



TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

JUHO RUUSUNEN
**DEEP NEURAL NETWORKS FOR EVALUATING THE QUALITY
OF TEMPERED GLASS**

Master's thesis

Examiners:
Seppo Syrjälä
Heikki Huttunen

Examiners and topic approved:
8.8.2018

TIIVISTELMÄ

JUHO RUUSUNEN: Syväoppivat neuroverkot karkaistun lasin laadun arviointiin

Tampereen teknillinen yliopisto

Diplomityö, 73 sivua

Syyskuu 2018

Konetekniikan diplomi-insinöörin tutkinto-ohjelma

Pääaine: Voimalaitos- ja polttotekniikka

Tarkastajat: Seppo Syrjälä, Heikki Huttunen

Avainsanat: neuroverkko, lasi, karkaisu, murtuminen

Tässä työssä tarkastellaan automaattisia menetelmiä murtuneen lasin sirulukumäärän laskemiseen kameralla otetusta kuvasta. Sirujen minimilukumäärä tietyllä tarkastelualueella on standardoitu vaatimus karkaistulle lasille. Laskennan suorittaa tyypillisesti karkaisukoneen operaattori manuaalisesti. Manuaalinen lasikonta on työlästä, aikaa vievää ja altis inhimillisille virheille.

Työssä tutkitaan erilaisia kuvankäsittelymenetelmiä ja koneoppimiseen perustuvia menetelmiä sirulaskennan automatisoimiseksi. Parhaaksi menetelmäksi valikoitui syväoppiva konvolutiivinen neuroverkko yhdistettynä yksinkertaiseen jälkikäsitteilyyn. Neuroverkko erittelee sirut kuvasta ja jälkikäsitteilyllä saadaan laskettua luotettavasti lukumäärä siruille. Systemin rakenne esitetään työssä seikkaperäisesti ja sen suorituskykyä arvioidaan kattavasti. Menetelmällä saavutettiin alle 5 % keskimääräinen laskentavirhe käytetyllä validointidatalla.

ABSTRACT

JUHO RUUSUNEN: Deep neural networks for evaluating the quality of tempered glass

Tampere University of Technology

Master of Science Thesis, 73 pages

September 2018

Master's Degree Programme in Mechanical Engineering

Major: Power Plants and Combustion Technology

Examiners: Seppo Syrjälä, Heikki Huttunen

Keywords: neural network, glass, tempering, fragmentation

In this work, automated methods for counting the number of shards in a picture of broken glass are reviewed. The minimum number of shards in a specific observation field is a standardized requirement for tempered glass. The operator of the tempering machine typically counts the shards manually. Manual counting is laboursome, time consuming and prone to human errors.

Several image processing and machine learning methods for automating the counting task are experimented with in this work. The best performing method proved to be a deep learning convolutional neural network combined with a simple postprocessing scheme. The neural network segments each shard in an image and with postprocessing the shard count can be obtained robustly. Architecture of the framework is presented in detail and its performance is evaluated extensively in this work. The framework reached below 5 % mean absolute counting error on the used validation data.

PREFACE

This research was done as a part of Machine Learning group in Laboratory of Signal Processing in Tampere University of Technology. The research topic was provided by Glaston Finland Oy. I would like to thank my supervisor, associate professor Heikki Huttunen, for offering me this interesting topic and for guidance throughout the project. This research allowed me to refine my skills and gather more knowledge in the field of machine learning. I am grateful to Heikki for giving me free hands to do the research at my own style and pace. It allowed me to work in a way that is most efficient for me and at no point did the amount of work seem overwhelming.

Additionally, I would like to thank people at Glaston Finland Oy for providing this research topic and the facilities required to complete the research. You were very supportive throughout the project and it has been an enjoyment working with you. I am especially grateful to Antti Aronen for sharing his expertise in the fields of glass tempering and heat transfer.

Tampere 22.08.2018

Juho Ruusunen

TABLE OF CONTENTS

1.	INTRODUCTION	1
2.	GLASS PROPERTIES AND TEMPERING PROCESS	2
2.1	Properties of glass.....	2
2.2	Tempering process.....	3
2.2.1	Heat transfer	4
2.2.2	Energy consumption	7
2.2.3	Residual stresses	10
2.3	Quality and fragmentation properties of tempered glass	11
3.	METHODS FOR IMAGE SEGMENTATION.....	16
3.1	Definition of the problem.....	17
3.2	Introduction to neural networks	18
3.2.1	Backpropagation and learning	20
3.2.2	Convolutional networks	23
3.3	Image processing approaches.....	25
3.4	Machine learning approaches.....	29
4.	DEEP NEURAL NETWORK FRAMEWORK FOR GLASS FRAGMENTATION ANALYSIS	35
4.1	Overview of the framework	36
4.2	DeepLab v3+ architecture and methods	37
4.2.1	Atrous convolution.....	38
4.2.2	Atrous spatial pyramid pooling (ASPP).....	40
4.2.3	Decoder module.....	42
4.3	Network backbones	43
4.3.1	Xception	44
4.3.2	ResNet.....	46
4.3.3	MobileNet v2.....	48
4.4	Postprocessing	50
5.	PERFORMANCE OF THE FRAMEWORK.....	52
5.1	Data.....	52
5.1.1	Annotation	52
5.1.2	Preprocessing.....	53
5.1.3	Augmenting	54
5.2	Training details.....	54
5.3	Performance	56
5.3.1	Evaluation metrics	57
5.3.2	Validation performance.....	58
5.3.3	Sensitivity and generalization.....	66
5.3.4	Summary	67
6.	CONCLUSIONS.....	69
	REFERENCES	70

NOMENCLATURE

Abbreviations

ANN	Artificial neural network
ASPP	Atrous spatial pyramid pooling
BSG	Borosilicate glass
CNN	Convolutional neural network
conv	Convolution
CRF	Conditional random field
DCNN	Deep convolutional neural network
FC	Fully connected
FCNN	Fully connected neural network
FN	False negative
FP	False positive
GPU	Graphics processing unit
low-e	Low emissivity
mIOU	Mean intersection over union
MAE	Mean absolute error
MLP	Multilayer perceptron
ILSVRC	ImageNet Large Scale Visual Recognition Competition
ReLU	Rectified linear unit
SLG	Soda-lime-silica glass
sep conv	Depthwise separable convolution
TN	True negative
TP	True positive

Latin symbols

A	Area
b	Neuron bias
c	Ground truth shard count
\hat{c}	Predicted shard count
c_p	Specific heat
C	Cost
ΔC	Gradient vector
D	Distance transform
E	Young's modulus, Error in page 20
f	Neuron activation function, General function in Section 3.2.2
F	Mapping function in residual block
F	General function in Section 4.2.1
g	Cost function, General function in Section 3.2.2
G	General function
h	Heat transfer coefficient
h	H-minima transform threshold in Section 3.3
H	Desired mapping function in residual block
I	Image
k	Thermal conductivity, Kernel size in Section 4.3.3

L	Glass thickness in Section 2
L	Number of layers in a neural network
\dot{m}	Mass flow
n	Number of neurons in a layer
N	Number of pixels in a batch
P_c	Compressor power
P_f	Fan power
Δp	Overpressure
q	Heat flux
q_1, q_3	25th and 75th percentiles of data
r	Atrous rate, Radius in Section 2.2.2
S	Source term for radiation inside glass
t	Time
T	Threshold, Temperature in Section 2
U	Elastic strain energy
\dot{V}	Volumetric flow
w	Neuron weight, Whisker length in Section 5.3.2
W	Weight vector
w_p	Pixel weight in loss function
x	Neuron input, coordinate
y	Neuron output, coordinate
\hat{y}	Predicted output
z	Input to activation function, coordinate

Greek symbols

γ	Ratio of specific heats
ϵ	Emissivity
θ	Model parameter
λ	Learning rate
ν	Poisson's ratio
ρ	Density
σ	Stefan-Boltzmann constant
σ	Stress
σ	Ratio of shard pixels to total number of pixels in Section 5.2
ϕ	Heat flow

Subscripts

a	Air
ac	Air, cooling
bot	Bottom surface
c	Compressor
c	Compressive stress
$cond$	Conduction
$conv$	Convection
f	Fan
g	Glass

<i>i</i>	General index
<i>m</i>	Mid-plane
<i>o</i>	Outer surfaces of the furnace
<i>top</i>	Top surface
<i>rad</i>	Radiation
<i>s</i>	Tensile stress
<i>w</i>	Wall
∞	Environment

1. INTRODUCTION

Glass tempering is a process whose goal is to increase the strength of glass. Strengthened glass has several useful properties, which allow it to be used in various demanding applications such as vehicle or building windows, doors and mobile screens. Tempered glass is often placed in visible areas so both strength and visual flawlessness are required. Both are a direct result of the quality of the tempering process. The tempering process of glass is a simple heat treatment technique where a sheet of glass is first heated and then rapidly cooled. However, for the product to be high quality tempered glass, sophisticated tempering machinery is needed.

To ensure that the produced tempered glass meets the standards of the industry, frequent manual testing is required. One requirement for the quality is the fragmentation density of the glass in case of breakage. European standard [1] defines the minimum number of shards in a specific observation field. For example, glass sheets of thicknesses 4 – 12 mm should have at least 40 shards in every 50 x 50 mm region. When tempered glass is continuously produced, the operator has to check at regular intervals that this requirement is met. This means breaking a sheet of glass and then manually counting the shards. It is a laboursome process and prone to human errors.

In this work, a software-based method is proposed to automate the counting task. Different image processing and machine learning approaches for shard segmentation are reviewed. The objective of this research is to determine how accurate shard counts the automated approaches can reach. The best-performing method relies on deep neural networks to segment an image of broken glass into individual shards and a simple postprocessing scheme to obtain the shard count. The neural network architecture, the postprocessing scheme and the results obtained are described in detail in this work. The framework proposed uses only raw images as input, so a single smartphone with a camera and enough processing power should be sufficient for the fragmentation analysis. The practical implementation however is not in the scope of this thesis.

The remainder of this thesis is organized as follows. First, an overview of the glass properties and the tempering process is given. Fragmentation properties of tempered glass are thoroughly reviewed. Next, widely used methods for image segmentation are presented. These include image processing and machine learning approaches. An introduction to neural networks is also given since understanding them is essential for understanding the proposed framework. Finally, the proposed deep neural network framework is presented in detail and its performance is evaluated. Conclusions and possible directions for future research are given.

2. GLASS PROPERTIES AND TEMPERING PROCESS

In this chapter, the general properties of glass and the tempering process of glass are described. The heat transfer between glass and the environment during the cooling phase has a major effect on the resulting stress distribution. The effects of heat transfer and residual stress distribution to the quality of the end result are explained. Finally, the fragmentation properties of tempered glass are reviewed. The fragmentation properties are especially important in the scope of this thesis.

2.1 Properties of glass

Glass has many properties that make it a widely used structural element in various applications. The term glass often refers to a hard, brittle and transparent solid most often seen in windows, cookware, lenses et cetera. In technical terms, glass is an amorphous solid, which means that it transforms from liquid to solid state without crystallization given a reasonably low cooling rate. This suggests that the atomic structure of solid glass is similar to that of a liquid. Glass has a long history reaching all the way to 3 000 B.C. or earlier. However, the concept of safety glass was first patented in 1909. This was due to the need for strong glass in the automotive industry. [2]

Most of the commercial glass is soda lime silica glass (SLSG). Some applications also use borosilicate glass (BSG), which has the property of very high resistance to temperature changes as well as high hydrolytic and acid resistance. The chemical compositions of these glass types are given in the Table 2.1.

Table 2.1. Chemical composition of soda lime silica and borosilicate glass. [3]

Component		Soda lime silica glass	Borosilicate glass
Silica sand	SiO ₂	69-74 %	70-87 %
Calcium oxide	CaO	5-14 %	-
Soda	Na ₂ O	10-16 %	0-8 %
Boron-oxide	B ₂ O ₃	-	7-15 %
Potassium oxide	K ₂ O	-	0-8 %
Magnesia	MgO	0-6 %	-
Alumina	Al ₂ O ₃	0-3 %	0-8 %
Others	-	0-5 %	0-8 %

Soda lime silica glass accounts for 90 % of all manufactured glass. As seen from the table above, its main components are silicon dioxide, calcium oxide and sodium oxide. Magnesium oxide can be used to lower the melting point and aluminum oxide has the property

of improving durability. Other components are used to alter the properties in a desired way, e.g., color or physical and chemical properties. [2] In this work, the term glass refers to soda lime silica glass.

Typical float glass coming from a glass manufacturer is annealed glass. A schematic of the float glass production line is shown in Figure 2.1 below.

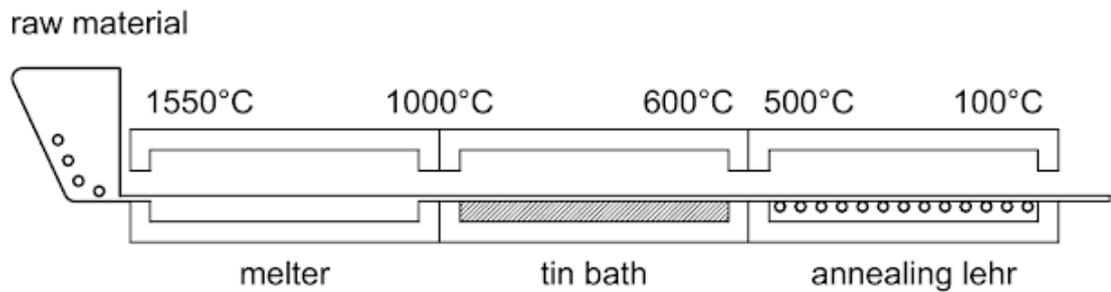


Figure 2.1. Sketch of a float glass production process and the corresponding process temperatures. [3]

As shown in Figure 2.1, glass production is a continuous process, where raw materials gradually turn into annealed float glass. The raw materials are melted in a furnace to form molten glass. Molten glass is then poured on to a shallow pool of tin at approximately 1000 °C. Tin is used because it remains in liquid state through a wide temperature range of 232 – 2270 °C. Tin is also denser than glass, causing the glass to float on the tin bed forming a smooth surface. The glass sheet is gradually cooled just below solidification point before feeding it to the annealing lehr at around 600 °C. Lehr is a temperature-controlled kiln whose purpose is to slowly cool the glass to prevent any residual stresses within the end result. [3] On exiting the annealing lehr, the sheet of glass is usually cut by machines into smaller pieces. Float glass has the useful properties of high quality and structural flexibility. Hence, it is used widely in applications such as automobile glass, mirrors, tables and shelves, windows, doors just to name a few. Many applications however require strengthened glass, which can be created in a process called tempering.

2.2 Tempering process

Annealed glass has very low residual stresses. This is useful because it allows the glass to be cut into smaller sheets or otherwise manipulated. Annealed glass can be easily broken, and it fragments into large and sharp shards. This makes it unsuitable for applications requiring high level of strength and safety features. The purpose of the tempering process is to increase the strength of glass by inducing a parabolic stress profile over the glass thickness so that there is compression on the surface and tension in the middle. In addition to the increased strength, tempered glass also has the desired property of fragmenting into very small unharmed crumbs upon breakage.

During the tempering process, a sheet of annealed glass is fed to a furnace on top of rotating ceramic rollers. The plate is then heated with thermal radiation and forced convection to above 600 °C and then rapidly cooled below transition temperature by air jets in a process called quenching. In the quenching phase, the sheet of glass is moved back and forth to prevent isolated conductive heat transfer from the rollers. Rapid cooling causes the surface of the glass to cool faster than the inside, inducing residual stresses within the glass. After quenching, the glass is slowly cooled back to near room temperature. The final cooling phase can be slow because the residual stresses are formed during the quenching phase. A sketch of the tempering process is shown in Figure 2.2.

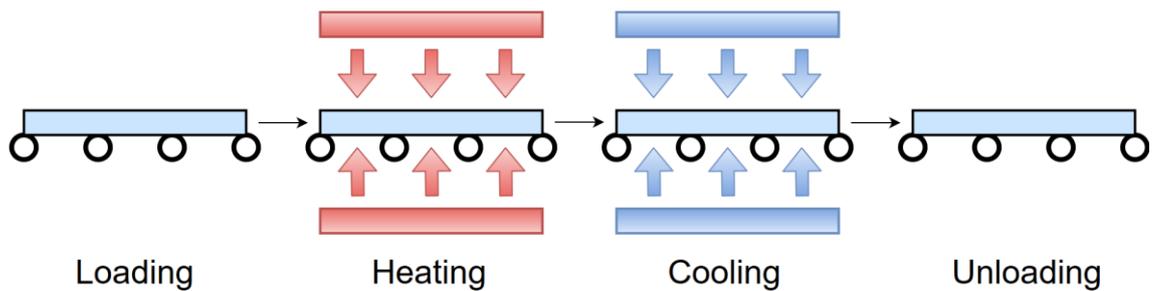


Figure 2.2. Sketch of the tempering process.

The typical range of tempered glass thickness varies from 3 mm to 19 mm. Smaller thicknesses pose a challenge for the heat transfer because the temperature gradient between the surface and the mid-plane during the cooling phase must be high enough to create the required residual stresses. If the sheet is too thin, the required temperature difference between surface and mid-plane cannot be reached. Thus, thinner glass requires higher glass temperature due to the higher cooling rate. Increasing glass temperature arises another difficulty as glass stiffness decreases with increasing temperatures. This leads to various visual defects such as roller waves and edge lifts [2]. In order to reach the required rate of heat transfer from the surface of a thin glass, forced convection is used to assist the radiation. This is done in a process called quenching, where the glass is cooled with high pressure air jets. It has been shown that the needed overpressure Δp is inversely proportional to the cube of the glass thickness L [4], i.e.,

$$\Delta p \sim \frac{1}{L^3}. \quad (2.1)$$

Equation (2.1) implies that thin glass tempering is also problematic in terms of energy consumption since very high quenching pressures are needed. Energy consumption of the tempering process is described in Chapter 2.2.2.

2.2.1 Heat transfer

Because glass tempering is a heat treatment process, the heat transfer during the process has a major impact on the tempering quality. Both the heat transfer in the heating phase

and in the cooling phase are important. In the heating phase, both surfaces of the glass should be heated with similar rates. Otherwise, the residual stresses formed during the cooling phase will be nonuniform, resulting in deformations of the glass. The surface-wise heat transfer should also be uniform, or the glass will fragment unevenly in case of breakage. This also applies to the cooling phase if the heat transfer coefficient is not constant throughout the glass.

In the furnace, heat transfer to the glass is governed by radiation from the resistors, conduction between glass and the rollers and forced convection. Radiation also occurs inside the glass, which is a complicated process for such semi-transparent material. In the 80s, low emissivity glasses appeared in the market, which gave rise to the need for forced convection inside furnaces. Low-e coated glasses can reflect as much as 90% of the radiation emitted by the resistors [2]. Forced convection is nowadays used for clear glass also to assist in heat transfer and to balance the temperature field.

In the cooling phase, heat transfer is mostly governed by forced convection. The glass is cooled by blowing high velocity air jets through an arrangement of nozzles. The velocity of jets depends on the thickness of the glass: thinner glasses require higher velocities. This is because, for thinner glass, a higher heat transfer coefficient is needed to create the required temperature difference between glass surface and mid-plane. The arrangement of air jets during the cooling phase is complicated and it has been reviewed by Rantala & Karvinen [5].

Next, the analytical formulation of the heat transfer in the glass plate during the tempering process is given. Inside the furnace, the glass is subject to radiative heat transfer between the glass and other surfaces, convective heat transfer between the glass and hot air and conductive heat transfer between the glass and the rollers. Schematic of the heat transfer phenomena in plate glass during tempering process is given in Figure 2.3.

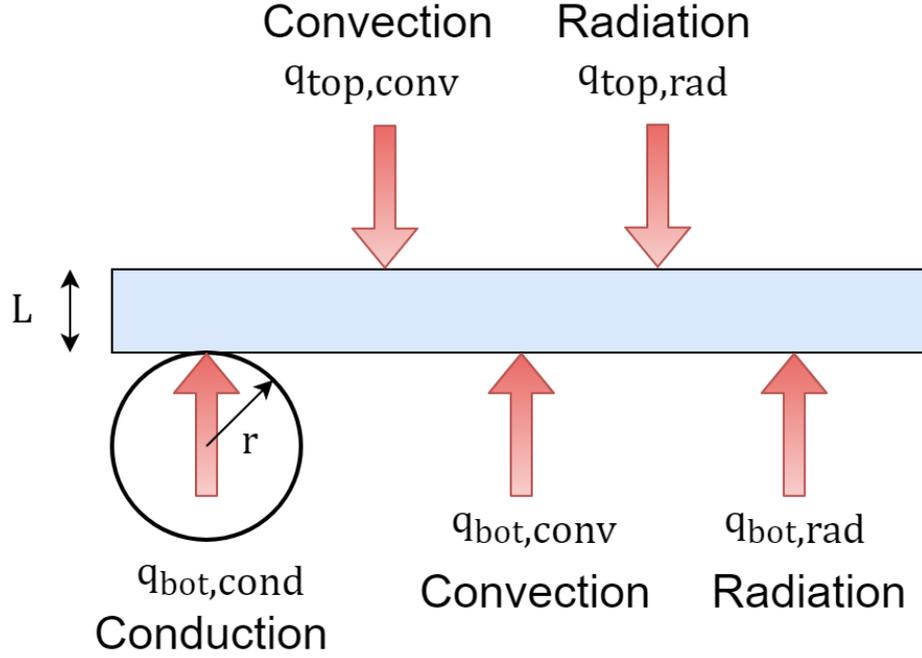


Figure 2.3. Heat transfer phenomena in plate glass during the tempering process. On the top surface, only convective and radiative heat transfer are present. On the bottom surface, contact with the rollers also induces conductive heat transfer.

Analytical formula for the thickness-wise temperature profile as a function of time can be established under some assumptions. The sheet of glass can be thought as a semi-infinite solid since the thickness of the glass is very small compared to the other dimensions. Thus, the edge behavior and the width- and length-wise heat transfer can be neglected. Considering only the temperature gradient in the thickness-wise direction, or in the z -direction, the 1-dimensional energy equation for glass is

$$\rho_g c_{pg} \frac{\partial T}{\partial t} = \frac{\partial}{\partial z} \left(k \frac{\partial T}{\partial z} \right) + S. \quad (2.2)$$

In equation (2.2), ρ_g is glass density, c_{pg} is specific heat of glass, k is thermal conductivity and S is the source term for radiation inside the glass.

For opaque materials such as steel the source term can often be omitted. Semi-transparent materials such as glass are however complicated because the radiation is not a surface but a bulk phenomenon. The upper and lower layers of the glass plate radiate heat contributing to the source term. Accurate modeling of the source term is important especially when the temperature of the glass or the surroundings is high [2]. Several formulations for the source term have been made in the literature and they have been reviewed by Rantala [2]. Most of the existing formulations are very complicated because they are developed to account for temperatures above the glass melting point at around 725 °C. At high temperatures the effect of internal net radiation between volume elements is increased, making the radiation heat transfer in molten glass significantly more complicated. The effect is further magnified by higher iron-oxide impurities within molten glass

[2]. In his thesis, Rantala proposes a simplified method that neglects the internal net radiation. Using the method can be justified in cases where the tempering temperatures are below 700 °C.

To solve Equation (2.2), the following initial and boundary conditions are needed. On the bottom surface of the glass, convective, conductive and radiative heat transfer occurs. If coordinate system origin is at the center of the glass volume, the glass thickness is L and surrounding temperature is T_∞ , the heat flux at the bottom of the glass can be expressed as

$$k \left(\frac{\partial T \left(-\frac{L}{2}, t \right)}{\partial z} \right) = h_{bot} \left(T \left(-\frac{L}{2}, t \right) - T_\infty \right). \quad (2.3)$$

Similarly, for the top of the glass, the heat flux can be expressed as

$$-k \left(\frac{\partial T \left(\frac{L}{2}, t \right)}{\partial z} \right) = h_{top} \left(T \left(\frac{L}{2}, t \right) - T_\infty \right). \quad (2.4)$$

Problem in solving Equations (2.3) and (2.4) is finding the effective heat transfer coefficients h_{bot} and h_{top} , which combine different heat transfer mechanisms occurring at the surfaces. For the bottom of the glass, heat transfer is a combination of conductive, convective and radiative heat transfer, i.e., $h_{bot} = h_{bot,cond} + h_{bot,conv} + h_{bot,rad}$. For the top of the glass, where contact with the rollers is not present, $h_{top} = h_{top,conv} + h_{top,rad}$. Finding the heat transfer coefficient is a common problem in the field of heat transfer. Some formulations for them in the case of tempering process are presented by Rantala [2].

With initial condition $T(z, 0) = T_0$ and boundary conditions (2.3) and (2.4), the energy equation (2.2) can be solved.

2.2.2 Energy consumption

When energy consumption of the tempering process is evaluated, the process can be divided into two phases: the heating phase and the cooling phase. The energy balance in the tempering process is shown in Figure 2.4.

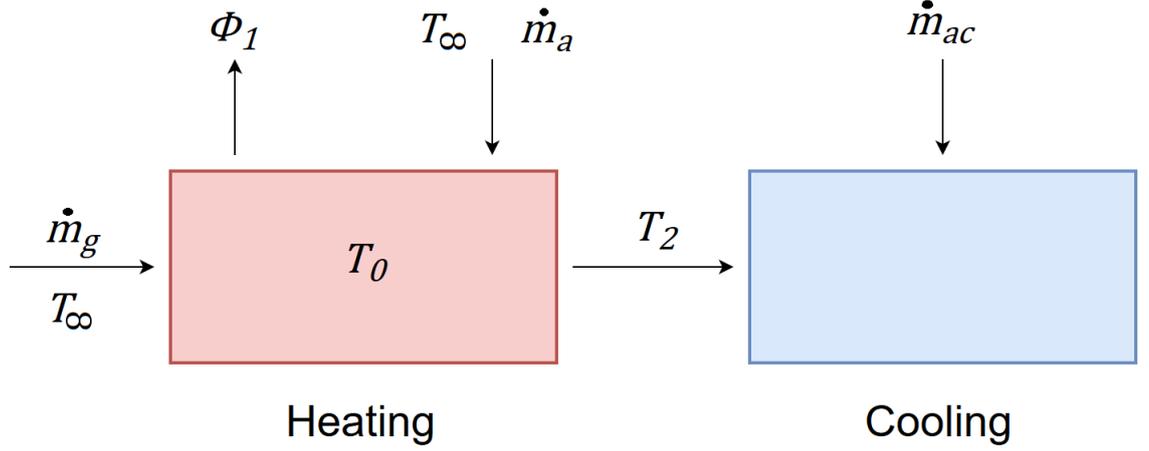


Figure 2.4. Energy balance in the tempering process.

According to the Figure 2.4, in the heating phase, if the amount of glass to be heated is known, the total rate of heat flow to the furnace can be expressed as

$$\phi = c_{pg}\dot{m}_g(T_2 - T_\infty) + c_{pa}\dot{m}_a(T_0 - T_\infty) + \phi_1, \quad (2.5)$$

where c_{pg} , c_{pa} and \dot{m}_g , \dot{m}_a are the heat capacities and mass flows of glass and air and T_∞ , T_0 are the temperatures of the room and the furnace. Temperature of the glass after heating is T_2 is and ϕ_1 is the heat loss through the furnace walls. The heat loss is a result of convective and radiative heat transfer to the inner surfaces of the furnace. The heat flux out of the furnace to the environment must be equal to the heat flux to the inner surfaces of the furnace. For the outer surfaces, the heat flux q_o can be expressed as

$$q_o = \frac{\phi_1}{A_w} = h_w(T_w - T_\infty) + \epsilon_w\sigma(T_w^4 - T_\infty^4), \quad (2.6)$$

where h_w is the heat transfer coefficient between surroundings and the outer surfaces, T_w is the furnace outer wall temperature, ϵ_w is the emissivity of the outer wall and σ is the Stefan-Boltzmann constant. Similar equation can be expressed for the inner surface of the furnace. The energy consumption of the resistors is equal to the energy transferred to the furnace in Equation (2.5), since all electrical energy can be converted to heat. However, when total energy consumption is considered, the energy required to produce compressed air must also be considered. It can be calculated as

$$P_c = \frac{c_{pa}\dot{m}_a(T_c - T_\infty)}{\eta_c}, \quad (2.7)$$

where η_c is the compressor efficiency and the temperature after compression is

$$T_c = \left(\frac{p_c}{p_\infty}\right)^{\frac{\gamma-1}{\gamma}} T_\infty, \quad (2.8)$$

where p_c is the pressure before the nozzles in the furnace. Parameter γ is the ratio of specific heats. [6] Since pressurized air is flowed to the furnace, an equal amount of air must also exit the system. This adds another source of energy loss to the system as the air exiting the system is usually still hot. This can be partly remedied by using a heat exchanger, which transfers heat from the hot air exiting the furnace to the cold air entering the compressor.

During the cooling phase, energy consumption is determined by the generation of high velocity air jets. It can be calculated according to the Equation (2.7) if pressurized air is used. The amount of energy required is heavily dependent on the thickness of the glass. Forced convection can be attained either by means of fans or by pressurized air. For thin glasses (generally below 4 mm), the overpressure must be high to reach a sufficiently high heat transfer coefficient and pressurized air is used. Using pressurized air is beneficial also because it is easy to produce and store elsewhere. For thicker glasses, fans are used because the air is easier to circulate and there are less thermal losses. The energy consumption of a fan depends on overpressure and can be defined as

$$P_f = \frac{\Delta p \dot{V}_a}{\eta_f}, \quad (2.9)$$

where \dot{V}_a is the volumetric flow rate of air and η_f is the fan efficiency. Equation (2.9) does not account for pressure losses in the air channels. In Figure 2.5, the fan power requirement in the cooling phase is plotted against glass thickness.

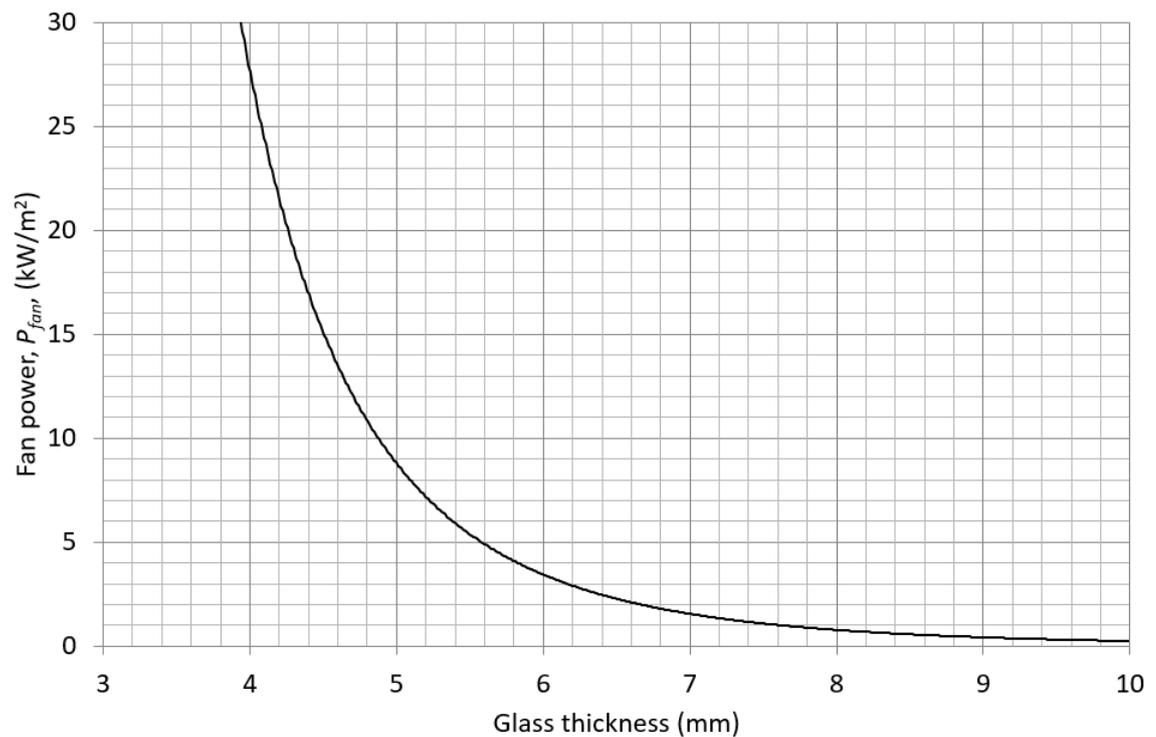


Figure 2.5. Required fan power as a function of glass thickness. [2]

Figure 2.5 shows that decreasing glass thickness significantly increases the required fan power: for example, decreasing glass thickness from 4 to 8 mm makes P_f 42-times higher. Figure 2.5 is based on Equation (2.9) and data from real tempering process [2]. Glass thickness also has an effect on the required cooling time. In Table 2.2, typical quenching and cooling times are shown for different glass thicknesses.

Table 2.2. *Quenching and cooling times for different glass thicknesses.*

Thickness (mm)	3	4	5	6	8	10	12
Quenching time (s)	5	6	9	12	24	36	60
Cooling time (s)	33	50	70	110	150	215	275

Quenching time means the rapid, high pressure cooling in the beginning of the cooling phase from above 600 °C to around 400 °C. After that, glass is cooled to near room temperature. Only the quenching phase influences the forming residual stresses.

In his thesis, Aronen [7] shows, for an example furnace, that for 3 mm glass, 1/3 of the total energy consumption is used to heat the glass. For 5 mm glass, the ratio is 1/2. In the end result, the energy is stored in the form of residual stresses. It is noteworthy that the energy content of the residual stresses is very small compared to the total energy used in the tempering process.

2.2.3 Residual stresses

In the cooling phase, the surface of the glass cools faster than the mid-layer, inducing a compressive stress on the glass surface and a tensile stress in the mid-layer. For comparison, the typical surface compression in annealed glass is 2 MPa, in heat-strengthened glass 40 – 60 MPa and in tempered glass 90 – 110 MPa. Heat-strengthened glass is similar to the tempered glass, but the cooling phase is slower. [2]

Stress profiles are often approximated with a second-order parabola by assuming symmetry around mid-plane and equilibrium. Thickness-wise stress profiles for different glass types with the given assumptions are presented in Figure 2.6.

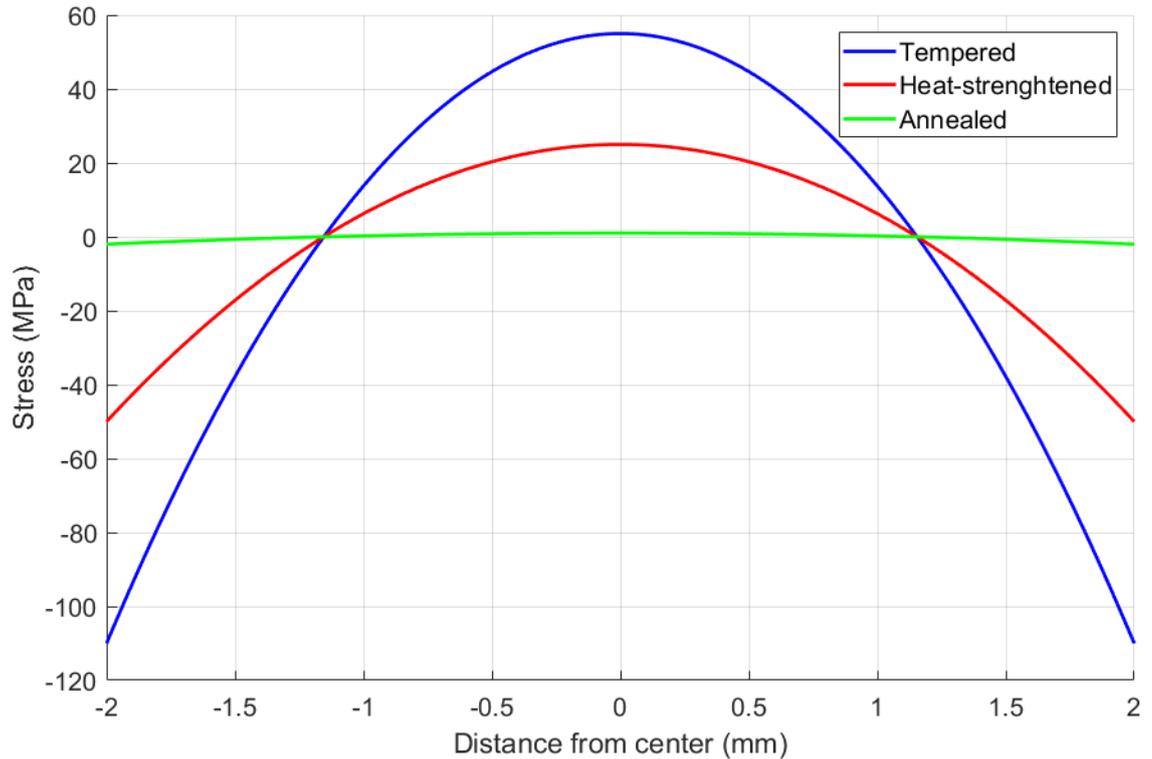


Figure 2.6. Thickness-wise stress profiles for different glass types.

Mid-plane symmetry and equilibrium imply that the compressive surface stress is twice the interior tensile stress in magnitude, i.e., $|\sigma_s| = 2|\sigma_c|$.

The reason why compressive stress on the surface is desired in safety glass is the fact that it increases the magnitude of external load needed to break the glass. Glass is broken when a tensile stress on the surface exceeds a certain threshold. This is due to the surface flaws in the glass, which grow fast under tensile stress. When compressive stress is present in the surface, the external load must first exceed it to create tensile stress, which breaks the glass. In case of breakage, the internal tensile stress causes the glass to fragment into multiple small shards as the strain energy obtained from the tempering process is rapidly released throughout the glass.

2.3 Quality and fragmentation properties of tempered glass

This thesis focuses on evaluating the quality of tempered glass, specifically its fragmentation properties. Tempered glass however also has other qualities that should be evaluated. In this chapter, these qualities are presented, and the fragmentation properties in particular are more thoroughly investigated.

In tempered glass, good visual properties are often wanted. There are a variety of possible visual defects and in general they are all caused by an improper heating process. For example, roller waves, longitudinal or cross patterns, distorted reflections or transparency are usually caused by too high process temperatures. Roller waves are formed when the

softened glass is sagged between the rollers during heating. Several standards define the requirements for strength and visual properties and customers might have their own special needs.

In addition to the strength and visuals, fragmentation properties are an important metric of quality in tempered glass. The broken glass is wanted to fragment densely and completely into small harmless crumbs. It not only improves the safety of the glass post breakage, but also increases its structural capacity. The difference in fragmentation of annealed and tempered glass is illustrated in Figure 2.7.

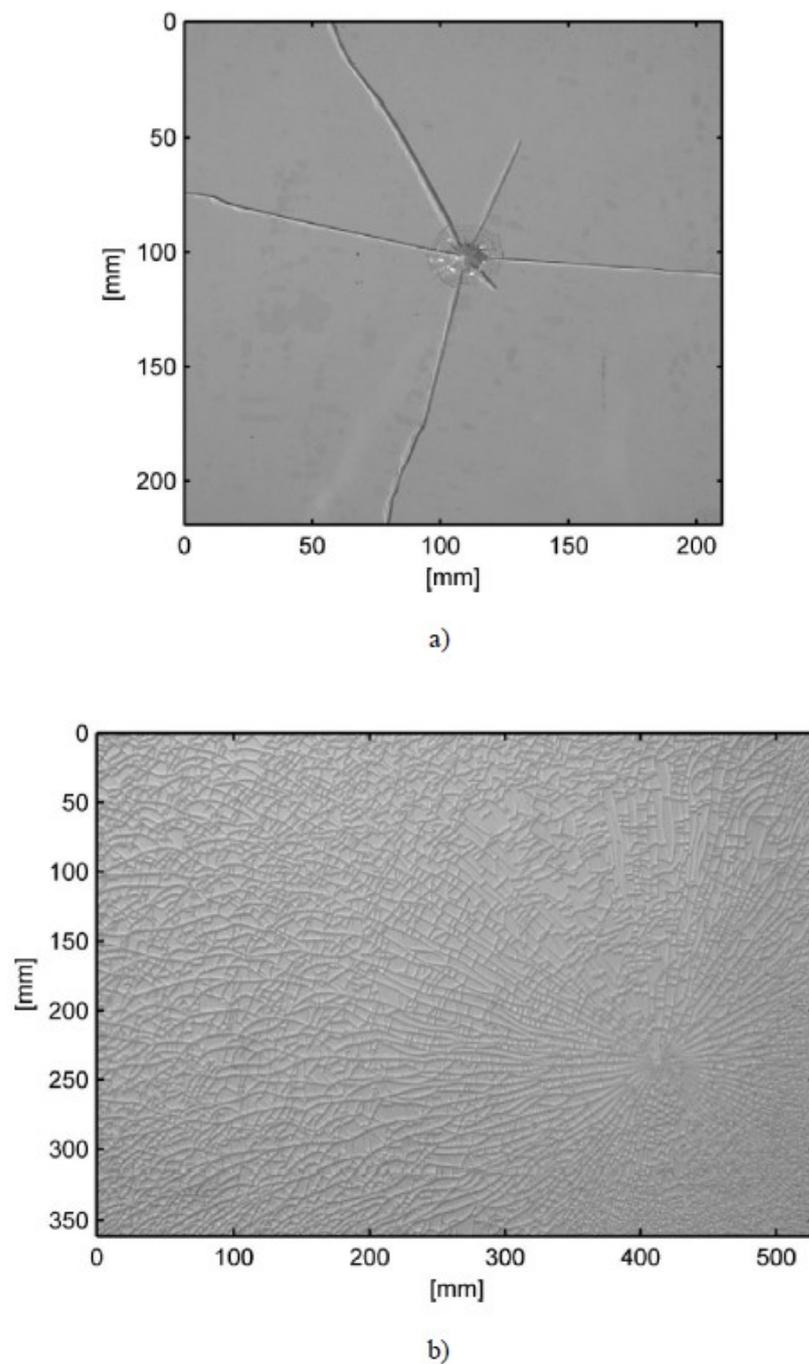


Figure 2.7. Fracture pattern in (a) annealed glass and (b) tempered glass. [8]

A standard defines the minimum number of shards in a specific observation field [1]. For example, glass sheets of thicknesses 4 – 12 mm should have at least 40 shards in every 50 x 50 mm region. However, the whole sheet of glass need not be considered in this analysis because the shard density is affected by the proximity of edges and the impact point. The area of interest for fragmentation analysis is defined by the standard and it is shown in Figure 2.8.

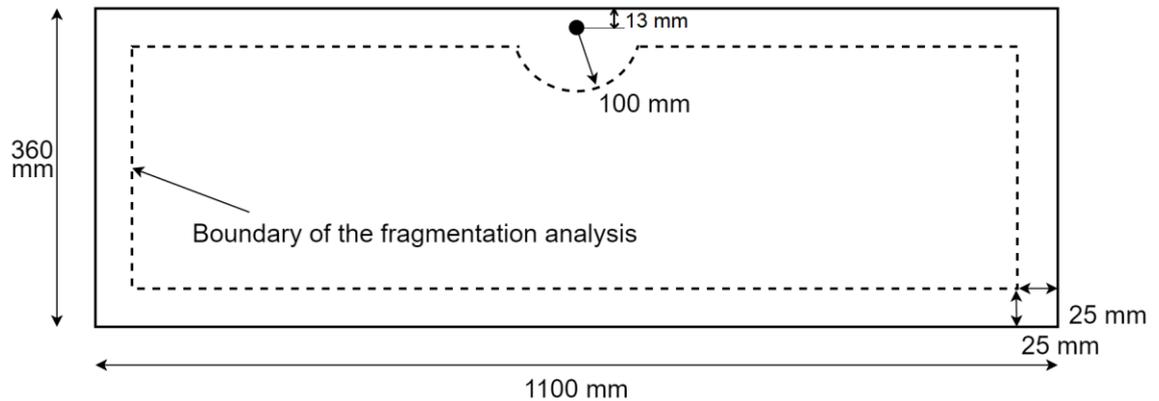


Figure 2.8. Area of interest in the fragmentation analysis. All areas inside the dashed line should be included in the analysis.

All regions inside the dashed line in Figure 2.8 should be considered in the fragmentation analysis.

The fracture process of tempered glass is not completely understood due to the rarity of researches into the matter. The fracture front propagates very fast and the process time is very short, making it a difficult matter to investigate. Nielsen [9] has made a proposal for the fragmentation mechanism. Nielsen hypothesizes that the fragmentation process is primarily driven by the interior tensile stresses. When the fracture front propagates in the tensile zone, the tensile stresses are released and the material contracts locally. The compressive surface stresses are then released, and temporary tensile stress might occur at the surface. This initiates a secondary fracture front propagating from the surface towards the center explaining the appearance of three fracture fronts as shown in Figure 2.9 below.

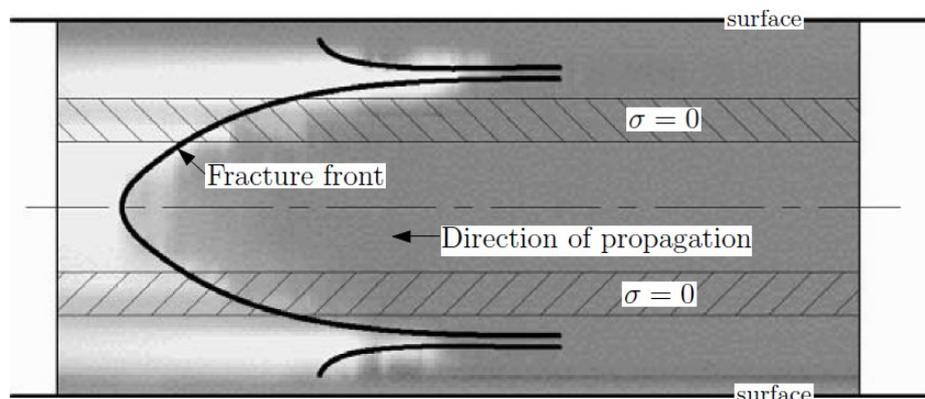


Figure 2.9. Fracture front inside the glass, based on high-speed observations. [9]

Pourmoghaddam & Schneider [10] have made experiments to find the relation between residual stresses and fragmentation behavior. However, their observations show that the area below the impact point over the whole width to the opposite edge of the specimen is influenced by the proximity of the impact point in terms of fragmentation density. Therefore, in their experiments, they only consider fragments in the areas under an angle of 45° left and right from the impact point.

In the experiments, Pourmoghaddam & Schneider analyze the relation between the fragmentation properties and the residual stresses over various glass thicknesses. Correlations between the residual stress and particle counts, particle weights and particle sizes for glass plates with thicknesses 4, 8 and 12 mm are established. Plates with different thicknesses are tempered such that the resulting residual stress is in a predetermined range. After tempering, the residual stress is measured using a scattered light polariscope. Glass plates are then shattered according to the standard SFS-EN 12150-1 [1]. Fracture patterns are scanned with a commercial device CulletScanner [11] and the individual particles are counted and weighed. The results and conclusions of their experiments are briefly presented in this chapter.

Elastic strain energy is the energy governing the fragmentation behavior within the glass and by Hooke's law it can be defined as

$$U = \frac{4(1-\nu)}{5} \frac{L\sigma_m^2}{E}, \quad (2.10)$$

where E is the Young's modulus, ν is Poisson's ratio, L is glass thickness and σ_m is the mid-plane tensile stress. While the fragmentation density increases with the elastic strain energy, Equation (2.10) shows that the fragmentation density is also proportional to the glass thickness, i.e., thicker glass will fragment more densely. [10]

In Table 2.3, the shard counts given by the experiments of Pourmoghaddam & Schneider are presented. Shards are counted for every specimen over varying thicknesses L , mid-plane tensile stresses σ_m and elastic strain energies U .

Table 2.3. *Shard counts for different specimens in relation to the elastic strain energy and mid-plane tensile stress. [10]*

N ₅₀ [-]	L = 4 mm		L = 8 mm		L = 12 mm	
	U [J/m ²]	σ _m [MPa]	U [J/m ²]	σ _m [MPa]	U [J/m ²]	σ _m [MPa]
20	61.2	41.7	72.1	32.0	86.4	28.6
40	72.6	45.4	95.3	36.8	125.0	34.4
80	95.9	52.2	140.7	44.7	201.7	43.7
120	119.2	58.2	186.0	51.4	279.0	51.4

The quantity N_{50} is the average shard count over 8 distinct 50 x 50 mm observation fields. Even figures are obtained by interpolating the results. Multiple observation fields are used to increase the shard count per specimen for quantitative results. The results in Table 2.3 show that higher the residual stress, higher the shard count. It can also be seen that thicker glass requires lower residual stresses to achieve the same shard count as the thinner glass. This behavior was previously suggested by Equation (2.10).

3. METHODS FOR IMAGE SEGMENTATION

Image segmentation is a process of partitioning an image into multiple meaningful segments. It is one of the most important and most common problems in image processing [12]. The goal of the segmentation is usually to simplify the representation of an image such that it is much easier and more efficient to analyze. As an example, an image containing a dog and a cat could be segmented such that the image has only three different pixel values, one for the cat, one for the dog and one for the background. In other words, every pixel in an image is assigned a label such that pixels with the same label share certain characteristics. Image segmentation is a natural approach for fragmentation analysis, where each shard should be separated as its own instance. An ideal segmentation mask of a broken glass would contain every shard as its own separate region. The segmentation mask could easily be further analyzed to obtain shard density, shard count and other interesting quantities.

As mentioned before, fragmentation analysis is one of the components of quality control in tempered glass. When tempered glass is continuously produced, a sheet of glass must be broken at regular time intervals to ensure it meets the fragmentation requirements [1]. It is a laboursome process and subject to human error. Automating the process would make the quality control not only more objective but also more robust and more reliable. The task of the operator is to count the number of shards in the 50 x 50 mm region where the shard density is the lowest as described by the standard. Operator is responsible for both determining the area of lowest shard density and the actual counting. Both tasks are subjective and prone to errors due to human fatigue and lack of concentration.

In addition to easing the counting task, the fragmentation pattern of the broken sheet of glass would be a valuable data source towards the goal of better understanding the fracture process. As explained in Chapter 2.3, fragmentation of tempered glass is a complicated process and not fully understood. Automated fragmentation analysis from an image would significantly ease the analysis and increase the amount of data available for further research. With the increased amount of data, a direct relationship between the tempering process and the fragmentation behavior could possibly be established.

A commercial device CulletScanner by Softsolution [11] designed for this purpose already exists. A broken sheet of glass is placed on top of the device and a camera scans through the sheet. The scan image is passed through an algorithm that automatically performs the analysis. The result of the analysis is a complete segmentation mask of the broken glass sheet. The mask can then be further analyzed with the provided software. The scanner is said to fulfill all relevant standards and norms. There are however problems with a device like this. It is expensive and requires a lot of space. This thesis proposes a new software-based method for fragmentation analysis, requiring only a camera

to take the image and a computer to perform the computation. The method utilizes deep learning neural networks, which are robust against variations in the raw image input. Given that the model is not sensitive to varying illumination and noisy images, a single mobile smartphone might be sufficient for the task. There are no insurmountable problems in mobile implementation and it is only a matter of computational resources.

In this chapter, the problem of automated fragmentation analysis from an image is defined. An introduction to the concept of neural networks is presented, since knowledge of them is vital for understanding the framework used in this work. Several widely used methods for similar segmentation tasks, including image processing and machine learning techniques, are reviewed. Their applicability to the problem of fragmentation analysis is also examined. It should be noted that a usual segmentation pipeline consists of a combination of different methods rather than one single method. The order of the methods applied is usually task specific and domain knowledge might be needed.

The neural network framework used in this work is detailed in Chapter 4 and the performance of the framework is demonstrated in Chapter 5. The methods presented in this chapter are those that were experimented with during this research.

3.1 Definition of the problem

The problem in image segmentation is to somehow define a set of properties that distinguish one pixel from another. Usually the segmentation task is easy for a human but very difficult to explicitly define for a computer. Let us consider for example an image of a dog and a cat. Image is only an array of intensity values ranging from 0 to 255. The human brain is able to use a very high level of contextual information to easily segment the image and classify individual pixels. For a computer to succeed in the task, this contextual information must be somehow programmatically defined. Making explicit rules for such classification is extremely hard.

In Figure 3.1, some images of broken glass and their respective ground truth segmentation masks are shown. The annotation process is further detailed in Chapter 5.1.1.

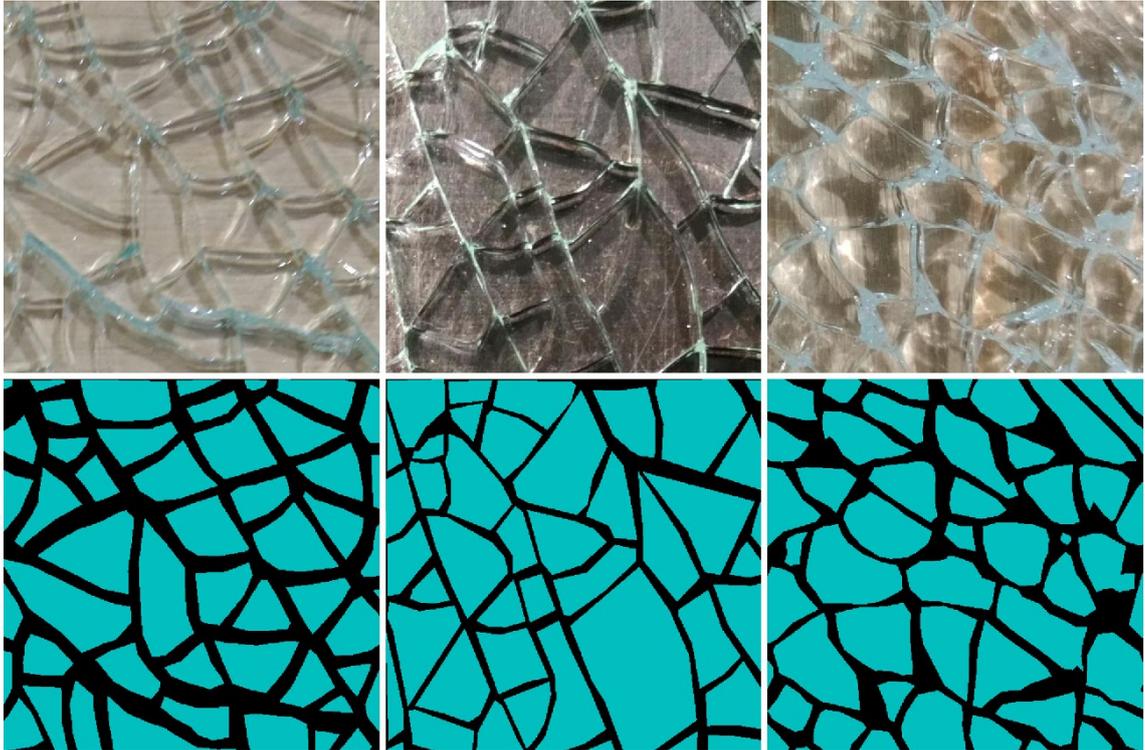


Figure 3.1. Raw images of shards and the corresponding ground truth segmentation masks.

As it can be seen from the figure, there are many difficulties in the segmentation of shards: variance in the background, variance in the fracture pattern, vanishing edges and noise. The segmentation algorithm should be robust to such variations in the input image. The segmentation mask should require as little postprocessing as possible to obtain interesting quantities such as the shard count. The mask is not very useful if a lot of noise or misclassified pixels are present.

Traditional image processing approaches completely lack the contextual information because they have no prior knowledge of what constitutes the objects to be segmented. They only operate on the color information of pixels and usually are able to find features such as edges and regions. Machine learning approaches however, especially deep neural networks, are thought to be able to learn more abstract representations of the objects of interest [13]. Furthermore, traditional approaches rely on local image properties while deep learning methods can extract global image features.

3.2 Introduction to neural networks

The DeepLab architecture used in this work utilizes a deep convolutional neural network (DCNN). To understand how it works, a basic level of knowledge of how neural networks work is required. This chapter briefly presents the basic concepts and gives some insight on how they learn. A review on convolutional networks is also presented. Convolutional

neural network is a specific class of neural networks used especially in analyzing visual imagery.

Artificial neural network (ANN) is a collection of artificial neurons, which loosely model the real neurons in animal brains. A single neuron in the network receives inputs from multiple other neurons, performs a simple arithmetic operation on them and then passes the result through an activation function to other neurons. A collection of these simple units forms a complex network capable of “understanding” abstract information. A schematic of an artificial neuron is shown in Figure 3.2.

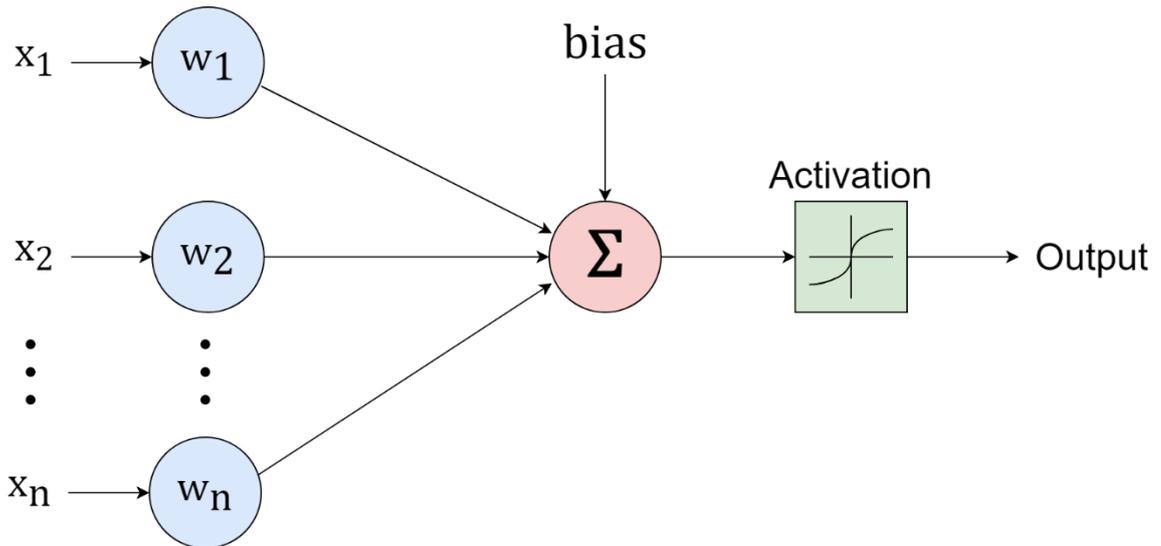


Figure 3.2. Schematic of an artificial neuron.

Mathematically the neuron can be expressed as

$$y = f(\mathbf{w} \cdot \mathbf{x} + b). \quad (3.1)$$

In Equation (3.1), y is the output for the neuron, \mathbf{w} is the weight vector to be learned, \mathbf{x} is the vector of inputs from other neurons and b is the bias also to be learned. The activation function f is usually a non-linear function such as sigmoid, but it can also be a simple threshold function.

There are various network topologies, probably the simplest being a *feedforward network*, which consists of an input layer, a number of hidden layers and an output layer. In a feedforward network each neuron in a layer outputs their activation only to the neurons in the next layer towards the output layer. These kinds of networks are often *fully connected* (FC), which means that every neuron in a layer is connected to every neuron in the next layer. A schematic of a fully connected feedforward network is shown in Figure 3.3. Various more complex topologies exist, such as skip layers and recurrent networks, but they are not addressed in this introductory chapter. [14]

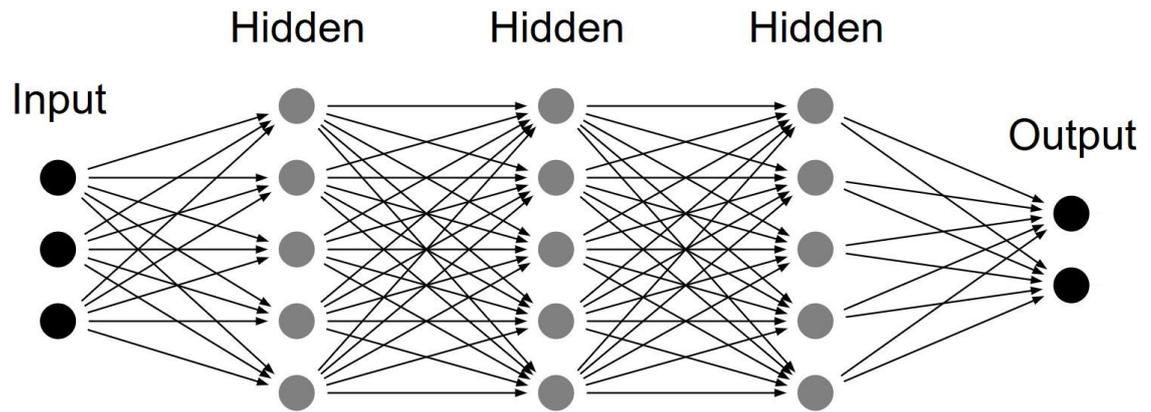


Figure 3.3. Schematic of a fully connected neural network with three hidden layers.

The network in Figure 3.3 has three hidden layers but there is no limitation to their number. Each grey circle in Figure 3.3 corresponds to a single neuron shown in Figure 3.2. The number of neurons in each hidden layer can be arbitrary. The number of neurons in the input and output layers are determined by the input size and the desired output size. For example, in the case of recognizing hand-written digits, the input layer could have 1024 neurons corresponding to an input image size of 32 x 32 pixels and the output layer could have 10 neurons, each corresponding to one digit. During learning, the weights and biases of each neurons are tuned such that a specific input results in a desired output. The learning procedure where the network is taught against a known output is called *supervised learning*. The weights and biases are tuned in a process called *backpropagation*.

3.2.1 Backpropagation and learning

Backpropagation

When a neural network gives an output for a given input x , there is an error $E = y - \hat{y}$ between the ground truth y and the network output \hat{y} . Each parameter of the network is responsible for the error E in some proportion. Backpropagation is an algorithm used to calculate the parameter changes in order to minimize a *cost* or *loss* function, which is usually some function of the error E . In this chapter, the concept of backpropagation is explained by a very simple example. The mathematical formalism is then expanded to arbitrary sized networks.

Suppose a very simple network with L layers each containing only one neuron. The output layer of the network has an output \hat{y}_L and the desired output is y . The output of a neuron is often called an *activation*. Subscript L indicates the number of layer in the network, L being the last layer. The cost to be minimized is then $C = g(\hat{y}_L, y)$, where g is the cost function. The cost function could for example be a squared difference between the predicted output and the desired output. The activation of the output layer is a result of the activation of the previous layer and weights and biases of the output neuron, i.e.,

$$\hat{y}_L = f(w_L \hat{y}_{L-1} + b_L) = f(z_L), \quad (3.2)$$

where f is the activation function for the neuron. Each term in Equation (3.2) is contributing to the total cost C . To simplify the following equations, it is convenient to denote the input to the activation function by z . Now it is possible to calculate the contribution of each parameter to the cost. For example, by applying the chain rule, the change in the cost C with respect to the change in weight w_L is

$$\frac{\partial C}{\partial w_L} = \frac{\partial z_L}{\partial w_L} \frac{\partial \hat{y}_L}{\partial z_L} \frac{\partial C}{\partial \hat{y}_L}. \quad (3.3)$$

There is only one weight since every layer has only one neuron in this simple example. Similarly, for the bias b_L and the activation \hat{y}_{L-1} , the gradient can be calculated by changing the term $\frac{\partial z_L}{\partial w_L}$ in Equation (3.3) to $\frac{\partial z_L}{\partial b_L}$ or to $\frac{\partial z_L}{\partial \hat{y}_{L-1}}$, respectively. The activation of the previous layer is similarly the result of the activation before that and the weights and biases of the neuron in that layer. The chain rule can be applied to find the effect of any parameter in the network to the final cost. Hence the name backpropagation: the gradients are propagated backwards through the network.

The previous example was for the simple case where each layer of the network had only a single neuron. For networks that are more complicated the idea is essentially the same. Consider a fully connected network with an arbitrary number of neurons and hidden layers. For multiple output neurons, the total cost is simply the sum of individual costs over all neurons j , i.e., $C = \sum_{j=1}^{n_L} g(\hat{y}_L^j, y^j)$, where n_L is the number of neurons in the output layer. The difference is in the expression for the gradient of the cost C with respect to the activation \hat{y}_{L-1}^k , where subscript k indicates the number of the neuron in the layer $L - 1$. In a fully connected network, the activation of each output neuron depends on the activation \hat{y}_{L-1}^k . Thus, the effect must be summed over all of the output neurons $1 \dots n_L$, i.e.,

$$\frac{\partial C}{\partial \hat{y}_{L-1}^k} = \sum_{j=1}^{n_L} \frac{\partial z_L^j}{\partial \hat{y}_{L-1}^k} \frac{\partial \hat{y}_L^j}{\partial z_L^j} \frac{\partial C}{\partial \hat{y}_L^j}. \quad (3.4)$$

Again, the chain rule can be applied to find the gradients for any parameter in the network.

Weight updating

By calculating the gradient of C with respect to every parameter in the network, a gradient vector ∇C can be formed. Each element in the vector indicates how a specific weight should be tuned to minimize the total cost. When the gradient vector is computed, the weight vector W can be updated as

$$W \leftarrow W - \lambda \nabla C, \quad (3.5)$$

where λ is a parameter called *learning rate*. The learning rate is typically a small number, e.g., 0.001, and it is used to control how quickly the network adapts to new information. Usually a small learning rate is used to reduce the effect of noise in the training data and to make the gradient descent smoother. One interpretation of the learning rate is that while the gradient vector is the direction of the step, the learning rate is the length of the step. The minima can be approached reliably by taking small steps to the correct direction.

The weights can be updated in several ways. The most accurate method is to iterate through all of the training examples, calculate average loss, calculate gradients by means of backpropagation and then update the weights to the direction indicated by the negative gradients. However, the learning process is very slow if the whole dataset must be iterated between every weight update. One iteration over the entire training set is called an *epoch*. To speed up the learning, a mini-batch approach is often employed. It means that a subset of the whole dataset is used for updating the weights. In this way, multiple weight updates can be done in a single epoch. The path of the gradient descent is not as accurate, but each mini-batch gives a reasonable estimation for the correct direction of the gradient. The size of the mini-batch can be controlled to compromise between training time and smoothness of the learning process. Very small batch sizes tend to make the path of the gradient descent noisy and the model might get stuck at local minima. Gradient descent, where each batch consists of a number of random samples is called *stochastic gradient descent*. The effect of batch size to the path of the gradient descent is illustrated in Figure 3.4 below.

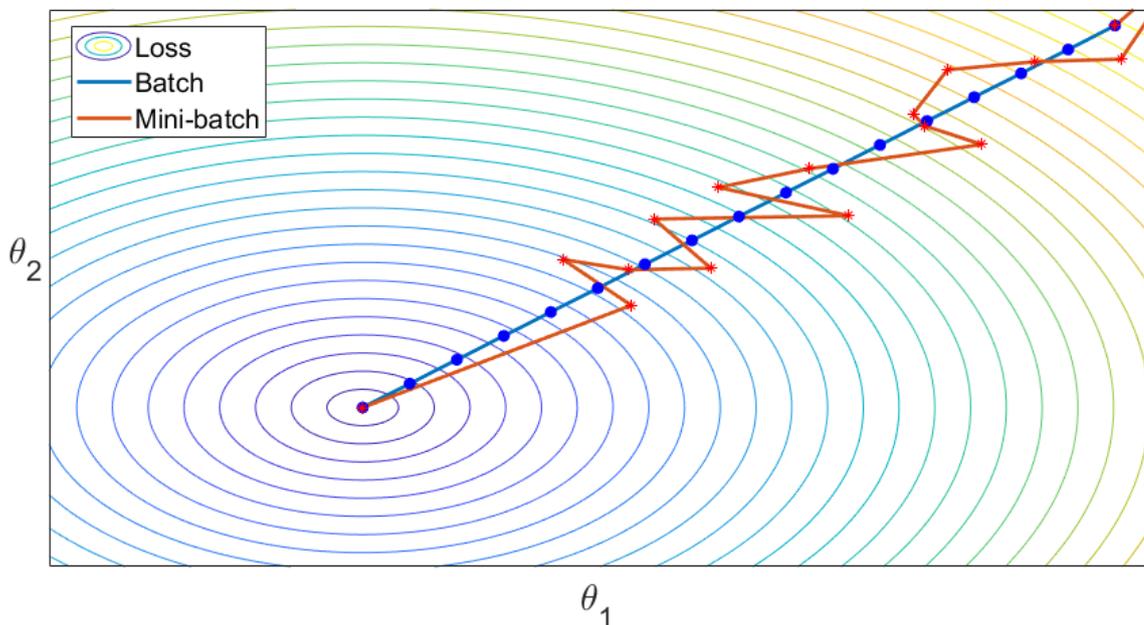


Figure 3.4. Effect of batch size on the path of the gradient descent. *Batch*: all samples are used for each weight update. *Mini-batch*: a number of random samples are used for each weight update. θ_1 and θ_2 are model parameters.

The mini-batch approach results in a noisier learning process. Using the whole training set for each weight update produces a very smooth learning curve but requires very long

training times. Figure 3.4 does not correspond to any real network training but it simply illustrates how the loss might behave during training.

3.2.2 Convolutional networks

Convolution is a function that blends one function with another. Mathematically, it is an integral that expresses the amount of overlap of function g as it is shifted over another function f [15]. Convolution of functions f and g in 1-dimensional case can be expressed as

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t - \tau)d\tau. \quad (3.6)$$

Equation (3.6) can be also interpreted as a weighted average of function f while g is the weighting function. Convolution is often applied to image processing in the form of *kernels* or *filters*.

In image processing, convolution is a process of adding each element of the image to its neighbors, weighted by a convolution filter. The filter can be shifted over every image position to generate an image convolved by the filter. Various image processing operations are achieved with different convolutional filters, such as edge detection, sharpen and blur. A filter with some weights can be thought as a feature extractor that highlights certain features and patterns in the input image.

The pattern and feature extracting property of convolution makes it a powerful tool in neural networks in the case of image analysis. By utilizing backpropagation, the network can learn the filter weights (i.e., features and patterns) relevant to a specific task. A typical convolutional neural network consists of a cascade of convolutional layers, each one with differing filters. The network can also be supplemented with pooling layers and fully connected layers. Main components of a convolutional network are described in this chapter.

Convolutional layer

A convolutional layer is defined by a filter with fixed height and width, stride, padding and the number of filters. The filter depth is usually determined by the depth of the input volume. For example, in the case of color image input, the filter has dimensions $w \times h \times 3$, where the last dimension corresponds to the three RGB color channels of the image. Width and height of the filter are typically small numbers such as 3 or 5 to limit the number of parameters and computational requirements. Each element of the filter is a learnable weight that is tuned during network training by backpropagation.

The convolution operation is performed by sliding the filter around the image with a step size defined by the stride value. With $stride = 1$, the filter moves one pixel at a time thus operating at every image position. With $stride = 2$, the filter moves 2 pixels at a time et

cetera. Stride can be increased to reduce the spatial dimensions of the output channels or to reduce overlap among receptive fields. At each image location, output value is a dot product between the filter and the image patch.

Padding determines how the edges of the input are handled. For example, by applying a 5×5 convolution filter on 32×32 image, the output dimensions would be 28×28 . One way to resolve this is to pad the input borders with equal number of zeros, so that the output spatial dimensions are same as the input spatial dimensions. This scheme is often called *same*-padding. The former mentioned case, where the filter only operated on valid image locations, is often called *valid*-padding.

Last parameter of a convolutional layer is the number of filters. Each filter produces a single feature channel in the output, i.e., the number of output channels equals the number of filters. On top of convolution there is usually an activation layer and a batch normalization layer. A widely used activation layer is ReLU (rectified linear unit), which applies a function $\text{relu}(x) = \max(0, x)$ to every value in the output. Using this kind of activation has been shown to introduce nonlinearity and computational efficiency to the system [16]. Batch normalization layer normalizes the output of the layer. Typically, zero mean and unit variance normalization is used. This normalizes the inputs to the subsequent layers, generally speeding up the learning.

Pooling layer

Pooling layers are often added between sequential convolutional layers. Their purpose is to simplify the representation by reducing spatial dimensions. This also reduces model parameters, adds invariance to small spatial distortions of the input and reduces overfitting. [17] Pooling layer is sometimes also called a downsampling layer since it combines several input values into one output value.

The pooling layer, similarly to the convolutional layer, is defined by filter size, strides and padding. Often used pooling scheme is max pooling, where each output value is the maximum value in the corresponding filter position. Max pooling for 4×4 input array with a 2×2 filter and a stride of 2 is illustrated in figure below.

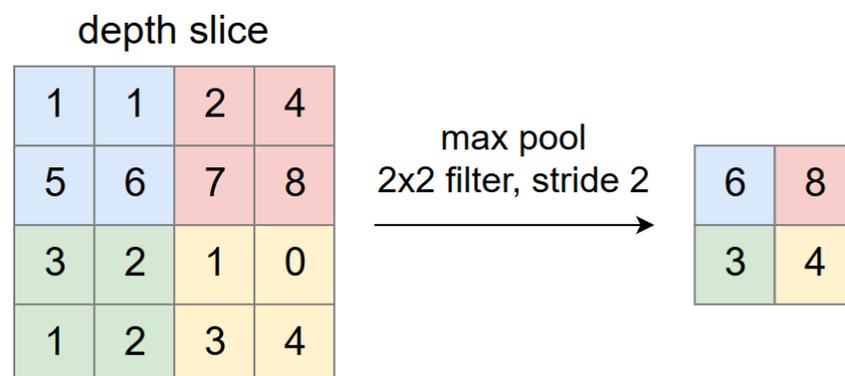


Figure 3.5. Max pooling performed on 4×4 input with 2×2 filter and a stride of 2.

In the case of Figure 3.5 the max pooling operation effectively halves the spatial dimensions, thus reducing the number of parameters in the subsequent layers. The pooling operation always operates channel-wise so depth dimensions are not reduced.

Fully connected layer

Fully connected layers are often added to the end of the convolutional network to capture global information. This is useful for example in image classification tasks, where the local spatial information is not important and the classification can be done based on global feature space. However, adding fully connected layers makes the network not fully convolutional, requiring a fixed size input. One way to convert fully connected layers into convolutional layers is by using a filter whose spatial dimensions are the same as the layer input dimensions [18].

3.3 Image processing approaches

In this work, the image processing approaches refer to unsupervised methods that operate on pixel intensity values and use a predetermined set of rules to classify pixels. In this chapter, several popular methods are presented. Usually multiple image processing techniques are used in series to reach a desired outcome. Image segmentation is a psycho-visual task and therefore not susceptible to any analytical solution [19]. Therefore, image processing approaches should be augmented with domain specific knowledge and heuristics. This includes the task-specific serialization of different methods. Problem with image processing approaches in general is the difficulty of finding an algorithm that generalizes well and is robust to variations in input images. If frequent hand-tuning of the parameters is required, the purpose of automation is defeated.

Image processing methods for segmentation can be roughly divided to the following groups: discontinuity detection and similarity detection [20]. An example of the former is edge detection, where image is segmented into regions based on intensity changes along edges. Under the similarity detection falls many widely used techniques such as thresholding, region growing and clustering. Some of these methods are briefly reviewed in this chapter.

Thresholding

Thresholding is one of the oldest, simplest and most popular approaches to image segmentation [19]. In the simplest form, thresholding can be defined as

$$I_T(x, y) = \begin{cases} 1, & I(x, y) > T \\ 0, & I(x, y) \leq T \end{cases} \quad (3.7)$$

where I is the original image, $I_T(x, y)$ is the resulting thresholded binary image at coordinates (x, y) , and T is the chosen threshold value. There are several methods for choosing the threshold value and it depends on the task in question. One popular way is to use

Otsu's method [21], which chooses the threshold to minimize the intraclass variance of the black and white pixels. Example image of a broken glass thresholded by Otsu's method is shown in Figure 3.6.



Figure 3.6. Fragmented glass and the Otsu thresholded binary image.

Simple thresholding is very sensitive to uneven illumination and it fails to capture edges that are hard to notice. The resulting image is also very noisy and on its own not useful in terms of fragmentation analysis. There are also adaptive thresholding techniques that use local statistical features as in the pixel neighborhood as opposed to global statistics to perform the classification. An example of adaptive thresholding is shown in Figure 3.7.



Figure 3.7. Fragmented glass and the adaptively thresholded binary image.

Adaptive thresholding handles variations in illumination much better than the global one. One suitable application for the method is for example the binarization of images of text

with uneven illumination. However, an image of fragmented glass is too complicated to segment using simple thresholding.

Watershed transform

Watershed transform is a type of image segmentation algorithm. It can be classified as a region growing approach. The transform can be intuitively visualized geographically: the image is thought as a topographic surface, which is flooded by water. The water level starts rising from areas with lowest intensity values. These areas are also called catchment basins. Watersheds, or dividing lines, are built at points at which water coming from different basins would meet. Schematic of the watershed transform is shown in Figure 3.8.

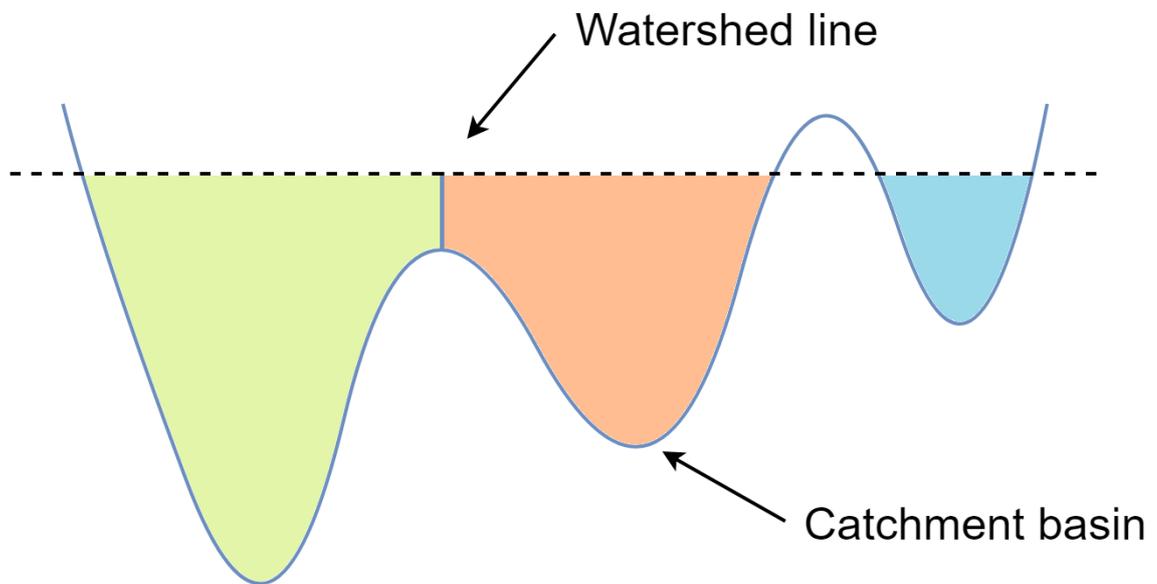


Figure 3.8. Schematic of the watershed transform.

In Figure 3.8, water level is gradually rising everywhere from the lowest intensity value and watershed lines are formed at where water coming from separate basins would first meet. In the case of broken glass, the catchment basins would ideally be the center points of each shard and the watershed lines would be formed at the edges of each shard. However, in reality, the original image is not such that the local minima would always be at the shard centers. In fact, one of the problems with the watershed transform is the severe oversegmentation occurring with unprocessed natural images [22]. According to Roerdink & Meijster, this problem can be remedied in several ways, e.g., by applying the watershed transform to the gradient magnitude image instead of the original image. This enforces the boundaries to form at the boundaries where the intensity change is the highest.

Another way to overcome the oversegmentation problem is to use the H-minima transform. It suppresses all minima whose depth is less than a threshold h . H-minima transform has been used in conjunction with watershed for example in medical image segmentation

[23]. Example of the watershed transform applied to the original fragmented glass image and to H-minima transformed grayscale image is shown in Figure 3.9.

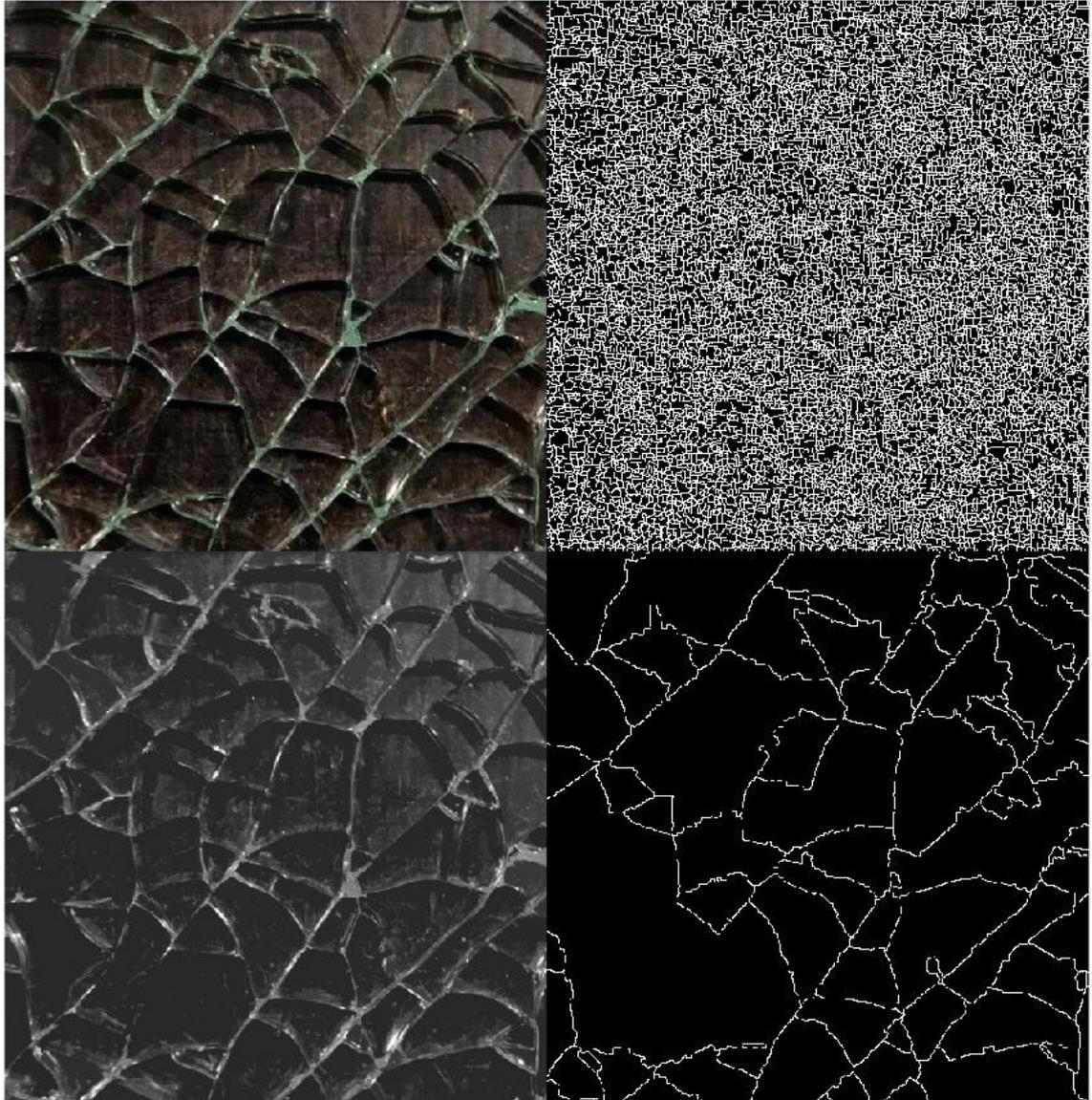


Figure 3.9. Watershed transform applied to the original image (up-right) and to H-minima transformed grayscale image (bottom-right). On the top-left is the original image for reference and on the bottom-left the H-minima transformed grayscale image.

The original image is severely oversegmented but in the case of the H-minima transformed image, the watershed transform correctly segments some of the shards. Nevertheless, many errors are present and the H-minima parameter h must be tuned individually for each input image. In this case, an iteratively found value of $h = 40$ was used. It should also be noted that the image used is a relatively easy case due to the clear edges and the absence of noise. For harder cases, the H-minima transform does not necessarily suppress the unwanted minima as reliably.

Distance transform

Distance transform is a useful tool in many image processing applications. Given a binary image, a value in the distance transform depicts the distance from the nearest nonzero pixel in the binary image. The distance can be defined in several ways, most common being *Euclidean*, *Chessboard* and *Cityblock* -distances. Equations for each method are presented in Table 3.1.

Table 3.1. Equations for different methods to calculate the distance between two pixels.

Method	Equation
Euclidean	$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$
Chessboard	$\max(x_1 - x_2 , y_1 - y_2)$
Cityblock	$ x_1 - x_2 + y_1 - y_2 $

Example of the Euclidean distance transform is shown in Figure 3.10.

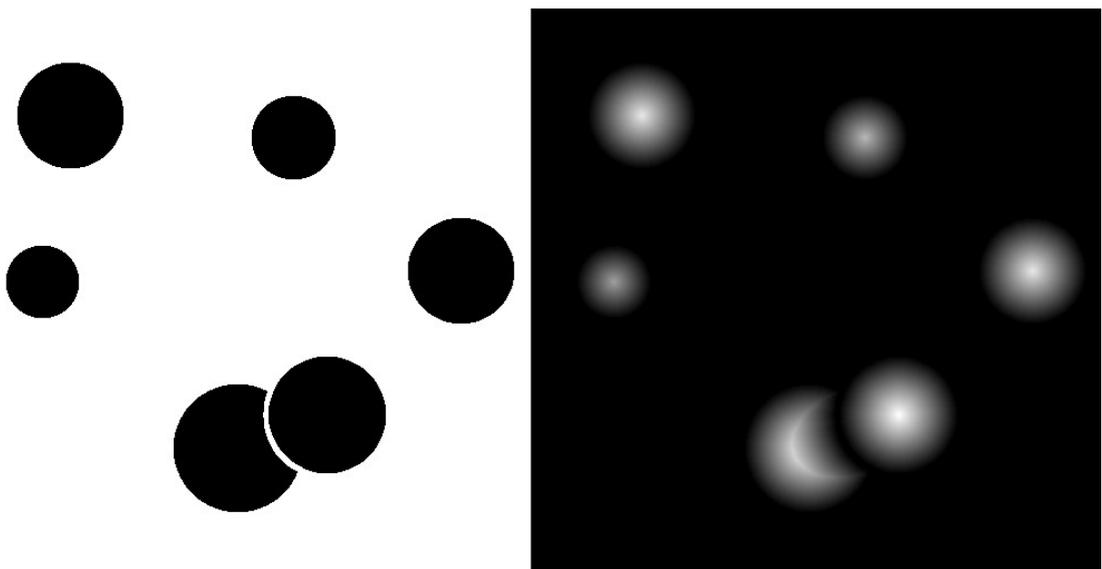


Figure 3.10. Example of Euclidean distance transform. Each pixel value in the distance transform (right) is the distance from the corresponding pixel in the binary image (left) to the nearest white pixel.

The distance transform is also used as a postprocessing step in the framework used in this work. As shown later in Chapter 4.4, it can be used in conjunction with thresholding to effectively remove small connections between separate regions in a binary image.

3.4 Machine learning approaches

A widely used quote by Tom Mitchell [24] defines the concept of machine learning: "A computer program is said to learn from experience E with respect to some class of tasks

T and performance measure P if its performance at tasks in T , as measured by P , improves with experience E ." In the scope of this work, the approaches used are those of *supervised learning*, where the computer learns with the help of example inputs and their desired outputs. In *unsupervised learning* on the other hand, the computer learns the patterns and structure of the input data without any given labels. Unsupervised learning has many applications in cases where the goal is to build representations of the input, which can be used for decision making, prediction, clustering and dimensionality reduction [25]. Supervised methods are suitable for tasks where the desired output is known and labeled data is available. In this chapter, machine learning approaches refer to supervised methods.

One of the advantages of machine learning approaches in the field of image segmentation is overcoming the problem of explicitly defining the rules for classification. Machine learning models learn the relevant features in the input data with the help of training examples. Machine learning approaches are typically much more accurate than human-crafted rules due to the fact that although humans are very good at classifying, they are incapable of accurately defining the rules by which the classification is done [26]. Training examples are typically human-generated samples that the model should learn to replicate. In the case of fragmentation analysis, examples of the original input images and the corresponding human-generated ground truth annotations are shown in Figure 3.1. The goal of the machine learning model is to learn to generate an output that is as close to its ground-truth counterpart as possible. Input to the model is the original image. In some cases, the original image can be preprocessed to make it easier to process for the model. One disadvantage of the machine learning approaches is that they usually require high amounts of human generated training data.

In image segmentation, the desired output is an image where each pixel defines the class label of the corresponding pixel in the original image. Problems where each input sample is assigned to a discrete category are called classification problems [27]. In dense prediction, a label is produced for each pixel in the image [28]. Image segmentation falls under the field of dense prediction and there are several approaches for the task. Semantic segmentation and pixel-wise classification are practically synonyms. In semantic segmentation, the goal is to separate the image into semantically meaningful regions and the system usually utilizes global contextual information. This means that one object in an image does not easily get assigned multiple labels.

In this chapter, several machine learning approaches that were experimented with during this research are presented. These include linear classifiers and neural networks. The deep learning framework used in this work is presented in Chapter 4.

Linear classifiers

Linear classifier is a classifier that uses the value of linear combination of characteristics to determine the class label. For example, if a m -dimensional feature vector $\mathbf{x}_i = (x_{i,1}, x_{i,2}, \dots, x_{i,m})$ represents a pixel p_i in an image I , the output score for the pixel is

$$y_i = f(\mathbf{w} \cdot \mathbf{x}_i + b). \quad (3.8)$$

In Equation (3.8), \mathbf{w} is a real vector of weights and b is a real number, both learned by the classifier. Feature vector \mathbf{x}_i is the vector of characteristics that the classification is based on. It can be simply pixel intensity values or some other values computed from the image. Note that Equation (3.8) is similar to Equation (3.1), which defines an artificial neuron. Function f maps the dot product into the desired output, i.e., class probability or class label. In the simplest, f could be a function that maps all values above a certain threshold to the first class and all other values to the second class.

Logistic regression is a type of linear classifier that is often applied to problems with a binary outcome. It is a natural choice for fragmentation analysis, as only two classes, a shard and a non-shard, are present. As an output, the logistic regression model gives the probability of a sample belonging to a certain class according to the Equation (3.8). In logistic regression, the mapping function f is a sigmoid function defined as

$$f(\mathbf{w} \cdot \mathbf{x} + b) = \frac{1}{1 + e^{-(\mathbf{w} \cdot \mathbf{x} + b)}}. \quad (3.9)$$

Although the sigmoid function is a non-linear function, the classifier is still linear because it produces linear decision boundaries. The sigmoid function is convenient because it maps any real value into a range of $[0, 1]$, which can directly be interpreted as a probability. Parameters for the logistic regression are learned by maximum likelihood estimation [29]. Linear classifiers such as logistic regression can be applied to multiclass cases as well.

Sequential neural network

In this work, two main types of neural networks are experimented with. First one is a sequential structure of multiple shallow networks inspired by Jurrus et al. [30]. Second one is a deep neural network structure which, in this work, was found to by far outperform every other tested method. In this chapter, the sequential ANN structure is presented. The deep neural network framework is described in detail in Chapter 4.

The original framework proposed by Jurrus et al. consists of two main components: a stencil operator to efficiently sample a patch in pixel neighborhood and a simple multi-layer perceptron with one hidden layer. The concept of the stencil is illustrated in Figure 3.11.

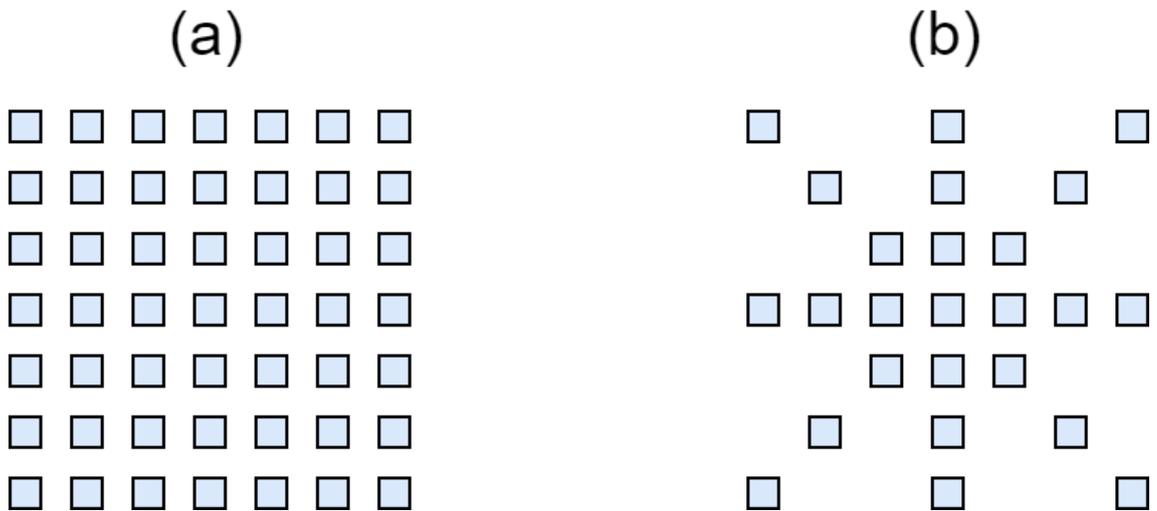


Figure 3.11. Examples of image sampling techniques: a patch (a) and a stencil (b). With the stencil approach, a same area of the image can be covered with less parameters. Effect is magnified as the area is increased.

The sequential structure comes from having these MLP classifiers in series, each one receiving as input the original image and the prediction from the previous MLP classifier. The flow of information in the network is shown in Figure 3.12.

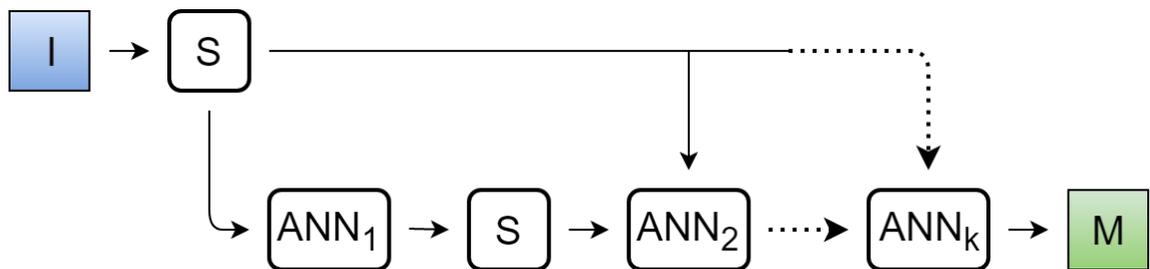


Figure 3.12. Flow of information in the serial ANN structure. I is the original image, S is the stencil operator that samples the input data and M is the probability map from the final classifier.

In Figure 3.12, each ANN is a MLP consisting of one hidden layer. Each ANN receives as input the original image intensity values and the probability values from the previous classifier, both sampled through a stencil. Therefore, every network in the series has different inputs but a common desired output. Jurrus et al. state that this structure has two main benefits. First, the classifiers use raw image intensities rather than hand-crafted filter banks or statistical features. Second, the classifiers can gather more and more contextual information as the network progresses as each classifier in the series can utilize the results from the previous classifiers.

In this work, the sequential network is implemented with Keras using TensorFlow backend. Performance of the system in the task of shard segmentation is demonstrated in Figure 3.13.

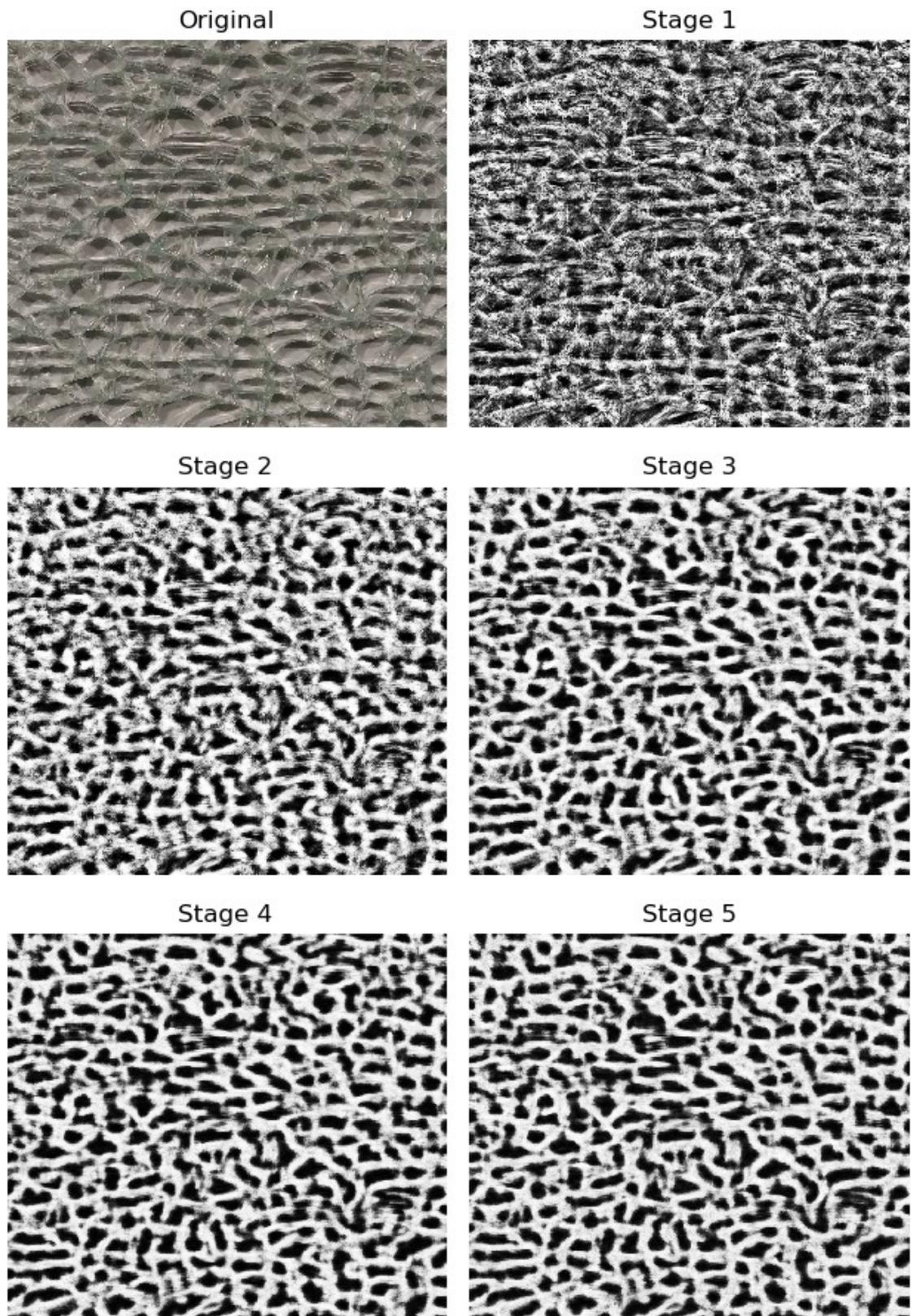


Figure 3.13. Original image and the ANN predictions for all stages in a sequential network consisting of 5 ANN classifiers.

The network used in Figure 3.13 consists of 5 networks in series. The prediction clearly gets more precise as the network progresses. After stage 1, the probability map is very

noisy and there are no clear borders. At later stages, noise is gradually removed and the network is more confident of its predictions. However, the network is unable to close gaps and form clear borders along individual shards even after stage 5. The network was trained with sparsely annotated data, where pixels indicating a shard were always at the approximate center point of the shard. An equal number of pixels indicating an edge were added to the training data. The network could possibly perform better if it was trained with fully annotated images. In later published paper [31], Jurrus et al. improved the performance of their model by applying a tensor-voting algorithm at the end of the network. They showed that the algorithm effectively closed remaining gaps and further removed noise.

Another significant problem with this kind of framework is the inference time. In practice, each ANN has to be inferred once because they require the output of the previous ANN as input. Processing high-resolution images is very slow and not doable in real time applications. However, experimenting with this framework shows that even very simple neural networks are, at least on some level, able to learn the concepts of shards and edges in an image.

4. DEEP NEURAL NETWORK FRAMEWORK FOR GLASS FRAGMENTATION ANALYSIS

Convolutional neural networks have for long been dominating the field of image classification and object recognition. In 2012, a deep convolutional neural network architecture first reached state-of-the-art classification performance on ImageNet Large Scale Visual Recognition Competition (ILSVRC) [32]. ImageNet is a highly challenging dataset consisting of over 15 million images belonging to roughly 22 000 different categories [33]. The images are collected from the internet and labeled by humans. The AlexNet network proposed by Krizhevsky et al. in 2012 reached an error rate of 15.3 % compared to the second place of 26.2 %, considerably increasing state-of-the-art performance. The error rates mentioned are top-5 error rates, where the rate is a fraction of images for which the correct label is not among the five most probable labels proposed by the model. For comparison, in 2017, the error rates for best performing models are close to 2 % [34]. This is due to the significant advancements in computer technology and neural network architectures in the recent years.

Long et al. [18] showed that the advances made in image classification and object recognition can be transferred to the task of semantic segmentation. According to the authors, they were the first to train fully convolutional neural networks (FCNN) end-to-end for pixelwise prediction and from supervised pre-training. How they achieve this is by casting the existing classifiers into FCNNs and augmenting them for dense prediction. Augmentation is done by in-network upsampling and by adding a pixelwise loss. Upsampling is required because the classification networks usually have very low output resolution, hampering the localization accuracy. The existing networks are fine-tuned for the segmentation task, thus transfer learning is utilized. The AlexNet network by Krizhevsky et al. mentioned earlier was also converted to the segmentation task in their work. The convolutional networks achieved reasonable performance by standard metrics, but the output was coarse. Long et al. addressed this issue by proposing a new architecture that was augmented with skip layers. The proposed architecture reached state-of-the-art performance on many segmentation challenges such as PASCAL VOC, NYUDv2 and SIFT Flow. [18]

Since the work by Long et al., deep convolutional neural networks (DCNN) have pushed the state-of-the-art performance in segmentation challenges. One property of traditional DCNNs is the gradual reduction in signal resolution due to the constant striding and pooling operations between the convolutional layers. This is desired in high-level tasks such as image classification, where abstraction of local features is important. However, it hampers low-level tasks such as semantic segmentation because the final layer of the network is not localized enough for pixelwise prediction. [35] The Deeplab system proposed by

Chen et al. in 2014 further increased the performance in PASCAL VOC 2012 segmentation challenge. In this work, the latest DeepLab v3+ architecture is adopted.

In this chapter, the framework adopted for semantic segmentation in the case of fragmentation analysis is presented. The main methods and components used in the Deeplab v3+ architecture are explained in detail. The architecture supports several network backbones, namely Xception [36], ResNet [37] and MobileNet v2 [38], which are also briefly presented. The actual performance of the framework and dataset details are presented in Chapter 5.

4.1 Overview of the framework

The framework used in this work for the task of fragmentation analysis consists of four main components: a DCNN for feature extraction, an atrous spatial pooling pyramid (ASPP) scheme for capturing multiscale context, a decoder module for restoring spatial accuracy and a postprocessing scheme for obtaining the quantities of interest. The DCNN, ASPP and decoder modules are provided in the DeepLab v3+ system. Illustration of the fragmentation analysis pipeline is shown below.

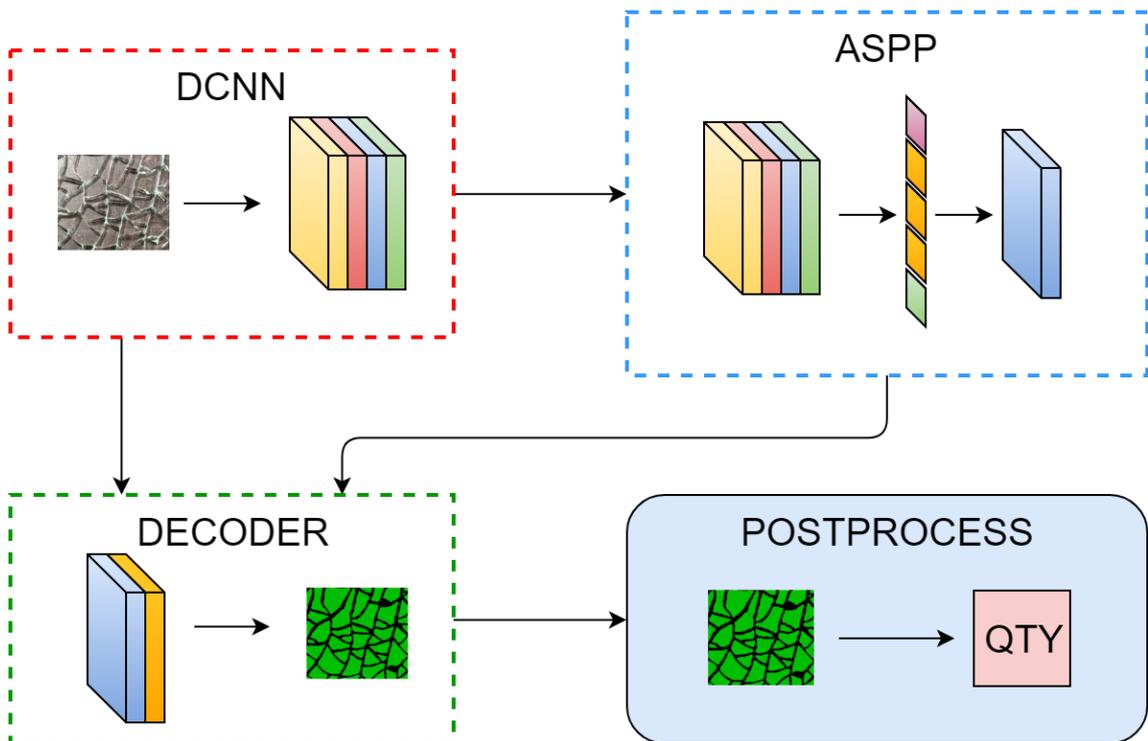


Figure 4.1. Pipeline of the fragmentation analysis framework. The DCNN module extracts features from the raw image and the ASPP module captures multi-scale information. Decoder module generates the segmentation map by combining low resolution semantic information from the ASPP and low-level features from the DCNN. Postprocessing is used for obtaining the quantities of interest.

Each of the modules shown in Figure 4.1 are described in the following sections. The structure of the DCNN depends on the network backbone used. In this work, DeepLab v3+ variants of Xception, ResNet and MobileNet v2 are experimented with. A more detailed structure of the DCNN for each network backbone is given in Chapter 4.3. MobileNet v2 differs from the others as it does not utilize the ASPP and decoder modules. Instead, the DCNN directly generates the segmentation mask.

DeepLab v3+ provides several novel techniques to improve the segmentation results over existing solutions. Main components of the system are presented in Chapter 4.2.

4.2 DeepLab v3+ architecture and methods

The DeepLab architecture has several versions, namely v1 and v2 [39], v3 [40] and v3+ [41], each achieving state-of-the-art performance on release. The first two versions of DeepLab architecture integrate two components: a DCNN for coarse pixelwise prediction and fully connected conditional random fields (CRF) as postprocessing for edge refinement. Version 2 adds the ASPP module for efficient segmentation at multiple scales. Version 3 omits the CRF postprocessing and adds an improved ASPP module. The latest v3+ improves over v3 by adding the decoder module at the end of the network to further refine localization especially along object boundaries. In this work, the latest DeepLab v3+ architecture is adopted. DeepLab is a system built to address three main problems in applying DCNNs to semantic segmentation tasks: reduction in signal resolution, spatial invariance and objects at multiple scales [35].

The first problem is resolved by employing *atrous convolution*, also known as dilated convolution. In atrous convolution, the filters are upsampled by adding zeroes between active filter taps. This effectively enlarges the field of view of the filter and allows computing feature maps more densely without increasing computational cost or sacrificing receptive field. Atrous convolution is described in Chapter 4.2.1.

The second problem, the spatial invariance, is an inherent result of utilizing classification DCNNs for the task of semantic segmentation. In image-level classification, abstraction is desired and spatial accuracy is not important to correctly predict the image-level label. However, in semantic segmentation, the spatial accuracy is very important. In DeepLab v3+, this problem is alleviated by employing a decoder module at the end of the network for restoring the local spatial information. According to the authors, the decoder module allows for accurate object boundary recovery without a need for postprocessing. In previous versions of DeepLab (v1 & v2), the boundary recovery was achieved by means of conditional random fields (CRF). In particular, an efficient dense CRF implementation was employed [42]. The experiments of Chen et al. however showed that a simple decoder module achieves better performance at object boundaries.

The last difficulty arises from objects existing at multiple scales. To address this issue, DeepLab v3+ applies atrous spatial pyramid pooling (ASPP) with different atrous rates (i.e., different field of view) and fuses the result into one feature map. The decoder module also utilizes multi-scale information from the encoder module to further refine the object boundaries.

All of the methods above enabled DeepLab v3+ to achieve state-of-the-art performance on a challenging PASCAL VOC 2012 segmentation dataset [41]. The relevant methods used in the network are described in this chapter.

4.2.1 Atrous convolution

Atrous or dilated convolution is a modified convolution operation inspired by *algorithme à trous*, an algorithm for wavelet decomposition [43]. It can be used to increase the receptive field of the convolution filter without increasing model parameters. In practice, it works by adding zeroes between active filter taps. The distance between active filter taps, and therefore the number of zeroes, is defined by an *atrous rate* parameter. Mathematically, the output of discrete 2-dimensional convolution operation at any point \mathbf{i} can be expressed as

$$Y(\mathbf{i}) = \sum_{\mathbf{k}} F(\mathbf{i} + \mathbf{k})G(\mathbf{k}), \quad (4.1)$$

where Y, F, G are two-dimensional values. In equation (4.1), F is the input signal, G is the filter and Y is the output value. Atrous convolution can be expressed similarly by introducing an additional parameter, the atrous rate r :

$$Y(\mathbf{i}) = \sum_{\mathbf{k}} F(\mathbf{i} + r\mathbf{k})G(\mathbf{k}). \quad (4.2)$$

In Equation (4.2), the input signal F is dilated by a factor of r when calculating the convolution. Note that the filter G remains constant and therefore atrous rate can be tuned to effectively calculate feature responses at different scales. The concept of atrous convolution is illustrated visually in Figure 4.2.

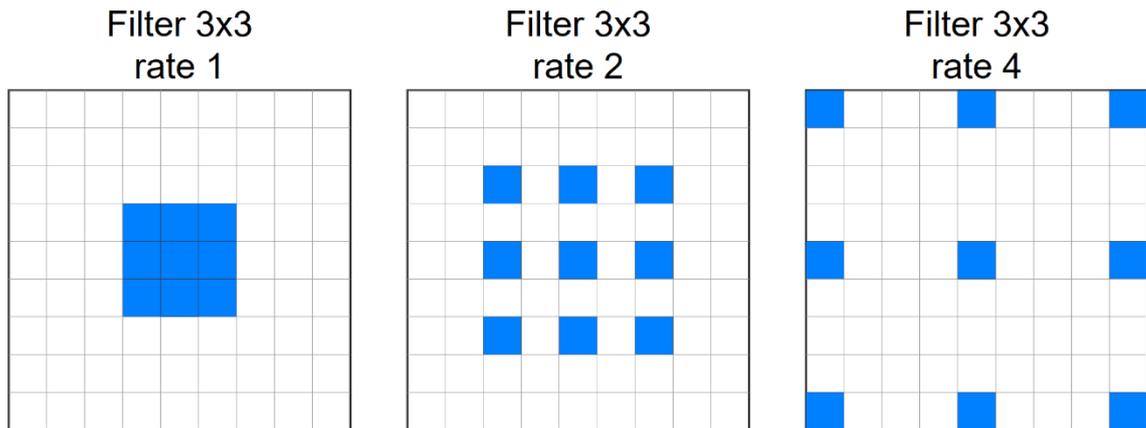


Figure 4.2. Atrous convolution with 3x3 filter and different rates applied on a 9x9 feature map. Standard convolution corresponds to rate = 1. Increasing the atrous rate effectively increases the filter’s field of view without increasing the number of parameters.

As depicted in Figure 4.2, the field of view of the filter increases with atrous rate while the number of parameters remains the same. Without atrous convolution, constant downsampling of the signal is required to capture contextual information at multiple scales. In classification tasks it is not a problem, as the output is just a low-resolution class label. However, in dense prediction tasks, the output feature map should have the same resolution as the input. Atrous convolution can be utilized to compute feature maps more densely without increasing computational cost or complexity of the model.

In many DCNNs implemented for classification tasks the resolution of final output feature response is up to 32 times smaller than the input image resolution [40]. The ratio between input spatial resolution and output spatial resolution is called *output stride*. Thus, in the former case, $output_stride = 32$. The effectiveness of atrous convolution in retaining spatial resolution can be demonstrated with the following example. Given an image, suppose there is first a downsampling operation that reduces the resolution by a factor of 2 and then a convolution operation. The convolution thus only computes feature responses at 1/4 of the original image locations. Feature response at all image positions can be achieved by removing the downsampling operation and instead upsample the subsequent convolution filter by a factor of 2, i.e., use atrous convolution with $rate = 2$. This way, the feature response can be computed at original resolution without degrading field of view or increasing number of parameters. This scheme could be extended to any network by removing downsampling layers and by replacing the subsequent convolutions by atrous convolutions with appropriate rates. However, it is too costly to compute all feature responses at original image resolution. Instead, a hybrid approach can be adopted: increase atrous rate and upsample the feature responses to original resolution by bilinear interpolation. It is acceptable if the class probabilities given by the model are sufficiently smooth. [39]

In Figure 4.3, the effectiveness of atrous convolution in retaining spatial dimensions is demonstrated.

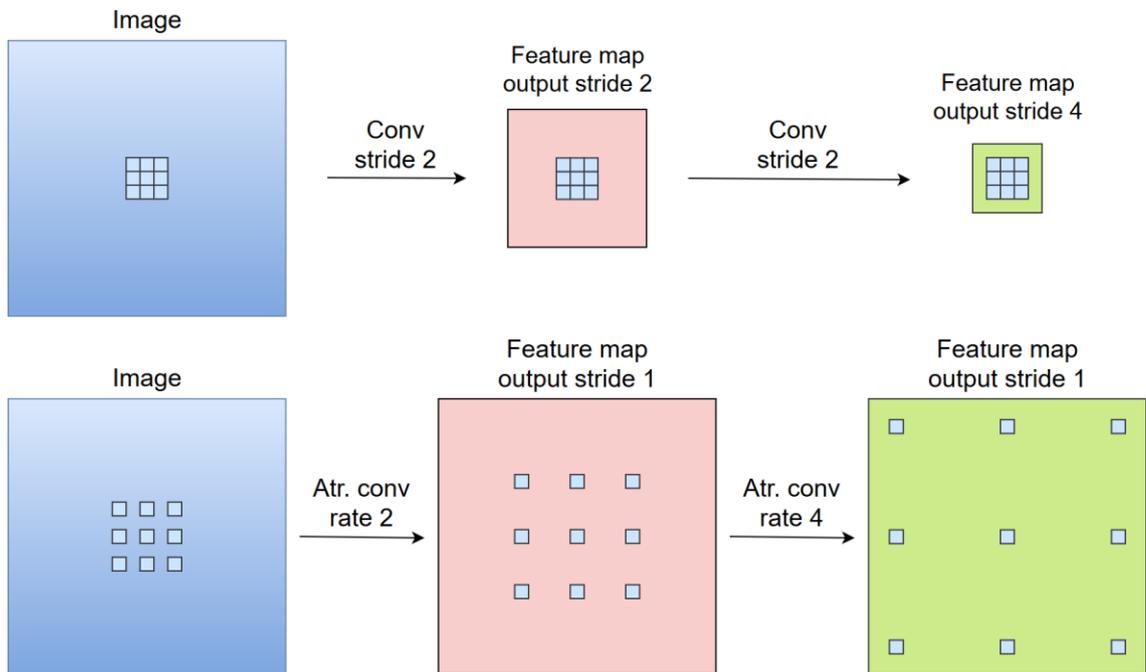


Figure 4.3. Networks with (above) and without (below) atrous convolution. If atrous convolution is not used, the final output stride equals to 4 due to the striding operations. When atrous convolution is used, the network is able to retain original image resolution without sacrificing field of view or increasing the number of parameters.

In Figure 4.3, the network (a) uses standard convolution accompanied with constant striding to capture long-range information. The striding eventually decimates the performance in semantic segmentation tasks due to the low resolution of the output. By utilizing atrous convolution, the network (b) can similarly capture long-range information whilst also retaining output stride and localization accuracy. Note that the number of parameters and computations per filter position also remain unchanged.

4.2.2 Atrous spatial pyramid pooling (ASPP)

In atrous spatial pyramid pooling, the input feature map is convolved in parallel with different atrous rates and the resulting feature maps from different branches are fused to form the final output. The ASPP module is inspired by the work of He et al. [44], who were the first to implement spatial pyramid pooling in the context of CNNs.

The ASPP module is located after the last convolutional layer in the encoder part of the network. The convolutional feature map is sampled with different atrous rates using a 3x3 filter. The pyramid is also augmented with a 1x1 convolution and image pooling. Schematic of the ASPP module is shown in Figure 4.4.

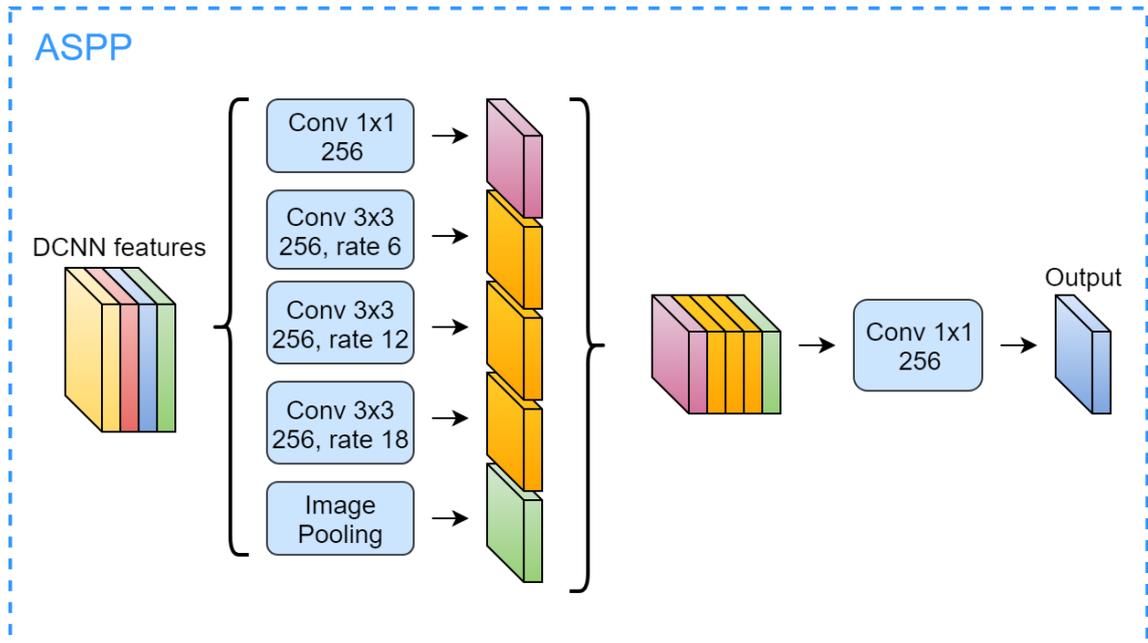


Figure 4.4. Schematic of the DeepLab v3+ ASPP module. The ASPP module is expanded in the middle. The DCNN feature map is sampled with a 1x1 convolution and multiple 3x3 convolution with varying atrous rates. Image pooling is added to capture whole-image context.

In Figure 4.4, the input to the ASPP is the feature map generated by the DCNN, whose structure depends on the network backbone (see Chapter 4.3). The feature map is then sampled with varying atrous rates to capture multiscale context. Each convolution branch in the ASPP module has 256 filters, thus producing an output with a depth of 256. The outputs from all branches are then concatenated and fed through 1x1 convolution also with 256 filters. The output of the ASPP module is directed to the decoder module, which is described in Chapter 4.2.3.

The image-pooling branch performs global average pooling on the feature map before passing it through a 1x1 convolution. Global average pooling is an operation that takes an average over the input channels. For example, performing global average pooling on tensor with dimensions $W, H, C = 10, 10, 3$ would result in a tensor with dimensions $1, 1, 3$. Global average pooling is sometimes used to replace fully connected layers in classification networks. Convolutional feature maps need to be vectorized before feeding them to FC layers. One approach is to perform the vectorization via global average pooling and feeding the result directly to a softmax layer. This approach has been shown to reduce overfitting and improve generalization over traditional fully connected layers [45]. In the case of DeepLab, the global pooling layer is added because atrous convolution with large rates fails to capture global context information. This happens because when the atrous rate becomes large, most of the active filter weights are outside image borders. This essentially degrades the atrous convolution into a 1x1 convolution since only the center filter weight is active. By introducing a global pooling layer, whole image context can be captured.

4.2.3 Decoder module

Encoder-decoder networks have been successfully applied to many tasks where accurate spatial information is needed. These include human pose estimation, object detection and semantic segmentation [41]. The encoder module encodes the input to a small resolution feature vector, which contains rich semantic information but the spatial information is mostly lost. The output of the encoder can be used for tasks of high abstraction such as classification. For tasks like semantic segmentation however, a decoder module can be used to recover the lost spatial information.

How the decoder works is by combining the rich low-level semantic information from the encoder module with the accurate spatial information from the earlier layers of the network. The decoder structure of DeepLab v3+ is shown in Figure 4.5.

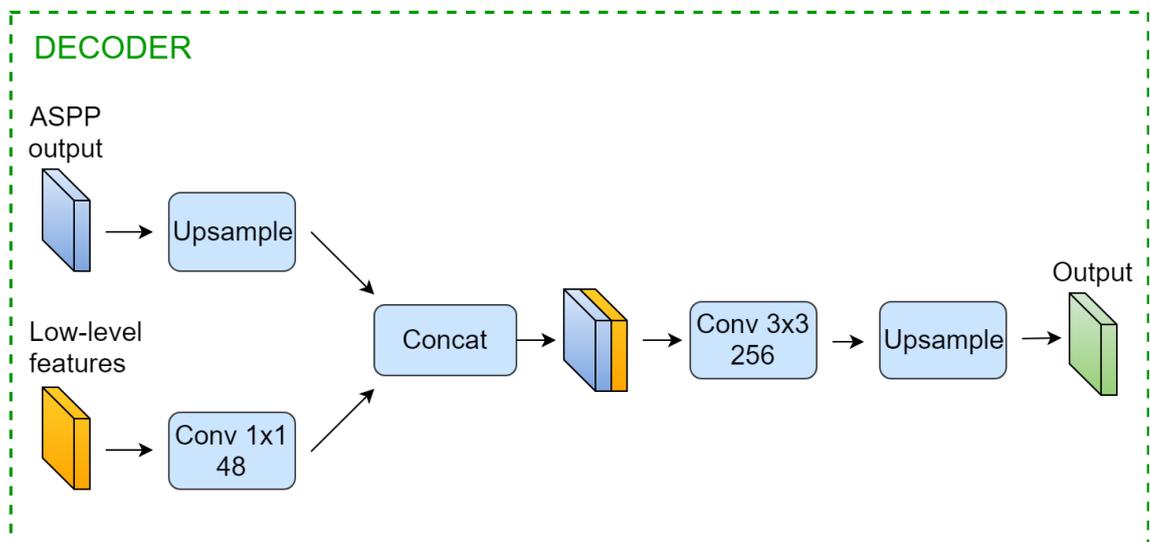


Figure 4.5. DeepLab v3+ decoder structure. Semantically rich ASPP features and spatially accurate low-level features from the earlier layers of the network are combined to form the final segmentation mask.

As shown in Figure 4.5, DeepLab v3+ uses a relatively simple decoder module, where low-level image features are combined with encoder output feature map. In this context, the low-level features refer to a hypercolumn representation proposed by Hariharan et al. [46]. They define the hypercolumn at a pixel as a vector of activations of all layers above that pixel. Hariharan et al. state that the idea is motivated by the hypothesis that the information of interest is distributed over all levels of the CNN. Visualization of the hypercolumn approach is presented in Figure 4.6.

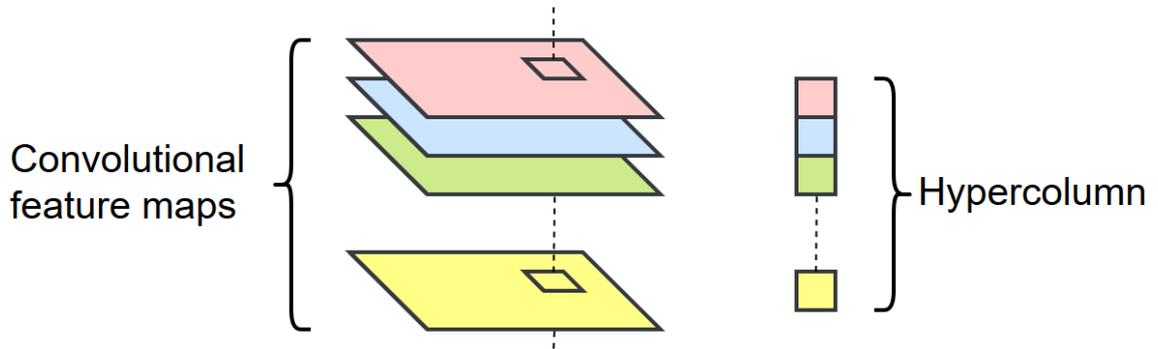


Figure 4.6. Pixel hypercolumn representation. Feature maps are from different layers of the DCNN.

The 256-channel encoder output is directed to the decoder module. The low-level features also contain many channels since multiple feature maps are concatenated to form the hypercolumn tensor. Therefore, low-level features are convolved with 1×1 filter to reduce the number of channels. Chen et al. [41] state that this helps training the model by preventing the large number of channels in the low-level representation from outweighing the encoder output. After experiments, they ended up reducing the channels of low-level features to 48.

The decoder module has two upsampling operations. In DeepLab v3+, the encoder and decoder modules typically have output strides of 16 and 4, respectively. Therefore, the encoder output must first be upsampled by a factor of 4 to match the decoder spatial dimensions. The output of the decoder is then further upsampled by 4 to match the input image dimensions. The system allows for different output strides, but in their experiments, Chen et al. noticed that a value of 16 attained the best trade-off between accuracy and speed. Output stride of 8 marginally improved the results with the cost of increased computational cost. However, one interesting property of the system is that the inference output stride need not be the same as the stride used during training. This allows for more accurate inference if enough computational resources are available.

4.3 Network backbones

DeepLab is a versatile system capable of supporting various network backbones. In this work, the backbone refers to the DCNN block in Figure 4.1, i.e., the part of the encoder whose output is passed to the ASPP module. The DCNN is the most important part of the network since it is responsible for learning the features that best represent the objects to be segmented. The ASPP and decoder modules on the other hand serve the purpose of improving multi-scale performance and refining object boundaries.

In this work, models with different backbones are trained with same hyperparameters for qualitative results. The backbones experimented with are Xception, ResNet and Mo-

bileNet v2, each described in detail in this chapter. In DeepLab v3+, the Xception backbone has several variants, namely Xception 41, 65 and 71, the number indicating network depth (i.e., number of layers). Similarly, ResNet has 2 variants, 50 and 101, the number also indicating depth.

4.3.1 Xception

The Xception network is based on the following hypothesis: cross-channel and spatial correlations in CNN feature maps can be entirely decoupled [36]. This is in contrast to usual convolution operation, in which a single filter learns both cross-channel and spatial correlations. The idea is inspired by the Inception architecture [47], which partly decouples spatial and cross-channel correlations. In Inception, the input to the layer is convolved with different filters in parallel. Each parallel branch is convolved with 1×1 filter to reduce dimensions. Xception architecture takes this one step further by completely decoupling spatial and cross-channel correlations, hence the name “Extreme Inception” or “Xception”. In particular, Xception uses a scheme known as *depthwise separable convolution*, where a spatial convolution is first performed separately on each channel and a 1×1 convolution follows to capture cross-channel correlations. In his experiments, Chollet showed that Xception outperforms Inception in several classification challenges without increasing model parameters. The concept of depthwise separable convolution is illustrated in Figure 4.7.

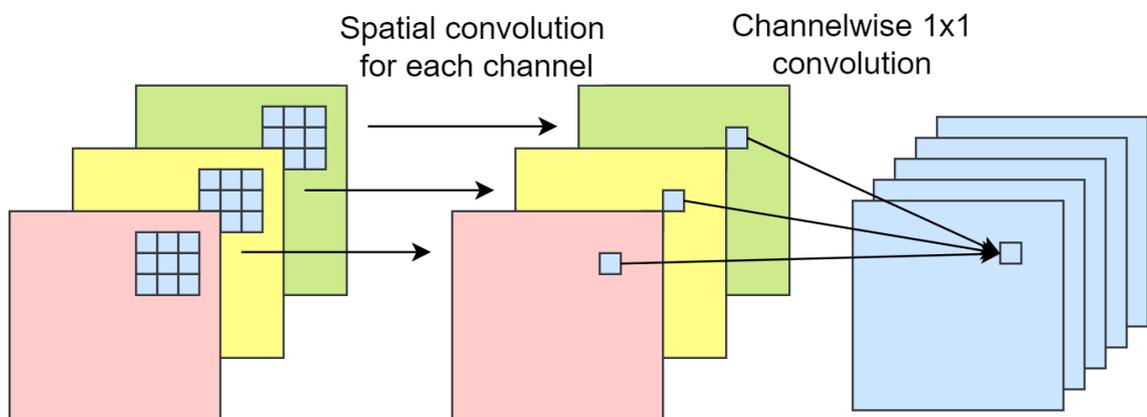


Figure 4.7. *Depthwise separable convolution. Each channel of an input is convolved with one kernel keeping the channels separate. Resulting feature maps are convolved with an arbitrary number of 1×1 kernels.*

Inspired by He et al. [37], the Xception architecture also adds residual connections around convolution modules. For more details on residual networks, see Chapter 4.3.2.

The original Xception architecture consists of 36 layers [36]. In DeepLab v3+, Chen et al. modify the Xception architecture by adding more layers, by replacing max pooling operations with convolutions and by adding batch normalization and ReLU activation

after each 3x3 convolution. The modified Xception DCNN model used in this framework is shown in Figure 4.8.

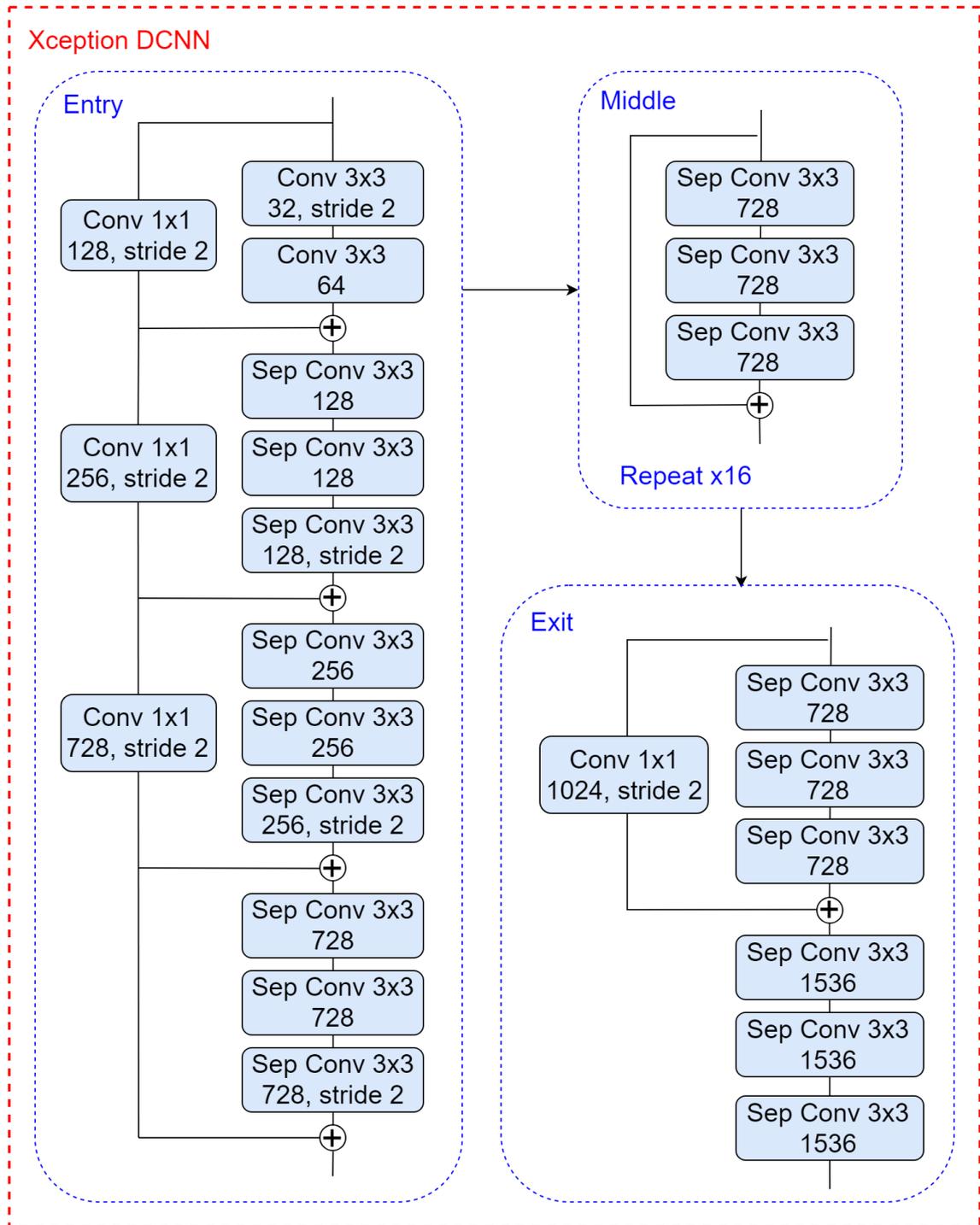


Figure 4.8. Modified Xception 65 architecture employed in DeepLab v3+. Sep conv = depthwise separable convolution.

Figure 4.8 depicts the Xception 65 network since there are 65 layers without counting the skip connections. Xception 41 is otherwise similar but the middle block is repeated only 8 times. In Xception 71, there are 2 additional blocks in the entry flow. In particular, the first and the last block of the entry flow are duplicated with a stride of 1.

4.3.2 ResNet

Residual networks (ResNet) were first created to address the problem of training deep neural networks. For long, “deep” networks were considered to consist of 16 to 30 layers. At the same time, it was widely acknowledged that network depth is crucial for the performance of the model. However, improving the existing networks is not as easy as just adding more layers. One problem is the vanishing gradient, which means that in deep networks, the gradient eventually degrades to a very small number as it is backpropagated through the network. This problem however has been mostly addressed with various initialization and normalization schemes. Another problem is the degradation of performance as the model performs worse even during training when the number of layers is increased. The idea for residual networks comes from the following insight: a deeper model should not perform worse than its shallower counterpart because the layers could be effectively deactivated by performing a simple identity mapping, i.e., by not changing the input in any way. [37]

The proposal by He et al. makes it easier for the network to perform the identity mapping. They add direct connections around the typical convolutional layers. Example of a residual block is shown in Figure 4.9.

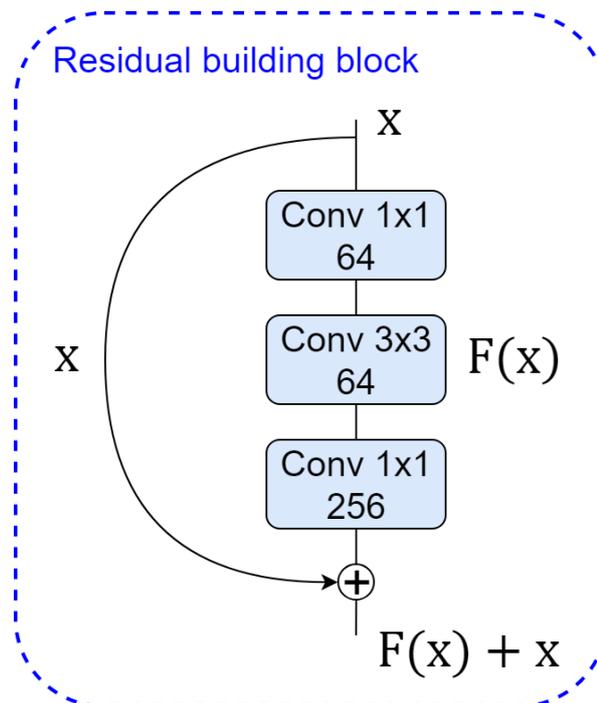


Figure 4.9. An example of residual building block.

The building block shown in Figure 4.9 is a so-called *bottleneck* block, where the 1x1 convolutions first decrease and then restore dimensions. This is done for computational efficiency since the input to the block usually has many channels in very deep networks (256 in the example figure). [37]

Denoting the desired mapping to be learned as H , in residual block, the network has to instead learn the mapping $F(x) = H(x) - x$. He et al. hypothesize that this kind of mapping is easier to optimize than a direct mapping. In an extreme case, if the optimal mapping was the identity $H(x) = x$, it might be easier to learn the mapping $F(x) = 0$ instead. In a way, the residual network itself can decide the optimal depth since it is easier to skip layers. In fact, the experiments by He et al. show that by using residual connections, the performance increases along with the number of layers and the networks show no sign of optimization difficulties. However, they experimented with an extreme 1201-layer network on CIFAR-10 dataset [48] and it performed slightly worse than its 152-layer counterpart. They stated that this was probably due to overfitting to a relatively small dataset.

In DeepLab v3+, the 50- and 101-layer residual networks are implemented for the most parts in their original form. The architectures used in this work for ResNet DCNNs with 50 and 101 layers are shown in Figure 4.10.

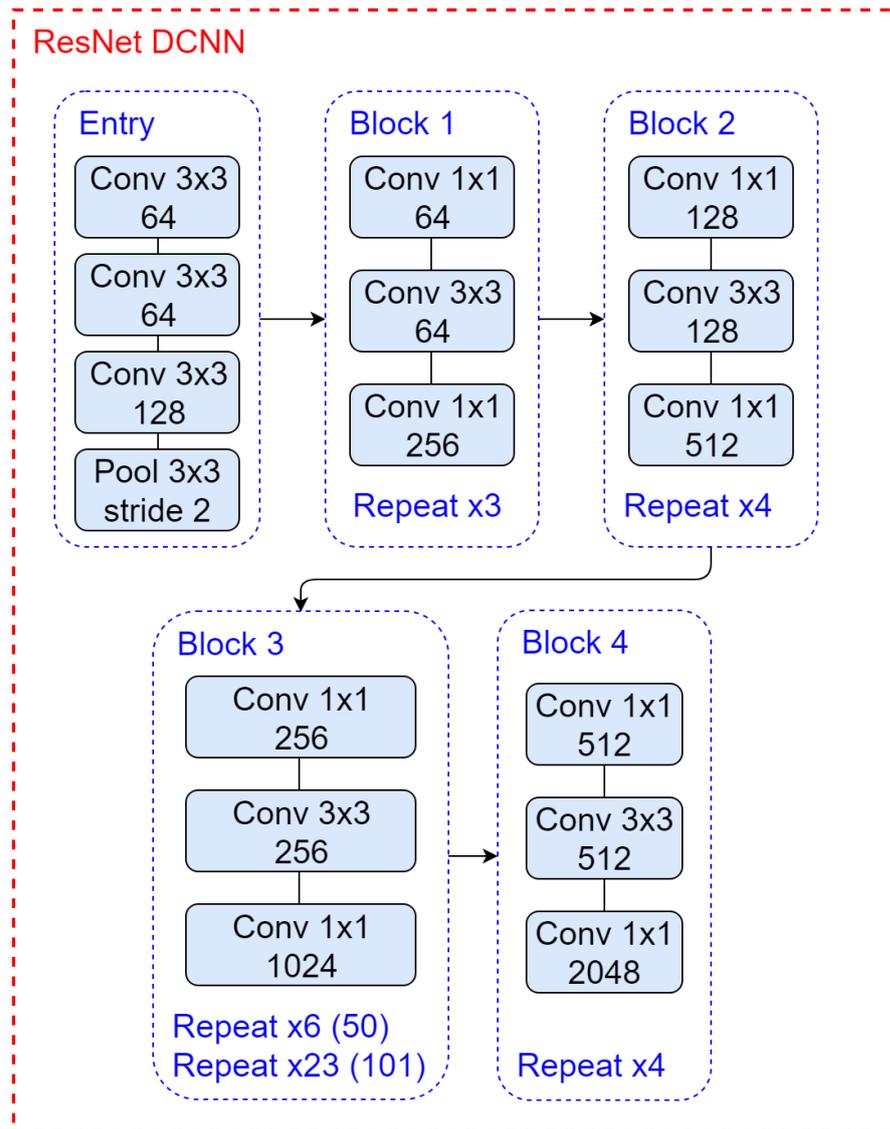


Figure 4.10. ResNet architectures for 50- and 101-layer versions. Residual bottleneck blocks 1-4 are repeated a number of times before moving on to the next block. Each Block 1-4 contains a residual connection over the convolutional layers as illustrated in Figure 4.9.

Main difference in this implementation is the Entry block, where there is only a single 7x7 convolution and the pooling layer in the original paper by He et al. [37].

4.3.3 MobileNet v2

MobileNet v2 is a lightweight deep neural network architecture proposed by Sandler et al. [38]. It is developed for mobile devices. Similar to Xception, it utilizes depthwise separable convolution to significantly decrease computational costs without heavily affecting model accuracy. Sandler et al. show that by applying depthwise separable convolution with a kernel size k , the computational cost is decreased almost by a factor of k^2 compared to the standard convolution.

MobileNet v2 extends the original architecture by two components: inverted residuals and linear bottlenecks. The original residual connections utilized in ResNets (see Figure 4.9) connect the input and the output of the block, while the 1×1 bottlenecks reduce channels in between. In MobileNet v2 however, the process is inverted by expanding the intermediate layers by a factor called *expansion ratio*. The number of parameters is still much lower since depthwise separable convolution is used between the bottlenecks. Example of inverted residual building block used in MobileNet v2 is shown in Figure 4.11.

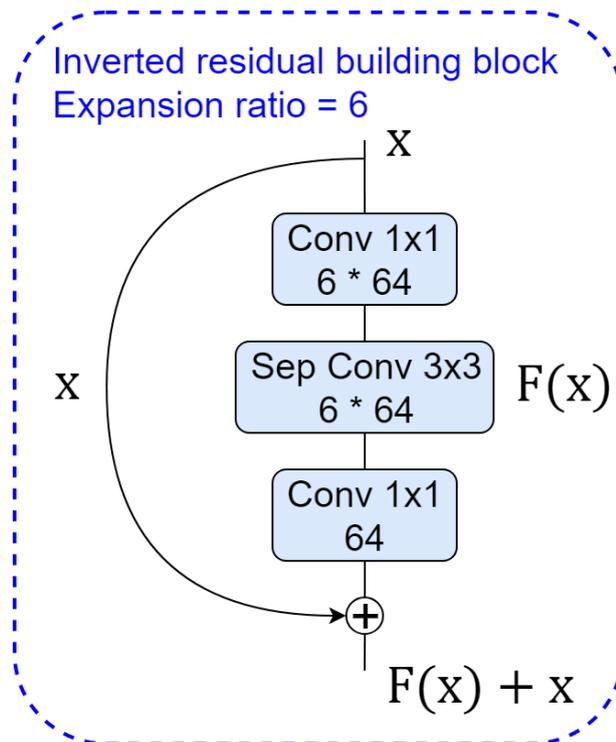


Figure 4.11. Example of an inverted residual building block. A 64-channel input is connected to the output. First 1×1 convolution expands the depth by a factor of 6 (expansion ratio). Intermediate 3×3 convolution is depthwise separable to reduce the number of parameters. The final 1×1 convolution restores the original dimensions. The output channels of the block need not be same as the input channels, but then the residual connection must be expanded appropriately.

Another contribution is *linear bottlenecks*. Traditional activations such as ReLU discard information since all negative values are set to zero. This problem is often tackled by adding more channels to the layers. In MobileNet v2, the activation function in the last 1×1 bottleneck in Figure 4.11 is simply omitted. They further show that using linear activation indeed improves the model performance. [38]

In this work, the original MobileNet v2 architecture proposed by Sandler et al. is used. Spatial convolutions utilize atrous convolution. However, the DeepLab v3+ decoder and ASPP modules are not used since the mobile network aims for fast inference with reduced accuracy. The architecture used is presented in Figure 4.12.

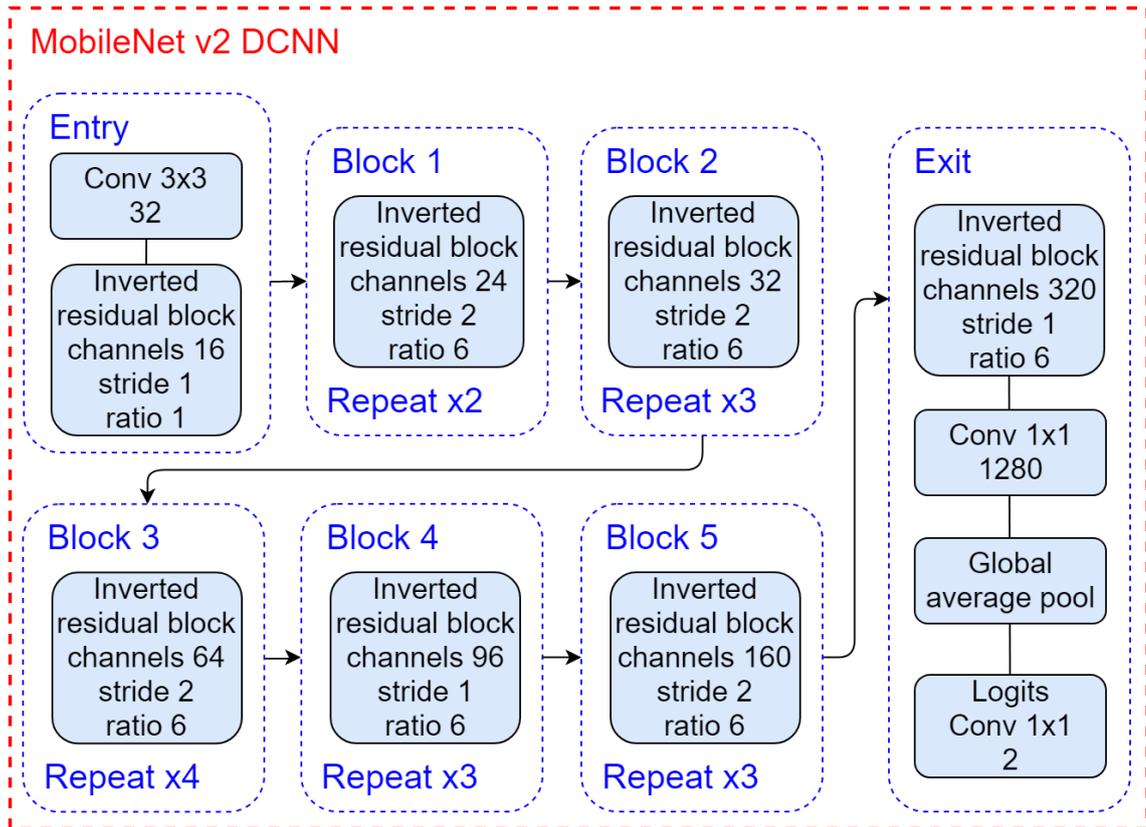


Figure 4.12. MobileNet v2 architecture. The DCNN directly generates the final logits since the decoder and ASPP modules are not utilized. Each inverted residual block is similar to the example in Figure 4.11.

4.4 Postprocessing

The output of the DeepLab v3+ system is the semantic segmentation of the input image. In many cases, the system is not able to completely separate each individual shard and some small connections remain between adjacent shards. The simplest way to count the shards is to determine the number of distinct regions in the binary segmentation mask. However, to reach accurate counts using this method, some postprocessing of the segmentation mask is needed.

By inspecting the segmentation masks, one can conclude that the shard borders are very clear and there are no misclassified pixels in the shard interiors. Only problem is the small connections that remain between some adjacent shards that are hard to separate. In this work, distance transform followed by instance-wise thresholding is used. In practice, this means that the raw segmentation mask is first distance transformed, and then each separate region in the distance transform is individually thresholded. Instance-wise thresholding is used because global thresholding of the distance transform would also remove smaller shards completely.

This scheme works by first taking the distance transform, then iterating through all separate regions in the distance transform. Denoting the distance transform of the whole segmentation mask by D , the thresholded distance transform D' for each region i can be expressed as

$$D'_i(x, y) = \begin{cases} D_i(x, y), & D_i(x, y) > T \max(D_i) \\ 0, & D_i(x, y) \leq T \max(D_i) \end{cases} \quad (4.3)$$

where T is a thresholding factor. The value of T is chosen such that it removes small connections but does not oversegment the regions. After experiments, a value of $T = 0.3$ was chosen. Before instance-wise thresholding, very small regions that can be considered noise are first removed.

In Figure 4.13, original image, raw segmentation mask and postprocessed segmentation mask are shown for one test sample.

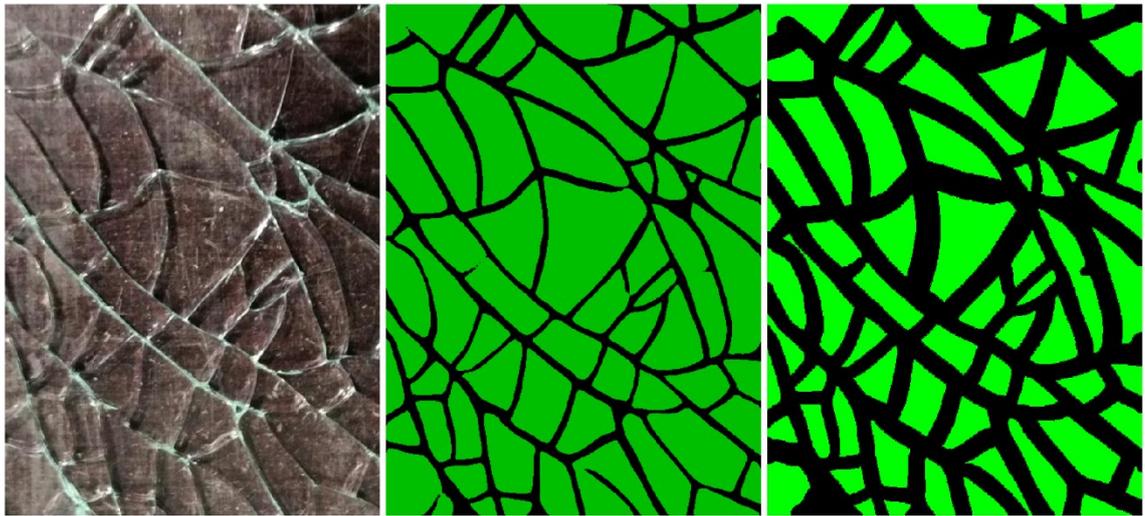


Figure 4.13. Postprocessing scheme. On the left is the original image, in the middle the raw segmentation mask from the model and on the right is the postprocessed segmentation mask. The postprocessing effectively removes small connections without removing small individual shards.

5. PERFORMANCE OF THE FRAMEWORK

In this chapter, the fragmentation analysis performance of the framework described in Chapter 4 is examined. First, the data, annotation methods and the preprocessing steps used are presented. The training scheme used with the DeepLab v3+ system is described in detail. Next, the used evaluation metrics are presented and a comparative analysis is done between different model backbones and with varying hyperparameters. Finally, sensitivity to variations in input images and generalization to unseen data are examined for the best performing model variant.

5.1 Data

The data used in this work consists of raw images of broken glass and their corresponding ground truth segmentation masks. Each sample is manually collected and labeled, making the data acquisition phase one of the most laboursome parts of this work. Nevertheless, the dataset is big enough to provide a proof-of-concept for the framework.

In this chapter, the annotation process, the preprocessing steps taken before feeding the images to the model and data augmentation methods used during model training are presented.

5.1.1 Annotation

Annotation is one of the most time-consuming parts of this work. Annotation was purely manual since the automated tools are not accurate enough for the task. The annotation tool used was MATLAB Image Labeler. Experiments showed that the model would benefit from a larger dataset. Due to time constraints, only glasses with thicknesses 4 mm and 6 mm were annotated. The dataset also contains some previously annotated glasses, which have unknown thicknesses. Each sample is labeled with a tag *4mm*, *6mm* or *mixed* to compare shard counting accuracy over varying thicknesses. In Figure 5.1, dataset samples from different categories and their ground truth annotations are shown.

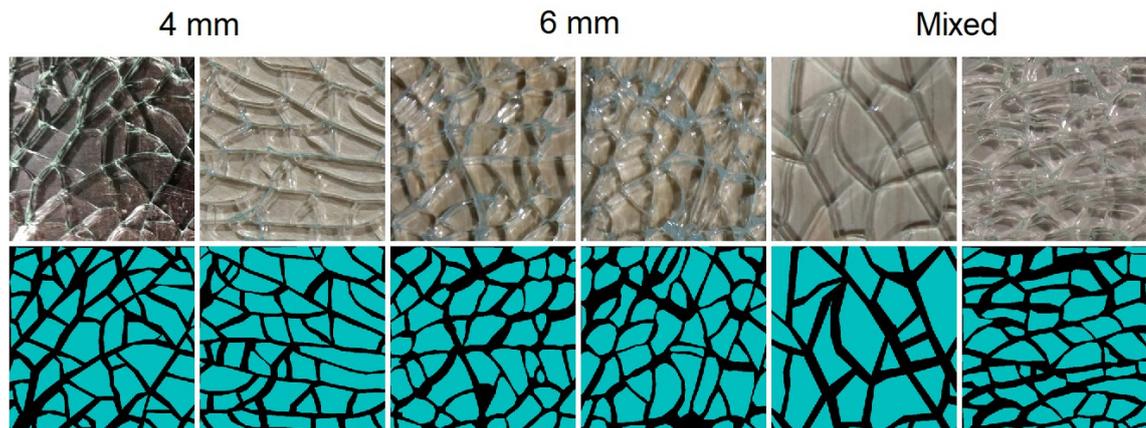


Figure 5.1. Samples of raw images and corresponding ground truth annotations for different glass categories.

Each shard in an image is labeled with a polygon by defining its vertices. It is often up to the annotators subjective judgement which pixels belong to shard regions and which do not. Each shard is however labeled such that there are sufficiently wide gaps between any adjacent shards. This enforces the model to reliably separate adjacent shards, which is critical to obtain quantities such as the shard count. In a raw image, the shard edge might sometimes only be a few pixels wide and practically indistinguishable from the background. In these cases, wider gaps are left between the label polygons. Given this annotation style, the resulting ground truth segmentation maps have approximately the following class ratio: 60 % shard and 40 % edge. During training, the loss function is weighted appropriately to balance the classes, see Chapter 5.2 for details.

5.1.2 Preprocessing

Data preprocessing is kept minimal to make the model as general as possible. The goal is that the user takes a raw image for example with a mobile phone, feeds the raw image to the network and obtains the results. Some preprocessing is however done to keep the training samples and data augmentation consistent.

First, each full sized labeled image is scale-normalized such that a 513 x 513 pixel region in the image contains approximately 50 shards. Normalization is done by up- or downsampling the image appropriately. The raw images are taken from varying distances so there are high variations in scale. Normalization is done partly because, when it comes to shard counting, the interesting property is whether a 50 x 50 mm region contains over 40 shards, as described by the standard [1]. A 513 x 513 image size gives a reasonable trade-off between image resolution and model inference time.

Second reason behind scale-normalization is to make the inputs to the model more consistent. During training, scale augmentation is done. If the raw images were not scale-normalized, the training samples would have an undesirably large variance in scale as the raw images are further scaled during augmentation.

Lastly, the scale-normalized images are cropped to 513 x 513 pixel samples. This step does not modify any pixel values. Training-wise, this step is not necessary as the model randomly crops the input images to a desired size. However, cropping before training increases the number of samples and allows the dataset to be divided into constant train and validation sets. If the image cannot be equally divided, the last blocks are concatenated into a larger sample. This means that the maximum dimensions for a sample are 1025 x 1025 pixels.

After cropping the images to 513 x 513 pixel samples, the whole dataset consists of 220 images. The dataset is splitted into training and validation sets with 85 % and 15 % respective proportions. Thus, the training and validation sets contain 187 and 33 samples. Model validation is performed on the validation set and training on the training set. The split is not completely random because it was made sure that the validation set contains samples of varying thicknesses and backgrounds.

5.1.3 Augmenting

During training, the samples are randomly flipped left-right and top-down with a 50 % probability. This effectively increases the number of samples by a factor of 4. Of course, flipped samples are very similar and they cannot be considered fully distinct samples. However, it probably removes overfitting to some extent since the network filters must work with different input values. Rotating the images with arbitrary angles was also tested, but it didn't provide any noticeable boost in performance. Thus, a simple flipping approach is adopted.

Another augmentation method during training is random scaling within a predetermined range. To keep the samples consistent, a scaling range of $[0.5, 2]$ is used with a step size of 0.25. For each input sample, the scaling factor is randomly selected from the range with the given step size, giving the possible scaling factors of $\{0.5, 0.75, 1, 1.25, 1.5, 1.75, 2\}$. In this case, scaling means that both image width and height are scaled with the given factor. Given that the input samples are preprocessed so that each 513 x 513 pixel sample contains approximately 50 shards, the model should, in theory, learn to segment samples that have from 12 to 200 shards. Performance of the model with different scales is evaluated in Chapter 5.3.3.

5.2 Training details

In this chapter, the training protocol employed is described. All of the different model backbones are trained with same set of hyperparameters for comparative results. The loss function used is cross entropy, which for a single pixel p is defined as

$$C_p = - \sum_{i=1}^2 y_i \log(\hat{y}_i), \quad (5.1)$$

where y is the target label probability distribution and \hat{y} is the distribution given by the model. The target distribution is *one-hot* encoded, where only the correct label has value 1 and others are zero. Since this is a binary classification task, the cost in Equation (5.1) is equivalent to

$$C_p = -(y_i \log(\hat{y}_i) + (1 - y_i) \log(1 - \hat{y}_i)), \quad (5.2)$$

where the index i is either 1 or 2, indicating a shard or an edge pixel. The total loss C for a batch is the average over all individual pixels, i.e.,

$$C = \frac{1}{N} \sum_{p=1}^N C_p, \quad (5.3)$$

where N is the total number of pixels in the batch. In this case, the dataset samples have an unbalanced class distribution. In particular, about 60% of the total pixels are labeled as *shard*. Training with such dataset would bias the model towards preferring labeling pixels as shards. Therefore, in the loss function, each pixel p is weighted with a factor of w defined as

$$w_p = \begin{cases} 1/\sigma, & y_p = \text{shard} \\ 1/(1 - \sigma), & y_p = \text{edge} \end{cases}. \quad (5.4)$$

In Equation (5.4), the parameter σ is the ratio of shard pixels to the total number of pixels and y_p is the ground truth label for pixel p . Weighted loss for an individual pixel can be expressed as

$$C_{p,weighted} = w_p C_p. \quad (5.5)$$

This causes the edge pixels to have larger weight in the loss function proportional to the class balance. Experiments show that balancing the classes this way indeed improves the model performance.

Hyperparameters

For qualitative results, models with different backbones are trained with the following hyperparameters. A polynomial learning rate policy is used with a base learning rate $\lambda = 0.005$. Polynomial learning policy decays the learning rate polynomially with respect to the global training step as $\lambda \leftarrow \lambda(1 - \text{step}/\text{total_steps})^\gamma$, where learning power $\gamma = 0.9$. Total number of training steps is 100 000. A batch size of 6 is used to make sure that the model generalizes over the varying glass backgrounds and thicknesses. Batch normalization is not used since it requires too high a batch size for the GPUs available. A crop size of 513 x 513 is used during training, which means that larger samples are randomly

cropped to this size. Scale augmentation is done within $[0.5, 2]$ range with a step size of 0.25. Atrous rates of 6, 12 and 18 are employed in the ASPP module. Encoder output stride is 16 and decoder output stride is 4. Values for hyperparameters are summarized in Table 5.1.

Table 5.1. *Hyperparameters for backbone comparison.*

Hyperparameter	Value
Learning policy	Poly
Base learning rate	0.005
Training steps	100 000
Batch size	6
Batch normalization	False
Crop size	513 x 513
Scale aug. range	$[0.5, 2]$
Scale aug. step	0.25
Atrous rates	6, 12, 18
Encoder output stride	16
Decoder output stride	4

Model weight initialization

Every model tested in this work uses an ImageNet pretrained checkpoint to initialize the model weights. Using pretrained model weights is known as *transfer learning*, a powerful technique for reusing trained models. The idea behind transfer learning is that knowledge gained from one dataset can be applied to other datasets. In practice, it works by reusing the trained filters. In many cases, the filters learned especially in the earlier layers of the network focus on low-level information such as shapes, edges etc. This information can be utilized in other tasks as well. Deep CNNs are usually quite difficult to train from scratch so it is useful to use a pretrained model that can be fine-tuned to a specific task.

The authors of DeepLab provide several models pretrained with different datasets such as ADE20K, COCO and ImageNet. ImageNet is the most general of them so it is used in this work. All the model weights are reused except those of the logits layer since the number of classes is different in this case.

5.3 Performance

In this chapter, performance of each model is examined with various metrics. Most of the metrics used measure the segmentation quality, which is not the only interesting quantity in this work. Additionally, shard counting accuracy is tested with each model. The accuracy is measured for the whole validation set and individually for 4 mm, 6 mm, and mixed glasses.

First, the best performing DCNN backbone is chosen by training each with same hyperparameters. Next, further hyperparameter optimization is done to the best performing backbone. Sensitivity analysis is also performed to see how the model reacts to variations in image scale.

5.3.1 Evaluation metrics

The following metrics are used during training to evaluate the performance of the model: mean intersection over union (mIOU), precision, recall and shard counting accuracy. Visual inspection of the segmentation masks is also performed to verify the numerical results. Sometimes traditional metrics are unable to express subtle differences in segmentation results, which are clearly visible to a human eye. Each of the used metrics are described in this section.

Most of the used metrics can be explained with the help of the confusion matrix shown in Table 5.2.

Table 5.2. 2x2 confusion matrix.

		True class	
		Positive	Negative
Predicted class	Positive	True Positive (TP)	False Positive (FP)
	Negative	False Negative (FN)	True Negative (TN)

The confusion matrix is often used to evaluate the performance of machine learning or statistical models. Many metrics are a function of the TP, FN, FP and TN values. In this case, the segmentation is a binary problem since there are only two classes. A shard is encoded as value 1 and an edge as value 0. All of the metrics except IOU are calculated with respect to the shard label. For example, recall gives the fraction of correctly predicted shard pixels out of total predicted shard pixels.

Intersection over union (IOU)

IOU is a metric that defines the ratio of area of overlap to area of union between predicted and ground truth pixels. In terms of confusion matrix, it is defined as $IOU = \frac{TP}{TP+FP+FN}$. In semantic segmentation, the IOU metric can be thought as giving a measure of similarity between the predicted and ground truth segmentation masks.

Recall

Recall measure gives the fraction of correctly predicted shard pixels over all true shard pixels and it is defined as $\text{recall} = \frac{TP}{TP+FN}$. Recall is a measure of how much of the true shard pixels the model found. However, recall in its own does not necessarily tell the whole truth since labeling all pixels as shards would result in a perfect recall of 1. Therefore, recall is often accompanied with a precision metric.

Precision

Precision is quite a similar metric as recall, but it gives the fraction of correctly labeled shard pixels out of all predicted shard pixels. It is defined as $\text{precision} = \frac{TP}{TP+FP}$. A precision of 1 would mean that all the pixels labeled as shards by the model are also ground truth shard pixels. In many cases, hyperparameters can be tuned to compromise between precision and recall. For example, if it is important to find every instance of an object, recall can be emphasized more.

Shard counting accuracy and error

One of the goals of the system is to accurately count the number of shards in an image. The accuracy is counted both by utilizing the postprocessing scheme described in Chapter 4.4 and by naïve approach, which means simply counting the number of connected regions in the raw binary segmentation mask. In both approaches, very small regions that can be considered as noise are first removed. Accuracies are tabulated for the whole validation set and individually for 4 mm, 6 mm and mixed glasses.

In addition to overall accuracy, the counting error is also illustrated in several plots. The validation set contains 33 samples. Relative counting errors are plotted separately for each sample and for each model. Mean absolute error (MAE) over the validation set is also plotted for each model.

Inference time

Computational efficiency is often an important property in a machine-learning model. Depending on the application, even real time processing might be required. In this work, the feedforward capacity of the models are tested by taking an average processing time over 100 images. Inference is run for a total of 105 times, but the 5 first are excluded from the average since there might be some unrelated processing in the first few runs.

5.3.2 Validation performance

In this section, model performance is examined for different backbones and hyperparameters. All the results presented are for the validation set, which comprises 15% of the total dataset. The remaining 85% were used for training.

Backbone comparison

First, different model backbones are compared by training the model with different backbones and constant hyperparameters. The parameters used for backbone comparison are presented in Table 5.1. These parameter values are used because they represent the default training protocol utilized by the authors of DeepLab v3+. However, batch size is tuned such that the models are trainable with two GPUs, each with 12 Gb memory. A large total number of steps is used to see if the models overfit to the training data. Note that polynomial learning policy is used so the learning rate gradually degrades to zero during training.

In the following figures, mIOU, precision and recall are plotted as a function of training step for different model backbones. The raw plots are quite noisy so they are smoothed with a moving average filter to clarify the differences between the models.

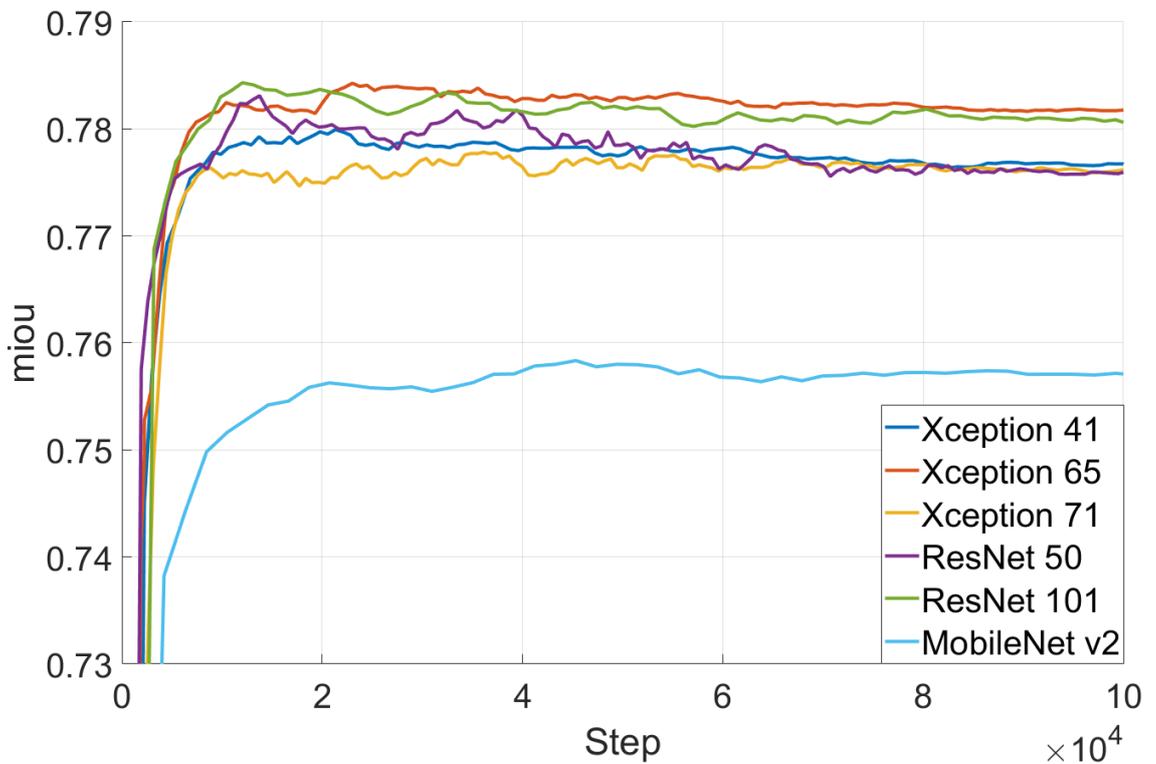


Figure 5.2. Mean intersection over union (mIOU) for different model backbones.

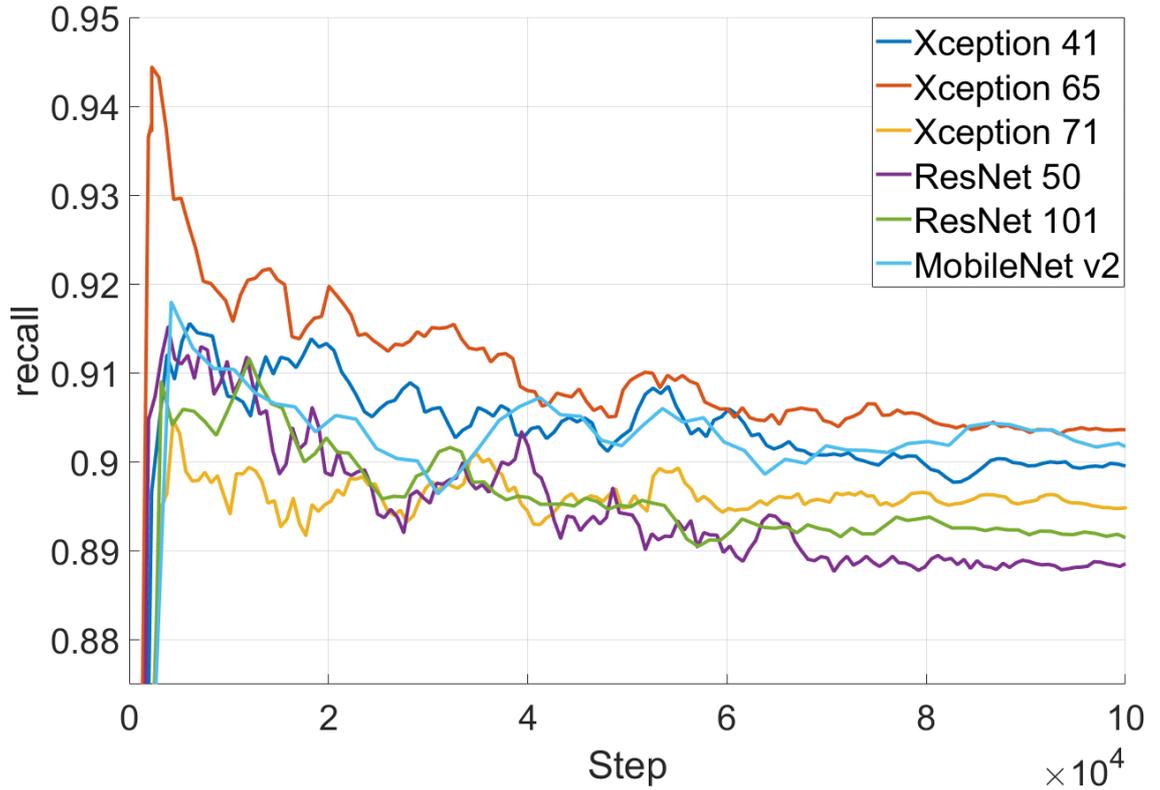


Figure 5.3. Recall for different model backbones.

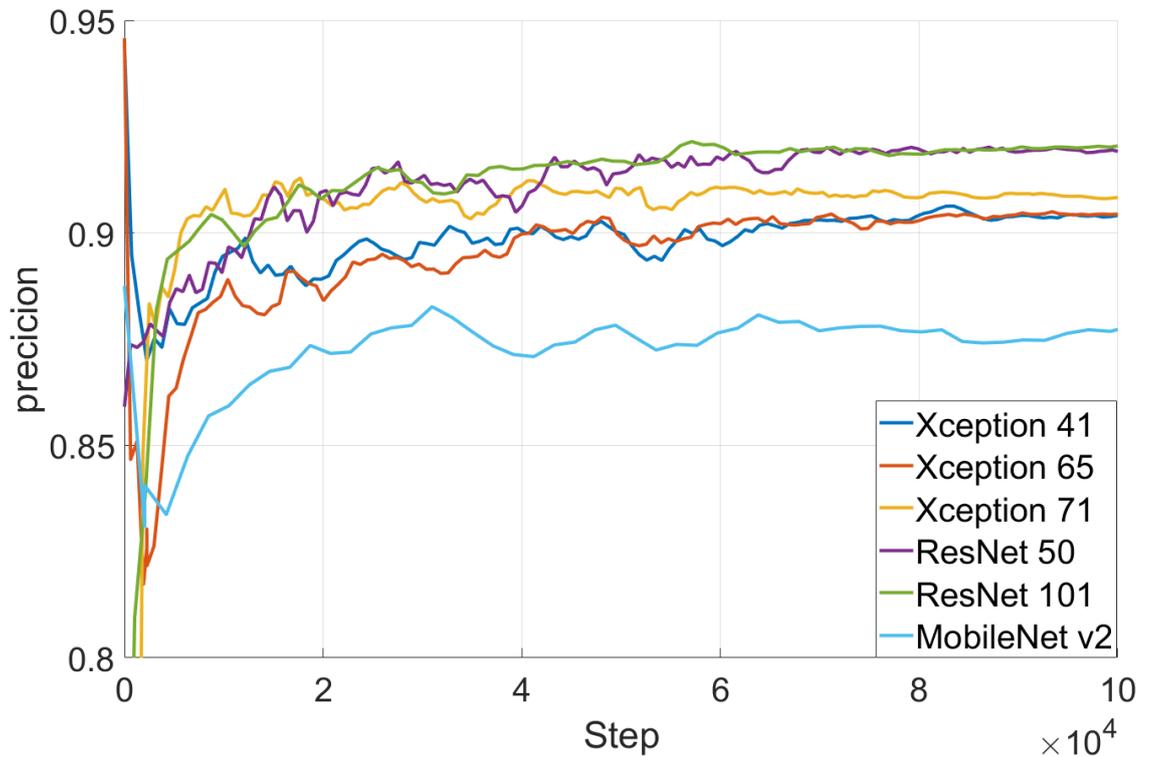


Figure 5.4. Precision for different model backbones.

Figure 5.3 and Figure 5.4 show that the Xception models excel in recall while ResNets are more precise. MobileNet also has high recall, but the low precision indicates that the

segmentation results are coarse. The plots also show that the models show no sign of overfitting even when the number of training steps is as high as 100 000. Therefore, the following shard counts in Table 5.3 and visual comparisons in Figure 5.8 are done with the final model checkpoint after training.

Table 5.3. *Shard counts given by different model backbones with postprocessing (P) or with naïve approach (N). In the naïve approach, shard count is simply the number of connected regions in the binary segmentation mask. Inference times for different models are tabled.*

DCNN / Set	All		4mm		6mm		mixed		Mean inference time
	P	N	P	N	P	N	P	N	
Xception 41	2057	1992	580	551	676	658	801	783	94.5 ms
Xception 65	2058	1995	588	571	680	654	790	770	118.6 ms
Xception 71	2100	2033	593	575	689	670	818	788	135.0 ms
ResNet 50	2108	2028	602	579	694	667	812	782	80.0 ms
ResNet 101	2109	2039	604	586	704	669	801	784	104.2 ms
MobileNet v2	2017	1673	586	438	665	545	766	690	30.4 ms
Ground truth	2115		614		718		783		-

In Table 5.3, shard counts are presented for each model backbone with and without the postprocessing scheme described in Chapter 4.4. In the cases of 4 mm and 6 mm glasses, postprocessing significantly increases the count accuracy. MobileNet seems to especially benefit from it, which is probably due to the coarse segmentation masks, which have a lot of small connections between adjacent shards. The postprocessing effectively removes any such connections. Inference times show that the MobileNet variant is by far the fastest, while Xception and ResNet variants are almost equally fast, both slowing down as the number of layers increases. Inference times are determined using a system with single NVIDIA GeForce GTX 970 GPU.

The shard counts in Table 5.3 are for the whole validation set and therefore might not tell the whole truth, since the model could overpredict the counts for some samples and underpredict for others. Thus, in Figure 5.5, the image-wise errors are plotted for each model.

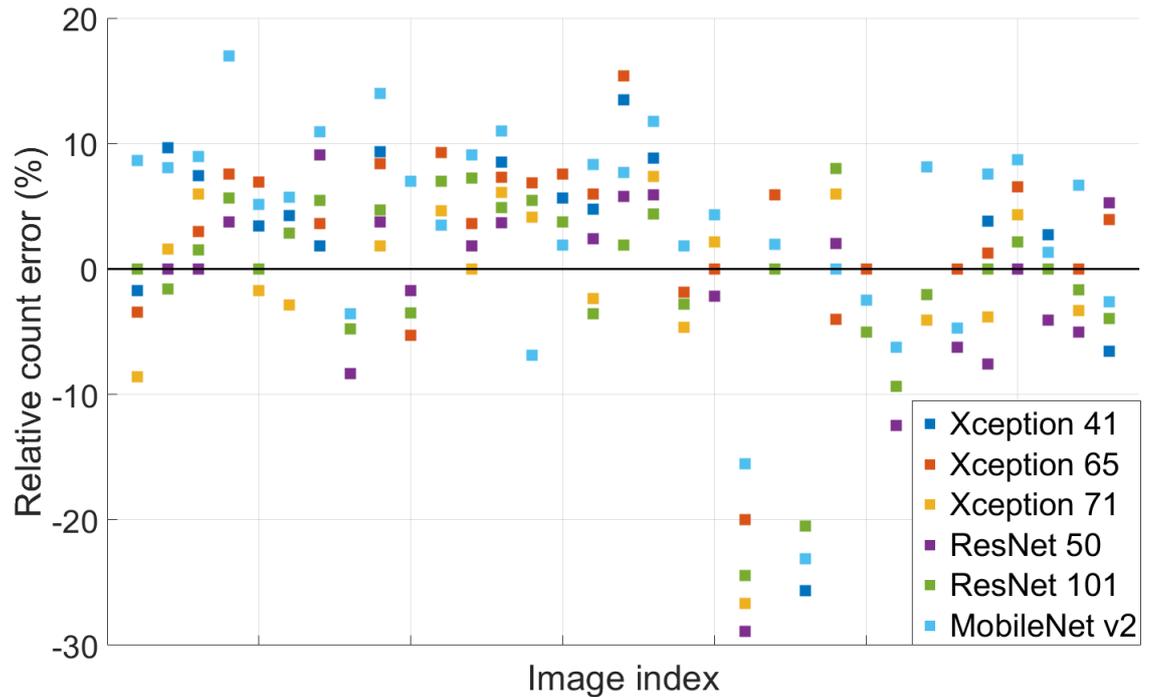


Figure 5.5. Image-wise counting errors for each model variant. The error is calculated as $(\hat{c} - c)/c$, where \hat{c} is the predicted shard count and c is the ground truth count. Positive error implies overprediction while negative error implies underprediction.

In Figure 5.5, the error is calculated as $(\hat{c} - c)/c$, where \hat{c} is the predicted shard count and c is the ground truth count. Thus, positive values correspond to overprediction and negative values to underprediction. It should be noted again that the ground truth counts are not absolute truths, and the “erroneous” count given by the model might well be as correct as the ground truth count, since in many cases an error of 5 % means a difference of only 2-3 shards. The same data can be presented also in the form of box plot or Whisker plot, which is shown in Figure 5.6.

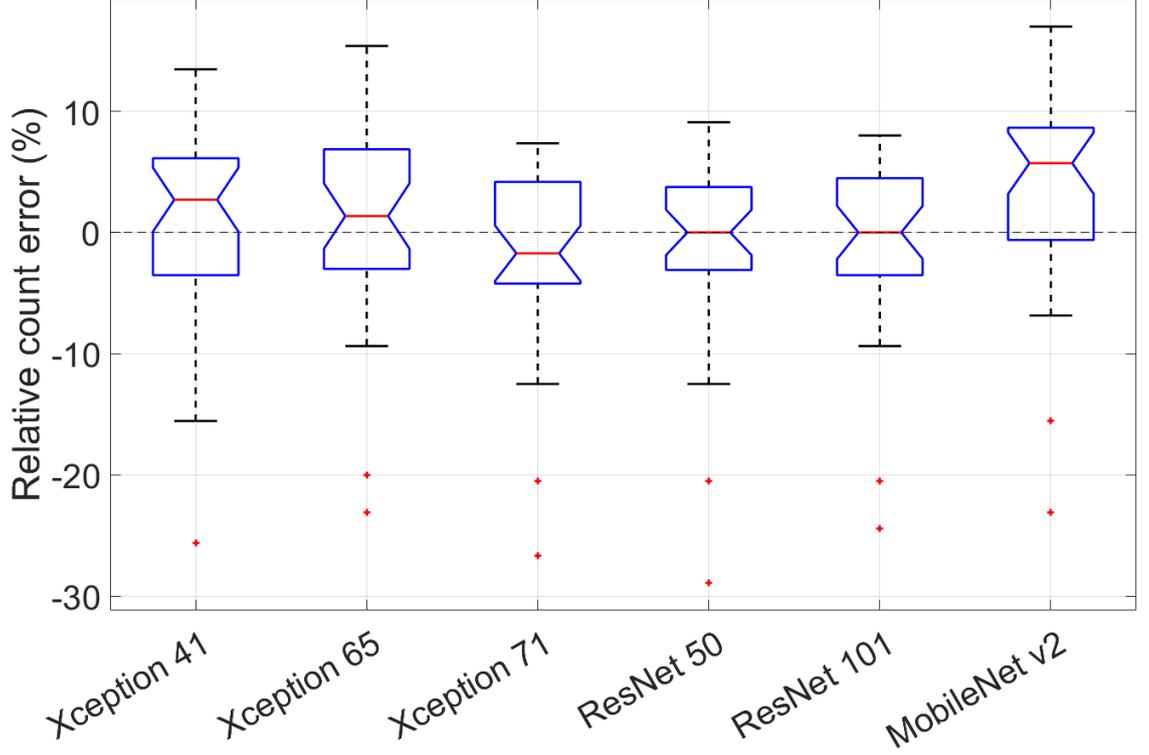


Figure 5.6. Box plot of prediction errors for each model variant computed over 33 validation images. The error is calculated as $(\hat{c} - c)/c$, where \hat{c} is the predicted shard count and c is the ground truth count. Positive error implies overprediction while negative error implies underprediction.

The red bands inside the blue boxes are the medians of the prediction errors. As can be seen, for the ResNet variants the median of the error is zero. The top and bottom edges of the boxes indicate the 25th and 75th percentiles of the data, respectively. The dotted lines extend to the last data points not considered outliers, which in turn are plotted as red ‘+’ symbols. A data point d is considered an outlier if it satisfies the following condition:

$$q_3 + w(q_3 - q_1) < d < q_1 - w(q_3 - q_1), \quad (5.6)$$

where $w = 1.5$ is the Whisker length, and q_1 and q_3 are the 25th and 75th percentiles of the data, respectively. ResNet 101 variant has the lowest total deviance, excluding outliers. The 50-layer variant however has smaller deviance within the q_1 and q_3 percentiles.

In Figure 5.7, mean absolute errors (MAE) over validation samples are plotted for each model. The error is calculated as

$$\text{MAE} = \frac{1}{N} \sum_{i=1}^N \left| \frac{\hat{c}_i - c_i}{c_i} \right|, \quad (5.7)$$

where \hat{c} is the predicted shard count, c is the ground truth count and N is the total number of validation samples.

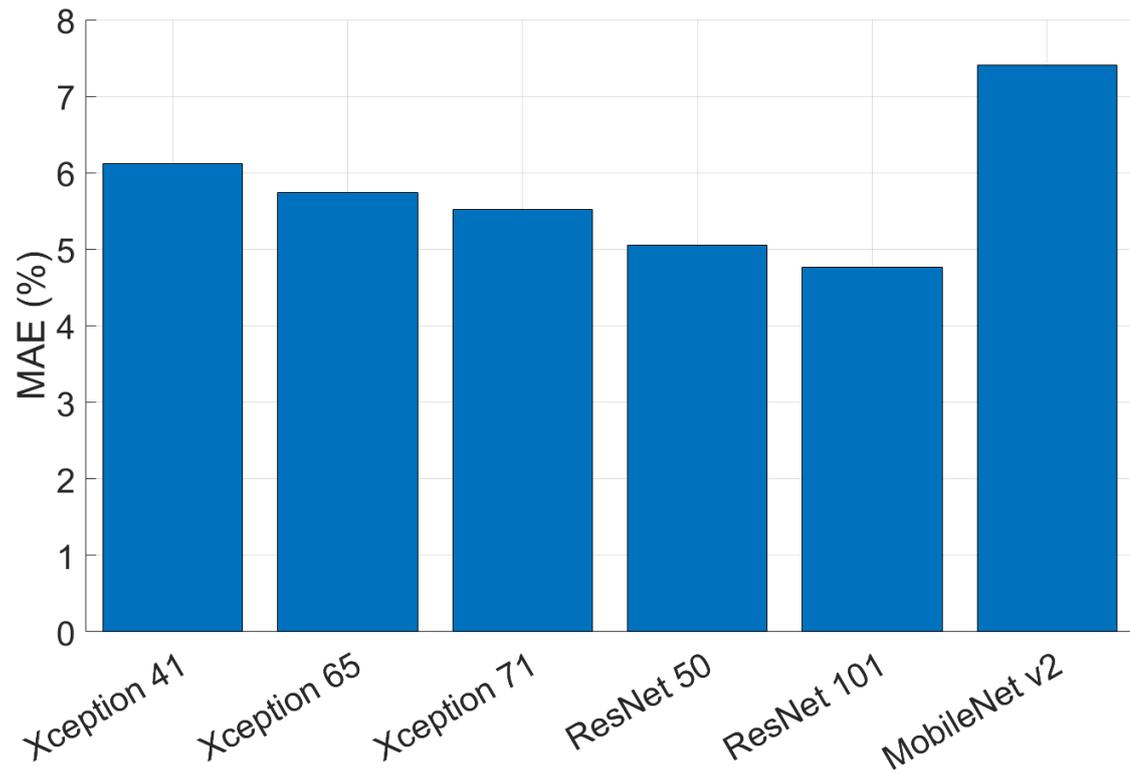


Figure 5.7. Mean absolute prediction error over validation samples for each model.

The bar plot shows that MAE gradually decreases for Xception and ResNet models as the number of layers increases. The 101-layer variant of ResNet reaches the best performance of below 5 % MAE over all validation samples. For ResNet 101, 90 % of the validation samples have an absolute error below 8 %.

In Figure 5.8, a visual comparison of the segmentation results is presented for several samples.

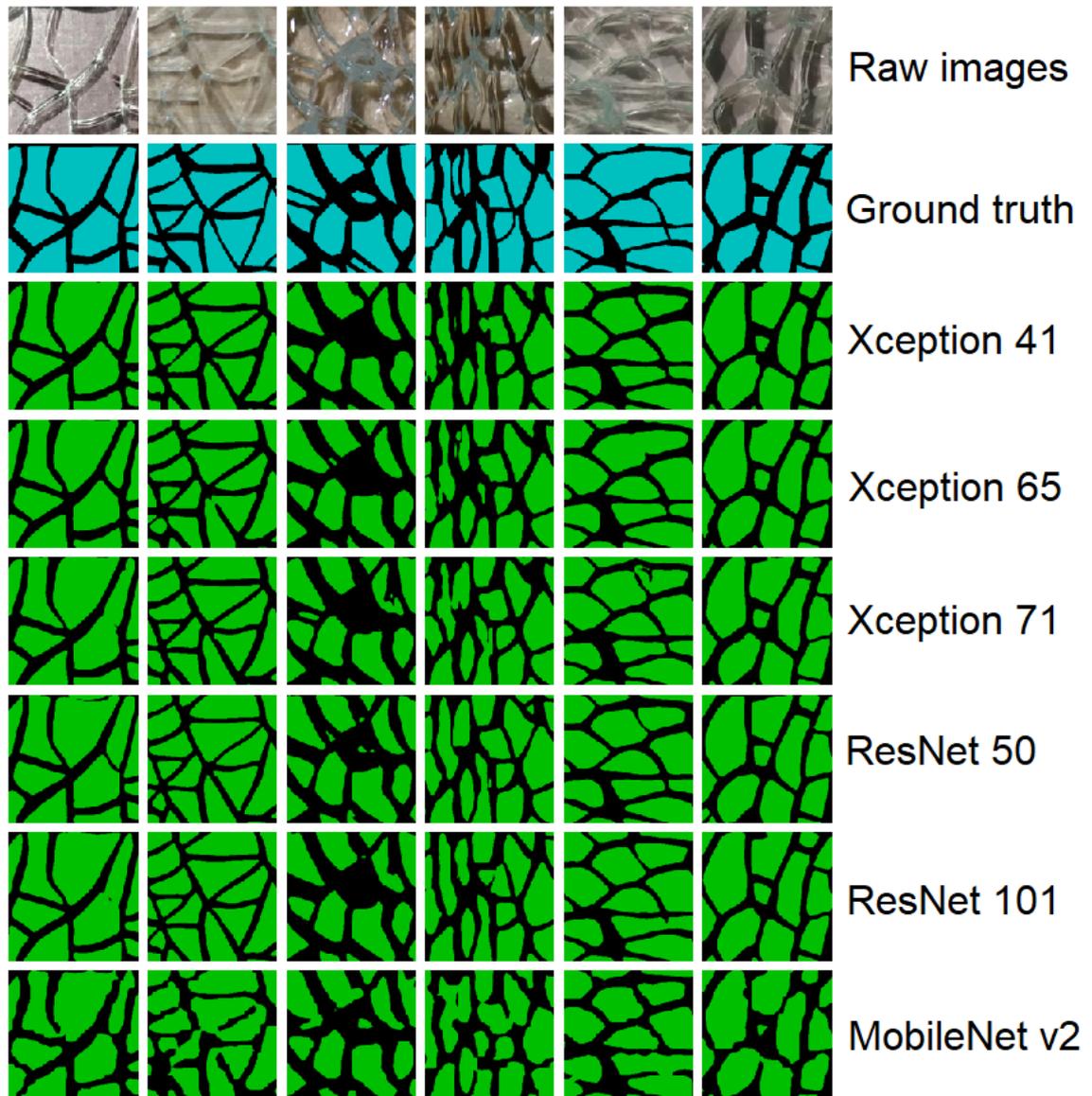


Figure 5.8. Visual comparison of segmentation results for different backbones. First two columns are from the 4 mm set, second two from the 6 mm set and the last two from the mixed set. Images are cropped to 256x256 size to retain readability.

Figure 5.8 verifies that the segmentation results by MobileNet are indeed coarse and many small connections between adjacent shards are present. Results for Xception and ResNet models are quite similar and visual comparison does not offer much information. However, as the recall and precision curves suggest, Xception models are more eager towards labeling pixels as shards. Upon closer visual inspection of the segmentation results, it indeed seems that ResNet models are more reliable at separating each shard. In the Xception results, there are too many connections that the postprocessing is not able to remove, thus the smaller shard counts.

As a conclusion for backbone comparison, ResNet 101 is the best performing model in terms of precision and shard count accuracy. It seems that in this case, precision of the

model should be emphasized more than recall to receive accurate counts. Out of Xception models, the 71-layer variant achieves the best performance.

Hyperparameter optimization

Experiments were done with varying hyperparameters. However, most of the parameters did not noticeably affect the model performance. Output stride is one of the most important parameters, since it determines the amount of image resolution downsampling in the encoder part of the network. Default value for the output stride is 16, and values of 8 and 4 were also tested. Problem is that decreasing the output stride significantly increases the memory requirements of the model. For example, a GPU with 16 GB memory is not able to train the Xception 71 model with *output stride = 4*, *crop size = 513 x 513* and *batch size = 1*. ResNet variants are more memory efficient and they can be trained with the above parameters.

The models were trained for 30 000 steps with a base learning rate of 0.001. A total of 3 GPUs was used, allowing a batch size of 3. Other hyperparameters are the same that were used in backbone comparison and they are presented in Table 5.1. The results obtained for a ResNet 101 model are summarized in Table 5.4.

Table 5.4. *mIOU, Precision and Recall for a ResNet 101 model variant with varying output stride and atrous rates.*

Output stride	Atrous rates	mIOU	Precision	Recall
16	6, 12, 18	0.783	0.897	0.910
8	12, 24, 36	0.779	0.921	0.890
4	24, 48, 72	0.784	0.916	0.897

It is hard to make any conclusions from the results in Table 5.4. The precision of the model seems to be increasing when decreasing the output stride. This is expected, since the model can utilize a higher resolution feature map and less upsampling is required in the decoder part. However, more testing and possibly a larger dataset would be needed to make any definitive conclusions. The default output stride of 16 probably yields a good compromise between accuracy and computational costs since the differences in evaluation metrics are reasonably small.

5.3.3 Sensitivity and generalization

In this chapter, sensitivity and generalization of the model are evaluated. Sensitivity to the scale of the raw image is inspected. Scale is important since a shard can be presented with an arbitrary number of pixels. During training, the samples were scale augmented so small variations should not matter. In Table 5.5, mIOU, precision and recall are shown

for different input image scales. Scaling means that the raw input image and the corresponding ground truth label are bilinearly resized to the new dimensions defined by the scale factor.

Table 5.5. *mIOU, Precision and Recall for a ResNet 101 model variant with varying input image scales.*

Scale	mIOU	Precision	Recall
0.5	0.747	0.862	0.904
0.75	0.775	0.906	0.897
1	0.781	0.920	0.891
1.5	0.775	0.929	0.881
2	0.767	0.933	0.873

In terms of mIOU, unscaled images achieve best results. However, increasing scale increases precision and conversely decreases recall. This implies that the gaps left between adjacent shards are larger when using higher scale, which can be verified from visual segmentation results. Therefore, shard count accuracy might benefit from increasing scale. Some of the performance hit on lower scales might be due to the fact that atrous convolution has to operate beyond image borders as the dimensions are scaled down.

Generalization of the model means how well the model performs on new samples that are somehow different from the samples it has seen before. In this case, it means glasses whose thickness is over 6 mm. Additionally, different backgrounds and varying lightning conditions could be experimented with. Quantitative analysis for different thicknesses and conditions cannot be performed as there is no annotated data available. However, inferring with varying thicknesses show that the model seems to generalize well and segmentation results are quite precise. Problem with thicknesses above 10 mm is that the image must be taken from straight above or else the shard edges mix together and make the counting impossible even for a human.

5.3.4 Summary

To conclude performance analysis, the Xception and ResNet backbones yielded very similar performance. MobileNet v2 variant is up to 4-times faster than the other variants, but it discards some of the key components in the DeepLab v3+ system, namely the ASPP and decoder modules, resulting in very coarse segmentations. Additionally, the speed-up is not important since all of the models are capable of real-time inference when using a reasonably modern GPU.

ResNet outperforms Xception in terms of precision metric, which is important when it comes to shard counting accuracy. This is because precise shard edges are the most important factor when counting the individual shards. Out of the two ResNet models, the 101-layer variant reached marginally better overall count accuracy, as shown in Table

5.3, and a lower total deviance excluding outliers, as shown in Figure 5.5 and Figure 5.6. The ResNet 101 variant reaches below 10 % error with all validation samples excluding the outliers, while for 90 % of the samples the error is below 8 %. Resnet 101 also achieved below 5 % MAE as shown in Figure 5.7. These error rates are quite good given that the annotations are not perfect and each sample contains approximately 50-100 shards. In many cases, it is ambiguous if some shards should be counted or not and the prediction by the model might be as correct as the count given by a human.

Hyperparameter optimization did not provide any conclusive results. It slightly confirmed the hypothesis that lowering output stride increases precision, as can be seen from Table 5.4. It also significantly increases the resource costs of the model, so the trade-off is probably not justifiable, given the negligible increase in performance. The default value *output stride = 16* was found to be a good compromise also in the original work by Chen et al. Other hyperparameters presented in Table 5.1 were also tuned but they did not noticeably affect model performance.

The model is not very sensitive to scale, as can be seen from Table 5.5. Increasing scale increases model precision, which may imply that larger gaps are left between adjacent shards, benefiting the counting accuracy. Model generalizes for glass thicknesses beyond 6 mm, which is the highest thickness in the training data. More data with varying lightning conditions, noisy images and otherwise imperfect conditions would be useful for comprehensive evaluation.

6. CONCLUSIONS

In this work, several approaches for automated shard segmentation were experimented with. The experiments showed that the task is too complicated for simple image processing methods and machine learning approaches must be utilized. State-of-the-art deep learning image segmentation networks proved to excel in the task. The final framework proposed in this work consists of DeepLab v3+ system to perform the segmentation and a simple postprocessing scheme to obtain the shard count.

The framework reached good overall accuracy in terms of segmentation results and shard counts. With the best performing model variant, count accuracy error for all validation samples is below 10 %, while the mean absolute count error over all validation samples is below 5 %. Given the ambiguity of the annotation process and errors in the annotated data, the achieved error rate is very good. It can be concluded that this framework provides a proof-of-concept for an automated shard counting system.

One of the time-consuming parts of this research was the generation of data for the network since each sample had to be manually annotated. However, the model generalizes quite well even with the relatively small dataset. The model would likely benefit from a larger dataset. More data would also allow for a more comprehensive validation. Automated methods for data annotation could be explored in the future to speed up the data generation.

The system heavily relies on the deep convolutional neural network that generates the segmentation mask. Such networks have seen major improvements in the recent years, and the trend may very well continue. Advancements in semantic segmentation could be directly transferred to this application as well. As of finishing this thesis, a version of the model is being field tested at Glaston Finland Oy. The testing provides a continuous flow of data and ideas for further development, which is essential for making this kind of application as robust as possible for daily usage.

REFERENCES

- [1] SFS-EN 12150-1, Glass in building -- Thermally toughened soda lime silicate safety glass. Part 1: Definition and description, SFS, 2016, 69 p.
- [2] M. Rantala, Heat transfer phenomena in float glass heat treatment processes, Tampere University of Technology. Publication 1355, 2015, 135 p. Available: <http://URN.fi/URN:ISBN:978-952-15-3692-2>.
- [3] M. Haldimann, A. Luible, M. Overend, Structural use of glass, International Association for Bridge and Structural Engineering, 2008, 215 p.
- [4] J. Barr, The Glass Tempering Handbook — Understanding the Glass Tempering Process, 2015, 52 p.
- [5] M. Rantala, Reijo karvinen, Heat transfer under an impinging jet at long nozzle-to-surface distances, Annals of the Assembly for International Heat Transfer Conference 13, Begel House Inc., 2006, 12 p.
- [6] R. Karvinen, A. Aronen, M. Rantala, Energy consumption in different types of glass treatment processes, Glass Performance Days GPD 2007, Tampere, Finland, 15-18 June 2007, 2007, pp. 715-717.
- [7] A. Aronen, Energy Consumption of Tempering Furnace, master's thesis, Tampere University of Technology, 2006, 60 p.
- [8] A. Aronen, Modelling of deformations and stresses in glass tempering, dissertation, Tampere University of Technology. Publication 1036, 2012, 76 p. Available: <http://urn.fi/URN:ISBN:978-952-15-2816-3>.
- [9] J.H. Nielsen, Tempered Glass: bolted connections and related problems, Ph.D thesis, Technical University of Denmark (DTU), Civil Engineering, 2009, 78 p.
- [10] N. Pourmoghaddam, J. Schneider, Experimental investigation into the fragment size of tempered glass, Glass Structures & Engineering, Vol. 3, Iss. 2, 2018, pp. 167-181.
- [11] Softsolution Automatic glass fragmentation test for glass | culletScanner - Glass IQ, Softsolution, web page. Available (accessed 18.08.2018): <https://www.glass-iq.com/en/products/culletscanner>.
- [12] S. Matta, Various image segmentation techniques, International Journal of Computer Science and Information Technologies (IJCSIT), Vol. 5, Iss. 6, 2014, pp. 7536-7539.
- [13] A.J. Simpson, Abstract Learning via Demodulation in a Deep Neural Network, CoRR, 2015, Available (accessed 01.06.2018): <http://arxiv.org/abs/1502.04042>.
- [14] D. Kriesel, A brief introduction on neural networks, available at <http://www.dkriesel.com>, 2007, 244 p.

- [15] E.W. Weisstein Convolution, From MathWorld -- A Wolfram Web Resource (accessed 05.06.2018), <http://mathworld.wolfram.com/Convolution.html>.
- [16] V. Nair, G.E. Hinton, Rectified linear units improve restricted boltzmann machines, Proceedings of the 27th international conference on machine learning (ICML-10), Omnipress, Haifa, Israel, pp. 807-814.
- [17] D. Scherer, A. Müller, S. Behnke, Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition, in: Anonymous (ed.), Artificial Neural Networks – ICANN 2010, Springer Berlin Heidelberg, Berlin, Heidelberg, 2010, pp. 92-101.
- [18] J. Long, E. Shelhamer, T. Darrell, Fully convolutional networks for semantic segmentation, 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, pp. 3431-3440.
- [19] N.R. Pal, S.K. Pal, A review on image segmentation techniques, Pattern Recognition, Vol. 26, Iss. 9, 1993, pp. 1277-1294.
- [20] D. Kaur, Y. Kaur, Various image segmentation techniques: a review, International Journal of Computer Science and Mobile Computing, Vol. 3, Iss. 5, 2014, pp. 809-814.
- [21] N. Otsu, A threshold selection method from gray-level histograms, IEEE transactions on systems, man, and cybernetics, Vol. 9, Iss. 1, 1979, pp. 62-66.
- [22] J.B. Roerdink, A. Meijster, The watershed transform: Definitions, algorithms and parallelization strategies, Fundamenta informaticae, Vol. 41, Iss. 1, 2, 2000, pp. 187-228.
- [23] C. Jung, C. Kim, Segmenting clustered nuclei using H-minima transform-based marker extraction and contour parameterization, IEEE transactions on biomedical engineering, Vol. 57, Iss. 10, 2010, pp. 2600-2604.
- [24] T.M. Mitchell, Machine learning, WCB/McGraw-Hill, Boston, Mass. [u.a.], 1997, 432 p.
- [25] Z. Ghahramani, Unsupervised learning, in: Anonymous (ed.), Advanced lectures on machine learning, Springer, 2004, pp. 72-112.
- [26] R. Schapire Machine learning algorithms for classification, Princeton University, web page. Available (accessed 06.06.2018): <https://www.cs.princeton.edu/~schapire/talks/picasso-minicourse.pdf>.
- [27] C.M. Bishop, Pattern recognition and machine learning, 8th ed. Springer, New York, NY, 2009, 738 p.
- [28] T. Sercu, V. Goel, Dense Prediction on Sequences with Time-Dilated Convolutions for Speech Recognition, CoRR, 2016, Available (accessed 4.8.2018): <http://arxiv.org/abs/1611.09288>.

- [29] J. Friedman, T. Hastie, R. Tibshirani, The elements of statistical learning, Springer series in statistics New York, NY, USA:, 2001, 265-267 p.
- [30] E. Jurrus, A.R. Paiva, S. Watanabe, J.R. Anderson, B.W. Jones, R.T. Whitaker, E.M. Jorgensen, R.E. Marc, T. Tasdizen, Detection of neuron membranes in electron microscopy images using a serial neural network architecture, Medical image analysis, Vol. 14, Iss. 6, 2010, pp. 770-783.
- [31] E. Jurrus, S. Watanabe, R.J. Giuly, A.R. Paiva, M.H. Ellisman, E.M. Jorgensen, T. Tasdizen, Semi-automated neuron boundary detection and nonbranching process segmentation in electron microscopy images, Neuroinformatics, Vol. 11, Iss. 1, 2013, pp. 5-29.
- [32] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, Advances in neural information processing systems, Curran Associates, Inc., pp. 1097-1105.
- [33] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, Imagenet large scale visual recognition challenge, International Journal of Computer Vision, Vol. 115, Iss. 3, 2015, pp. 211-252.
- [34] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla & M. Bernstein, ILSVRC 2017, ImageNet, web page. Available (accessed 18.08.2018): <http://image-net.org/challenges/LSVRC/2017/results>.
- [35] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, A.L. Yuille, Semantic Image Segmentation with Deep Convolutional Nets and Fully Connected CRFs, CoRR, 2014, Available (accessed 20.06.2018): <http://arxiv.org/abs/1412.7062>.
- [36] F. Chollet, Xception: Deep Learning with Depthwise Separable Convolutions, 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, pp. 1800-1807.
- [37] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, Deep Residual Learning for Image Recognition, 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, pp. 770-778.
- [38] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, L. Chen, MobileNetV2: Inverted Residuals and Linear Bottlenecks, CoRR, 2018, Available (accessed 20.06.2018): <http://arxiv.org/abs/1801.04381>.
- [39] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, A.L. Yuille, Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs, IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 40, Iss. 4, 2018, pp. 834-848.
- [40] L. Chen, G. Papandreou, F. Schroff, H. Adam, Rethinking atrous convolution for semantic image segmentation, CoRR, 2017, Available (accessed 02.07.2018): <http://arxiv.org/abs/1706.05587>.

- [41] L. Chen, Y. Zhu, G. Papandreou, F. Schroff, H. Adam, Encoder-decoder with atrous separable convolution for semantic image segmentation, CoRR, 2018, Available (accessed 01.07.2018): <http://arxiv.org/abs/1802.02611>.
- [42] P. Krähenbühl, V. Koltun, Efficient inference in fully connected crfs with gaussian edge potentials, Advances in Neural Information Processing Systems 24, Curran Associates, Inc., pp. 109-117.
- [43] F. Yu, V. Koltun, Multi-Scale Context Aggregation by Dilated Convolutions, CoRR, 2015, Available (accessed 22.06.2018): <http://arxiv.org/abs/1511.07122>.
- [44] K. He, X. Zhang, S. Ren, J. Sun, Spatial pyramid pooling in deep convolutional networks for visual recognition, CoRR, 2014, Available (accessed 25.06.2018): <http://arxiv.org/abs/1406.4729>.
- [45] M. Lin, Q. Chen, S. Yan, Network in network, CoRR, 2013, Available (accessed 03.06.2018): <http://arxiv.org/abs/1312.4400>.
- [46] B. Hariharan, P. Arbelaez, R. Girshick, J. Malik, Hypercolumns for object segmentation and fine-grained localization, 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, pp. 447-456.
- [47] C. Szegedy, Wei Liu, Yangqing Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, pp. 1-9.
- [48] A. Krizhevsky, Learning multiple layers of features from tiny images, master's thesis, University of Toronto, 2009, 60 p.