covered mainly salient and a small number of rare defect structures.

In order to evaluate a possible employment of the developed network algorithm in an industrial environment, occurring misclassifications must be studied further: as described in chapter 2, at present, photoluminescence measurements are performed after wafer

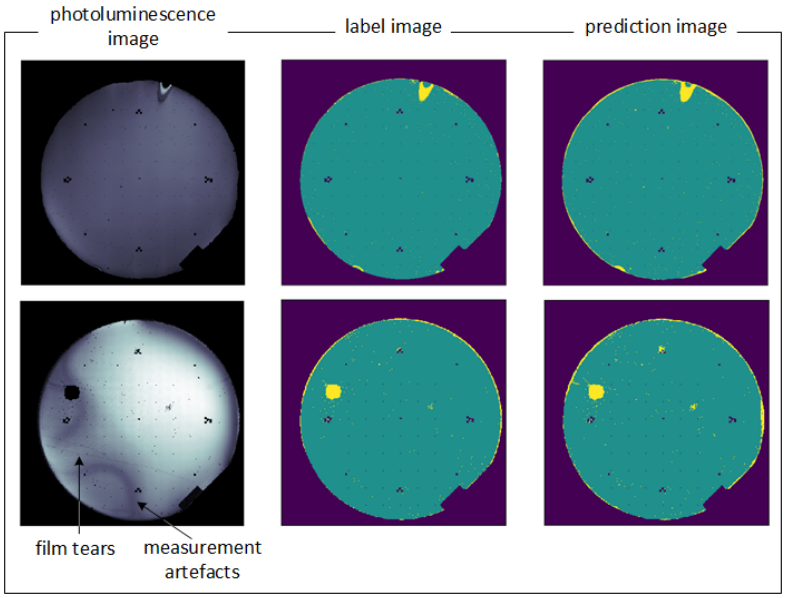



Figure 8.2.: Prediction images, generated by a *dense ASPP Vaughan* network, where the network has learned to distinguish salient non-defect and functional structures as well as measurement artefacts from actual defect structures. Due to the special network design, the network is able to accurately segment defect structures of multiple scales, such as single defective LED chips, voids, cracks and defect clusters.

probing and a subsequent chip-separation step. Therefore, possible separation damages, especially at the wafer edge, are not yet present while wafer probing and thus cause wrongful misclassifications. Additionally, not all defects detected by thorough electrical and optical tests manifest in conspicuous brightness values, resulting in defective LED chips that cannot be detected via photoluminescence images. Assuming that an additional photoluminescence measurement prior to wafer probing will be performed in the future, possible changes in the photoluminescence-measurement results along with the accordance of input images and label images must be investigated. Finally, monitoring and network re-training procedures must be developed that address the possible appearance of defect structures not yet portrayed in the dataset, due to changes in LED manufacturing. As an instance, it must be guaranteed that the network algorithm does not misclassify new defect structures due to their possible resemblance to film tears. It is therefore advisable to implement the algorithm in the manufacturing environment for test purposes for a continuous period, so as to acquire a thorough understanding of the network's performance in a running production. Note that industrial applications usually aim for a defect detection rate in the range of *parts per million* (ppm), where only $n$ defects per a million chips may be undetected, with $n \leq 50$ for example. Given the aforementioned restrictions, it seems unlikely that the developed network architecture

or any other algorithm can achieve according defect detection rates. Therefore, fully convolutional networks may be considered as one tool in an ensemble of data-analysis methods used to process available LED-wafer measurements in order to render wafer probing unnecessary. Moreover, given the network's ability to reliably segment input images despite highly variable brightness values, fully convolutional networks may also be employed for the detection of other defect structures occurring in photoluminescence measurements performed at the beginning of LED manufacturing.

Beyond that, network performance may be further improved by increasing the dataset size with additional training examples from the running production over time. In this regard, developing a network architecture that is able to input and output differently sized images would facilitate the addition of photoluminescence images of other LED-chip types to the dataset. Hereby, dataset size could be distinctively increased, next to enabling a more generic application of the network algorithm. Theoretically, fully convolutional networks are independent of the input and output image sizes, due to the thorough implementation of convolutional layers. Practically, however, input image, feature map and output-image dimensions are predetermined by the network's graph in the TensorFlow framework. Here, the soon to be released TensorFlow version 2.x, which omits the graph concept, might offer new possibilities. Moreover, due to the vast number of possible network architecture designs, hyperparameter values and novel concepts that are frequently published [61, 106, 129], future experiments based on the developed *dense ASPP Vaughan* network may further increase network performance as well. In order to increase the current segmentation accuracy, an ensemble of network algorithms may be implemented in addition to analysing all rotated versions of the photoluminescence wafer image, where the inferred defect maps are eventually combined. Furthermore, the developed network architecture may be employed to analyse other imaging measurements performed while LED manufacturing, where a combination of multiple defect maps derived from different measurements would presumably further increase prediction accuracy.

Next to improvements of the network algorithm, additional applications may be studied in the future: on the one hand, the inferred defect maps may be forwarded to an unsupervised clustering algorithm, which classifies the defect structures depending on their geometrical appearance and enables further evaluations and early identifications of process deviations. Alternatively, the *dense ASPP Vaughan* network may be used as starting point for the development of an unsupervised network architecture, which could be applied in cases where label images cannot be provided. On the other hand, experiments in chapter 6 revealed that the network adopts repeated input-label mismatches. This property may be used to perform measurement post-processing steps, such as the enlargement of predetermined defect structures, given an appropriately labelled dataset. Additionally, the network's ability to distinguish salient non-defect structures from defect structures, such as film tears from cracks, may be helpful for other data-analysis applications. Another advantage of neural-network algorithms is the transferability to similar

tasks: as an instance, the developed network can easily be applied to photoluminescence images of other LED-chip types, where the trained network parameters can be transferred to the new task and used as network initialisation. LED wafers that are composed of chips with similar dimensions and thus related photoluminescence-image sizes may even be analysed by the network algorithm without any changes, where the input images must be resized to the original input-image dimensions before processing. Moreover, the specially designed network architecture may also be beneficial for other industrial datasets, where the measurement images are similarly composed as photoluminescence images, such as the detection of voids in X-ray images of solder pads [130].

Altogether, the developed fully convolutional network, denoted as *dense ASPP Vaughan*, represents a novel network architecture for the segmentation of multiple-scaled objects in photoluminescence-measurement images and provides a pixel-fine and thus chip-fine output resolution. The thorough study of network design, hyperparameter tuning and dataset compilation in addition to an analysis of prediction images in this work leads to a specialised network design, which accurately segments single defective LED chips as well as large defect structures, while distinguishing measurement artefacts, non-defect structures and functional structures from actual defect structures.

# A. Notation

| | |
|---|---|
| $\boldsymbol{a}_i$ | Element $i$ of vector $a$, with indexing starting at 1 |
| $\boldsymbol{A}_{i,j}$ | Element $i, j$ of matrix $A$ |
| $X$ | Space of input images |
| $Y$ | Space of label images |
| $m$ | Number of i.i.d. examples in the dataset $\mathcal{D}$ |
| $\{(x_i, y_i)\}_{i=1}^m$ | Minibatch $\mathcal{B}$ of $m$ input-label pairs |
| $y$ | True label |
| $\hat{y}$ | Inferred label |
| $\mathcal{D}$ | Empirically distributed dataset with examples from $P^*$ |
| $P^*$ | Distribution that generated the data |
| $\tilde{P}$ | Learned or estimated distribution |
| $\mathbb{E}$ | Expected value |
| $\mathcal{F}$ | Hypothesis space of all candidate mapping functions $f \in \mathcal{F}$ |
| $f(x)$ | True underlying solution function of training images $x$ and labels $y$ |
| $f^*(x)$ | Correct solution function as determined by the algorithm |
| $\tilde{f}(x)$ | Learned approximation to the correct function as determined by the algorithm |
| $a \sim P$ | Random variable $a$ has distribution $P$ |
| $L(\hat{y}, y)$ | Loss function |
| $R(f)$ | Restriction term |
| $W, B$ | Weights and biases |
| $n$ | Input size / number of input pixels |
| $c$ | Output size / number of class categories |
| $H$ | Number of units / neurons in a fully-connected layer |
| $H \times W$ | Kernel size (height $\times$ width) |
| $\boldsymbol{p}$ | Prediction vector of normalised values between 0 and 1 |
| $\lambda \|w\|_2^2$ | L2-regularisation term, where $\lambda$ determines the regularisation strength based on the L2 norm $\|w\|_2^2$ |
| $\boldsymbol{\theta}$ | Parameter vector, commonly subsumes $W$ and $B$ |
| $\eta$ | Learning rate |
| $\boldsymbol{v}_t$ | Update vector |
| $\epsilon$ | Value in the order of $10^{-8}$ to avoid division by zero |
| BN | Batch normalisation |
| $\phi(z)$ | Activation function |
| $z_{i',j',f'}$ | Output of a convolution operation, located at the kernel centre in output feature map $f'$ |
| $s$ | Stride |
| $r$ | Dilation rate |

# B. Tables

The following tables provide an overview of the results and details of the corresponding plots in chapter 6.

Table B.1.: Influence of residual shortcuts and skip connections on the training progress. The table lists validation pixel accuracy (PA), mean pixel accuracy (MPA) and defect-class accuracy (DCA). *Blank*: neither residual shortcuts nor skip connections implemented, *no skips*: residual shortcuts in every three-layer convolutional block but no skip connections. 1 *skip* - 5 *skips*: gradually added skip connections, where the first skip connection fuses the most inner layers. See also figure 6.2.

| | PA | MPA | DCA |
|---|---|---|---|
| blank | 91.3 | 61.8 | 0.01 |
| no skips | 91.2 | 61.7 | 0.03 |
| 1 skip | 91.2 | 61.7 | 0.02 |
| 2 skips | 98.4 | 80.8 | 43.7 |
| 3 skips | 98.6 | 82.6 | 48.8 |
| 4 skips | 98.7 | 84.3 | 53.8 |
| 5 skips | 98.7 | 84.1 | 53.7 |

Table B.2.: Effect of different activation functions on the network performance. The table lists validation pixel accuracy (PA), mean pixel accuracy (MPA) and defect-class accuracy (DCA) of a network trained with ReLU, leaky ReLU, softplus and tanh activation functions in all intermediate layers, respectively. See also figure 6.7.

| | PA | MPA | DCA |
|---|---|---|---|
| ReLU | 98.4 | 80.4 | 43.9 |
| leaky ReLU | 98.3 | 79.6 | 41.3 |
| softplus | 98.3 | 79.3 | 40.3 |
| tanh | 98.4 | 79.9 | 41.4 |

Table B.3.: Influence of different learning-rate values on the training progress, where the employed learning rates were determined randomly on a logarithmic scale. The table lists validation and training pixel accuracy (PA), mean pixel accuracy (MPA) and defect-class accuracy (DCA). See also figure 6.8.

| L2 value | | validation | | | test | | |
|---|---|---|---|---|---|---|---|
| learning rate | | PA | MPA | DCA | PA | MPA | DCA |
| $1 \cdot 10^{-1}$ | 0.1 | 98.0 | 74.6 | 25.6 | 97.9 | 77.6 | 32.0 |
| $1 \cdot 10^{-1.15}$ | 0.071 | 98.3 | 79.5 | 39.1 | 98.4 | 84.3 | 50.1 |
| $1 \cdot 10^{-2.21}$ | 0.00617 | 98.6 | 83.5 | 51.7 | 99.0 | 90.3 | 69.2 |
| $1 \cdot 10^{-2.62}$ | 0.00239 | 98.7 | 84.4 | 54.5 | 99.2 | 91.7 | 74.0 |
| $1 \cdot 10^{-3.1}$ | 0.000794 | 98.7 | 84.4 | 54.4 | 99.3 | 92.8 | 77.0 |
| $1 \cdot 10^{-3.22}$ | 0.0006 | 98.7 | 84.1 | 53.5 | 99.3 | 92.9 | 77.6 |
| $1 \cdot 10^{-3.84}$ | 0.000145 | 98.6 | 83.1 | 50.5 | 99.2 | 92.4 | 75.4 |
| $1 \cdot 10^{-4.44}$ | 0.0000363 | 98.4 | 80.1 | 41.7 | 98.9 | 88.2 | 61.3 |
| $1 \cdot 10^{-5}$ | 0.00001 | 98.2 | 76.4 | 30.0 | 98.1 | 79.7 | 35.7 |

Table B.4.: L2 regularisation results in combination with a decreasing learning rate ($1 * 10^{-2.62}/0.00239$). The table lists validation and training pixel accuracy (PA), mean pixel accuracy (MPA) and defect-class accuracy (DCA). See also figure 6.10 and figure 6.11

| L2 value | | validation | | | test | | |
|---|---|---|---|---|---|---|---|
| | | PA | MPA | DCA | PA | MPA | DCA |
| $1 \cdot 10^{-1}$ | 0.1 | 98.1 | 73.3 | 21.7 | 97.6 | 73.7 | 20.9 |
| $1 \cdot 10^{-1.15}$ | 0.071 | 98.2 | 74.7 | 25.8 | 97.7 | 75.2 | 25.1 |
| $1 \cdot 10^{-2.21}$ | 0.00617 | 98.5 | 79.9 | 41.1 | 98.2 | 80.8 | 40.3 |
| $1 \cdot 10^{-2.62}$ | 0.00239 | 98.6 | 81.6 | 46.1 | 98.4 | 83.6 | 48.3 |
| $1 \cdot 10^{-3.1}$ | 0.000794 | 98.6 | 82.6 | 49.1 | 98.6 | 85.8 | 54.8 |
| $5 \cdot 10^{-4.0}$ | 0.0005 | 98.6 | 83.2 | 51.0 | 98.8 | 87.4 | 59.8 |
| $1 \cdot 10^{-3.84}$ | 0.000145 | 98.7 | 83.7 | 52.3 | 98.9 | 89.5 | 66.2 |
| $1 \cdot 10^{-4.44}$ | 0.0000363 | 98.7 | 84.1 | 53.5 | 99.0 | 90.4 | 69.3 |
| $1 \cdot 10^{-5}$ | 0.00001 | 98.7 | 84.4 | 54.5 | 99.2 | 91.7 | 73.5 |

Table B.5.: L2 regularisation results in combination with a decreasing learning rate ($5 * 10^{-4}$/0.0008). The table lists validation pixel accuracy (PA), mean pixel accuracy (MPA) and defect-class accuracy (DCA). See also figure 6.11.

| L2 value | | validation | | | test | | |
|---|---|---|---|---|---|---|---|
| | | PA | MPA | DCA | PA | MPA | DCA |
| $1 \cdot 10^{-1}$ | 0.1 | 98.4 | 78.3 | 36.6 | 98.1 | 79.2 | 36.3 |
| $1 \cdot 10^{-1.15}$ | 0.071 | 98.4 | 79.1 | 38.8 | 98.1 | 79.8 | 37.9 |
| $1 \cdot 10^{-2.21}$ | 0.00617 | 98.6 | 82.4 | 48.5 | 98.6 | 85.2 | 53.3 |
| $1 \cdot 10^{-2.62}$ | 0.00239 | 98.7 | 83.5 | 51.8 | 98.8 | 88.0 | 61.7 |
| $1 \cdot 10^{-3.1}$ | 0.000794 | 98.7 | 84.1 | 53.5 | 99.0 | 90.2 | 68.6 |
| $5 \cdot 10^{-4.0}$ | 0.0005 | 98.7 | 84.2 | 54.0 | 99.0 | 90.6 | 70.1 |
| $1 \cdot 10^{-3.84}$ | 0.000145 | 98.7 | 84.3 | 54.0 | 99.2 | 92.0 | 74.4 |
| $1 \cdot 10^{-4.44}$ | 0.0000363 | 98.7 | 84.0 | 53.0 | 99.2 | 91.9 | 74.3 |
| $1 \cdot 10^{-5}$ | 0.00001 | 98.7 | 84.1 | 53.5 | 99.3 | 92.8 | 77.1 |

Table B.6.: Influence of varying L2 values on the training progress, illustrated by a network trained with low (L2 = $1 \cdot 10^{-5}$), medium (L2 = $5 \cdot 10^{-4}$) and strong regularisation (L2 = 0.1). See also figure 6.12.

| | PA | MPA | DCA |
|---|---|---|---|
| low regularisation | 98.3 | 82.8 | 51.0 |
| medium regularisation | 98.4 | 83.5 | 53.3 |
| strong regularisation | 98.0 | 80.4 | 44.6 |

Table B.7.: Comparison of different optimiser methods, namely gradient descent, gradient descent with momentum, AdaGrad, RMSprop and Adam. The table lists validation pixel accuracy (PA), mean pixel accuracy (MPA) and defect-class accuracy (DCA). See also figure 6.13.

| | PA | MPA | DCA |
|---|---|---|---|
| learning rate $1 \cdot 10^{-2.62}$ / 0.00239 | | | |
| Gradient Descent | 98.2 | 78.0 | 35.0 |
| Momentum | 97.9 | 77.2 | 33.3 |
| AdaGrad | 98.3 | 79.2 | 38.7 |
| RMSprop | 98.6 | 83.3 | 51.3 |
| Adam | 98.6 | 83.2 | 51.0 |
| learning rate $8 \cdot 10^{-4}$ / 0.0008 | | | |
| Gradient Descent | 98.3 | 76.8 | 31.0 |
| Momentum | 98.3 | 79.5 | 40.0 |
| AdaGrad | 98.3 | 77.4 | 32.9 |
| RMSprop | 98.7 | 84.3 | 54.0 |
| Adam | 98.7 | 84.2 | 53.7 |

Table B.8.: Effects of differently weighted loss calculations on network performance, where *none* refers to no loss weights, *all 100* refers to the multiplication of all loss gradients with 100, despite the class category, and the remaining values refer to additional loss weights added to the defect-class category alone. The table lists validation pixel accuracy (PA), mean pixel accuracy (MPA) and defect-class accuracy (DCA). See also figure 6.14.

| loss value | PA | MPA | DCA |
|---|---|---|---|
| none | 98.5 | 80.1 | 41.7 |
| all 100 | 98.7 | 84.3 | 54.2 |
| 200 | 98.5 | 87.3 | 64.2 |
| 500 | 98.3 | 89.1 | 70.0 |
| 1000 | 98.0 | 89.9 | 73.0 |
| 2000 | 97.6 | 90.4 | 75.8 |
| 5000 | 96.8 | 91.0 | 78.8 |
| 7500 | 96.0 | 91.4 | 81.5 |
| 10000 | 95.7 | 91.1 | 81.3 |
| 15000 | 94.5 | 91.1 | 83.5 |
| 30000 | 92.5 | 90.3 | 84.6 |

Table B.9.: Validation pixel accuracy, mean pixel accuracy and defect-class accuracy of the *Vaughan* network architecture developed in the first part of chapter 6 as well as modified architectures, all trained on ultrasonic-embedded photoluminescence images. The first modification are densely connected layers, followed by dense networks with atrous-spatial-pyramid-pooling (ASPP) modules. See also figure 6.18.

| | PA | MPA | DCA |
|---|---|---|---|
| *Vaughan* | 98.0 | 94.3 | 86.6 |
| dense *Vaughan* | 97.4 | 95.6 | 90.6 |
| 1 ASPP module (downsampling) | 97.3 | 95.5 | 91.0 |
| 2 ASPP modules | 97.5 | 96.2 | 92.0 |
| 1 ASPP module (upsampling) | 97.5 | 95.4 | 91.2 |

Table B.10.: Validation pixel accuracy, mean pixel accuracy and defect-class accuracy as well as test defect-class accuracy (on test dataset 2) of a network trained with 4-fold cross validation, in comparison to a network trained with a manually split dataset. Analysing the metrics reveals the distinct influence of the training experience on network performance. See also figure 6.21.

| split | PA | MPA | DCA | DCA test2 |
|---|---|---|---|---|
| $k\,1$ | 96.9 | 92.3 | 76.8 | 63.5 |
| $k\,2$ | 97.1 | 97.9 | 93.3 | 75.1 |
| $k\,3$ | 97.8 | 97.9 | 93.4 | 86.5 |
| $k\,4$ | 97.5 | 95.9 | 87.8 | 75.7 |
| average | 97.3 | 96.0 | 87.8 | 75.2 |
| manual | 97.5 | 96.2 | 92.0 | 75.5 |

# Bibliography


[1] Martin Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. 2015.

[2] Abrar H. Abdulnabi, Stefan Winkler, and Gang Wang. "Beyond Forward Shortcuts: Fully Convolutional Master-Slave Networks (MSNets) with Backward Skip Connections for Semantic Segmentation". In: *CoRR* abs/1707.05537 (2017). arXiv: 1707.05537.

[3] Charu C. Aggarwal. *Neural Networks and Deep Learning - A Textbook*. Springer, 2018.

[4] Said Amirul Anwar and Mohd Zaid Abdullah. "Micro-crack detection of multicrystalline solar cells featuring an improved anisotropic diffusion filter and image segmentation technique". In: *EURASIP Journal on Image and Video Processing* 2014 (2014), p. 15.

[5] Avi Ben-Cohen et al. "Fully Convolutional Network for Liver Segmentation and Lesions Detection". In: *LABELS/DLMIA@MICCAI*. 2016.

[6] Yoshua Bengio. "Practical recommendations for gradient-based training of deep architectures". In: *CoRR* abs/1206.5533 (2012). arXiv: 1206.5533.

[7] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006.

[8] Léon Bottou and Olivier Bousquet. "The Tradeoffs of Large Scale Learning". In: *Advances in Neural Information Processing Systems 20*. Ed. by J. C. Platt et al. Curran Associates, Inc., 2008, pp. 161–168.

[9] Y-Lan Boureau, Jean Ponce, and Yann LeCun. "A Theoretical Analysis of Feature Pooling in Visual Recognition". In: *27th International Conference on Machine Learning, Haifa, Israel*. 2010.

[10] Gabriel J. Brostow, Julien Fauqueur, and Roberto Cipolla. "Semantic Object Classes in Video: A High-Definition Ground Truth Database". In: *Pattern Recognition Letters* (2008).

[11] Daniel J. Butler et al. "A naturalistic open source movie for optical flow evaluation". In: *European Conf. on Computer Vision (ECCV)*. Ed. by A. Fitzgibbon et al. (Eds.) Part IV, LNCS 7577. Springer-Verlag, 2012, pp. 611–625.

[12] Alfredo Canziani, Adam Paszke, and Eugenio Culurciello. "An Analysis of Deep Neural Network Models for Practical Applications". In: *CoRR* abs/1605.07678 (2016). arXiv: 1605.07678.

[13] Albert Cardona et al. "An integrated micro- and macroarchitectural analysis of the Drosophila brain by computer-assisted serial section electron microscopy". In: *PLoS Biol.* (2010).

[14] Young-Jin Cha, Wooram Choi, and Oral Büyüköztürk. "Deep Learning-Based Crack Damage Detection Using Convolutional Neural Networks". In: *Computer-Aided Civil and Infrastructure Engineering* 32.5 (2017), pp. 361–378.

[15] Liang-Chieh Chen et al. "DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs". In: *CoRR* abs/1606.00915 (2016). arXiv: 1606.00915.

[16] Liang-Chieh Chen et al. "Rethinking Atrous Convolution for Semantic Image Segmentation". In: *CoRR* abs/1706.05587 (2017). arXiv: 1706.05587.

[17] Patrick Ferdinand Christ et al. "Automatic Liver and Tumor Segmentation of CT and MRI Volumes using Cascaded Fully Convolutional Neural Networks". In: *CoRR* abs/1702.05970 (2017). arXiv: 1702.05970.

[18] Özgün Çiçek et al. "3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation". In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2016*. Ed. by Sebastien Ourselin et al. Cham: Springer International Publishing, 2016, pp. 424–432.

[19] Marius Cordts et al. "The Cityscapes Dataset for Semantic Urban Scene Understanding". In: *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[20] George Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of Control, Signals and Systems* 2.4 (1989), pp. 303–314.

[21] Jifeng Dai, Kaiming He, and Jian Sun. "Instance-aware Semantic Segmentation via Multi-task Network Cascades". In: *CoRR* abs/1512.04412 (2015). arXiv: 1512.04412.

[22] Nando de Freitas. *University of Oxford: Machine Learning Course 2014-2015 - Lecture Notes*. Published online. Retrieved 05.10.2018. 2014.

[23] Jeffrey Dean et al. "Large Scale Distributed Deep Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1223–1231.

[24] Chao Dong et al. "Image Super-Resolution Using Deep Convolutional Networks". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 38.2 (Feb. 2016), pp. 295–307.

[25] John Duchi, Elad Hazan, and Yoram Singer. "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization". In: *Journal of Machine Learning Research* 12 (July 2011), pp. 2121–2159.

[26] Vincent Dumoulin and Francesco Visin. "A guide to convolution arithmetic for deep learning". In: *ArXiv e-prints* (Mar. 2016). eprint: 1603.07285.

[27] Dumitru Erhan et al. "Visualizing Higher-Layer Features of a Deep Network". In: *Technical Report, Univeristé de Montréal* (Jan. 2009).

[28] Mark Everingham et al. "The Pascal Visual Object Classes (VOC) Challenge". In: *International Journal of Computer Vision* 88.2 (2010), pp. 303–338.

[29] Shahrzad Faghih-Roohi et al. "Deep convolutional neural networks for detection of rail surface defects". In: *2016 International Joint Conference on Neural Networks (IJCNN)*. June 2016, pp. 2584–2589.

[30] Li Fei-Fei, Andrej Karpathy, and Justin Johnson. *Stanford University CS231n: Convolutional Neural Networks for Visual Recognition - Lecture Notes*. Published online. Retrieved 17.08.2018. 2017.

[31] Philipp Fischer et al. "FlowNet: Learning Optical Flow with Convolutional Networks". In: *CoRR* abs/1504.06852 (2015). arXiv: 1504.06852.

[32] Wolf-Joachim Fischer. *Mikrosystemtechnik*. 1st ed. Vogel Buchverlag, 2000.

[33] Kunihiko Fukushima. "Cognitron: A Self-Organizing Multilayer Neural Network". In: *Biological Cybernetics* 20 (1975), pp. 121–136.

[34] Kunihiko Fukushima. "Neocognitron: A Self-Organizing Neural Network Model for a Mechanism of Pattern Recognition Unaffected by Shift in Position". In: *Biological Cybernetics* 36 (1980), pp. 193–202.

[35] Gerald Gerlach and Wolfram Dötzel. *Einführung in die Mikrosystemtechnik*. 1st ed. Carl Hanser Verlag, 2006.

[36] Ross B. Girshick et al. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *CoRR* abs/1311.2524 (2013). arXiv: 1311.2524.

[37] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS'10). Society for Artificial Intelligence and Statistics*. 2010.

[38] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep Sparse Rectifier Neural Networks". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Geoffrey Gordon, David Dunson, and Miroslav Dudík. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, Apr. 2011, pp. 315–323.

[39] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[40] Kristen Grauman and Trevor Darrell. "The Pyramid Match Kernel: Efficient Learning with Sets of Features". In: *J. Mach. Learn. Res.* 8 (May 2007), pp. 725–760.

[41] Ryuhei Hamaguchi et al. "Effective Use of Dilated Convolutions for Segmenting Small Object Instances in Remote Sensing Imagery". In: *CoRR* abs/1709.00179 (2017). arXiv: 1709.00179.

[42] Kaiming He et al. "Deep Residual Learning for Image Recognition". In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385.

[43] Kaiming He et al. "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification". In: *CoRR* abs/1502.01852 (2015). arXiv: 1502.01852.

[44] Kaiming He et al. "Identity Mappings in Deep Residual Networks". In: *CoRR* abs/1603.05027 (2016). arXiv: 1603.05027.

[45] Kaiming He et al. "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition". In: *CoRR* abs/1406.4729 (2014). arXiv: 1406.4729.

[46] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. *University of Toronto, CSC321: Neural Networks for Machine Learning - Lecture Notes.* Published online. Retrieved 01.12.2018. 2012.

[47] Matthias Holschneider et al. "A Real-Time Algorithm for Signal Analysis with the Help of the Wavelet Transform". In: *Wavelets*. Ed. by Jean-Michel Combes, Alexander Grossmann, and Philippe Tchamitchian. Berlin, Heidelberg: Springer Berlin Heidelberg, 1990, pp. 286–297.

[48] Kurt Hornik. "Approximation Capabilities of Multilayer Feedforward Networks". In: *Neural Networks* 4.2 (Mar. 1991), pp. 251–257.

[49] Andrew G. Howard et al. "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications". In: *CoRR* abs/1704.04861 (2017). arXiv: 1704.04861.

[50] Gao Huang, Zhuang Liu, and Kilian Q. Weinberger. "Densely Connected Convolutional Networks". In: *CoRR* abs/1608.06993 (2016). arXiv: 1608.06993.

[51] Mi-Young Huh, Pulkit Agrawal, and Alexei A. Efros. "What makes ImageNet good for transfer learning?" In: *CoRR* abs/1608.08614 (2016). arXiv: 1608.08614.

[52] John D. Hunter. "Matplotlib: A 2D Graphics Environment". In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95.

[53] Sergey Ioffe and Christian Szegedy. "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift". In: *CoRR* abs/1502.03167 (2015). arXiv: 1502.03167.

[54] Joel Janai et al. "Computer Vision for Autonomous Vehicles: Problems, Datasets and State-of-the-Art". In: *CoRR* abs/1704.05519 (2017). arXiv: 1704.05519.

[55] Andreas Jansson et al. "Singing voice separation with deep U-Net convolutional networks". 2017.

[56] Kevin Jarrett et al. "What is the best multi-stage architecture for object recognition?" In: *2009 IEEE 12th International Conference on Computer Vision*. 2009, pp. 2146–2153.

[57] Simon Jégou et al. "The One Hundred Layers Tiramisu: Fully Convolutional DenseNets for Semantic Segmentation". In: *CoRR* abs/1611.09326 (2016). arXiv: 1611.09326.

[58] Alex Kendall, Vijay Badrinarayanan, and Roberto Cipolla. "Bayesian SegNet: Model Uncertainty in Deep Convolutional Encoder-Decoder Architectures for Scene Understanding". In: *CoRR* abs/1511.02680 (2015). arXiv: 1511.02680.

[59] Diederik P. Kingma and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *CoRR* abs/1412.6980 (2014). arXiv: 1412.6980.

[60] Daphne Koller and Nir Friedman. *Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning*. The MIT Press, 2009.

[61] Jean Kossaifi et al. "T-Net: Parametrizing Fully Convolutional Nets with a Single High-Order Tensor". In: *arXiv e-prints* (2019). arXiv: 1904.02698.

[62] Philipp Krähenbühl and Vladlen Koltun. "Efficient Inference in Fully Connected CRFs with Gaussian Edge Potentials". In: *CoRR* abs/1210.5644 (2012). arXiv: 1210.5644.

[63] Alex Krizhevsky. "One weird trick for parallelizing convolutional neural networks". In: *CoRR* abs/1404.5997 (2014). arXiv: 1404.5997.

[64] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1097–1105.

[65] Svetlana Lazebnik, C. Schmid, and J. Ponce. "Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories". In: *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*. Vol. 2. June 2006, pp. 2169–2178.

[66] Yann LeCun. "Generalization and network design strategies". In: *Connectionism in perspective*. Ed. by R. Pfeifer et al. Elsevier, 1989.

[67] Yann LeCun. "Learning Process in an Asymmetric Threshold Network". In: *Disordered Systems and Biological Organization*. Ed. by E. Bienenstock, F. Fogelman Soulié, and G. Weisbuch. Berlin, Heidelberg: Springer Berlin Heidelberg, 1986, pp. 233–240.

[68] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[69] Ruirui Li et al. "DeepUNet: A Deep Fully Convolutional Network for Pixel-level Sea-Land Segmentation". In: *CoRR* abs/1709.00201 (2017). arXiv: 1709.00201.

[70] Yi Li et al. "Fully Convolutional Instance-aware Semantic Segmentation". In: *CoRR* abs/1611.07709 (2016). arXiv: 1611.07709.

[71] Yiting Li et al. "Research on a Surface Defect Detection Algorithm Based on MobileNet-SSD". In: *Applied Sciences* 8 (Sept. 2018), p. 1678.

[72] Guosheng Lin et al. "RefineNet: Multi-Path Refinement Networks for High-Resolution Semantic Segmentation". In: *CoRR* abs/1611.06612 (2016). arXiv: 1611.06612.

[73] Tsung-Yi Lin et al. "Microsoft COCO: Common Objects in Context". In: *CoRR* abs/1405.0312 (2014). arXiv: 1405.0312.

[74] Geert J. S. Litjens et al. "A Survey on Deep Learning in Medical Image Analysis". In: *CoRR* abs/1702.05747 (2017). arXiv: 1702.05747.

[75] Wei Liu, Andrew Rabinovich, and Alexander C. Berg. "ParseNet: Looking Wider to See Better". In: *CoRR* abs/1506.04579 (2015). arXiv: 1506.04579.

[76] Wei Liu et al. "SSD: Single Shot MultiBox Detector". In: *CoRR* abs/1512.02325 (2015). arXiv: 1512.02325.

[77] Jonathan Long, Evan Shelhamer, and Trevor Darrell. "Fully Convolutional Networks for Semantic Segmentation". In: *CoRR* abs/1411.4038 (2014). arXiv: 1411.4038.

[78] Hans-Jürgen Lugauer. *Grundlagen der Halbleiter-Epitaxy - Vorlesung LED-Technologie OTH Regensburg*. unpublished. 2013.

[79] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. "Rectifier Nonlinearities Improve Neural Network Acoustic Models". In: *ICML Workshop on Deep Learning for Audio, Speech and Language Processing*. 2013.

[80] Moritz Menze, Christian Heipke, and Andreas Geiger. "Object Scene Flow". In: *ISPRS Journal of Photogrammetry and Remote Sensing (JPRS)* (2018).

[81] Thomas M. Mitchell. *Machine Learning*. 1st ed. New York, NY, USA: McGraw-Hill, Inc., 1997.

[82] Alexander Mordvintsev, Christopher Olah, and Mike Tyka. *Inceptionism: Going Deeper into Neural Networks*. 2015.

[83] Vinod Nair and Geoffrey E. Hinton. "Rectified Linear Units Improve Restricted Boltzmann Machines". In: *Proceedings of the 27th International Conference on International Conference on Machine Learning*. ICML'10. Haifa, Israel: Omnipress, 2010, pp. 807–814.

[84] Dhanasekharan Natarajan. *ISO 9001 Quality Management Systems*. 1st. Springer Publishing Company, Incorporated, 2017.

[85] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. "Learning Deconvolution Network for Semantic Segmentation". In: *CoRR* abs/1505.04366 (2015). arXiv: 1505.04366.

[86] Augustus Odena, Vincent Dumoulin, and Chris Olah. "Deconvolution and Checkerboard Artifacts". In: *Distill* (2016).

[87] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. "Feature Visualization". In: *Distill* (2017).

[88] Travis Oliphant. *NumPy: A guide to NumPy*. USA: Trelgol Publishing. Online; accessed 17.04.2019. 2006.

[89] Aäron van den Oord et al. "WaveNet: A Generative Model for Raw Audio". In: *CoRR* abs/1609.03499 (2016). arXiv: 1609.03499.

[90] Nobuyuki Otsu. "A Threshold Selection Method from Gray-Level Histograms". In: *IEEE Transactions on Systems, Man, and Cybernetics* 9.1 (Jan. 1979), pp. 62–66.

[91] Sinno Jialin Pan and Qiang Yang. "A Survey on Transfer Learning". In: *IEEE Transactions on Knowledge and Data Engineering* 22.10 (Oct. 2010), pp. 1345–1359.

[92] Je-Kang Park et al. "Machine learning-based imaging system for surface defect inspection". In: *International Journal of Precision Engineering and Manufacturing-Green Technology* 3.3 (June 2016), pp. 303–310.

[93] Razvan Pascanu et al. "On the saddle point problem for non-convex optimization". In: *CoRR* abs/1405.4604 (2014).

[94] Fabian Pedregosa et al. "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12 (2011), pp. 2825–2830.

[95] Tobias Pohlen et al. "Full-Resolution Residual Networks for Semantic Segmentation in Street Scenes". In: *CoRR* abs/1611.08323 (2016). arXiv: 1611.08323.

[96] Christopher J. Raymond and Zhiqiang Li. "Photoluminescence metrology for LED characterization in high volume manufacturing". In: *Metrology, Inspection, and Process Control for Microlithography XXVII*. Vol. 8681. 2013.

[97] S. Hussain Raza, Matthias Grundmann, and Irfan Essa. "Geometric Context from Video". In: *IEEE CVPR* (2013).

[98] Joseph Redmon and Ali Farhadi. "YOLO9000: Better, Faster, Stronger". In: *CoRR* abs/1612.08242 (2016). arXiv: 1612.08242.

[99] Shaoqing Ren et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *Advances in Neural Information Processing Systems 28*. Ed. by C. Cortes et al. Curran Associates, Inc., 2015, pp. 91–99.

[100] Brian D. Ripley. *Pattern Recognition and Neural Networks*. Cambridge University Press, 1996.

[101] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *CoRR* abs/1505.04597 (2015). arXiv: 1505.04597.

[102] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: *CoRR* abs/1609.04747 (2016). arXiv: 1609.04747.

[103] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1". In: ed. by D. E. Rumelhart, James L. McClelland, and CORPORATE PDP Research Group. Cambridge, MA, USA: MIT Press, 1986. Chap. Learning Internal Representations by Error Propagation, pp. 318–362.

[104] Olga Russakovsky et al. "ImageNet Large Scale Visual Recognition Challenge". In: *CoRR* abs/1409.0575 (2014). arXiv: 1409.0575.

[105] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. 3rd. Upper Saddle River, NJ, USA: Prentice Hall Press, 2009.

[106] Sara Sabour, Nicholas Frosst, and Geoffrey E. Hinton. "Dynamic Routing Between Capsules". In: *NIPS*. 2017, pp. 3859–3869.

[107] Andrew M. Saxe et al. "On Random Weights and Unsupervised Feature Learning". In: *ICML*. Omnipress, 2011, pp. 1089–1096.

[108] E. Fred Schubert. *Light-Emitting Diodes*. 2nd ed. Cambridge University Press, 2006.

[109] Pierre Sermanet et al. "OverFeat: Integrated Recognition, Localization and Detection using Convolutional Networks". In: *CoRR* abs/1312.6229 (2013). arXiv: 1312.6229.

[110] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. New York, NY, USA: Cambridge University Press, 2014.

[111] Kanishka Sharma et al. "Automatic Segmentation of Kidneys using Deep Learning for Total Kidney Volume Quantification in Autosomal Dominant Polycystic Kidney Disease". In: *Scientific Reports* 7.1 (2017), p. 2049.

[112] Wenzhe Shi et al. "Is the deconvolution layer the same as a convolutional layer?" In: *CoRR* abs/1609.07009 (2016). arXiv: 1609.07009.

[113] Jamie Shotton and Pushmeet Kohli. "Semantic Image Segmentation". In: *Computer Vision: A Reference Guide*. Ed. by Katsushi Ikeuchi. Boston, MA: Springer US, 2014, pp. 713–716.

[114] Patrice Y. Simard, Dave Steinkraus, and John C. Platt. "Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis". In: *Proceedings of the Seventh International Conference on Document Analysis and Recognition - Volume 2*. ICDAR '03. Washington, DC, USA: IEEE Computer Society, 2003, pp. 958–.

[115] Karen Simonyan and Andrew Zisserman. "Very Deep Convolutional Networks for Large-Scale Image Recognition". In: *CoRR* abs/1409.1556 (2014). arXiv: 1409.1556.

[116] Samuel L. Smith and Quoc V. Le. "A Bayesian Perspective on Generalization and Stochastic Gradient Descent". In: *CoRR* abs/1710.06451 (2017).

[117] Daniel Soukup and Reinhold Huber-Mörk. "Convolutional Neural Networks for Steel Surface Defect Detection from Photometric Stereo Images". In: *Advances in Visual Computing*. Ed. by George Bebis et al. Cham: Springer International Publishing, 2014, pp. 668–677.

[118] Nitish Srivastava et al. "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *J. Mach. Learn. Res.* 15.1 (Jan. 2014), pp. 1929–1958.

[119] Maike Lorena Stern, Hans Lindberg, and Klaus Meyer-Wegener. "Rethinking Fully Convolutional Networks for the Analysis of Photoluminescence Wafer Images". In: *Submitted to the Journal of Computer Vision and Image Understanding* (2020).

[120] Maike Lorena Stern and Martin Schellenberger. "Fully Convolutional Networks for Chip-wise Defect Detection Employing Photoluminescence Images". In: *Journal of Intelligent Manufacturing* (2020).

[121] Chen Sun et al. "Revisiting Unreasonable Effectiveness of Data in Deep Learning Era". In: *CoRR* abs/1707.02968 (2017). arXiv: 1707.02968.

[122] Ilya Sutskever et al. "On the importance of initialization and momentum in deep learning". In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by Sanjoy Dasgupta and David McAllester. Vol. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, June 2013, pp. 1139–1147.

[123] Richard S. Sutton. "Two Problems with Backpropagation and Other Steepest-Descent Learning Procedures for Networks". In: *Proceedings of the Eighth Annual Conference of the Cognitive Science Society*. Hillsdale, NJ: Erlbaum, 1986.

[124] Christian Szegedy et al. "Going Deeper with Convolutions". In: *CoRR* abs/1409.4842 (2014). arXiv: 1409.4842.

[125] Lei Tai, Qiong Ye, and Ming Liu. "PCA-aided Fully Convolutional Networks for Semantic Segmentation of Multi-channel fMRI". In: *CoRR* abs/1610.01732 (2016). arXiv: 1610.01732.

[126] Martin Thoma. "A Survey of Semantic Segmentation". In: *CoRR* abs/1602.06541 (2016).

[127] Lisa Torrey and Jude Shavlik. "Transfer Learning". In: *Handbook of Research on Machine Learning Applications*. Hershey, PA, USA: IGI Global, 2009.

[128] Andreas Veit, Michael J. Wilber, and Serge J. Belongie. "Residual Networks are Exponential Ensembles of Relatively Shallow Networks". In: *CoRR* abs/1605.06431 (2016). arXiv: 1605.06431.

[129] Francesco Visin et al. "ReNet: A Recurrent Neural Network Based Alternative to Convolutional Networks". In: *CoRR* abs/1505.00393 (2015). arXiv: 1505.00393.

[130] Heribert Wankerl et al. "Fully Convolutional Networks for Void Segmentation in X-Ray Images of Solder Joints". In: *Submitted to the Journal of Manufacturing Processes* (2020).

[131] Shengping Wen, Zhihong Chen, and Chaoxian Li. "Vision-Based Surface Inspection System for Bearing Rollers Using Convolutional Neural Networks". In: *Applied Sciences* 8 (Dec. 2018), p. 2565.

[132] Jason Yosinski et al. "How transferable are features in deep neural networks?" In: *CoRR* abs/1411.1792 (2014). arXiv: 1411.1792.

[133] Jason Yosinski et al. "Understanding Neural Networks Through Deep Visualization". In: *CoRR* abs/1506.06579 (2015). arXiv: 1506.06579.

[134] Fisher Yu and Vladlen Koltun. "Multi-Scale Context Aggregation by Dilated Convolutions". In: *CoRR* abs/1511.07122 (2015). arXiv: 1511.07122.

[135] Sergey Zagoruyko and Nikos Komodakis. "Wide Residual Networks". In: *CoRR* abs/1605.07146 (2016). arXiv: 1605.07146.

[136] Matthew D. Zeiler and Rob Fergus. "Visualizing and Understanding Convolutional Networks". In: *CoRR* abs/1311.2901 (2013). arXiv: 1311.2901.

[137] Matthew D. Zeiler et al. "Deconvolutional networks". In: *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*. 2010, pp. 2528–2535.

[138] Chiyuan Zhang et al. "Understanding deep learning requires rethinking generalization". In: *CoRR* abs/1611.03530 (2016). arXiv: 1611.03530.

[139] Junsheng Zhang et al. "Defect Detection of Aluminum Alloy Wheels in Radiography Images Using Adaptive Threshold and Morphological Reconstruction". In: *Applied Sciences* 8.12 (2018).

[140] Zhengxin Zhang, Q. Liu, and Y. Wang. "Road Extraction by Deep Residual U-Net". In: *IEEE Geoscience and Remote Sensing Letters* 15.5 (2018), pp. 749–753.

[141] Hengshuang Zhao et al. "Pyramid Scene Parsing Network". In: *CoRR* abs/1612.01105 (2016). arXiv: 1612.01105.

[142] Shuai Zheng et al. "Conditional Random Fields as Recurrent Neural Networks". In: *CoRR* abs/1502.03240 (2015). arXiv: 1502.03240.

[143] Bolei Zhou et al. "Object Detectors Emerge in Deep Scene CNNs". In: *CoRR* abs/1412.6856 (2014). arXiv: 1412.6856.

[144] Yi Ting Zhou and Rama Chellappa. "Computation of optical flow using a neural network". In: *IEEE 1988 International Conference on Neural Networks*. July 1988, 71–78 vol.2.

[145] Xingquan Zhu and Ian Davidson. *Knowledge Discovery and Data Mining: Challenges and Realities*. Hershey, PA, USA: IGI Global, 2007.

[146] Yi Zhu and Shawn D. Newsam. "DenseNet for Dense Flow". In: *CoRR* abs/1707.06316 (2017). arXiv: 1707.06316.