Medical segmentation methods in wood panel defect detection

University of Turku
Department of Computing
Master of Science (Tech) Thesis
Master's in Health Technology
June 2025
Atte Rehnbäck

Supervisors:
Jari Björne
Jani Oikari

UNIVERSITY OF TURKU
Department of Computing

Atte Rehnbäck: Medical segmentation methods in wood panel defect detection

Master of Science (Tech) Thesis, 107 p.
Master's in Health Technology
June 2025

At present many quality inspection tasks in the wood processing industry are performed by production line workers if quality inspection is performed to begin with. At Maler Oy finished wooden panels are inspected and stacked for packaging by workers. This task is highly repetitive and labor intensive.
In this thesis I designed a visual wood panel inspection system based on deep learning-based methods used in medical image segmentation. Deep learning offers a versatile set of detection methods for image processing such as classification, object detection and segmentation. Segmentation has been studied extensively in the medical field and more specifically in the segmentation of tumors from magnetic resonance images. I performed a hyperparameter search in the form of an extensive literacy survey to design a robust training process for the wood feature and defect segmentation model. Tumor and wood feature segmentation are very similar problems since both use images with organic variation and imbalanced data so the results of the hyperparameter survey are transferable to the wood processing industry.
I trained and compared three networks for the wood panel feature segmentation task: U-net, DeepLabV3, and Segformer. To train the networks I collected and annotated over 6200 images of spruce interior panels with a white opaque varnish. The classes included features like cracks, edge clefts, knot holes, healthy and dry knots, pith streaks, and resin pockets. U-net performed the best of the three network architectures, so it was selected for the defect detection stage of the inspection system.
The inspection system grading component is based on contour detection and analysis. Contour analysis offers methods for extracting a feature's dimensions and other geometric properties. Using these methods I developed a system that checks a set of predefined requirements for each wood feature class. I tested the grading system by comparing the grading results produced by the model's test outputs with the results produced by the human made annotations. The model's segmentations produced very similar grading results to the annotations. Therefore, it is possible to design and develop an automated solution for visual wood panel quality inspection using deep learning-based detection.

Keywords: Segmentation, Deep learning, Machine Learning, Industry, Automation, Wood processing, Medical imaging

Contents

List of Figures

List of Tables

1 Introduction

This thesis explores the use of neural networks developed for medical image recognition in an industrial setting. Namely in the wood industry. The goal of this thesis is to develop a system capable of performing quality inspection for a finished wood panel.

The standard quality inspection method on wood panel painting lines is manual. This makes the task of quality inspection on the finished product extremely labor intensive, slow and unreliable. The only way to remedy this bottleneck on the production line is to develop an automated solution to replace the human quality inspector. However, with multiple types of finish applied to wood this automation task becomes very difficult with traditional machine vision approaches. Since the material is also organic, no two defects are identical. These factors make the development of handcrafted detectors extremely difficult. This is why I have chosen to explore the use of artificial neural networks for the quality inspection task in this thesis. With the latest developments in computation technology, it is now possible to run even computationally expensive neural networks in real time making the deployment of neural networks in automation environments possible.

Deep learning-based vision algorithms have been successfully used for determining the quality of industrial products and processes [1] such as steel [2], welds [3], semiconductors [4], fabric [5], and fruit [6]. Existing research in the domain of wood inspection area is rather scarce, so I decided to perform a study to the field of

medical imaging and how deep learning is applied for tumor detection in magnetic resonance images.

The thesis is structured in the following way: the first two chapters introduce the reader to the field on deep learning (2) of and its use in machine vision applications (3).

The fourth chapter studies the use of deep learning-based image segmentation methods in tumor segmentation from magnetic resonance images. The chapter focuses on studying the supervised training parameters of the segmentation models. In this chapter I performed an extensive hyperparameter survey in the form of a literacy survey. And so, the chapter has four research questions: What is the most popular loss function? What is the most popular optimizer? What are the most popular augmentation methods. What is the most popular transfer learning method? To answer these questions, I collected a large set of articles studying tumor segmentation using the U-net architecture. The fifth chapter then briefly discusses existing research of deep learning-based image segmentation in the wood processing industry.

The sixth chapter then uses the results of the hyperparameter survey to train a segmentation model to detect the wood's structural features of interest and defects, such as knots, resin pockets, and pith streaks, from finished wooden panels. The data was collected from a panel painting production line using an automated image capturing system and then annotated by hand. A rule-based quality inspection system is then designed based on the segmentation model and contour analysis. The inspection system is then tested on the test split of the dataset to evaluate the system's usability.

1.1 Motivation

The last stage where the panel's production quality is checked on a wood panel painting line is manual. The production line worker has two primary tasks at this

workstation. The primary task is to stack the incoming panels appropriately for packaging and visual quality inspection is a secondary concern. The worker sits still at their station as the panels move perpendicularly towards them on the conveyor belt. Stacking the panels is a fast-paced process as the maximum pace of the production line is measured at around 100m/min. This leaves very little time for the worker to inspect the quality of the finished product before it has to be stacked and packaged. In addition to the fast production speed, the panels can often be many meters long making it virtually impossible to properly inspect the entire front side of the panel. This leads to a situation where only the most noticeable defects are rejected, provided they get noticed.

In addition to the fast production speed and the non-optimal inspection set-up the repetitive nature of the work can be tiring. The level of performance and motivation of the worker can shift from hour to hour and day to day. Also, the level of scrutiny a worker applies to the quality of the product varies between employees. All these factors lead to a situation where the quality of the product can be inconsistent. This creates a need for an automated solution to perform visual quality inspection on the finished product.

The products have varying quality criteria but the ones I'll be focusing on in this thesis is the healthy knotted spruce panel. Its quality criteria are publicly available on Maler Oy's quality card [7] which is shown in figure 1.1. The goal of this thesis is to design an automated visual inspection system that is able to detect and extract the information required to adhere to the criteria specified in the quality card of figure 1.1.

The long term purpose of the research produced in this thesis is to apply the results in practice. One possible deployment platform for machine vision algorithms and deep learning-based detection methods is Beckhoff's TwinCAT automation environment. The TwinCAT platform offers various functionalities for real time image

MALER

Laatua Viimeistelyyn

CE

Maler Oy Joutsentie 2, 84100 YLIVIESKA
13
DoP: **002DoP2013-07-01**

EN 14915:2006+ AC:2007
Ponttiliitoksellinen massiivi kuusipaneeli
95...120 mm x 14...15 mm

Sisätiloissa käytettävä seinä- ja kattopaneeli

Tiheys ja paksuus	380, 14...15/8
Palokäyttäytyminen	D-s2,d2
Formaldehydin vapautuminen	E1
Pentakloofifenoli-pitoisuus	-
Vesihöyryn läpäisevyys	58
Äänen absorboituminen	0,1 / 0,3
Lämmönjohtavuus	0,107 W/m K
Muut tiedot:	www.maler.fi

LAATUKORTTI

TK-KUUSIPANEELI

- Tuotteessa sallitaan kiinteitä, tuoreen värisiä oksia, joiden koko on enintään ⅓ tuotteen leveydestä, sarvi ja lehtioksat saavat olla puolet tuotteen leveydestä.
- Sallitaan yksi 8 mm:n oksalohkeama kahden metrin matkalla käyttölappeella. Takapinnalla ei rajoituksia.
- Pihkakolot sallitaan vähäisessä määrin, mutta ei läpimeneviä.
- Tuotteessa sallitaan yksittäiset hiushalkeamat, jotka eivät ole läpimeneviä. Läpimenevät halkeamat sallitaan ainoastaan kappaleiden päissä, pituus enintään tuotteen leveyden mittainen. Päätypontatun tuotteen päissä sallitaan kiinni oleva halkeama, jonka pituus on enintään puolet tuotteen leveydestä.
- Lylyä sallitaan sen verran, ettei tuotteen asentaminen sen vaikutuksesta olennaisesti vaikeudu.
- Sydänjuova sallitaan, jos sen pituus on enintään puolet tuotteen pituudesta.
- Tuotteessa ei sallita oksanreikiä, koroja ja kaarnarosoja, sinistymää, värivikaa, lahoa, hyönteisvahinkoja eikä paikkoja. Sormijatkokset ovat sallittuja vain siitä erikseen tuotteessa mainittuna.

Figure 1.1: Healthy knotted spruce quality card of Maler Oy

capturing and processing. The TwinCAT platform also offers options to run neural network inference either synchronously in kernel space or asynchronously in application space. TwinCAT's machine learning server functionality offers the option to run inference on a discrete Nvidia GPU making it possible to apply deep learning-based vision algorithms in real time. TwinCAT's image processing tools include functionalities such as contour analysis tools, perspective transformations, and measurements. These tools make it easier for automation professionals to implement and deploy machine vision applications such as the one designed in this thesis in production line environments with minimal development time.

2 Deep learning

Conventional machine learning models struggle to process natural data in its raw form. These models required the data scientist to engineer a feature extraction process in which the raw data was transformed into an appropriate internal representation from which the machine learning model could then learn and recognize patterns. Deep learning offers us a way to skip the feature engineering phase, since deep learning models are able to discover the relevant features from the raw data [8]. This feature is the reason why I chose to base the defect detection stage of the quality inspection system on deep learning. This chapter explains the core principles on neural networks and their training.

2.1 Artificial neurons

Artificial neural networks as computational models are inspired by the basic function of the human brain. Neural networks consist of neurons organized into layers. A neuron is the smallest learning component within the network and so a neural network is a composition of neurons just like the human brain. [9]

A biological neuron consists of three main types of components: the cell body, dendrites and an axon (figure 2.1). The dendrites are connected to other neurons' axons and deliver electric impulses to the cell body. The axon on the other hand conducts electrical impulses produced by the cell body to other neurons' dendrites [10]. In other words, the dendrites act as the neuron's input channels and the axon

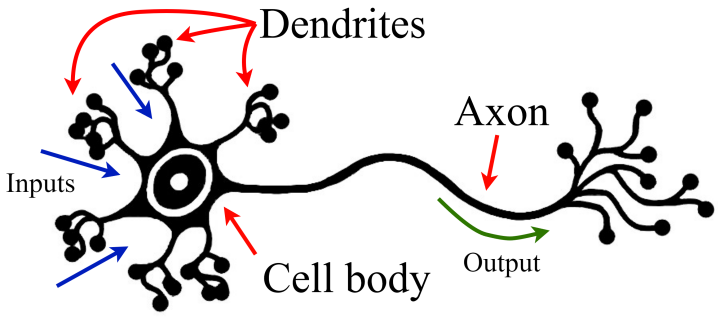

Figure 2.1: Structure of a neuron.

is the output channel. [9]

How a neuron works is relatively simple. If the stimuli arriving from the dendrites to the cell body are strong enough, then the cell body sends an electrical signal along its axon. The axon then conducts the impulse onward to other neurons. The neuron therefore acts analogously to an on-off switch. [9]

A neuron can then be thought of as an application of logistic regression. An artificial neuron maps a set of inputs and a single output. The process can be roughly divided into two steps: calculating the weighed sum with a bias term and then giving the weighed sum into a nonlinear activation function that then gives the activation value of the neuron (figure 2.2). [9]

The connections to previous neurons are portrayed by the weight/coefficient vector and the action potential of the neuron is portrayed by output of the linear model passed through the activation function. [9]

One way to try and understand logistic regression and neurons is through the dot product. Given a weight vector w and an input vector x the weighed sum is essentially the dot product of these vectors w*x. When the angle between these vectors is less than 90 degrees the sign of the dot product is positive and negative when the angle is greater than 90 degrees. When accounting for the factor that

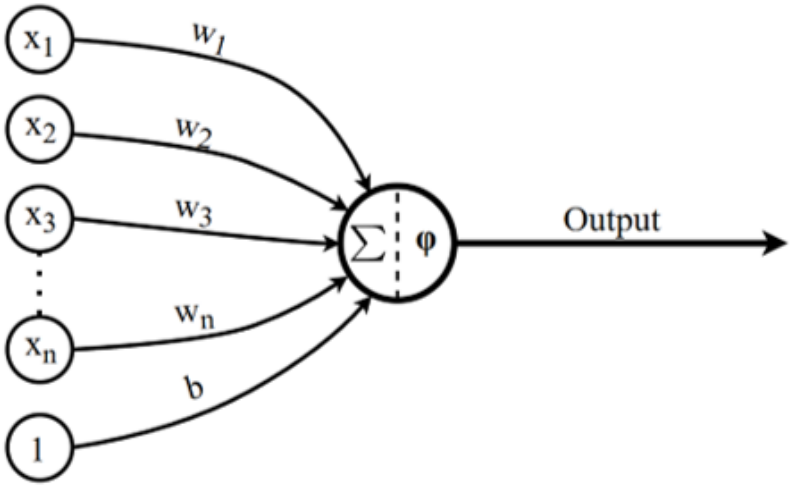

Figure 2.2: A mathematical representation of a neuron.

usually the threshold for the activation function of the neuron is zero then the weight vector defines an activation space for the neuron. The bias term when added to the dot product moves the activation space's edge in the direction of the weight vector. If the bias term is positive, then the edge moves to the opposite direction of the weight vector and vice versa when the bias is negative. The bias term therefore limits or expands the activation space depending on its sign. [9]

Activation functions are nonlinear functions that are essentially used for introducing a nonlinear characteristic to a neuron's behavior. With neural networks we wish to model nonlinear phenomena and without adding an activation function to a neuron this would not be possible. Without activation functions neural networks would become linear models and the composite structure would be redundant. The activation function acts as a switch for the neuron determining whether the neuron will activate or not. It takes the scalar value of the dot product as an input and produces an output known as the activation value. A threshold function is a simple example of an activation function and is often also referred to as the unit step. A threshold function that has zero as its threshold is defined $f(x) = \begin{cases} 0, & \text{if } x < 0. \\ 1, & \text{if } x >= 0. \end{cases}$ and the shape of the function resembles that of a step. [9]

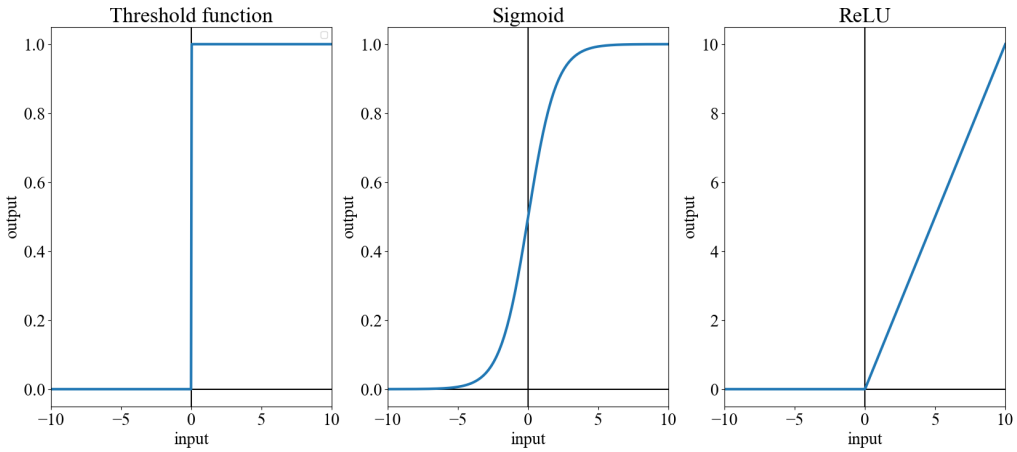

Figure 2.3: Three activation functions where the threshold is at 0. Threshold function: $f(x) = \begin{cases} 0, & \text{if } x < 0 \\ 1, & \text{if } x >= 0 \end{cases}$. Sigmoid function: $f(x) = \frac{1}{1+e^{-x}}$ and the ReLU function is defined $f(x) = max(0, x)$.

2.2 Neural networks

Neural networks consist of neurons. Each neuron in the network defines a simple nonlinear function mapping an array of inputs into a single scalar output. These neurons are organized into layers and these layers are stacked to form a network-like structure. In this network a neuron is depicted as a vertex and connections between neurons are marked by edges. An artificial neural network has at least two layers, an input layer and an output layer [9]. A network with multiple layers is also called a multilayer perceptron or MLP [11]. The input layer is essentially the network's input vector and doesn't contain any artificial neurons. An artificial neural network can also have layers between the input and output layers. These are called hidden layers. The number of hidden layers required for a network to qualify as a deep neural network is two. [9]

The neurons are connected to each other much like neurons in our brain but where biological neurons are connected by their dendrites and axons artificial neurons are connected by their corresponding weights and outputs. In a fully connected layer, every neuron receives an input vector that consists of every output of the previous

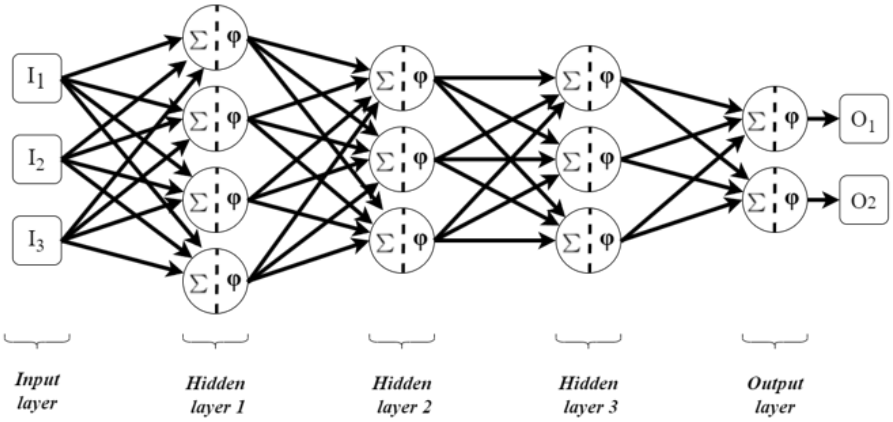

Figure 2.4: A feed forward neural network or MLP. Square is an input value, arrow is a connection between neurons, circle is an individual neuron. Each connection between neurons has a corresponding weight in the neuron in the receiving end of the connection.

layer's neurons. This is how every weight of a neuron corresponds to a single output value of the previous layer. If layer l_n has an output vector $y_n = [y_1, y_2, y_3, \ldots, y_m]$ then the output $y_{n+1,x}$ of neuron x in layer n+1 is calculated by passing the output vector of the previous layer as the input vector to the neuron and so $y_{n+1,x} = a(w_{n+1,x} * y_n + bias)$. [9]

Neural networks and deep learning in general are a form of representation learning. Representation learning is a form of learning where a machine learns representations from the raw data needed for the detection or classification task. At each network layer the abstraction level of the representation increases. [8]

In feed forward networks information flows in only one direction and there are no feedback loops within the network. In this network structure outputs of a layer can be passed as input only to following layers. Despite their name and their structure being largely inspired by its biological counterpart, the goal of neural networks is not to model the brain [11]. Instead, deep networks can rather be thought of as universal function approximation machines. In fact the universal approximation theory by Hornik et. al. [12] states that a network with a linear output layer and a hidden layer with a "flattening" nonlinear activation function can approximate

any Borel measurable function with any nonzero error. The function must be a mapping from a space with finite dimensionality to another and the network must have enough units to achieve the desired error. In theory this means that a network can map any relation between a set of inputs and outputs with arbitrary accuracy provided the network is large enough. This gives deep learning boundless potential in regard to what it can do. However, finding the desired relation between inputs and outputs in the form of a set of network weights is not a straightforward process as we will see in the next section.

2.3 Neural network training

Finding an optimal set of network weights is essentially an optimization problem. The goal of the process is to optimize the network weights in regard to the network's performance on new examples. The current method of network optimization is an iterative training process where the network is used to predict outputs based on a set of training data. Then the network weights are iteratively adjusted to minimize the difference between the predicted and desired outputs. [13]

The tools required for the training process include a loss function, backpropagation, an optimization algorithm, and an optional set of regularization methods. Since our learning problems in both medical and wood industry domains include a predefined set of classes of interest, this section describes how a neural network is trained using these tools in a supervised learning process.

2.3.1 Dataset splits

A machine learning model or in our case a neural network is trained by optimizing its performance on a set of training data. In this thesis the method of training is supervised so the network's optimization is with regard to some predefined set

of correct outputs. However, the network's performance on the data it is trained on is not what we are interested in. Instead, our primary concern is the model's performance on new data that the model has not been trained on [14]. A model's ability to perform well on new data is called generalization [13].

A simple way to test the model on new data is to hold out some of the data you have for a separate portion called a validation set. This validation set can then be excluded from the training process to obtain information on the model's generalization ability [14].

However, in many cases we want to also optimize some training parameters in a way that maximizes validation set performance. Unfortunately, this requires the training process to peek at the validation data to guide the training process. This makes the validation set to become part of the training process which if unaddressed will invalidate the test results. Therefore, we need a third set of data on which we will test our model to obtain final evaluation metrics. This set is called the test set [15].

Splitting the dataset into three separate parts results in a new problem. It limits the amount of data we can use for training and testing the model. K-fold cross-validation is a solution aimed to relieving this problem. In k-fold cross-validation dataset is split into k number of equisized parts. These parts then take turns in serving as the test set while the rest are used for training. The model's test scores on different partitions of the data are combined at the end of the training process. This way the whole dataset can be used for obtaining test results of the training process [14].

2.3.2 Loss functions

In a supervised learning process, we aim to produce a model that has captured the relation between the input examples and the desired outputs. If the learning

process has been performed successfully the model should perform well on various performance measures. Sometimes optimizing the model's performance directly on a given performance metric is unfeasible. For example, we might wish to maximize recall, precision or accuracy of a classification model at a given task but directly maximizing these metrics might not be achievable efficiently. Instead of trying to maximize model performance on a given performance metric the more efficient way to go about it is to try and minimize the error between the predicted and correct outputs. In fact, using a surrogate loss function instead of a correct-incorrect loss function produces better class separation and produces in a more robust classifier. [13], [16]

Some notable loss functions used for measuring neural network output include cross entropy for classification and Dice loss for segmentation [16]. Cross entropy is defined as

$$L_{CE}(y, \hat{y}) = -\sum_{c=1}^{C} y * \log \hat{y} \tag{2.1}$$

And Dice loss as

$$L_D(y, \hat{y}) = 1 - \frac{2(y * \hat{y}) + \epsilon}{y + \hat{y} + \epsilon} \tag{2.2}$$

2.3.3 Backpropagation

After the error of the network has been calculated, we need to find the relation between it and network weights. This relation is called the error gradient of the network $\frac{\delta E}{\delta W}$. The gradient is needed by the optimization algorithm to make adjustments to the network weights that decrease the network's loss. The process used to find the error gradient is called backpropagation and this section describes how it works.

The backpropagation technique was first used for finding the network error gradient in 1986 by Hinton et. al. [17]. The training of the neural network begins by performing a forward pass. The forward pass is performed by presenting the network

with an input example and computing the output. During the forward pass both the weighted sum values and activation values of the network's units are recorded for the backward pass phase.

Once the forward pass phase has been completed the error of the network must be calculated. For example, the error can be calculated with the mean squared error: $E = \frac{1}{2}\sum_{i}^{n}(y_i - \hat{y}_i)^2$. Where n is the number of output neurons, y_i and $\hat{y}_i$ are the actual and desired outputs of unit i respectively. Once the error has been calculated it will have to be backpropagated through the network. This is performed by using the chain rule: $\frac{df(g(x))}{dx} = f'(g(x))g'(x)$. However, in this context the Leibnitz's notation is more intuitive to understand: $\frac{df}{dx} = \frac{df}{dg}\frac{dg}{dx}$.

By using the chain rule, we can link the error first from the network outputs to the network output activations and then from the activations to the layer weights: $\frac{\delta E}{\delta w_k} = \frac{\delta E}{\delta Y_k}\frac{\delta Y_k}{\delta z_k}\frac{\delta z_k}{\delta w_k}$. Where k is the number of layers on top of the input layer. Y_k is the output of the network or the activations of the output layer k, z_k is the weighted sum vector of the layer k, and w_k is the weight matrix of the layer k. Here the bias term has been embedded into the weight vector making it a normal weight always receiving an input 1 allowing us to omit the bias vector from the equations. In order to propagate the error gradient back to the next layer we need to substitute the $\frac{\delta z_k}{\delta w_k}$ term with $\frac{\delta z_k}{\delta Y_{k-1}}$. So, we can get $\frac{\delta E}{\delta Y_{k-1}}$ which is needed to calculate error gradient for the weight matrix of layer $k-1$.

$\frac{\delta E}{\delta Y_{k-1}} = \frac{\delta E}{\delta Y_k}\frac{\delta Y_k}{\delta z_k}\frac{\delta z_k}{\delta Y_{k-1}}$ and then $\frac{\delta E}{\delta w_{k-1}} = \frac{\delta E}{\delta Y_{k-1}}\frac{\delta Y_{k-1}}{\delta z_{k-1}}\frac{\delta z_{k-1}}{\delta w_{k-1}}$. This procedure essentially consists of differentiating each operation's output in the network in regard to its inputs starting from the output layer until the input layer is reached.

In the example where the mean squared error is used as the loss function the error gradient in regard to the network's outputs becomes $\frac{\delta E}{\delta Y_k} = \frac{2}{k}(Y_k - \hat{Y})$. And in the case where the activation function that produces Y_k is the sigmoid ($\sigma(x) = \frac{1}{1-e^{-x}}$) function the derivative of Y_k with respect to its input z_k is simply the derivative of

the activation function: $\frac{\delta Y_k}{\delta z_k} = \sigma'(z_k) = \frac{d\sigma(z_k)}{dz_k} = \sigma(z_k)(1-\sigma(z_k))$. This only leaves us with the $\frac{\delta z_k}{\delta w_k}$ term. Since $z_k = Y_{k-1}w_k$ where each weight is only used once to multiply its corresponding input from Y_{k-1} then the gradient for each neuron in layer k is simply the vector Y_{k-1}. So $\frac{\delta E}{\delta w_k} = \frac{\delta E}{\delta Y_k}\frac{\delta Y_k}{\delta z_k}\frac{\delta z_k}{\delta w_k} = \frac{2}{n}(Y_k - \hat{Y})\sigma(z_k)(1-\sigma(z_k))Y_{k-1}^T$.

After computing the gradient for the weights in the layer k we need to compute the gradient with respect to the layer's inputs. This is done by replacing the Y_{k-1} vector with the weight matrix w_k since this time z_k derivated with respect to Y_{k-1} instead of w_k. This process is repeated until the error gradient has been calculated for each layer.

The backpropagation algorithm was proposed for training neural networks in 1986 by Rumelhart, Hinton, and Williams. They note that this method only works with networks that have connections going forward in the network and that backwards connections are forbidden. While connections going backwards in the network are forbidden connections going forward in the network can skip succeeding layers. The creators note that this method is not guaranteed to find a global minimum within the weight space. However, they also note that the optimization process rarely gets stuck in local minimum that is far worse than the global minimum. Additionally, they suggest that adding connections to the network can create additional paths to local minima close to the global minimum in the weights space. [17]

2.3.4 Gradient descent

After we have found the error gradient for the network, we need to change the network's weights in a way that reduces the error of the network. This is done with gradient based optimization.

In a case where the error of the system is calculated by some function $f(x)$ the derivative of this function is denoted as $f'(x)$. The derivative specifies the slope of the function at point x. It tells us how a small adjustment in the input affects the

output. The error gradient essentially tells us how we should modify the network weights to increase the error. This means that we can reduce the network error by changing the network weights to the opposite direction of the gradient. Since the optimization algorithm follows the error gradient to the perceived downhill, we call this algorithm gradient descent. [13]

In our case, a gradient consists of partial derivatives of a multidimensional function $f : R^n \rightarrow R$. A partial derivative of f is denoted as $\nabla_{x_i} f = \frac{\delta}{\delta x_i} f(x)$. The gradient tells how changing the value of x_i changes the value of f. The partial derivatives are stored in a vector that is then denoted as $\nabla_x f(x)$. To minimize f we need to find the direction in which we move x that decreases the value of f the fastest. When going to the direction of fastest descent we use the gradient to propose a new point: $x' = x - \epsilon \nabla_x f(x)$ where ϵ is the learning rate of the algorithm or the length of step we take to the direction of the slope. The value of ϵ is a simple scalar. [11], [13]

Stochastic gradient descent

When we optimize deep learning models, we use an extension of gradient descent known as stochastic gradient descent. In contrast to normal gradient descent, stochastic gradient descent takes only a small portion of the dataset for calculating a single step. In normal gradient descent the error is calculated by summing the errors produced by all training set examples[13]. In older material the term stochastic gradient descent is synonymous with online learning where the error gradient is calculated from the error produced on singular data points. [11]

With large datasets it is advantageous to use stochastic gradient descent since calculation time per step becomes significantly shorter. In Stochastic gradient descent (or SGD) the training data is sampled into minibatches. These minibatches are non-overlapping subsets of the training set with only a few examples. The size

of minibatch is usually chosen to be relatively low regardless of the dataset size. Usually, the minibatch size is in the range of tens to a few hundred. [13]

Momentum

Stochastic gradient descent can sometimes be a slow method of training neural networks. Adding momentum to the update step calculation process can speed up training significantly in situations featuring high curvature, weak but stable gradients, and noisy gradients for instance. Momentum is essentially an algorithm that accumulates a decaying moving average of past gradients. This method introduces a new variable called velocity that determines the direction and the update step length. The velocity variable is the decaying moving average of past and present negative gradients. [1] The idea behind adding a momentum term to the update rule is to use the real-world analogy the Newton's first law of inertia in moving the weight vector through the parameter space. If the update to the weight matrix in gradient descent is expressed as $\Delta x_t = -\epsilon \nabla_x f(x_t)$ then with the use of momentum the update to the weight becomes:

$$\Delta x_t = -\epsilon \nabla_x f(x_t) + p \Delta x_{t-1} \tag{2.3}$$

The idea of using momentum in the update rule is to preserve some amount of velocity from earlier gradients. From the equations we can see that when the new update step is calculated we first multiply the old velocity variable with multiplier p. the multiplier p is the rate of decay of the old gradients and determines how much the old gradients are allowed to affect the new update step. This can be thought of as the momentum carried over from the old gradients. After applying decay to the old gradients, we add the negative gradient multiplied by the learning rate ϵ to the momentum. Now that the new velocity term has been calculated it is used for updating the weights. [18]

Adaptive learning rate optimizers

An adaptive learning rate algorithm uses individual adaptive learning rates to optimize each weight in the network. Using adaptive learning rates is beneficial for the learning process since learning rate is a difficult hyperparameter to choose. The difficulty of learning rate choice stems from the high impact learning rate has on model performance. Making the learning rate adaptive instead of static eliminates the need to choose an appropriate learning rate. Making the learning rate adaptive for every network parameter allows for steps of different length depending on how sensitive the network error is to changes in individual weights. Therefore, the learning rate can be kept higher for weights having slight gradient slopes and lower for steeper slopes. [13]

AdaGrad is an adaptive learning rate optimizer that scales the learning rates inversely proportional to the square root of the sum of squared historical values of the gradient. This means that parameters having the most impact to the network error have their learning rate decrease the fastest and parameters having little effect on the error have slower learning rate degradation. The AdaGrad algorithm works well in convex areas of the error function surface. [19]

However, in most cases the learning process involves the optimizer to pass the parameter vector through nonconvex areas where the decrease of the learning rate could occur too quickly resulting in the training algorithm never reaching a local convex area. This is where The RMSProp algorithm improves upon the AdaGrad by adding decay to the impact past gradients have on the learning rates. RMSProp uses a decaying weighted moving average of gradients to adjust the learning rate. With this implementation the learning rate decreases when the gradient for a weight steepens and increases when the gradient diminishes. [20]

Adam is a combination of RMSProp and AdaGrad. Adam stands for adaptive moment estimation and this optimization technique was published in 2015 by

Kingma and Ba [21]. The Adam algorithm uses both first and second order moments of the gradients to find a good learning rate for individual weights. This approach aims to combine the advantages of RMSProp and AdaGrad.

2.3.5 Batch normalization

Batch normalization itself isn't an optimization algorithm but rather a way to ease the work of actual optimization algorithms. It is a form of adaptive reparameterization where the connections between neurons are standardized.

Deep neural networks consist of a large number of functions that are composed together to form the overall network. When a neural network is trained the training algorithm makes adjustments to individual weights and simultaneously changes the way the neural network's subfunctions operate. The problem with the training algorithm is that it changes each weight while assuming that the rest of the network remains unchanged. This is not the case in practice, however. The entire network is updated at once and this can have unforeseen effects on the overall network's operation even though the individual changes to the weights may have been small. This makes choosing an appropriate learning rate a challenging affair. Tackling the problem by taking all the effects into account would be computationally expensive and would still require approximations that might not be able to take all interactions into account.

Batch normalization aims to relieve this problem by reparametrizing the connections between neural network's layers. This alleviates the need to coordinate the weight updates across layers. This technique can be used for standardizing the input to any layer in a neural network.

Let A be a two-dimensional matrix containing the activations of a layer for a given example in each row. Matrix A contains the activations of a given layer for a single minibatch of input examples. A is normalized by subtracting a mean vector μ

containing the mean activations for every unit in the layer. The subtracted matrix is then divided by the standard deviation vector σ containing the standard deviations for each neuron. The normalized minibatch output matrix for the layer is then: $A' = \frac{A - \mu}{\sigma}$.

After the minibatch output matrix has been normalized it is fed as input to the subsequent layer in the network. Because the mean and standard deviation values are calculated over the activations for a minibatch this technique cannot be used as is when we wish to use the network on individual examples. The mean and standard deviation values are stored during training and then used in validation and test phases. $\mu = \frac{1}{m}\sum_i A_i$ and $\sigma = \sqrt{\delta + \frac{1}{m}\sum_i (A - \mu)_i^2}$, where m is the number of examples in the minibatch and δ is a small positive value close to zero. The δ term is introduced to avoid an undefined gradient $\sqrt{z}$ at $z = 0$.

With batch normalization the layers of the neural network lose some of their expressive ability as their means are now zero and have a standard deviation of one. The expressiveness of a standardized layer can be retained by giving it a new multiplier γ and a bias β, so the output of the batch normalization procedure is $\gamma A' + \beta$. These parameters are learned during the training process, and they allow the neuron outputs to have any mean and standard deviation.

At first glance, first removing the mean and standard deviation from a unit and then reintroducing them seems redundant. In fact, a layer without batch normalization and a layer using batch normalization can represent the same family of functions. However, they learn in a different manner. Without batch normalization the mean and variance of a neuron's output are determined by the complex interactions between the weights of the neuron and the weights of the layers preceding it. With batch normalization the mean and standard deviation of a neuron are now learned separately from the neuron's weights and this new arrangement is much easier to optimize with gradient descent.

When using batch normalization with convolutional layers the same standard deviation and mean values must be used to standardize the whole feature map. Otherwise, the statistical properties of the feature map will vary depending on spatial location. In their research article Ioffe and Szegedy demonstrate that by simply adding the batch normalization layers to the network the required training time for convergence is reduced to a fraction of the previous training time. In addition to sped up training they were able to improve the network's classification performance on ImageNet data. Batch normalization was also observed to enable the use of higher learning rates. [22]

2.3.6 Transfer learning

Transfer learning can be used in situations where we can use something previously learned in a new setting. In transfer learning we assume that some factors explaining variations in setting A remain relevant in setting B as well. Much like learning to operate a motorcycle clutch after learning to use a clutch on a car. The principle remains the same, but the method of operation has changed. [23]

In a supervised learning setting the nature of the output is usually the changing element in transfer learning. For example, in image processing we might face a very specific problem where we only have a very limited number of annotated examples available for training. By using transfer learning, we can first train the network in a very broad setting A where a large dataset is available. Then the network is retrained in the specific setting B. The representations learned in setting A might help in learning relevant representations in setting B. For example, in image recognition many categories share low level features such as edges, textures, lighting, shadows, etc. In a setting where we want to be able to tell whether the image contains a wolf or a tiger, we could be very limited in terms of available images of the two. We can therefore use a larger dataset containing cats and dogs to pretrain the model

to differentiate between the two and then retrain it to do the same on wolves and tigers. [23]

When transfer learning is used in image processing it usually involves training the main part of the network to detect basic features on a large general-purpose dataset and then retraining the last layers to perform a specific task. An example of such a general-purpose dataset would be the ImageNet dataset.[16]

2.3.7 Regularization

In his paper [24] Bishop describes regularization as limiting the degrees of freedom of a model. He also classifies regularization to be a complexity control method of a model in addition to architecture design, early stopping and noise injection. However, in the deep learning book [13] regularization is characterized as a collection of methods that aim to improve a model's generalization performance at the possible expense of increased training error. In this section, I will explain two regularization methods used in this thesis: augmentation and early stopping.

Here controlling the complexity of a model means limiting the model's capacity. Model capacity can be thought of as the size of the set of possible functions the model is capable of representing. Regularization often aims to restrict a model's capacity to avoid over-fitting. Over-fitting occurs when the model starts modeling the noise of the training data while the generalization error increases. [13]

Augmentation

Training a deep learning model with more data leads to better generalization. However, the amount of data available is often limited due to various reasons. The data might be difficult to annotate or scarce and in medical applications it is usually both. The easiest way to fix this problem would be to somehow increase the amount of available data [25] .

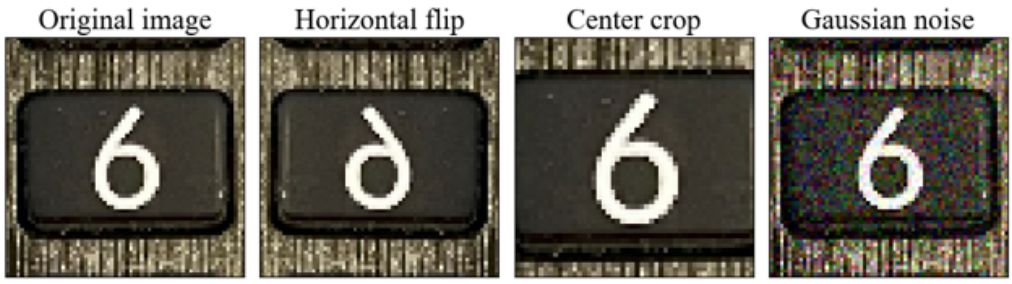

Figure 2.5: A demonstration of augmentation methods in the context of image processing. Horizontal flip, center crop, and Gaussian noise.

One way to increase the amount of data is to perform data augmentation to the training data. With data augmentation we artificially create more data out of already existing data points. For example, in classification tasks data augmentation is performed on the input values and the original label is retained. The augmentation operations must be selected carefully so that the labels do not change or otherwise the labels will also have to be changed accordingly. For instance, when classifying drawn numbers, we might want the classifier to be invariant to slight rotations, but the amount of rotation must be regulated accordingly. If the amount of rotation is allowed to be up to 180° then we will encounter situations where $6 \rightarrow 9$ and vice versa. [25], [26]

Since the augmentation operations introduce different types of distortions and translation to the data this improves the network's translation tolerance. When augmenting images, we might use multiple different translation techniques randomly. These include scaling, zooming, horizontal and vertical flips, etc. Neural networks are also quite sensitive to noise and so adding random noise to the training data helps to improve the network's operation with noisy inputs. [13]

Early stopping

Neural networks usually start to over-fit to the training data at some point during the training cycle. Over-fitting starts when the validation error stops decreasing

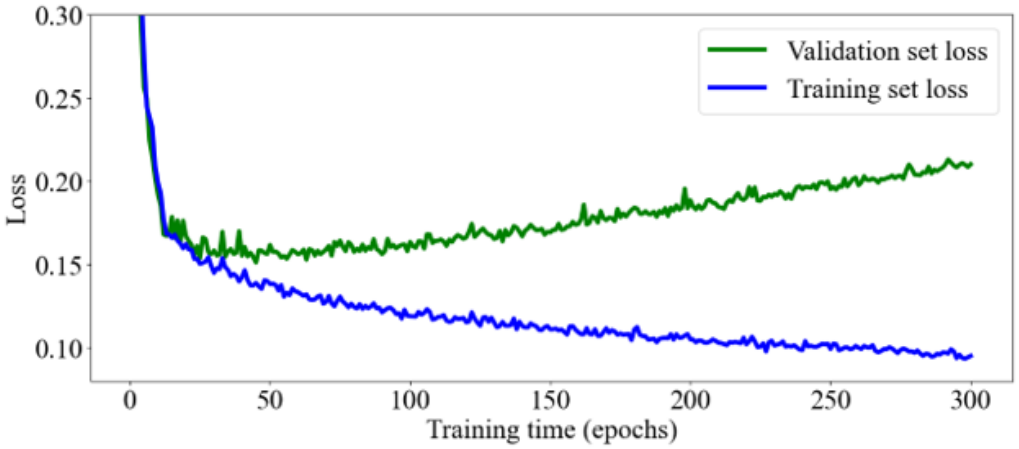

Figure 2.6: The development of validation and training loss over several epochs. The training loss diminishes throughout the whole training process. The Validation loss however starts to increase before the 50th training epoch and the model starts to over-fit.

and starts to grow again (figure 2.6). This happens because the training algorithm optimizes the network's performance based on the training error. Therefore, in order to produce a network that generalizes the best we should stop the training process at the point when the model starts to over-fit, and the minimum validation error has been reached. This is called early stopping. [24]

Early stopping treats the number of used training cycles as another hyperparameter. Restricting the number of training cycles acts as a regularizer in the sense that it limits the number of updates the training algorithm can apply to the network's weights. Where early stopping differs from other regularization techniques, is that simply running the training process on the model is a search for the optimal number of training cycles, whereas other regularization techniques feature a fixed hyperparameter that is searched for by executing the training process multiple times and by observing its effects on the validation error. The only requirement is that we test the network's performance on a validation set in between the training cycles. [24]

3 Neural networks in computer vision

3.1 Convolutional layer

Convolutional neural networks are a specialized type of network designed to process data with a known grid-like structure. Data with a grid like topology could be images or sound data. The name refers to the fact that these networks are structured in manner that the neurons perform a convolution operation to their inputs. A convolutional neural network employs a convolutional layer in at least one layer in its architecture. [13]

Originally convolutional neural networks were designed to process image data. The design goal was that the neurons in the early stages of the network would detect small local visual features, and neurons in the later stages of the network would form higher order features. A local visual feature is essentially a pattern whose extent is limited to a small patch in the input, a small group of pixels neighboring each other. For example, in the case of recognizing a dog from an image the neurons in the early layers detect small features such as edges and curves. The detection of these small features is then used by the later layers to detect body parts and ultimately the presence of the whole animal. [9], [27]

The use of convolution as part of a neural network's architecture was introduced

by LeCun et. al. in 1989 when they developed a way to apply backpropagation to a convolutional layer. [27] In the paper LeCun et. al. applied a convolutional network to a handwritten digit recognition task and found that a convolutional network requires little training data to reach convergence and can be trained very fast in comparison to a fully connected network. They also showed that a network can be trained on raw image data instead of using handcrafted features and then training a network with a set of features.

The concept of weight sharing comes into play when we try to understand the structure of a convolutional neural network. When we consider the local feature detector, we essentially have a neuron that receives a limited view of the original input. We refer to this small subset of the original input as the receptive field of the neuron. However, when we limit a neuron's input it can only detect a feature if the feature is located in the neuron's receptive field. The weights corresponding to the pixels in the receptive field are organized into a two-dimensional matrix known as a kernel. [9]

The network has to achieve translation invariant feature detection and so the neuron specialized to detect a certain feature is multiplied with each instance of the feature detecting neuron receiving a slightly shifted receptive field from the others. The shift is a hyperparameter known as the stride length. The stride length controls how much the receptive fields of the neurons overlap with each other. When combined these neurons form a feature detector that can detect the presence of the feature no matter where it is located in the image. Also, the combined receptive fields of the neurons cover the entire image. In convolutional neural networks this translation invariant feature detection is done by convolving the input with a feature kernel. The feature detecting kernel is also called a filter since the kernel goes through the image picking up instances of the searched feature and thus only allowing the searched features to pass into the feature map. [9]

Convolution is a mathematical operation described by the following equation $S(t) = \int x(a)w(t-a)da$. In our case for the convolution operation the outputs of w with negative arguments must be 0. Otherwise, the operation would allow us to see into the future. In addition to the aforementioned constraint the w must also be a valid probability density function. The function x in the convolution operation is the input and the function w is referred to as the kernel. The output of the operation is often called the feature map. [13] The convolution operation described above is the continuous version of the operation. However, when dealing with digital images the data is always in a pixel format and so we need to use the discrete version of convolution and so the equation transforms to the following form:

$$S(t) = (x * w)(t) = \sum_{a=-\infty}^{\infty} x(a)w(t-a) \tag{3.1}$$

And since we are dealing with two-dimensional image data we need to use the two axis version of discrete convolution:

$$S(i,j) = (I * k)(i,j) = \sum_{m}\sum_{n} I(m,n)K(i-m, j-n) \tag{3.2}$$

Where $S(i,j)$ is the output feature map and the I function is the input and K is the kernel. [13] Usually when constructing a CNN the used kernels are much smaller than the input and so the neurons have a very limited field of view of the original input. The function of a kernel is to act as a local feature detector and by dragging the kernel across the input space and by recording where the feature was detected we obtain the output also known as the feature map. The convolution operation in a CNN can be described with an analog of searching a painting for features with flashlight with a very narrow beam. We can go through the painting in a systematic way and record where a desired feature was detected and so obtain a map of the painting where the feature was detected. [9]

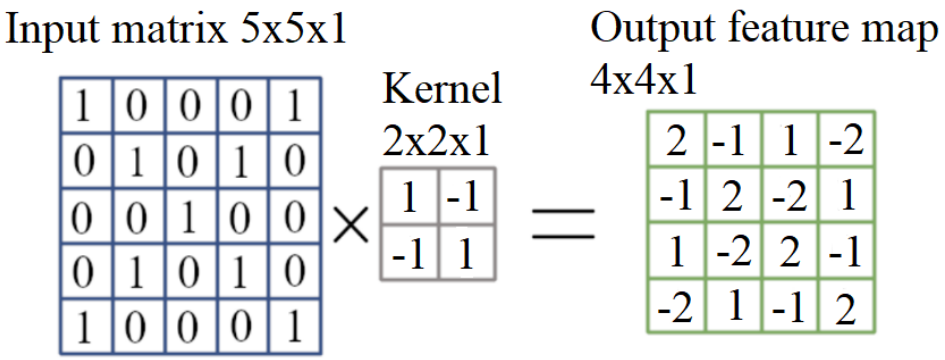

Figure 3.1: Example of convolution operation with a 2x2 kernel on a 5x5 input matrix.

In addition to the convolution operation a convolutional layer also usually involves an activation function. The activation function is applied to the feature map obtained by the convolution operation. Most often the used activation function is a rectified linear function. This essentially turns all negative values in the feature map to zero. [9]

Often it is not crucial for the network's operation to preserve the exact location of a detected feature. Because of this CNNs often discard location information by down sampling the obtained feature maps by using a pooling layer. This is done to generalize the network's ability to perform image classification. Pooling is similar to convolution at least in the sense that the same pooling function is applied across the input space. In case of pooling however the receptive fields of the pooling operations don't often overlap with each other. Most common pooling function is max pooling. This pooling method returns the maximum value of its individual inputs. Average pooling is also an option which acquires the mean of its inputs. [9]

A single convolution function or a kernel can only detect the presence of only one feature. In order for a network to detect multiple features these kernels have to be placed into layers in parallel to each other. This way a layer can learn to detect multiple different features producing a feature map for every feature searched. The

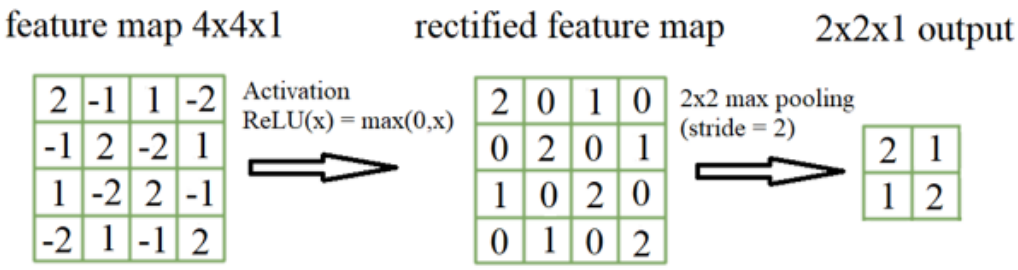

Figure 3.2: How a feature map is passed through an activation function and then max pooled.

feature maps can be combined together as a multifilter feature map or they can be flattened to a one-dimensional vector and given as input to a dense layer.

3.2 Vision transformer

In 2020 an alternative approach to deep learning-based image processing was developed. Inspired by the success transformer-based architectures have enjoyed in natural language processing, Dosovitskiy et. al. developed a transformer-based image recognition architecture called the Vision Transformer or ViT for short.

The ViT architecture's image processing pipeline follows closely to the original transformer. The image is first rearranged into 16 patches of the same size. Each patch is then positionally encoded and given an input embedding. The input embeddings are then fed into the transformer. The output of the transformer is then fed into another fully connected layer which acts as the classification head. [28]

3.3 Vision tasks

3.3.1 Classification

In image classification, a neural network is used for detecting the presence of features or objects in the image. The objects are given a class and then the images in the

dataset are assigned one (multi class) or several classes (multi label). After all an image can contain multiple classes depending on the setting. In a case where we have more than two classes, the output layer of the network contains as many neurons as there are classes. In this situation an output neuron is supposed to activate if its assigned class is present in the image. This is called one-hot-encoding. In one-hot-encoding the labels are encoded into a vector containing a binary value for each class. The binary value tells if the class is present in the image. For example, if we have a classification application containing three mutually exclusive classes: [green, blue, red] then the one-hot-encoded label for "red" would be [0, 0, 1] and [1, 0, 0] for "green". For example, the neural network architecture developed by Alex Krizhevsky that later became known as AlexNet performed image classification with 1000 classes and had 1000 neurons in the output layer. [25]

The architecture layout presented by Krizhevsky for image classification consists of two main components: the convolutional backbone and the densely connected classification head. In the case of AlexNet the convolutional base has 5 convolutional layers in succession followed by two densely connected layers and the output layer. The model developed by Krizhevsky et. al. won the 2012 ImageNet large-scale visual recognition challenge by a margin of over 10% in the top-5 error rate category.

In our application, the appropriate technique would be to use the multilabel technique since a single photo can contain many regions of multiple classes. Additionally, a binary case could also be used where the network tries to classify the photos into 'good' and 'bad' classes. However, these approaches would offer minimal transparency, special information of the detected class regions, and means to tune the behavior of the system. Changing the behavior of the system would require retraining the model. Because classification outputs are discrete values, their evaluation metrics are based on the binary option a classification can have: true or false. True positives and true negatives are examples the model predicts correctly, and

false positives and false negatives are examples the model predicts incorrectly. [16]

The most common metrics are accuracy, sensitivity, precision, recall and F1 score. Accuracy is the portion of examples the model classified correctly. [16]

TP = True Positive, TN = True Negative,

FP = False Positive, FN = False Negative

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \times 100 = \frac{\text{Correct classifications}}{\text{number of examples}} \times 100 \quad (3.3)$$

Specificity measures how well the model predicts true negatives. [16]

$$Specificity = \frac{TN}{TN + FP} \quad (3.4)$$

Recall is used for evaluating how well the model predicts samples correctly. In other words, it is used to measure the model's ability to make true positive predictions. Recall is also known as sensitivity. [16]

$$Recall = \frac{TP}{TP + FN} \quad (3.5)$$

Precision is a good metric when dealing with imbalanced classes. In an imbalanced scenario accuracy might give overly good results when a class with few examples get predicted poorly. [16]

$$Precision = \frac{TP}{TP + FP} \quad (3.6)$$

F1-score combines recall and precision into a metric describing both for situations where both properties of the model are important.[16]

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall} \quad (3.7)$$

3.3.2 Object detection

Object detection expands upon classification by also providing spatial information of the subject in the image. In essence object detection tells both what is in the image and where it is in the image. The information is provided in the form of a bounding box providing the x and y coordinates as well as the width and height of the object. In addition to the bounding box a probability distribution over the set of classes chosen for the task is also provided. An object detection algorithm such as YOLO is trained to provide both forms of afore mentioned information of the objects present in an image fed into it. [29]

Despite being a fast approach for object detection, it has some weaknesses. While the bounding box provides accurate information on the object's location and class, it lacks critical information of the object's shape. The size of the object can be somewhat estimated from the width and height of the bounding box but if the object is irregularly shaped then this becomes wildly inaccurate. For example, if an object of interest happens to be long and diagonally aligned in the image the total area of bounding box will be very large in comparison to the object.

3.3.3 Segmentation

Image segmentation is the process of grouping regions of an image together that share the same object class. The form of image segmentation where it is not necessary to differentiate between objects of the same class is called semantic segmentation [30]. When differentiation between objects is required then the task becomes instance segmentation.

Basically, image segmentation is an extension of the classification task described in section 3.3.1. In classification a single probability distribution is predicted for the entire image. In segmentation an image is submitted to the model and the model is supposed to produce a class mapping for each pixel in the original image. This

mapping looks like a high-level abstraction of the original image and indeed it is since every pixel can now only have a class value and thus all fine detail has been omitted from the image. This is depicted in figure 3.3.

The input data doesn't necessarily need to be simple images. Grayscale images can be used where color information either isn't needed or isn't available. An application domain where color information isn't usually available is medical imaging. Good examples of medical imaging methods that don't provide color information would be magnetic resonance imaging (MRI) and ultrasonography. [30]

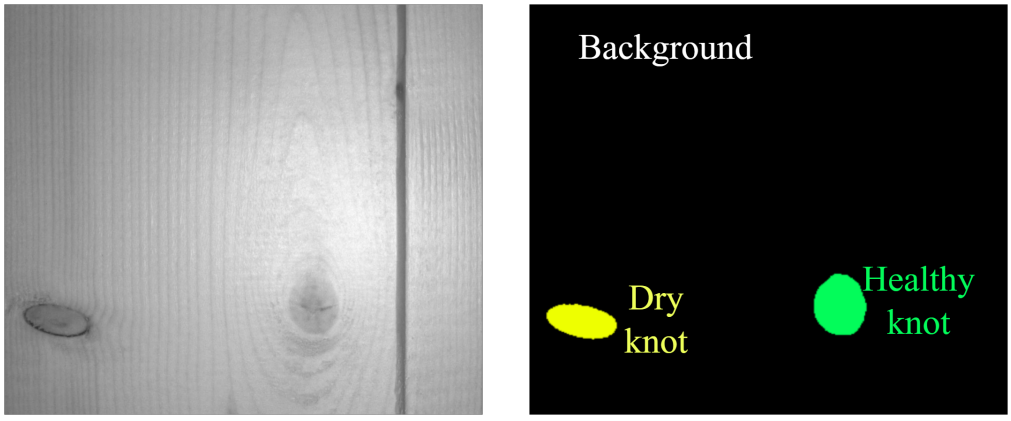

Figure 3.3: Example of a photograph and segmentation mask pair.

Segmentation can be done based on multiple images as well. With multiple images we might want to mimic stereo vision which animals with at least two front facing eyes possess. Use of stereo images aims towards achieving a more natural segmentation ability. This is very much like using depth data alongside the image. In co-segmentation the goal is to find a suitable segmentation for multiple images. Co-segmentation can be seen as finding common objects in two or more images.[30] This is often used in segmenting images produced by MRI. MRI can produce images with different modalities each containing slightly different information of the target tissue. An extension of using multiple images as input would be to use 3D-images. Again, using volumetric 3D data is common in medical imaging applications. When dealing with 3D data the smallest unit in the image is called a voxel which is a point

in the 3D matrix much like a pixel is in the 2D matrix of an image. [30]

Nowadays the most prevalent method of image segmentation is to use a convolutional neural network constructed following the encoder-decoder model. This model takes the entire image as an input and produces the complete segmentation map as its output [16]. Prior methods utilized hybrid proposal-classifier models and iterating through an image classifying each pixel to its respective class [31], [32] . In the hybrid proposal classifier approach an R-CNN is fine-tuned by sampling regions of interest for detection and segmentation resulting in a pipeline executing segmentation and classification separately [32], [33]. The iterative method involves iterating through the image and sampling small portions of the image at a time and then classifying it, producing a segmentation map [31], [34]. The weakness of the former model is that it cannot be trained end-to-end and the latter is slow since the model has to be executed for every pixel in the input image.

I chose segmentation for the feature detection component of the quality inspection system because segmentation is able to produce and abstracted version of the original image in the form of a feature map. The segmentation map preserves the shape of the features for further analysis unlike the bounding boxes in object detection. In this thesis I will test two convolutional neural network architectures called U-net [31] and DeeplabV3 [35] as well as a more recent vision transformer-based architecture called Segformer. These three architectures represent different ways of implementing a segmentation algorithm.

In evaluating image segmentation results, the goal is to compare the output segmentation map with the ground truth. This process is essentially checking which pixels were classified correctly and which ones weren't. Due to the different nature the output format the metrics used for evaluating classification tasks can sometimes give misleading results. The three most common metrics for comparing regions are pixel accuracy, intersection-Over-Union (IoU) and Dice score. [16]

Pixel accuracy is similar to the accuracy metric used to evaluate classification results. It tells the portion of pixels that were correctly classified. Pixel accuracy can be misleading in scenarios where classes are drastically imbalanced. A small class might occupy very small areas in comparison to background pixels and thus pixel accuracy may be high even though the small class objects aren't properly segmented. [16]

TP = True Positive, TN = True Negative,

FP = False Positive, FN = False Negative

$$PixelAccuracy = \frac{TP + TN}{TP + TN + FP + FN} \times 100 \tag{3.8}$$

Intersection-Over-Union measures the degree to which the predicted and ground truth areas overlap. This metric takes imbalanced classes into account better than pixel accuracy making it more prevalent in segmentation method evaluation. [16]

$$IoU = \frac{TP}{TP + FP + FN} \tag{3.9}$$

Dice score, which is also known as F1 score, is similar to IoU and their results are often interchangeable. It is usually up to the author's preference which method is used as both are common in medical studies. [16]

$$DiceScore = \frac{2 \times TP}{2 \times TP + FP + FN} \tag{3.10}$$

3.4 Segmentation architectures

3.4.1 FCN

The encoder-decoder model is derived from the idea of using convolution and deconvolution to form an end-to-end trainable network capable of segmenting an entire

image while also classifying the segments[16], [31], [32]. The first fully convolutional networks were constructed by replacing the dense layers with convolutional layers. This resulted in significantly smaller heatmaps compared to the original input in size. The network was thus modified to include a deconvolution stage to upsample the low-resolution feature maps into a segmentation map with the original image size. The resulting models were unable to produce segmentation maps with fine detail. The network was then modified further by adding skip connections allowing the deconvolution stage to use the lower-level features as context in creating the segmentation map in the late stages of the network structure.[32]

3.4.2 U-net

One of the first architectures to fully realize the encoder-decoder model was the U-net architecture. It was built upon the initial FCN-architecture proposed by Long et al. It successfully incorporated many of the FCN's design features such as skip connections and only using convolution and deconvolution as its learning processing stages. The U-net (figure 3.4) was named after how its drawn-out architecture appears. It appears symmetrical as its decoder component mirrors the encoder path in function [16], [31].

Together an encoder and a decoder form a neural network architecture called an autoencoder. The basic purpose of an autoencoder is to reconstruct the input to its output [13]. When an autoencoder is constructed using convolutional layers it is called a convolutional autoencoder. In their paper Dong et. al. constructed an autoencoder using convolution in the encoder and deconvolution in the decoder. The encoder serves as a feature extractor and the decoder then reconstructs the image based on the encoder's output. They also showed that such a network's segmentation performance can be improved with unsupervised pretraining. [36]

In the U-net the convolution stages in the contracting path involve two 3x3 un-

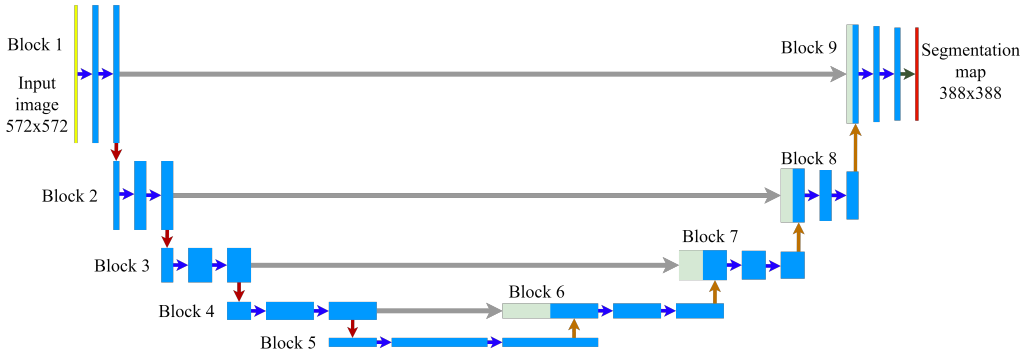

Figure 3.4: The U-net architecture proposed by Ronneberger et al. Blue arrow = 3x3 conv + Relu Red arrow = 2x2 max pooling Grey arrow = crop and concatenation Orange arrow = upsampling + 2x2 convolution Green arrow = 1x1 convolution

padded convolution operations with ReLU activation function followed by a 2x2 max pooling operation with a stride of 2. At every stage of the contracting path the number of feature channels doubles while the width and height of the feature maps halve. The decoder component utilizes a combination of an upsampling operation which doubles the resolution of each feature map followed by a 2x2 convolution operation which halves the number of feature channels. The feature maps from the corresponding stage in the encoder are cropped and concatenated with the upsampled feature maps. The cropping is necessary since the feature maps coming up the decoder have lost border pixels due to being subjected to more convolution operations than the ones forwarded from the encoder. Like the stages in the encoder path the stages in the decoder involve two 3x3 convolutions with a ReLU. The final layer maps the 64-channel feature matrix to the desired number of classes using 1x1 convolution. According to the writers the resulting network is relatively fast and capable of producing segmentations of fine detail. [31]

In [37] Iglovikov and Shvets show that the U-net can be used with a repurposed VGG-11 pretrained encoder. In their article they show that with the pretrained encoder gives better segmentation performance than if it was randomly initiated at the start of the training.

3.4.3 DeepLabV3

DeeplabV3 [35] is also a convolutional neural network architecture like U-net. However, these architectures differ quite a bit in their implementation. DeeplabV3 uses dilated convolutions with spatial pyramid pooling. Dilated or atrous convolution is a generalization of the previously described convolution operation where the input values have fixed length gaps between them where a normal convolution kernel has none. This form of convolution expands a convolutional filter's receptive field without increasing the number of parameters of a convolution layer.

Spatial pyramid pooling [38] is a technique where the output of the convolutional section of the network is pooled with various fixed number of bins. This enables the use of variable sized inputs. In deeplav3 this is used with parallel atrous convolutions with different rates to capture features at multiple scales.

3.4.4 Segformer

Segformer [39] is a vision transformer (ViT) [28] based model that abandons the convolution-based approach. The Segformer however also adopts an encoder-decoder architecture. The encoder produces both high resolution coarse features as well as low resolution fine features. The decoder then fuses the features to produce the final segmentation mask. The encoder section is called a mix transformer. It is inspired by the original ViT, but the authors have modified it to suit a segmentation task. The decoder part of the network consists of fully connected layers giving the decoder layers larger receptive fields than CNN based decoders. In the first step of the decoder the channel dimensions are unified with a fully connected layer. In the second step the features are then upsampled. Then a fully connected layer is used for fusing the features together. In the final step of the decoder another fully connected layer is used for producing the segmentation mask with desired number of classes.

The original ViT [28] paper notes that a transformer-based approach introduces less inductive biases to the network than a CNN based approach. In a CNN locality, translation invariance, and two-dimensional neighborhood structure are embedded throughout the network. This lack of inductive bias could pose a problem with smaller datasets such as the ones used in this thesis.

4 Deep learning in tumor segmentation

This section will examine the use of U-nets in a medical setting. The U-net architecture is relatively old at the time of writing this thesis. However, research on medical image segmentation using the U-net architecture is still being published in significant amounts as we will soon discover.

The purpose of this chapter is to discover a robust training method for the feature segmentation models of the wood grading system. This chapter has three main parts. In the first part I will provide brief explanations of what tumors are and how MR imaging works. Then I will perform a literacy survey to uncover popular recently used training methods of the U-net architecture. In the third part of this chapter, I will test the extracted training parameters discovered in the survey to validate their effectiveness. I will do this by training the networks with the BraTS-2020 data.

I chose MRI tumor segmentation for the subject of the hyperparameter survey because it shares three crucial characteristics with wood feature segmentation:

1. Organic material
2. Imbalanced classes
3. Amount available of data is limited

The first characteristic sets the difficulty of both segmentation tasks. With tumor segmentation the regions of interest have natural variation like wood features such

as knots. However, the nature of tumors are different in a way that they are always caused by uncontrolled cell growth and are therefore likely to exhibit more unique features. The wood features such as knots on the other hand occur more frequently in the wood tissue and therefore should have less variance in their appearance. Because of this, I would say that tumors have an extra degree of randomness in their appearance because of their inherently random nature. The wood feature segmentation task should therefore be easier for the training algorithm to learn despite also featuring natural variance.

The second characteristic is about the inherently small size of the tumor areas and wood features when compared to the background class and the whole image area. The model's training process is guided by the error gradient so the loss function has to be able to quantify the segmentation error appropriately when dealing with heavily imbalanced classes.

The third characteristic about data limitation isn't unique to these segmentation problems but still needs addressing none the less. The U-net architecture itself already addresses this issue by improving gradient flow with its skip connections. The data limitation problem can also be alleviated by using extensive augmentations, transfer learning and an appropriate optimization algorithm.

The survey will be carried out by reviewing articles using the U-net architecture for tumor segmentation in MR-images. Magnetic resonance imaging is able to produce high contrast images of soft tissues making it a good choice for tumor detection and why the survey will only include studies conducted on MR images.

4.1 Magnetic resonance imaging

MRI is an imaging technique based on the presence and properties of water residing within tissues of the body. Water quantities and properties can vary from tissue to tissue. The water properties of a tissue can become abnormal if it is subjected to

injury or disease. MRI can reliably detect these changes, making it a reliable and sensitive diagnostic tool for inspecting the anatomical features of the body and even organ function. [40]

An MRI machine consists of a few main components, the first of which is a large and powerful magnet that creates a constant magnetic field. This magnet is the reason why ferromagnetic materials aren't permitted in the same space as the MRI machine. The second component is an RF transmitter coil making an excitation magnetic pulse in the radio frequency range. The next component is the receiver coil used for detecting the RF signal from the excited hydrogen atoms. A magnetic field gradient is used for localizing the MR signals received by the receiver coil. In addition to these a computer is used for controlling the scanner, displaying the MR images and storing them. In order for a human to be placed in the machine it needs an input device holding the body in place. This is usually a bed, or a table equipped with comfort, positioning aids and physiological monitoring equipment. [40]

The MRI machine detects changes in the magnetic properties of atom nuclei and more specifically the nuclei of hydrogen atoms. The hydrogen atoms within the body are bound in substances such as water and lipids which make roughly $75 - 85\%$ of the body's mass. The most impactful properties of tissues regarding hydrogen are proton density, T1, and T2. Proton density is the frequency of atoms within a certain volume of tissue. For example, bodily fluids such as cerebrospinal fluid have higher proton densities than tissues like tendon and bone. T1 is the spin-lattice relaxation time and T2 is the spin-spin relaxation time. The relaxation times tell how long it takes for the magnetic moments of the hydrogen atoms to reach equilibrium after an excitation pulse. Fluids have the longest T1 and T2 relaxation times whereas fat-based tissues have the shortest relaxation times. Water based tissues fall in between fluids and fats in terms of relaxation times. [40]

The nucleus of a hydrogen atom is a positively charged subatomic particle known

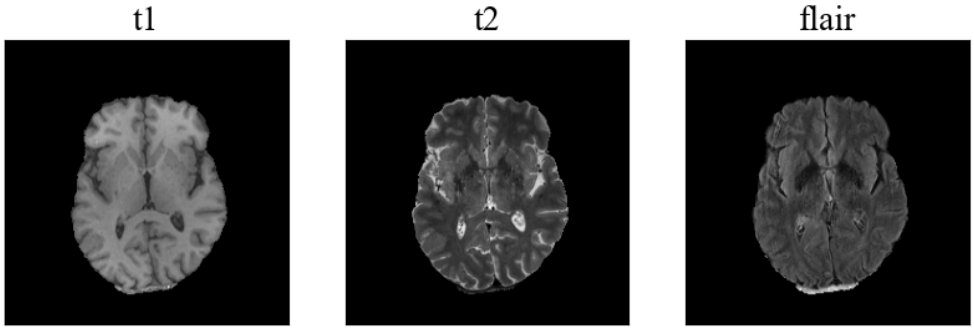

Figure 4.1: An axial slice of a brain MRI volume. T1(left), T2 in the middle, and FLAIR on the right. In T1 images white matter appears bright and grey matter appears dim and vice versa in T2 since white matter has a higher myelin density than grey matter. The ventricles appear bright in the T2 image since they are filled with CSF. The CSF has been suppressed in the FLAIR image.

as a proton. These protons have a property called a spin which creates a magnetic moment for the particle. The main magnet is used for creating a weak magnetization of bodily tissues where the magnetic moments are aligned with the main magnetic field. The individual magnetic moments however aren't perfectly aligned but instead precess around the axis longitudinal to the main magnetic field. The precessing magnetic moments are out of phase with one another and so the net magnetic field of the tissues aligns with the external magnetic field. [40]

The acquisition process of an image starts with the excitation of the protons' magnetic moments with the transmitter coil. The transmitter coil produces an external oscillating magnetic field that knocks the magnetic moments to a plain transverse to the static magnetic field and brings the precession of these moments to phase with one another. The when the magnetic moments are in phase with one another they form an oscillating magnetic field. This magnetic field induces a voltage in the receiver coil that can then be measured. [40]

As soon as the excitation pulse ends the precessing spins start to dephase decaying the net magnetic field they are forming. The rate of this decay is called T2 relaxation. The T2 relaxation time is the time at which the strength of the signal

emitted by a tissue has fallen to 37% of the peak strength. The T1 relaxation is the rate at which the hydrogen spins regain the net magnetic field strength 63% of the maximum magnetization of the tissue longitudinal to the main magnetic field. Both relaxation times tend to be longer in bodily fluids, medium length in water-based tissues and shortest in lipid-based tissues. The T2 relaxation time is generally much shorter than T1 for a given tissue type. [40]

4.2 Cancer and tumors

Cancer is a deadly disease caused by malignant tumors. Malignant tumors are human tissue formed by cancer cells. Cancer cells spawn when a normal cell's DNA develops undesirable mutations which damage the cell's functions. Undesirable effects on the cell's functions may include accelerated growth and uncontrolled replication. A tumor forms when cancerous cells start to uncontrollably multiply via cell division. Normally the body possesses countermeasures for dispatching cancerous cells before they can develop into tumors. These countermeasures usually decline in effectiveness as the body ages. This decline over age is part of the reason why cancers are more prevalent in the older population. [41]

Because the cause of cancer is in random mutations in the DNA each cancer is unique. The mutations can develop further as the cancer progresses resulting in different genetic changes within cells of the same cancer. Even cells in a single tumor can have different changes in DNA. These mutations can occur during cell division, or they can be caused by external factors called carcinogens.[41] Carcinogens include substances such as cigarette smoke, ionizing radiation such as X-rays, and pathogens such as the human papilloma virus (HPV) [41], [42]. Gene changes that lead to cancer can also be inherited.[41]

Cancers are named after their organ of origin. For example, cancer that has developed from brain cells is called brain cancer and cancer born of skin cells id

called skin cancer. Cancerous tumors often invade neighboring tissue and can also spread across the body in a process called metastasis creating even more tumors. A tumor originating from another part of the body is called a metastatic tumor. For example, when breast cancer creates a tumor in the lung it becomes metastatic breast cancer instead of breast cancer and lung cancer.[41]

Cancer leads to death by causing organ failure. Metastatic tumors can disrupt their host organ's functions to such a degree where the organ eventually stop performing its functions. For example in leukemia the blood cell producing part of the bone marrow starts to produce abnormal white blood cells in excessive amounts. These abnormal white blood cells eventually crowd over the normal blood cells hampering the circulatory system's ability to distribute oxygen, control bleeding and countering disease. [41]

Cancer is a highly prevalent disease with millions of new cases (19.7) and deaths (9.7) worldwide each year. The most prevalent cancer types are lung, breast, and colorectum cancers. Many face the reality of cancer as the probability of developing cancer before the age of 75 is appr. 20%. [43] Since cancer is a common cause of death and its mortality rate increases the longer it is not diagnosed early detection becomes even more important. However, manual detection and segmentation of tumors is very slow and so automatic detection and segmentation of tumors could offer faster and cheaper means of diagnosis lowering the threshold of using imaging tools.

4.3 U-net tumor segmentation hyperparameter survey

In this chapter we will examine a total of 363 papers studying tumor segmentation from MR images using CNNs based on U-net. The papers were searched from the

Web of Science database with the search parameters listed below. I chose to limit the architecture to only include the U-net to limit the scope of the survey. As these are general purpose methods and the different training parameters can be used for all three architectures described in chapter 3.4 we can expect that the results won't be biased towards the U-net architecture.

For the survey I had four research questions:

1. What are the most common augmentation methods?
2. What is the most common loss function?
3. What is the most common optimizer?
4. What is the most common transfer learning implementation?

I used the following search to find the articles:

(cancer OR tumor OR tumour) AND U-net AND segmentation AND (MRI OR magnetic resonance imaging)

This search gave 635 results, 363 of which satisfied the earlier criteria and were accessible via utu volter in November 2023. I examined the articles one by one and extracted the training parameters of the network training process from each article if they were disclosed. In this study the hyperparameters are selected by their popularity. This is because a proper one-to-one comparison between the hyperparameters would be impossible to do with a literacy survey since the individual articles use slightly different training methods, models, datasets, and evaluation metrics. Since the articles have also conducted their own design processes in selecting their hyperparameters we can assume that on average they have selected the best combination of hyperparameters available for their respective studies. So, on average the popular choice of a given hyperparameter should perform the best since it was the result of most design processes.

4.3.1 Optimizer

From the 363 articles studied in this study 75 articles didn't specify the used optimization algorithm. Out of the 288 articles the overwhelming majority of 232 used a form of the adaptive momentum or the Adam optimizer. 41 articles used the stochastic gradient descent optimizer. The SGD algorithm was used often with momentum (17) or Nesterov momentum (7) and weight decay (6). The use of other optimizers was quite rare. After Adam and SGD, the most used optimizers were the RMSprop (4), and Ranger (3) optimizers. Since the most used optimizer was Adam, we will also use it in our wood panel defect detection network training.

4.3.2 Loss function

The choice of loss function is crucial in image segmentation tasks because the class distribution of image pixels is prone to being unbalanced as foreground classes might occupy only a small area in an image otherwise dominated by the background class. Some loss functions such as cross entropy by default cannot account for class imbalances well. This is a problem in tumor segmentation and many studies state the class imbalance to be the primary feature around which they design their used loss functions. These functions can be standalone consisting of only one term or hybrid functions consisting of multiple terms. A hybrid approach can be a linear combination of two separate previously known loss functions such as cross entropy and dice loss.

Of the 363 articles studied, 300 disclosed their used loss function. The most frequent loss functions are the dice and cross-entropy losses being used in 222 and 132 papers respectively. Other notable loss functions are the focal and Tversky losses with 20 and 11 appearances respectively. In total dice is used as a standalone loss function in 129 papers or as part of a hybrid function in 222 therefore making it the most prevalent loss function in tumor segmentation. The cross-entropy function on

the other hand, either categorical or binary is used together with other loss functions or as standalone in 131 papers. Binary or categorical cross-entropy is used alone in 54 papers. 5 of which used the weighted version of cross-entropy. The weighted cross-entropy is used with other loss functions in 4 papers.

In tumor segmentation datasets each class is often present within an image and so the frequency of the classes is quite high despite the pixel distributions being unbalanced. In our wood defect detection, some of the defect classes can be quite scarce and so a loss function capable of accounting for this imbalance must be chosen. Therefore, following the majority and choosing Dice loss as our loss function should prove successful.

4.3.3 Transfer learning

Transfer learning proved a scarcely used technique in tumor segmentation with only 30 papers disclosing its use in some way. Of the 30 articles that used transfer learning, 13 used a pretrained image classification CNN as the encoder their networks. Of these 13 articles, 8 used a ResNet-based pre-trained encoder. Other articles used different approaches such as using segmentation datasets related to their own datasets to pretrain their networks.

In the former approach the pretrained convolutional backbone of the original network is extracted. Then the U-net model is constructed by concatenating a compatible decoder structure and skip connections to the encoder. This way the resulting network will have to only learn the weights of the decoding path assuming the pre-trained features are useful in the new segmentation problem. Since the wood defects don't have complex shapes or textures, we anticipate that the approach of using a pre-trained encoder will decrease the network's required training time. Since the most popular choice of pre-trained encoder was the ResNet-34 we will also utilize it in our wood panel defect detection network.

4.3.4 Augmentation methods

Of the 363 articles used in this study, 213 articles disclosed using a data augmentation scheme as part their training pipeline. On average tumor segmentation datasets are quite small. This largely explains the popularity of data augmentation in this application domain. Using more data to train a neural network generally improves its generalization ability. And so synthetic data generation provides a way to alleviate the problem of having to work with small datasets.

The most used data augmentation method was flipping or mirroring along an axis. This augmentation method was used in 156 articles. The second most used augmentation method was rotation with 144 articles using it. Since the variety of data augmentation methods present in the papers is very large, we decided to use the methods that were used in 10 or more articles. These methods and use times in articles respectively are:

1. Flip – 156
2. Rotation – 144
3. Scaling – 43
4. Elastic transform – 37
5. Shift – 28
6. Translation – 28
7. Zoom – 26
8. Shear – 23
9. Gaussian noise – 22
10. Gamma – 21
11. Brightness – 18
12. Contrast – 16
13. Crop – 16

4.4 Testing the survey results with BraTS-2020 data

Before we apply the discovered training parameters to our own data, we will test and see how they work in the environment they were derived from. The research question

in this chapter isTo accomplish this task, I will use the widely known BraTS 2020 training dataset [44]–[48]. This dataset contains 369 annotated MRI brain volumes containing glioblastomas and low-grade gliomas. To address the apparent limited number of patients in the BraTS-2020 dataset I will use 10-fold cross validation to attain reliable performance metrics. The models I chose for this experiment are U-net and DeeplabV3 with the Resnet34 encoder and Segformer-Mit_b2 models. These networks have similar number of parameters so they should have similar learning capacities. I gave the U-net model's decoder blocks the following number of parameters [512, 256, 128, 64, 32] for the blocks 5, 6, 7, 8, 9 in figure 3.4 respectively. In the decoder I also enabled the use of batch normalization between the convolution and activation. With the ResNet34 encoder the number of parameters of the U-net model is at appr. 30 million. For DeepLabV3 and Segformer models the number of parameters were appr. 26 and 25 million respectively.

4.4.1 Preparing the MRI volumes

The MR imagery comes in the form of three dimensional nifti-files. Each MR imaging modality and segmentation has its own volume image. For these 3D files to be used to teach a 2D segmentation network the volumes must be sliced. The orientation I chose to slice the volumes in was the transverse plane. To increase the relative portion of pixels belonging to tumor/lesion classes in the dataset I excluded all tumor/lesion free slices from the 2D dataset. The omission of tumor/lesion free slices resulted in a total number MRI slices of 24 422. And finally, since the original segmentations have an empty class in label 3, I relabeled the pixel label 4 (enhancing tumor) as 3. The volumes also contain varying value ranges, so I also chose to divide each slice with its highest value to bring each slice's value range to [0,1].

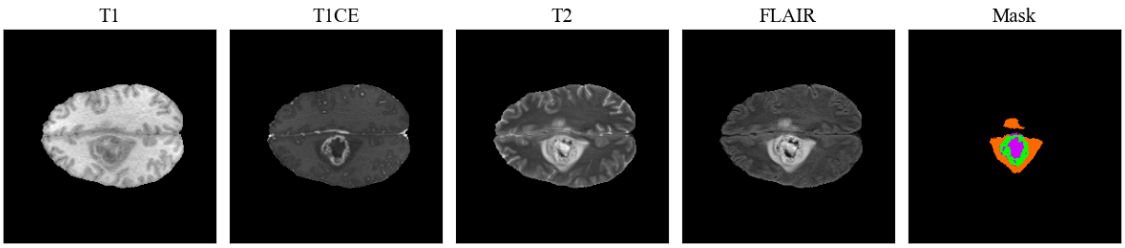

Figure 4.2: An axial slice of a brain MRI volume from the BraTS 2020 dataset. The colors of the mask depict different parts of the tumor/lesion. Orange: peritumoral edema, green: enhancing tumor, purple: tumor core

4.4.2 Cross validation training

I decided to use a cross-validation training scheme because the BraTS-2020 has only 369 annotated MRI volumes. Cross validation (CV) can be used as a work around when using small datasets. In cross validation the dataset is split into equisized folds. Then each fold is used as an evaluation set while the other folds are used for training. This means that the model is trained as many times as there are folds. In a deep learning setting using this technique takes a lot of time. On the other hand, this allows us to evaluate the model and the training algorithm on the whole dataset resulting in reliable results.

In this experiment I chose to use 10-fold CV to utilize as much data for training and testing as possible. For each CV iteration a single fold is used for testing while the nine others are used by the training algorithm. Of the nine folds used for training the model, one is held back for validation and monitoring the training process. I split the patients to 10 folds so that each fold had 37 or 36 patients. Splitting the data patient wise rather than slice wise ensures that there is no data leakage between the slices. The slices of an MRI-volume are statistically dependent on each other since successive slices are very similar to each other. Eliminating the data leakage between slices is critical for gaining accurate estimates of model performance on new data.

In the data loading process, I decided to center crop the slices from the original

224x224 size to 192x192. This reduces the relative amount of empty background pixels in the image and decreases training time.

The dataset has MRI volumes containing four modalities: T1, contrast enhanced T1, T2, and FLAIR. I decided to not use the T1 modality during training, because the information of T1 modality is present in T1ce images making the T1 modality mostly redundant. Also, the architectures I chose to use in this thesis have been pretrained with 3-channel RGB photos meaning that the first layers of the networks would have to be reinitialized with 4 channels weakening the effectiveness of pretraining in this setting.

The augmentation pipeline consisted of the methods mentioned in 5.3.4. The random gamma correction tool of the Albumentations library [49] didn't work on images with a value range [0,1] so I omitted this method from the pipeline. The augmentation pipeline consisted of the following components (and parameters):

1. Random horizontal flip ($p = 0.5$)
2. Random vertical flip ($p = 0.5$)
3. Random resized crop ($scale = (0.7, 0.9)$, $height = 192$, $width = 192$, $interpolation = cv2.INTER\_NEAREST$, $p = 0.4$)
4. Random brightness and contrast ($brightness\_limit = (-0.1, 0.0)$, $contrast\_limit = (-0.1, 0.1)$, $p = 0.4$)
5. Random Gaussian noise ($var\_limit = 0.002$, $mean = 0$, $p = 0.5$)
6. One of ($p = 0.5$)
    - Random shift, scale and rotation ($shift\_limit = 0.1$, $scale\_limit = 0.1$, $rotate\_limit = 0$, $border\_mode = cv2.reflect$, $interpolation = cv2.INTER\_NEAREST$)
    - Random rotation ($limit = (-45, 45)$, $border\_mode = cv2.reflect$, $interpolation = cv2.INTER\_NEAREST$, $p = 1$)

- Random elastic transform ($alpha = 1$, $sigma = 20$, $alpha\_affine = 20$, $interpolation = cv2.INTER\_NEAREST$, $p = 1$)

At the last stage of the input pipeline the MRI-slices were normalized with the channel wise means and standard deviations of the ImageNet dataset: mean=[0.485, 0.456, 0.406] and standard deviation=[0.229, 0.224, 0.225]. For the training I used the Adam optimizer and the Dice loss function as they were the most popular choices in the hyperparameter survey. For each iteration in the CV the models were trained for 100 epochs with a batch size of 8. Running these experiments was extremely time consuming even on a discrete GPU. Training the U-net, DeeplabV3, and Segformer models on an Nvidia RTX 3080 GPU took approximately 43, 36, and 58 hours respectively.

Table 4.1: Test Dice scores on the BraTS-2020 data

Model	background	tumor core	peritumoral edema	enhancing core
Unet-ResNet34	0.996	**0.737**	**0.811**	**0.853**
DeepLabV3-ResNet34	0.996	0.705	0.786	0.805
Segformer-Mit_b2	0.996	0.735	0.806	0.837

I trained the models with the pytorch-lightning [50] deep learning framework and segmentation models pytorch -library [51]. The results of the training can be observed in table 4.1. Based on the metrics in table 4.1 the models learned some relevant features from the data. Based on the metrics it would seem that the Segformer and U-net -based models perform somewhat similarly while the DeeplabV3 model lags behind the others. This could be the result of the DeeplabV3's architectural design. The DeeplabV3's base architecture doesn't use skip connections like the U-net nor does it omit as much inductive bias as the Segformer. The atrous spatial pyramid pooling module aims to improve the model's performance in problems where objects can appear at various scales. In a task such as tumor segmentation this feature seems to be redundant since the slices are presented to the networks in

uniform fashion. The U-net's skip connections seem to help the network to capture the small detail better. The difference between the networks can be seen in figure 4.4. The U-net's prediction manages to preserve most of the detail of the annotated mask while the Deeplab's prediction is less detailed. The Segformer's predictions are closer to the U-net's level of detail.

While the networks are able to learn the segmentation task quite well according to table 4.1 and figure 4.4, figure 4.3 tells a different story. The networks achieve their best validation scores quite early in the training process while the training loss continues to improve until the last epoch. This results in a situation where the gap between training and validation loss is quite wide at approximately 10 percentage points. This suggests that the networks are unable to generalize properly to new inputs. This can either be the result of small but consistent differences between the test predictions and annotations or the models fail to segment some of the new examples completely.

By taking a closer look at some of the test predictions I concluded that the latter is the case here. In figure 4.5 the models produced predictions that were entirely different to the annotated mask. In fact, the predictions were so different even amongst themselves that it seems like the models didn't learn to segment the image at all. The stark contrast in segmentation performance between figures 4.4 and 4.5 suggests that some of the tumors in the data are so different from the rest that the model isn't able to learn to segment them.

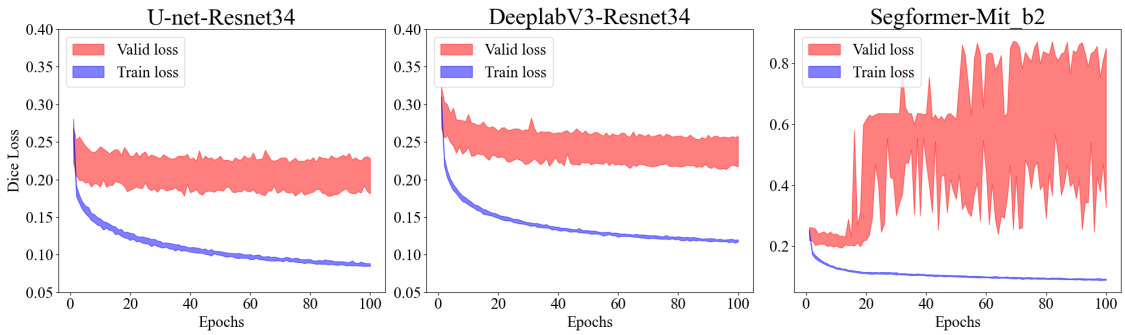

Figure 4.3: The training and validation loss development curves of the models. The figures combine the loss figures produced by training the models on different slices. The filled area depicts the range of values the loss could have at a given epoch.

The Segformer's validation loss figure differs from the other two. The Segformer's validation loss decays rapidly and consistently across the different training runs after the 15th epoch. This suggests that the Segformer models start to over-fit at this stage of the training process.

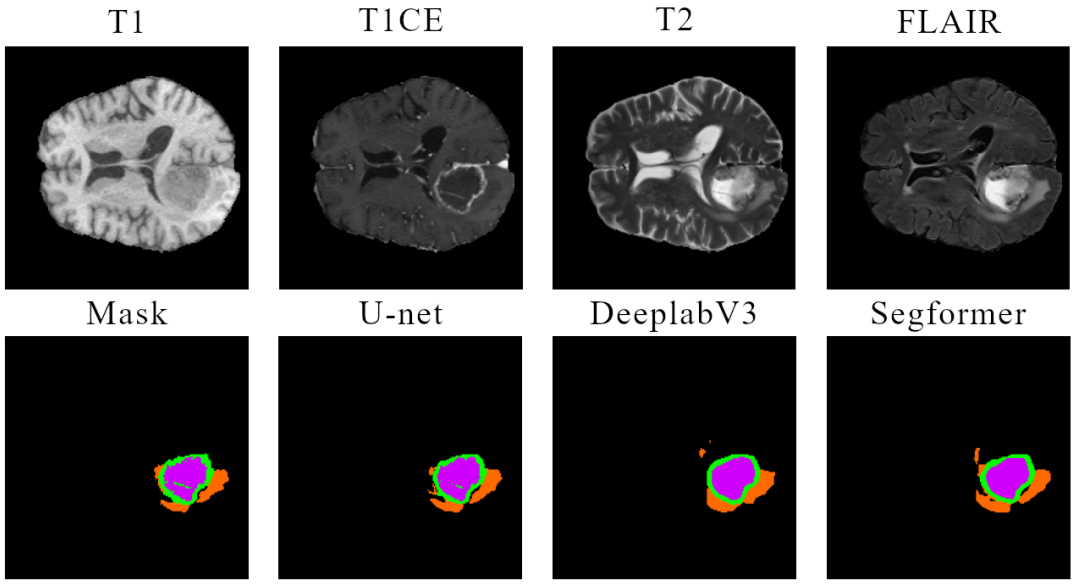

Figure 4.4: Good test segmentations on the BraTS-2020 dataset. Top row: Original Brain MRI slice modalities. Bottom row: The original annotated segmentation mask and the predictions made by the models.

The models' lackluster generalization is likely caused by the small number of

patients present in the dataset. While the total number of slices present in the dataset is fairly high at over 24 000, they produce a high degree of redundancy. Since a slice is very similar to its neighboring slices the amount of information provided by each slice is likely very small compared to a situation where the slices are statistically independent of each other. Since the slices are extracted from only 369 patients, we only have that many unique tumor cases.

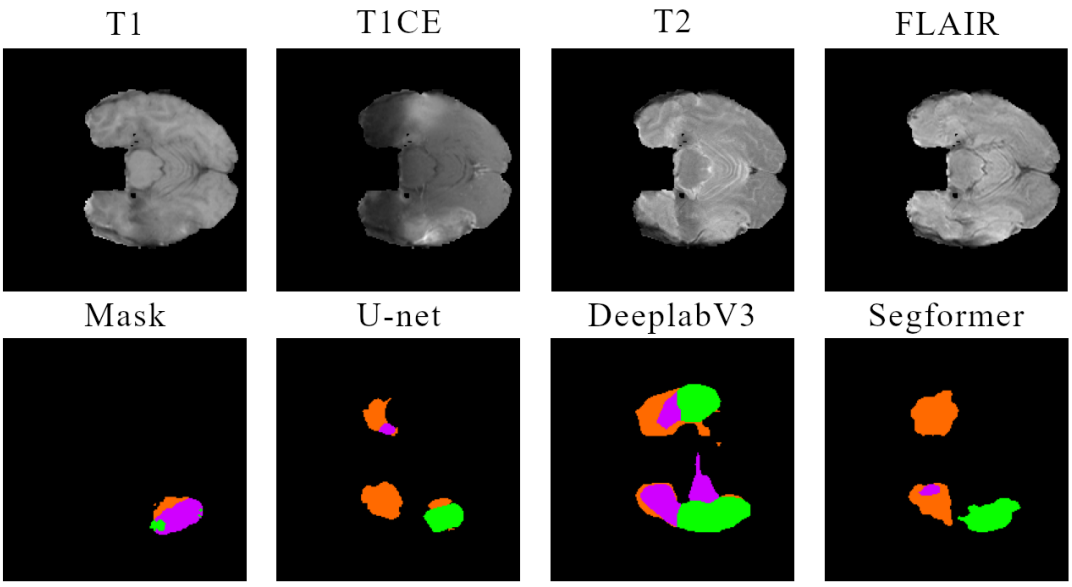

Figure 4.5: Bad test segmentations on the BraTS-2020 dataset.

This number of patients isn't enough to produce a model that would generalize well to new patients. We can see the inconsistent performance of the models on the test folds in figure 4.6. While the medians and interquartile ranges are consistent with the achieved levels of validation losses, we can also see that the 25th percentiles are quite low which means that the models are able to segment a portion of the data with a high dice score. However, when we examine the flier points of the boxplots and examine the histograms, we can see that the test losses have a prominent upper tail. This confirms the fact that some slices or patients are so different from the rest of the dataset that the models can't segment them when they are excluded from the training data.

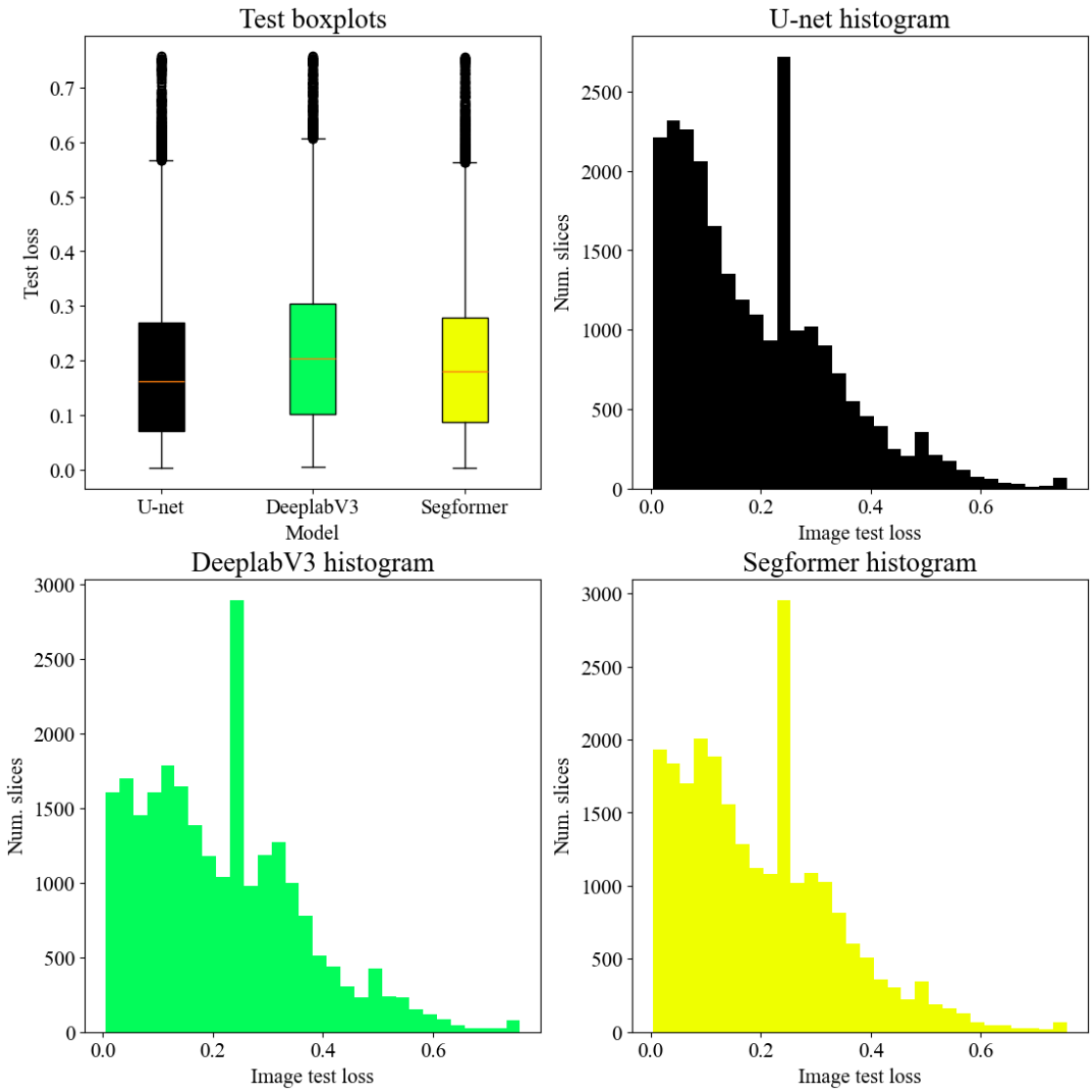

Figure 4.6: The BraTS 2020 test loss distributions as boxplots and histograms.

However, since the models were able to capture some useful features from the training dataset, I can conclude that in a situation where the amount of data is sufficient the chosen hyperparameters can be used for training segmentation models for wood defect detection. I will test this hypothesis in chapter 6.

5 Semantic segmentation in the wood industry

Before I conducted the hyperparameter survey of the previous chapter (4) I searched for related work from the wood processing industry. However as it turned out, existing research about wood feature segmentation is scarce. This is likely caused by the poor availability of good segmentation datasets in the field.

I performed the search of related research in the Web of science database using the search words presented below.

(wood OR timber OR lumber) AND (artificial intelligence OR AI OR machine learning OR neural network OR deep learning) AND (detection OR classification OR segmentation)

The search yielded 1408 search results. After filtering the search results based on the contents of the headers and abstracts leaving only the articles about machine learning and the wood industry, only 91 were deemed relevant. Of these 91 articles 22 researched the application of segmentation in the wood industry. Of these 22 articles 9 studied segmenting features from a tangential cross-sectional image of a tree's trunk. Of these articles two [52], [53] researched detecting dead and healthy knots, as well as cracks [52] or wormholes [53] from veneer, one researched detecting knots from oak planks [54], one researched sap and heartwood assessment [55], one researched detecting defects from processed bamboo [56], one researched the detection of cracks

[57] and one researched the detection of knots, cracks, and wormholes [58]. Two articles researched the detection of more than two or three features. The first [59] one aimed to detect healthy and dead knots, blue and brown stains, pitch steaks, and cracks. The other [60] attempted to detect birdseye and freckle, bark and pitch pockets, wane, cracks, blue and brown stains, holes, pith as well as healthy and dead knots. The articles, apart from the ones researching processed bamboo or veneer studied planks or an equivalent pieces of wood.

The photographs processed in this thesis differ from the articles described above by two factors. The first difference is the finish of the wood. Only the processed bamboo from the article by Hu et. al. is stated to have a finish as the pieces of bamboo are essentially finished products. These pieces of bamboo however have a transparent varnish rather than an opaque paint. The other difference is the fact that the wood examined in this thesis is in the form of a planed panel rather than a smooth plank. The panels in this study have a tongue and groove profile meaning the other side of the photographs always has a transitional area where the panel surface turns into the tongue. This area shows as a streak like pattern in the photographs adding to the complexity of the detection task.

The most popular network architectures used in the articles were based on the Mask R-CNN [52], [53], [58], the U-net [56], [57] and other FCNs [54], [59]. Three articles state that transfer learning accelerated the networks' learning process. One paper used the ImageNet dataset [59] and another used the COCO dataset [53]. The third paper by Hu et. al. also states that they used transfer learning but do not disclose the dataset they used [56]. The used loss functions and optimizers aren't disclosed in the articles very often. The article by He et. al. state that they use the categorical cross entropy to train their FCN based networks [59]. The article by Urtans et. al. uses binary cross entropy to train their FCN and DeepLab networks [54]. The article by Lin et. al. researching wood crack detection used the binary

Dice loss and the Adam optimizer to train their network based on U-net. The choice to use Dice loss was justified in the article with the fact that it is not influenced by class imbalance [57]. Class imbalance in a segmentation task means that certain classes in the dataset have significantly less pixels in the photographs. The Adam optimizer was also used in articles by Hu et. al. [56], and Li et. al. [58].

Unlike the used optimizers and loss functions, the used augmentation methods were disclosed more prevalently across the papers. Many papers used transformations such as flips [54], [58], [59], rotations [54], [57], [58], crops [57], [58], scaling [54], distortions [57], [59], gaussian noise [59], as well as altering brightness [57], [58] and color [54], [59]. In addition to these more conventional methods, two of the articles by Wang et. al. [53] and Li et. al. [58] employed generative adversarial networks to increase the size of the datasets. This shows the importance of augmentation methods in training segmentation models in the domain of wood industry where data is often very sparsely available.

Despite these useful findings the number of papers they were derived from was very limited and thus offer very limited insight to what methods we should use for training our wood feature detector. This is why I performed the hyperparameter survey in chapter 4. As it happens, the U-net model developed in 2015 by Ronneberger et. al. was originally developed for biomedical image segmentation [31]. In the next chapter I will perform my own study in the field of wood processing by using the training strategy I discovered in section 4.3.

6 Wood panel defect segmentation

In this chapter I collected and annotated an image dataset of wood panels with a white finish for the training and testing of the segmentation model. The purpose of this chapter is to design a processing pipeline for the purpose of the final quality inspection task of spruce interior panels. In other words, this chapter answers a simple research question: Can a deep learning based semantic segmentation model be used for the final wood panel quality inspection task? The pipeline can be divided into four stages: image acquisition, preprocessing, segmentation, and contour analysis. The aim of this pipeline structure is to capture the advantages of deep learning-based image processing while retaining the ability to adjust quality parameters without retraining and a high degree of transparency in the decision-making process. The proposed pipeline is illustrated in figure 6.1.

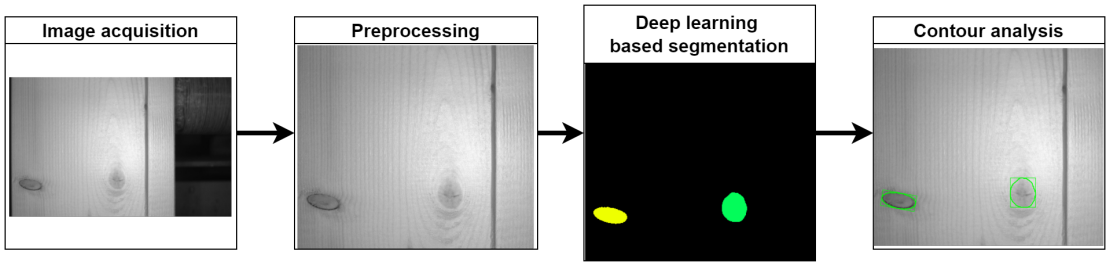

Figure 6.1: The proposed quality inspection pipeline. First the panel is photographed. Then the panel is cropped from the photo, resized and normalized. In the third step the section is segmented with a segmentation model. The segmentation results are then analyzed using contours.

In this chapter I will design and implement the proposed pipeline in a python programming environment. Building the complete system in an automation envi-

ronment is frankly out of scope for this thesis. Building a real time capable quality inspection system based on CNNs would require specialized hardware and testing the system against a human inspector would require additional modifications to the production line. Because of this limitation testing the proposed system will be restricted to the python environment. This chapter will first introduce the reader to wood grading. The following subchapter describes how the data was collected and annotated. The third subchapter shows how the results of the tumor segmentation hyperparameter study are used to train a segmentation model for extracting features from a wood panel surface. Finally, the fourth subchapter shows how the segmentation maps produced by the segmentation model can be used for determining the quality of the panel.

6.1 Wood grading

This study focuses on the structural defects of the wood prevalent in the TK-spruce panel. These defects are cracks, knot holes, edge clefts, resin pockets, and pith streak. In addition to these defects, we will also segment two different kinds of knots: healthy and dry knots. Examples of the segmented features are shown in figure 6.2.

TK is a quality class of spruce timber. In the Finnish language the abbreviation TK stands for "terveoksainen kuusi" which translates to English as "healthy knotted spruce". The TK quality class has clear requirements the timber must abide by. The requirements regarding the afore-mentioned defects according to puuinfo.fi [61] are the following:

1. A knot may not exceed 30% of the panel's width. Horned and leaf knots on the other hand are allowed a span of at most 50% of the panel's width. Knots must be solid. A solid knot can be either healthy or dead but with a fresh

color.

2. One detached knot 8mm in width is allowed at the edges of the panel's backside and none on the frontside.
3. Knot holes are not allowed.
4. Resin pockets can reach an accumulative maximum length 120mm in a span of 1 meter within a panel. No individual resin pocket however can exceed a length of 60mm.
5. Hair cracks at most 0.5mm wide with an accumulative length of at most 25% of the panel's length are allowed. Splits are allowed at the ends of the panel if their length don't exceed the width of the panel. If the panel has a tongue and groove at its ends, then the splits can be at most $\frac{1}{2}$ of the panel's width.
6. The length of a pith streak is allowed to be at most 50% of the panel's length.

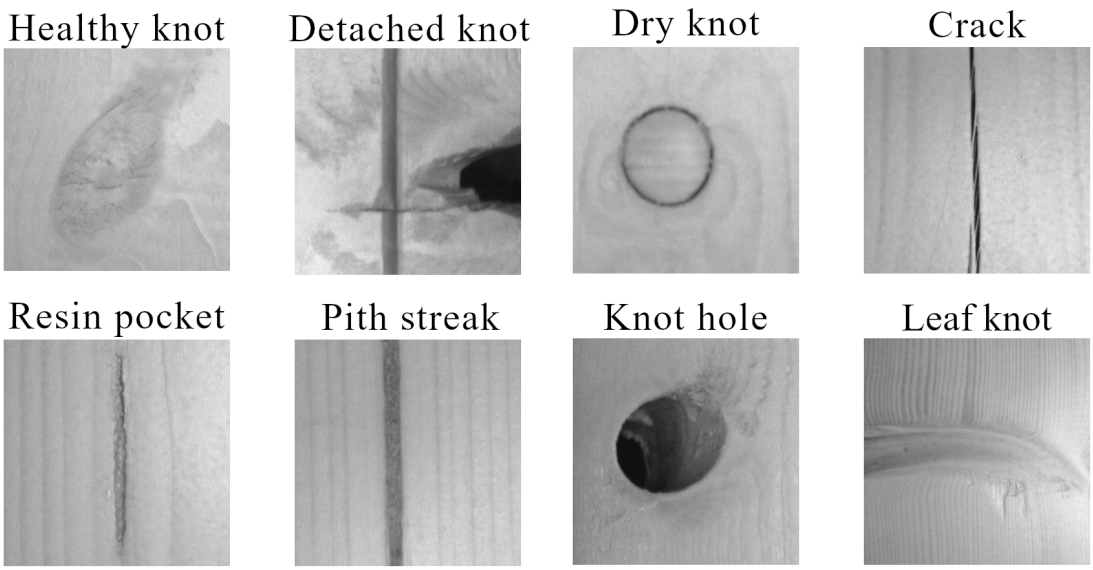

Figure 6.2: Examples of the wood defects and features looked for in this thesis.

However, the quality card of Maler Oy [7] regarding TK-spruce panels differs slightly from Puuinfo's requirement when it comes to the detached knots at the panel's edges. Maler allows at most one 8mm detached knot in 2-meter span on

the front side of the panel. While it would be advantageous to detect many of the described defects either before or right after or even before planing the panel and before applying the varnish, many of the defects can appear at multiple stages of the production process. The planing process itself could reveal resin pockets and pith streaks for example. The raw material is also subject to varying humidity levels and temperatures which can cause the panels to crack and loose knots to fall out. This creates a need for robust quality inspection to multiple stages in a wood panel's production process.

6.2 Data

6.2.1 Collection

The collection of the dataset was conducted over the span of 4 months, between November 2022 and March 2023. A camera was placed on top and near the end of the production line such that the panels would pass across the camera's field of view. The camera used in the experiment was a Cognex In-Sight D902C. The camera featured an inbuilt calibration functionality which was used in the data collection process. The object or product of the study was a healthy knotted spruce panel with a white lacquer finish.

The camera was fitted with a lens with an 8 mm focal length. The camera's resolution was 1920x1200 pixels. The camera was placed approximately 19 centimeters above the panel surface. The trigger mechanism was implemented with a combination of a light sensor and a time relay. Whenever the light sensor detected the presence of a panel the relay would send trigger pulses to the camera. Over the span of four months over 200 000 photographs of the white spruce panel were captured. Of these 200 000 pictures appr. 6200 pictures were sampled based on relevant features present in the photos.

6.2.2 Preprocessing

The photographs were calibrated with the inbuilt calibration tool of the D902C camera to correct the lens distortion caused by using a lens with relatively short focal length.

The raw photos were collected from the production as the panels were in motion. The motion caused the panel to often pass through the camera's field of view at an angle. And so, the position of the panel would shift in the camera's FOV. The panels would deviate from each other in width as well. Because of these factors the panel section could not be extracted with a simple static crop. Instead, an adaptive cropping method was developed for this task.

The photographs were mostly collected from the mid-section of the panels and photos containing the panels' ends were rarer. The panels would pass through the image frame from up to down. The first step in the cropping process was to recognize whether the picture was of an end section of a panel or midsection. This step was implemented by relying on the fact that the panels were white, and the background was dark. So, a section of the image where the panel would most of the cases be present. Then a horizontal mean was taken from the chosen ROI. The mean curve would have a significant jump as the image shifts from background to panel. If a rapid enough shift in pixel brightness was detected the image was cropped horizontally where the shift occurred. The preserved section was chosen by the polarity of the shift.

Next the angle of the panel was measured. This was done by taking vertical means of the pixel values near the bottom and top of the picture from left to right and right to left. The ROI for these means was the bottom and top 50 pixels of the photographs. The angles were determined based on their difference in x and y axes. Two angles were therefore calculated. One from each side of the panel and the photo was rotated the amount of the lesser angle. This was done due to the presence of the

edge clefts which when positioned near the bottom and top the image frame would result in exaggerated angles. After the image was rotated it was cropped again as the rotation would introduce black borders to the photos.

After a possible horizontal crop and angle correction the background left and right of the panels was cropped out as well. This was again done by taking a vertical mean at the height of the remaining photo. The first upward shifts from dark to bright pixel values from both sides of the photos were used at the points in the x-axis where the photos were cropped vertically. The resulting photographs were now angle corrected and free of any background elements.

6.2.3 Annotation

This thesis focuses on finding features and defects related to the wood rather than the lacquer /finish or planing of the panel. The main features/defects recognized during the data collection and annotation phase were cracks, dry knots, healthy knots, spiked knots, resin pockets, edge clefts, knot holes, and pith streak.

The features were annotated using the open-source annotation tool LabelMe [62]. The tool was used for creating a set of polygons for each image. Each polygon encapsulated a single feature/defect area and was given a label. The secondary objective of this thesis was also to produce a high-quality dataset for the wood feature detection task. The annotation process was a laborious, mind-numbing experience that took approximately a month's worth of work to complete. The amount of time spent on annotating each photo was non uniform. The time used for annotating a single photo was highly dependent on the number of segmentations and number of points required to mark a segment in the picture.

6.2.4 Dataset compilation

Because some of the defect and feature types present in the images are rather similar to each other I decided that some classes should be fused in the deep learning phase and be separated in the postprocessing phase. For example, knot holes and edge clefts are almost identical to each other in that they have the same cause, and their only difference is their location in the photograph. A knot hole is a region in the image that does not extend to the edge of the panel whereas edge clefts are found exclusively at the edge of the panel. While these are fundamentally the same defect, they have different rejection criteria in the panel grading process. The same can be done for knots, and spiked knots as they are visually similar but have different dimensions.

The label masks were compiled using *labelme.utils.shape.shape_to_mask* -function [62]. The joining of class labels was done at this phase. A mask was constructed for each polygon in an image. The masks were then joined with an elementwise maximum function. The masks were saved in npy-format.

The dataset's class composition is shown in table 6.1. Many of the defects were oversampled from the original sample of 200 000 photos and this table does not represent the actual prevalence of these classes in the TK-spruce panels.

Table 6.1: Class statistics of the annotated TK-spruce dataset. Count = number of polygons, Prevalence(%) = percentage of images where the class is present, Area(%) = class pixel percentage of the total number of pixels.

Class	**Count**	**Prevalence(%)**	**Area(%)**
background	6219	100	97.62
Healthy knots	8163	71.73	1.45
Dry knots	2051	24.71	0.16
Pith streaks	890	12.93	0.24
Resin pockets	3287	30.2	0.21
Cracks	2793	15.15	0.08
Knot holes	2039	29.92	0.23

6.3 Feature segmentation

6.3.1 Data loading and augmentation pipeline

The data loading pipeline was constructed using a custom dataset object extended from PyTorch's torch.utils.data.Dataset object. First the image and mask are opened with Open-CV and numpy respectively. The image and mask are then both resized to 384x384 with the Albumentations [49] library's resize function. After resizing the image and mask if the data is used for training a series of augmentations are performed, also with the Albumentations library's functionalities.

Finally, the input image is normalized with the Albumentations.normalize function with means [0.485, 0.456, 0.406] and standard deviation values: [0.229, 0.224, 0.225] for red, green and blue image channels respectively.

The training phase augmentations are implemented in the following order:

1. Random horizontal flip ($p = 0.5$)

2. Random vertical flip ($p = 0.5$)

3. Random resized crop ($scale = (0.7, 0.9)$, $height = 384$, $width = 384$, $p = 0.5$)

4. One of ($p = 0.8$)

    - Random brightness and contrast ($brightness\_limit = (-0.1, 0.1)$, $contrast\_limit = (-0.1, 0.1)$, $p = 0.5$)
    - Random gamma ($gamma\_limit = (80, 120)$, $p = 0.5$)
    - Random Gaussian noise ($var\_limit = (100, 200)$, $mean = 0$, $p = 0.3$)

5. One of ($p = 0.5$)

    - Random shift, scale and rotate ($shift\_limit = 0.2$, $scale\_limit = 0.3$, $rotate\_limit = 0$, $p = 0.5$)
    - Random rotate ($limit = (-10, 10)$, $p = 0.3$)
    - Random elastic transformation ($alpha = 2$, $sigma = 50$, $alpha\_affine = 60$, $p = 0.3$)

the geometric and affine transformations use the cv2.BORDER_REFLECT and cv2.INTER_NEAREST as arguments for the 'border_mode', and 'interpolation' parameters.

Translation, zoom and shear are omitted from the augmentation pipeline. Translation is essentially the same as shifting since both just move the image on x and axis. Zooming on the other hand is similar to cropping since after both operations the result is an area of the image. When the image is resized after cropping the effect is the same as zooming into a point in the image. Shear on the other hand has a similar effect to elastic deform and so it would be redundant to have two similar augmentation methods in the pipeline. The effects of the augmentation methods are depicted in figure 6.3.

The border mode was set to reflect since black areas at the image borders are associated with detached knots and having a constant value in the image could hinder the training process.

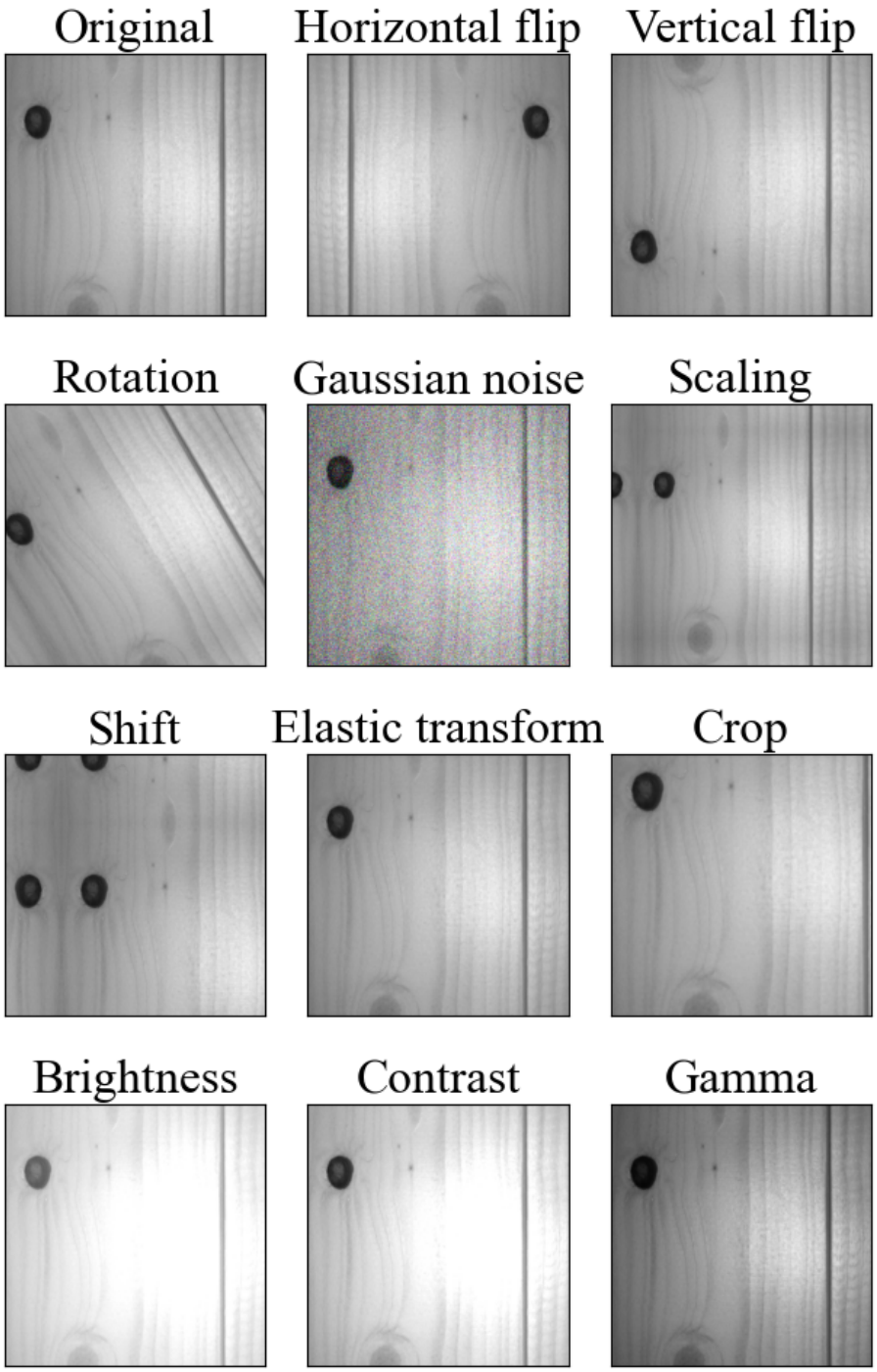

Figure 6.3: Visual demonstration of the used augmentation methods.

6.3.2 Training the models

For the training phase the dataset of total 6219 images were randomly split into training, validation, and test sets. 80% of the data was used for training and 10%

for both validation and test sets. The data was not split in a stratified manner since many of the classes coincide with each other in the images so a perfectly balanced split may not be achievable.

In the training phase we use the parameters we found in the literacy review. The most popular optimizer and loss function in tumor segmentation papers were the Adam optimizer and Dice loss. The learning rate set for the Adam optimizer was $1e^{-4}$. The training is implemented by using the pytorch lightning framework [50] and the models are constructed by using the segmentation models pytorch library [51]. I trained three models with approximately the same number or parameters. A U-net, a Deeplabv3, and a Segformer.

The U-net model was created with 3 input channels so the created model could also be applied to color images via transfer learning in the future. The number of output channels was set to 7. The chosen encoder was ResNet-34, and it was initialized with weights pretrained with ImageNet. The model used the first five encoding blocks of the ResNet34. I halved the number of channels of the decoder from chapter four's [512, 256, 128, 64, 32] to [256, 128, 64, 32, 16]. In the decoder block batch normalization layers were used between the convolution operations and activations.

Similarly to the U-net model the Deeplabv3 model was also initiated with a pretrained ResNet-34 encoder. The other parameters were left at default values as in the documentation of the library. For the Segformer model on the other hand I used the mit_b2 encoder which is one of the encoder architectures introduced in the original Segformer paper [39].

The halving of the U-net's decoder's channels reduced the number of parameters from appr. 30 to 24.4 million. The number of parameters of the DeepLabV3 and Segformer-Mit_B2 models were again appr. 26 and 24.7 million respectively.

The training, validation, and testing were performed on a desktop PC with an

AMD R5 5600X processor, 32 gigabytes of RAM and an Nvidia RTX 3080 GPU with 10 GBs of VRAM. The U-net and Deeplabv3 models were trained for 300 epochs and the Segformer was trained for 100 epochs. The used batch size was 16. The training took approximately 5, 12 and 2 hours for the U-net, Deeplabv3 and Segformer respectively. The models were trained for a set number of epochs and the checkpoints that achieved the lowest validation losses were then chosen for testing.

6.3.3 Segmentation results

The trained models achieved varying results. The models' class Dice scores are compared in figure 6.4 and table 6.2. The models' loss curves are shown and compared in figures 6.5, 6.6, 6.7, and 6.8. From the figures we can see that the model based on the U-net architecture performs the best. It achieves the best Dice score in all classes. The DeeplabV3 and Segformer models achieve comparable results to the U-net model with the exception of the crack class. In fact, the other two models perform significantly worse in the crack class to the point where it cannot be said that models would be able to detect this class of defects.

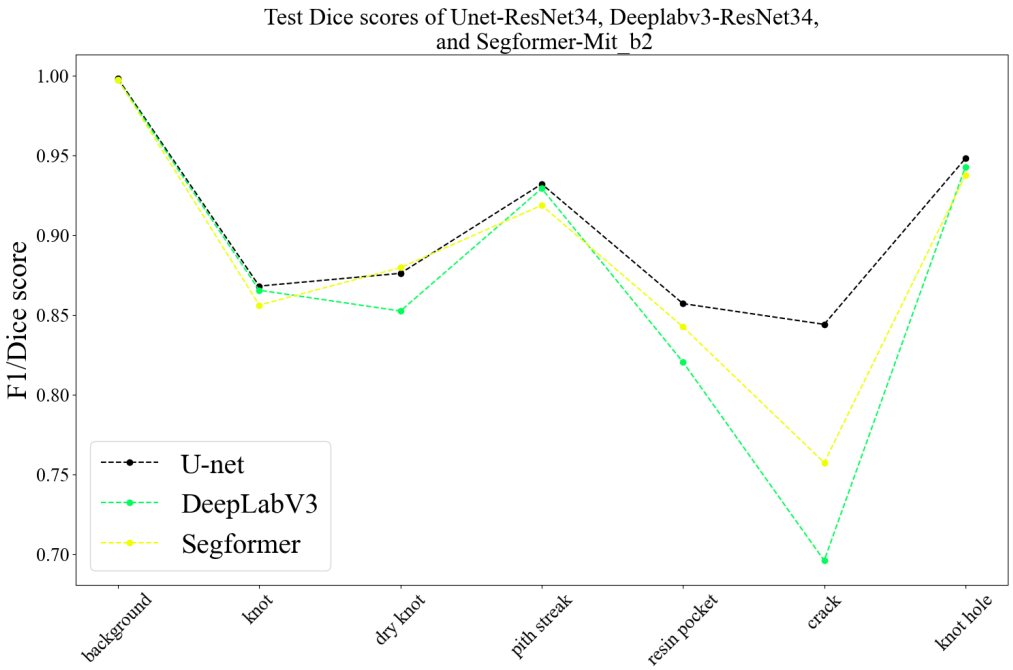

Figure 6.4: TK class test Dice score comparison between the models

Table 6.2: TK class test Dice score comparison between the models.

Class	Unet-ResNet34	DeepLabV3-ResNet34	Segformer-Mit_b2
background	**0.998**	0.997	0.997
Healthy knots	**0.868**	0.865	0.856
Dry knots	0.876	0.852	**0.88**
Pith streaks	**0.932**	0.929	0.919
Resin pockets	**0.857**	0.82	0.842
Cracks	**0.844**	0.696	0.757
Knot holes	**0.948**	0.943	0.937

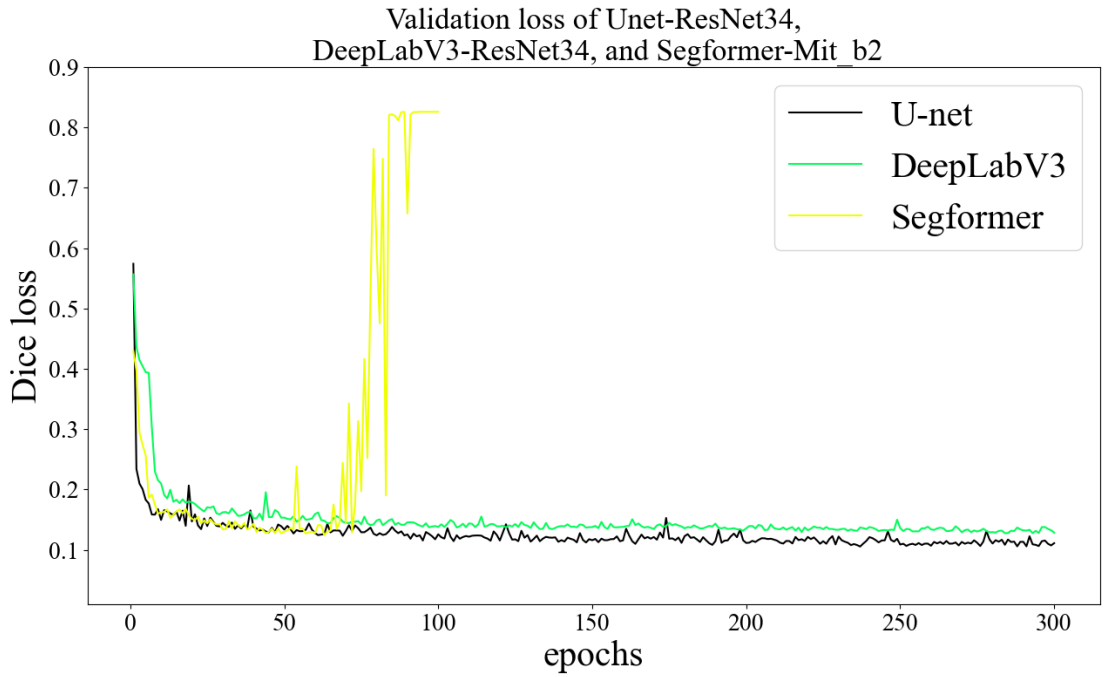

Figure 6.5: Comparison of the validation loss curves on the TK dataset.

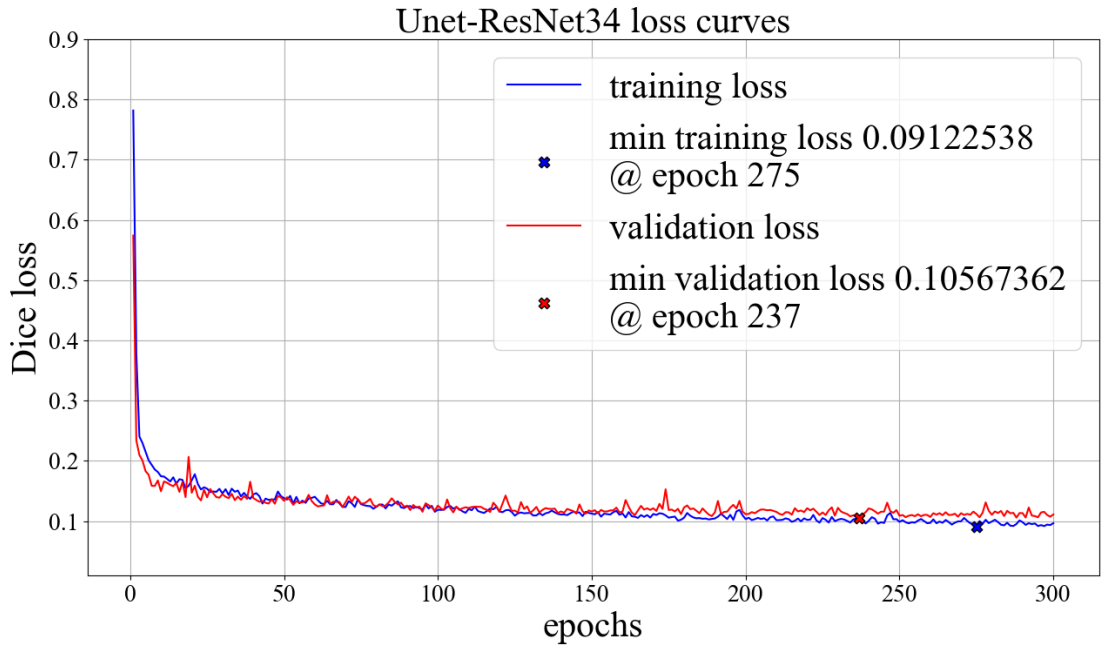

Figure 6.6: Training and validation loss curves of the Unet-ResNet34 model on the TK dataset.

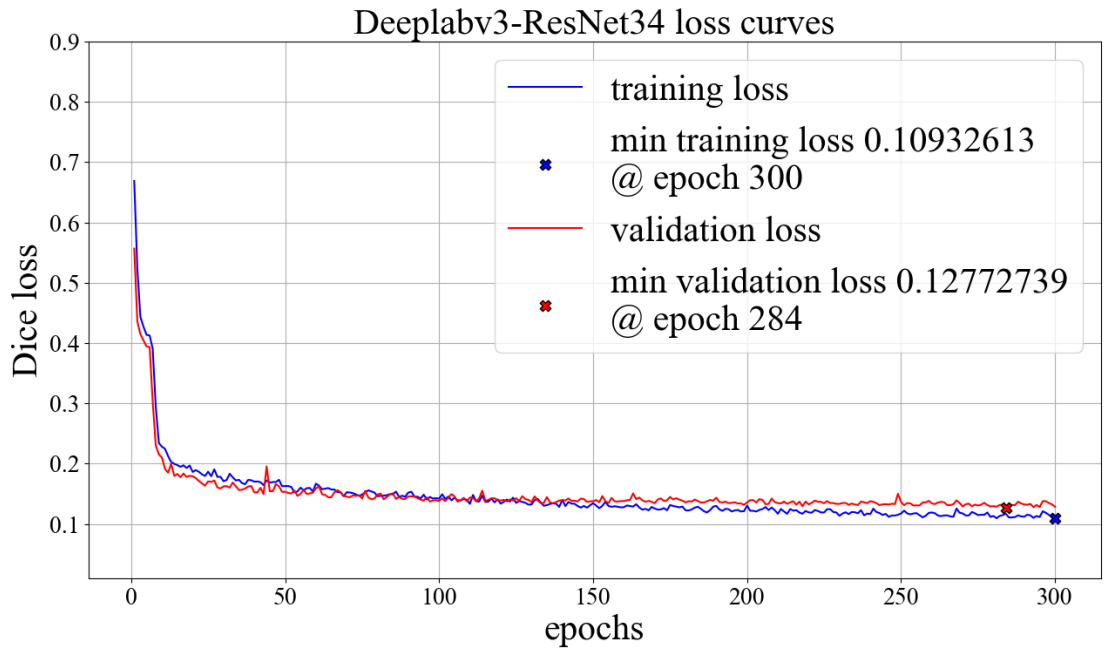

Figure 6.7: Training and validation loss curves of the DeepLabV3-ResNet34 model on the TK dataset.

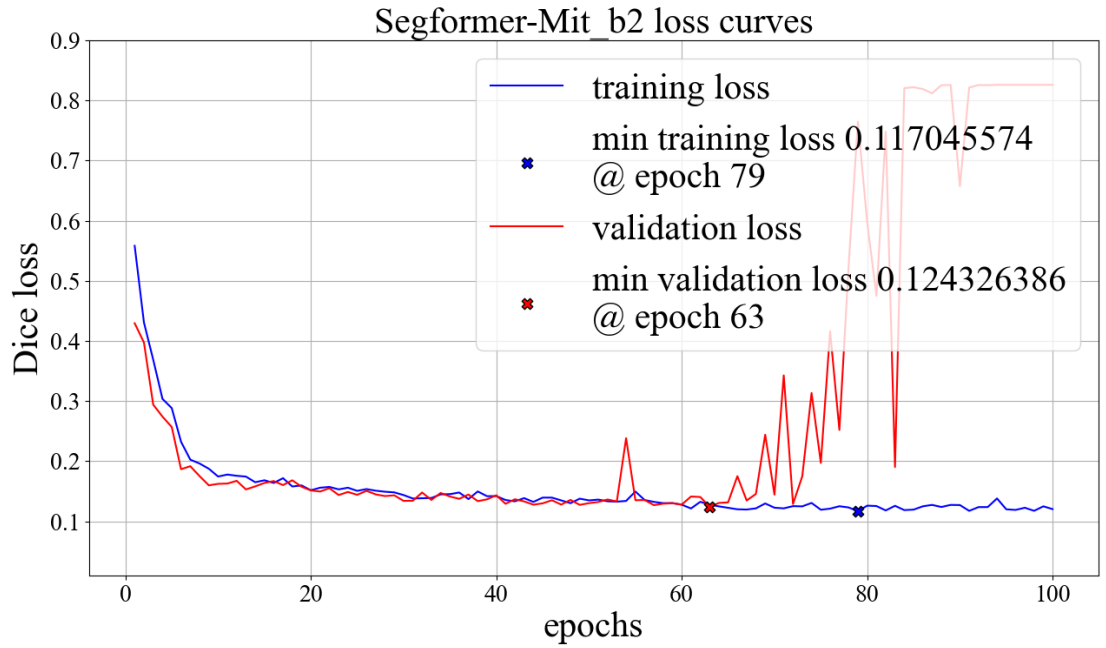

Figure 6.8: Training and validation loss curves of the Segformer-Mit_b2 model on the TK dataset.

After examining the loss curves, we can conclude that the U-net and Deeplabv3

based models are quite stable to train. The validation loss behaves very predictably and is closely coupled with the training loss. On the other hand, the Segformer model is an entirely different matter. Its validation loss starts to deteriorate rapidly after the 60th epoch. At this stage the model seems to start overfitting to the training data.

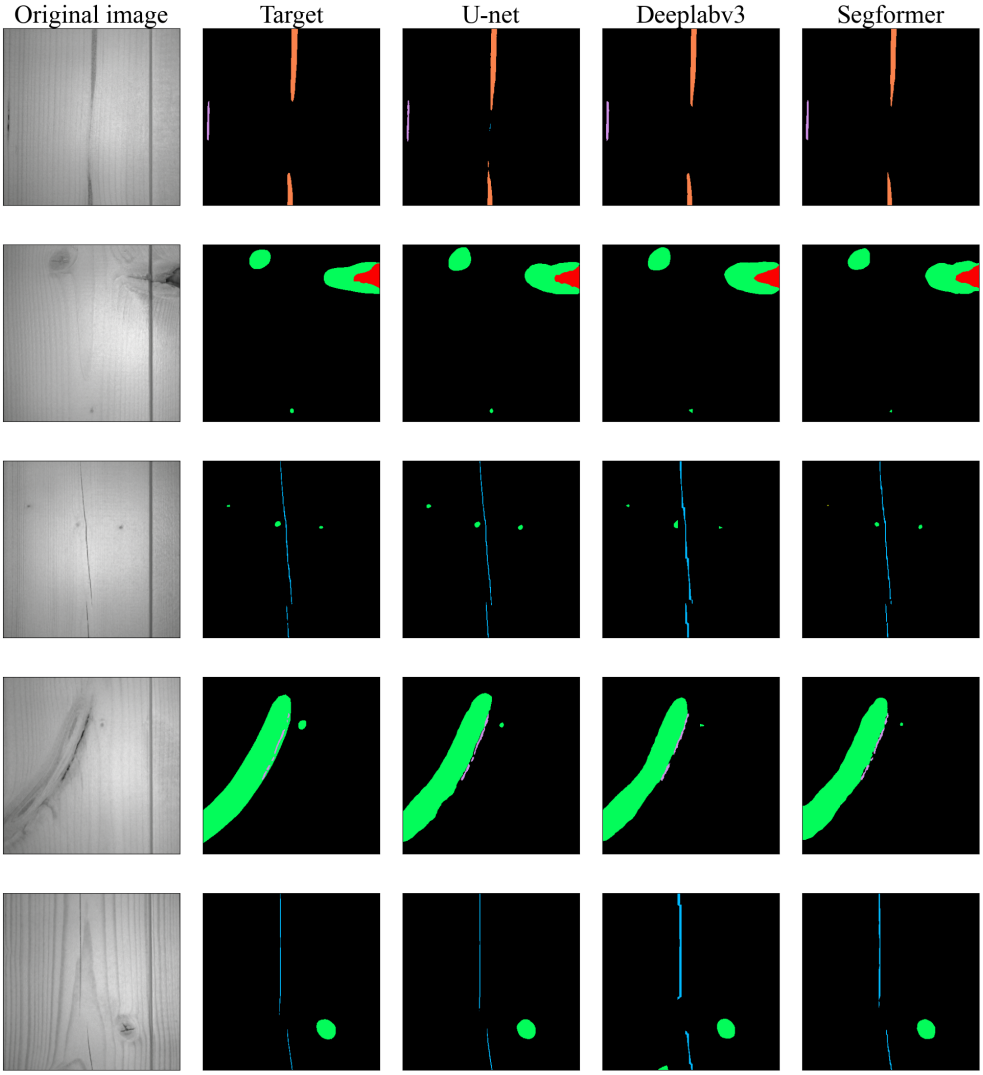

Figure 6.9: Comparing the TK test segmentations produced by the models.

The Segformer model's failure to achieve stable learning with this data could be because of its transformer-based architecture. The DeeplabV3 and U-net models are convolution layer based and so they have a stronger inductive bias embedded into

their architectures than the Segformer. The convolutional architecture forces these models to learn certain types of features from the data and so they are much more regularized by their architecture than the Segformer. In the original Segformer paper the models were trained with ADE20K, Cityscapes, and COCO datasets. The ADE20K dataset has 25 574 training images and 2 000 validation images. The Cityscapes dataset has 5 000 finely annotated and 20 000 coarsely annotated examples. The COCO stuff dataset has 164 000 images. Since our dataset of wood defect images is small relative to the Segformer's intended dataset size we can conclude that we might need more data to realize Segformer's full potential in this application domain.

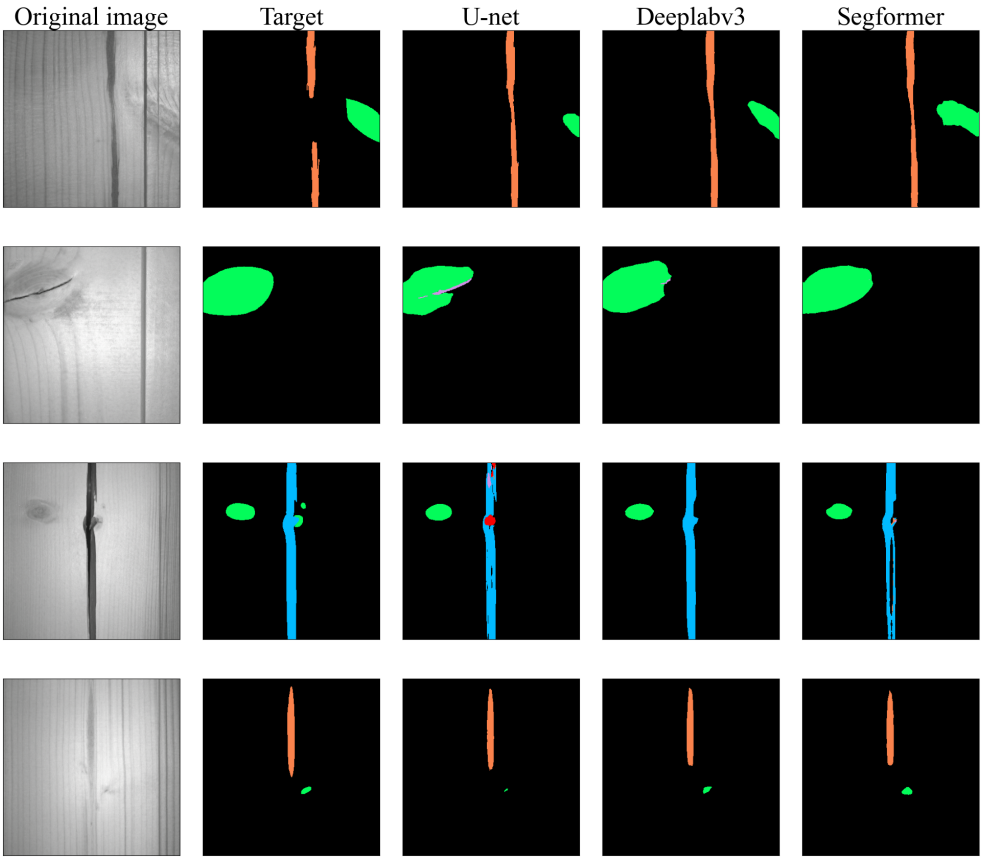

Figure 6.10: More TK test segmentations produced by the models.

When taking a look at the segmentations in figure 6.9 we can see that the

Deeplabv3 model's crack segmentations appear more jagged and thicker than the other models'. Other than that, the models appear to perform quite similarly when examining the segmentations.

The segmentations produced by the models sometimes differ from the target masks. This can be seen in figure 6.10. On the third row the U-net model has produced a knot hole region in the middle of the very wide crack. While the crack itself is detected as it should be the knot hole region will produce a false positive in the grading stage while the crack region will produce a true positive. The falsely detected knot hole therefore won't affect the final grading of the panel but makes it seem like the system is overly sensitive to the knot hole class even though these types of misclassifications make no difference in practice.

Sometimes the regions are very similar to each other but in cases such as in the figure 6.10's top and bottom rows the lengths of the predicted pith streak regions differ from their target counterparts. In edge cases sometimes an argument can be made about choosing any one of the proposed segmentations and each grading result can be equally correct. These cases also make it harder to quantitatively evaluate the final grading results. Since the U-net based model achieved the lowest validation loss, I chose it for the quality inspection system's segmentation stage. The U-net's performance is further examined in the next section.

6.3.4 Examining the U-net model

Based on the model results in the previous section, the U-net based segmentation model can be deemed the most suitable option for the quality inspection system's segmentation stage. The metrics used for further evaluating the U-net model are recall, precision, false positive rate, false negative rate, intersection over union and Dice score. These metrics were calculated for each class and are shown in table 6.3. In addition to these metrics a confusion matrix was also plotted to see how the

image pixels were usually misclassified. The confusion matrix is in figure 6.11.

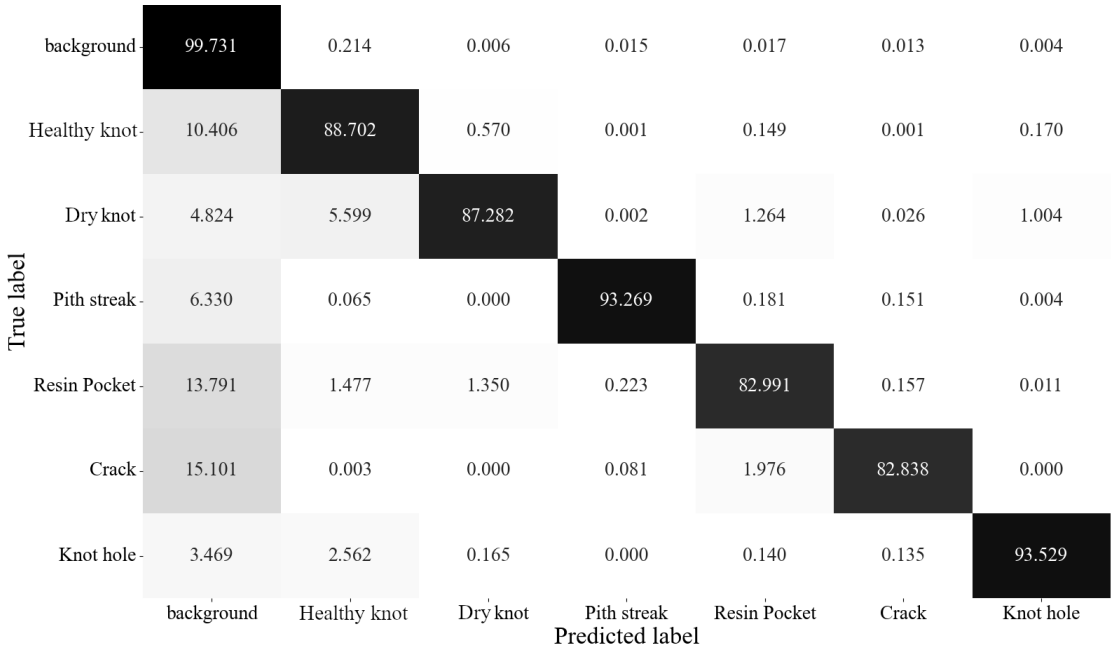

Figure 6.11: The U-net model's pixel prediction confusion matrix on the TK test set.

As we can see from the first metric graph, the model achieves decent performance in every class. The Dice score exceeds 0.8 in every class and the false positive rate is very low. However, the false negative rates are not as impressive. The low FPR leads to some impressive precision numbers, however recall suffers from the high FNR. This is alarming since a high number of false negatives implies that many defects or features pass undetected by the segmentation model, or classes are sometimes misclassified as other non-background classes. Upon a closer inspection of the confusion matrix, we can conclude that most of the confusion between classes is between the background/negative class and the feature classes. A closer look at the actual segmentations (figures 6.14, **??**, and **??**) performed by the model should tell us more about the nature of the model's misclassifications.

By observing the segmentations performed on the test data we can conclude that the model performs well on unseen data. The confusions between the background class and the other classes appears to be happening at the edges of the feature

Table 6.3: TK test set metrics of the Unet-ResNet34 model.

Class	Recall	Precision	FPR	FNR	IoU	Dice
background	0.997	0.998	0.095	0.003	0.995	0.998
Healthy knots	0.887	0.849	0.002	0.113	0.766	0.868
Dry knots	0.873	0.878	0.0	0.127	0.779	0.876
Pith streaks	0.933	0.932	0.0	0.067	0.873	0.932
Resin pockets	0.83	0.885	0.0	0.17	0.749	0.857
Cracks	0.828	0.861	0.0	0.172	0.731	0.844
Knot holes	0.935	0.961	0.0	0.065	0.901	0.948

contours. By comparing the shapes of the label and prediction regions we can observe that these shapes often differ from each other. This confusion at the region edges is the main reason for interclass confusion in the test data. Since the defect/feature classes rarely border each other, the confusion happens towards the background class and vice versa. The confusion from the background class to the other classes is masked in the results by the large volume of the background class pixels in the data leading to a high portion of true positives.

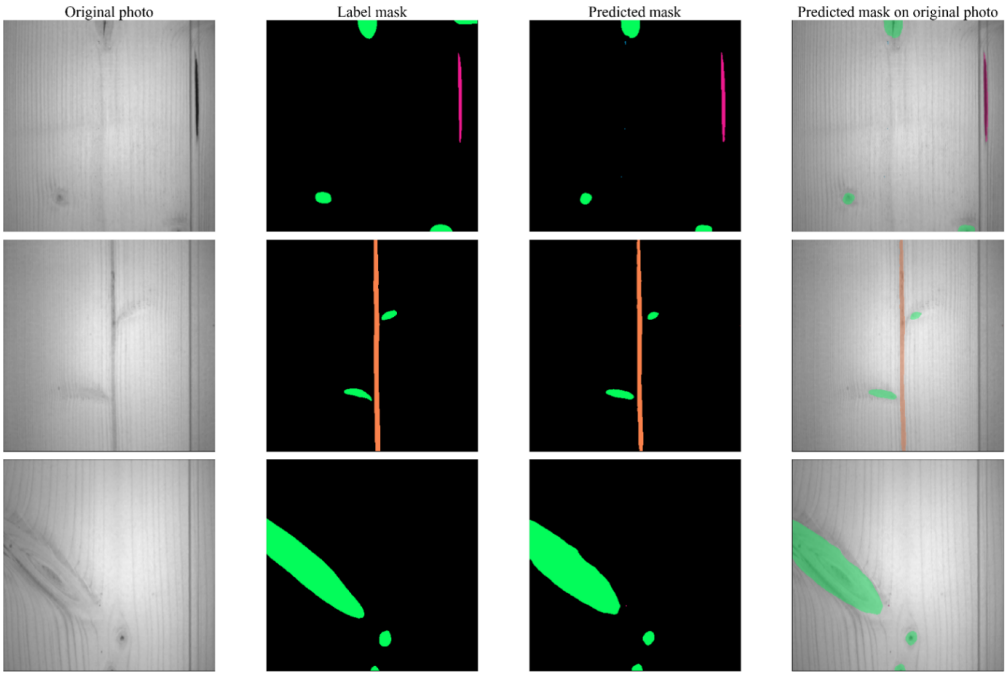

Figure 6.12: U-net TK test set segmentations of pith streaks (orange), resin pockets (purple), and knots (green).

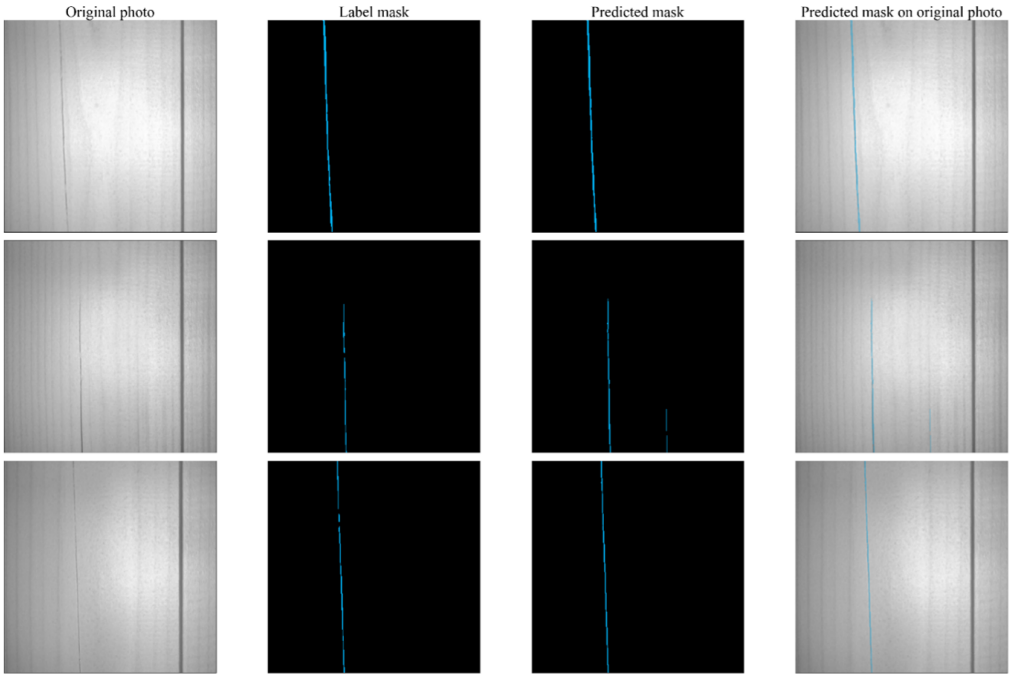

Figure 6.13: U-net TK test set segmentations of cracks (blue).

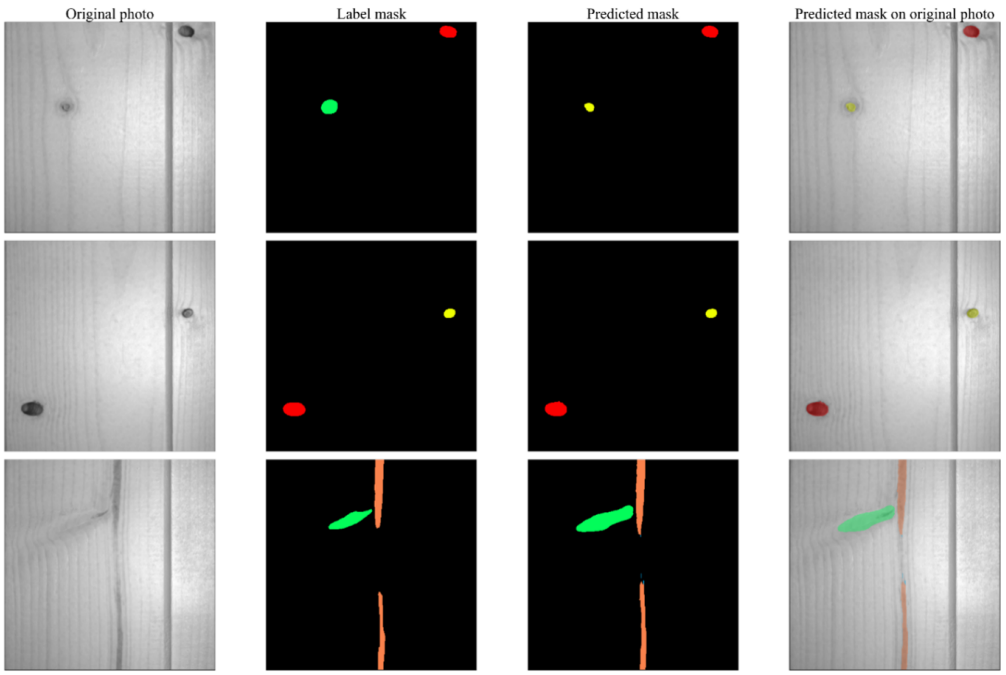

Figure 6.14: U-net TK test set segmentations of a pith streak (orange), knot holes (red), and knots (green).

Another key observation is the fact that the transition zone from the panel's

main surface to the tongue is not confused with the streak like classes (crack, resin pocket, and pith streak) despite having similar appearance. The transition zone is located on the right side of the photos. The zone suffers from bad lighting since the surface of the wood turns downward and is no longer perpendicular to the lighting direction. The model's ability to ignore this region implies that the model can learn to reliably ignore the optical effects of different types of panel surface profiles.

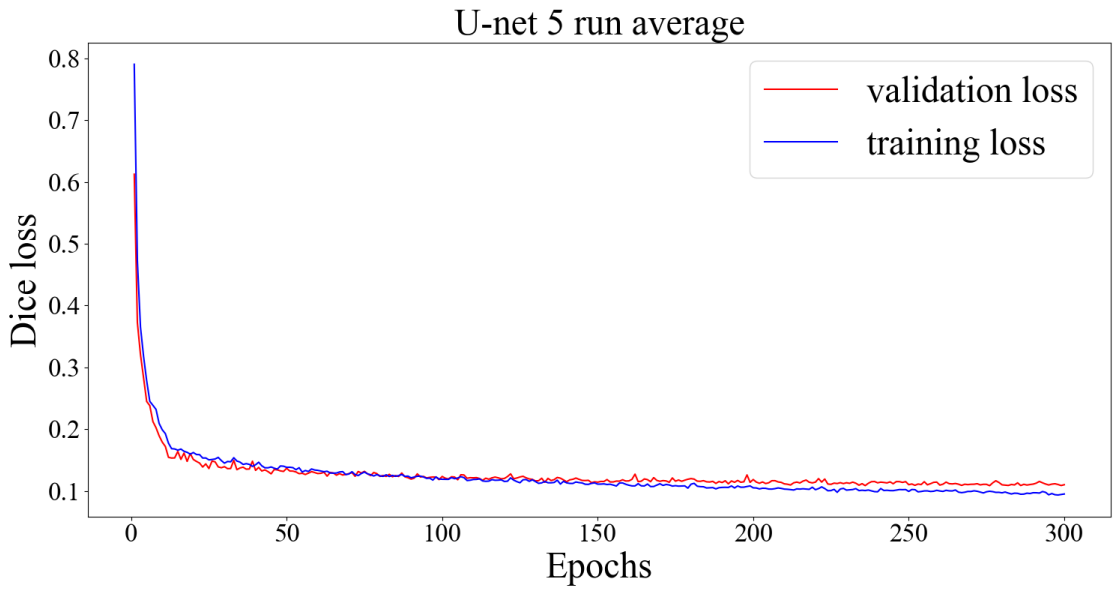

Figure 6.15: U-net model's training and validation mean loss curves of five TK training runs.

Figure 6.6 showed us how the training and validation loss developed during the training process. However, both training and validation loss curves are quite noisy. To obtain a less noisy plot of the losses I performed four additional training runs on the U-net model to obtain averaged loss development curves. The average loss curves of five training runs are plotted in figure 6.15. Figure 6.15 shows that validation loss overshoots the training loss at the start but approximately by the 100th epoch the training loss overtakes the validation loss. Both losses continue to decline, however, so even by the 300th epoch no overfitting has occurred.

To further evaluate the consistency of the segmentation model we can examine

the distribution of the loss produced on the test data. We can do this by plotting a boxplot and a histogram of the image specific losses.

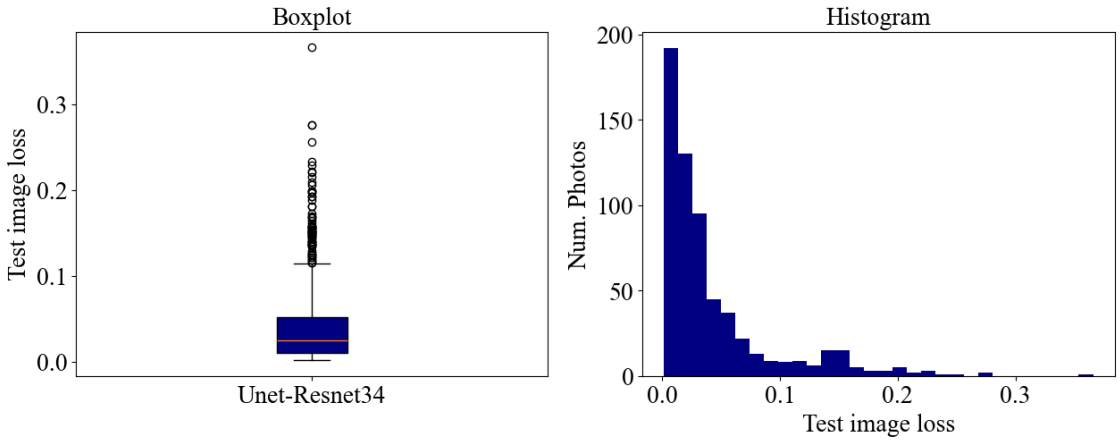

Figure 6.16: The U-net model's TK test set image loss boxplot and histogram.

From the plots in figure 6.16 we can see that the model performs quite consistently with the test data. The third quartile of the test loss is 0.052, meaning that the model produces a loss equal to or less than that on 75% of the test data. From the histogram we can see that the loss has a weak upper tail and most of the photos of the test data produce a loss less than 0.1. On the other hand 79 examples of the test dataset's 622 photos exceed a loss of 0.1 which could pose a problem at the contour analysis stage since the shape, location and size of the segmentation map's contours are the basis for the grading tools' operation.

6.3.5 Constraining the amount of data

To see whether more data would improve the segmentation results we can constrain the amount of data used for training. For this experiment I trained the Unet-Resnet34 model with 3.125, 6.25, 12.5, 25, 50, and 75 percent of the training data to see how the amount of data affects the training results. In figure 6.17 we can see that the validation loss development is affected by the amount of training data used and greatly affects the validation results. However, the effect seen in the

validation loss progression plot is relatively mild even when comparing the 12.5 and 100 percent runs. Only 621 photos were used in the 12.5 percent run. Keeping this in mind achieving a loss with only 621 photos comparable to using 4975 photos seems impressive.

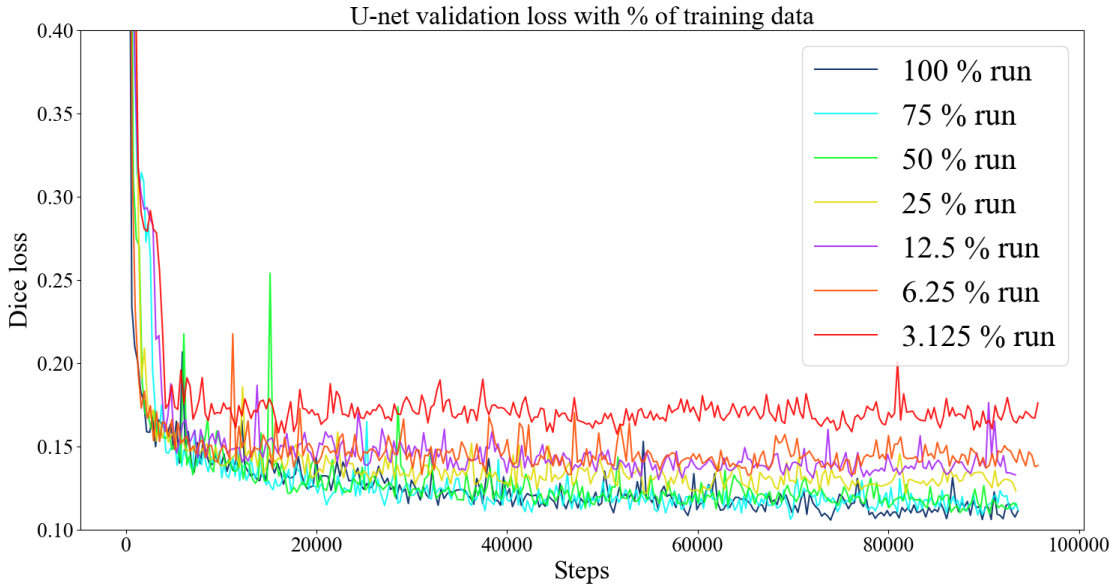

Figure 6.17: U-net model's validation loss development with different amounts of TK training data.

The loss curve is an average over the dice loss of each class. This means that the amount of data used could affect the loss on each class differently. In figure 6.18 the dice loss of each class is plotted against the training data percentage used. We can see that the classes most affected by the training data amount are resin pocket, knot, dry knot, and crack. We can see that the crucial pivot point after which the dice score is not greatly affected by the amount of used training data is around 25 percent. This is equivalent to about 1 200 photos. This is a surprising result given that usually adding more data improves performance. Since adding more photos past 1 200 affects the results very little we can conclude that the collected data has successfully captured the features present in the wood panels and adding more similar data points to the dataset might not improve performance.

Figure 6.18: U-net test Dice score of each class vs the used percentage of TK training data.

6.3.6 Role of data augmentation and transfer learning

Here I will examine the importance of both transfer learning and the used augmentation pipeline to the training process of the Unet-ResNet34 model. To obtain the necessary information, I performed three additional training runs on the Unet-Resnet34 model. One where the encoder was initialized with random weights i.e. no transfer learning was used. In the second run the designed augmentation pipeline was not used. And in the third one both transfer learning and augmentation were omitted from the training process.

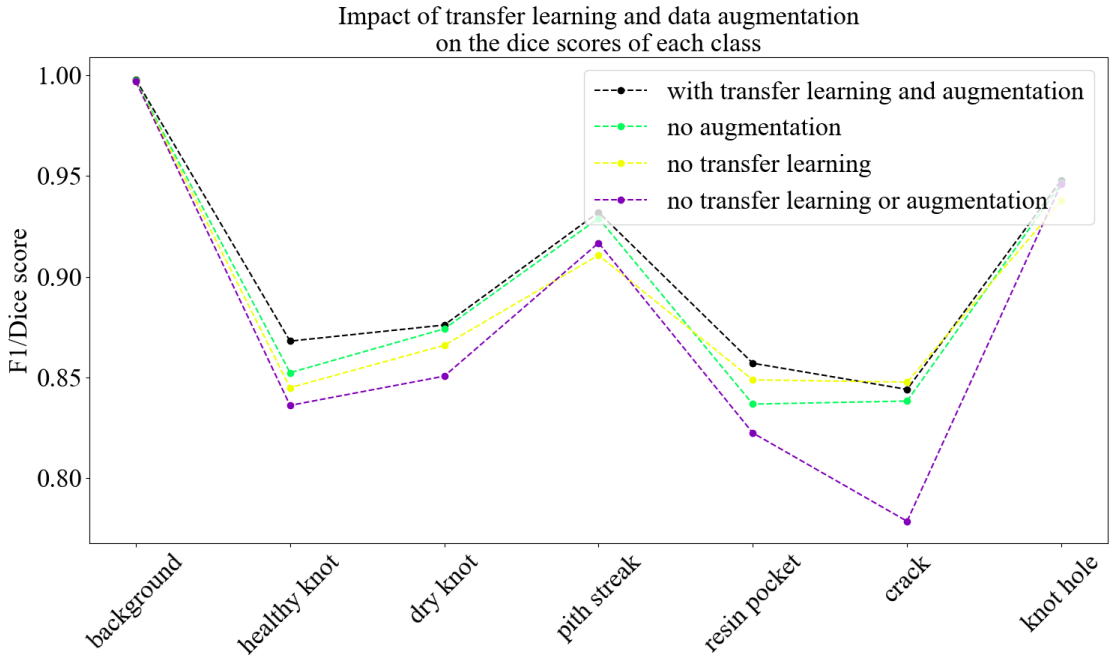

Figure 6.19: The impacts transfer learning and augmentation have on the test Dice score of each TK dataset class.

In figure 6.19 we can see that the training run where both transfer learning and augmentation were used achieves the best results and the one where neither was used performs the worst. The runs that omitted one of the techniques placed somewhere in the middle.

In figures 6.20, 6.21, 6.22, and 6.23 we can observe the training and validation loss curves of the different runs. These curves reveal that the usage of data augmentation ultimately has a much more significant impact on the development of the validation loss than transfer learning. The loss development curve without transfer learning is similar to the one where it is used. The losses decreased faster at the beginning of the training with transfer learning applied than without. Ultimately applying transfer learning to the encoder has a relatively small but still a measurable impact on the validation loss and test metrics.

Augmentation on the other hand had a very significant impact on the development of the training and validation losses. While the omission of the data augmen-

tation process from the training process has only a small impact on the test metrics and the validation loss curves its effects are very significant on the training loss. If we observe figures 6.22 and 6.23, we can see that the training loss gets much lower in the training runs where augmentation wasn't used. In fact, we can observe a slight upward trend in the validation loss curve on the right side of the graphs. This suggests that without data augmentation the Unet-ResNet34 model can start to overfit to the training set. This also suggests that the data augmentation pipeline could be the primary reason for why the training process is able to achieve comparable metrics even when trained on only a fraction of the training data as observed in the previous section.

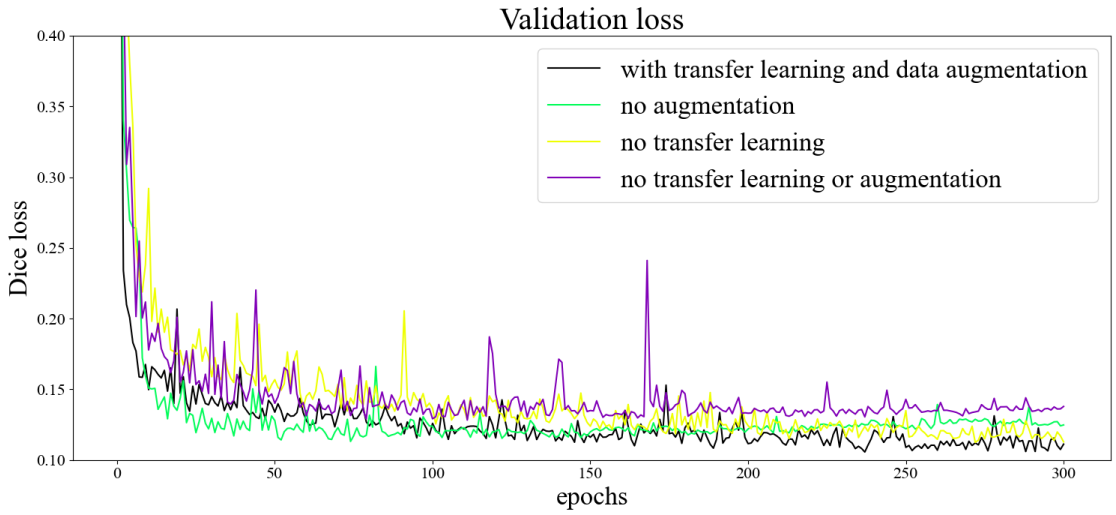

Figure 6.20: TK validation loss development comparison of augmentation and transfer learning combinations.

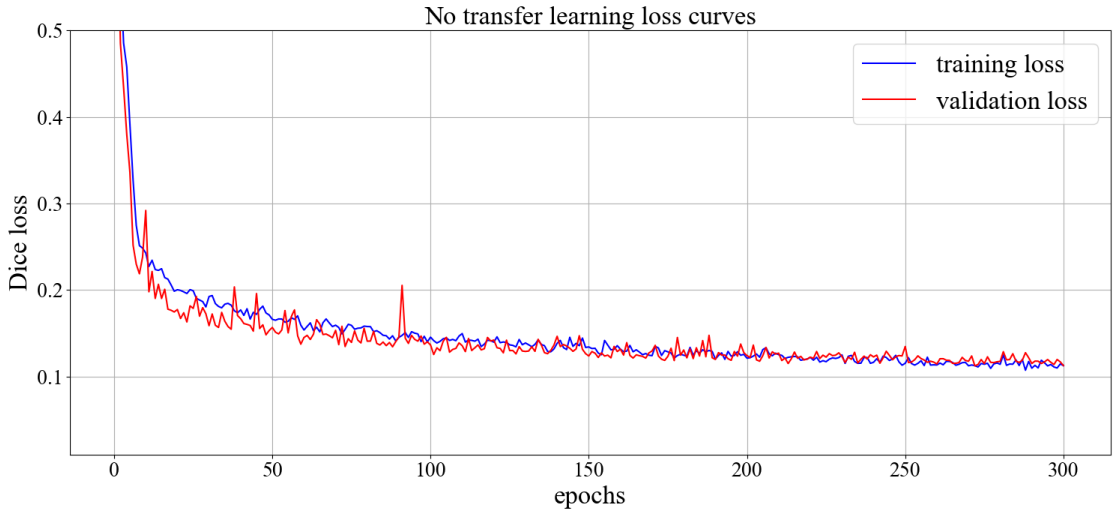

Figure 6.21: TK training and validation loss development of the U-net without transfer learning.

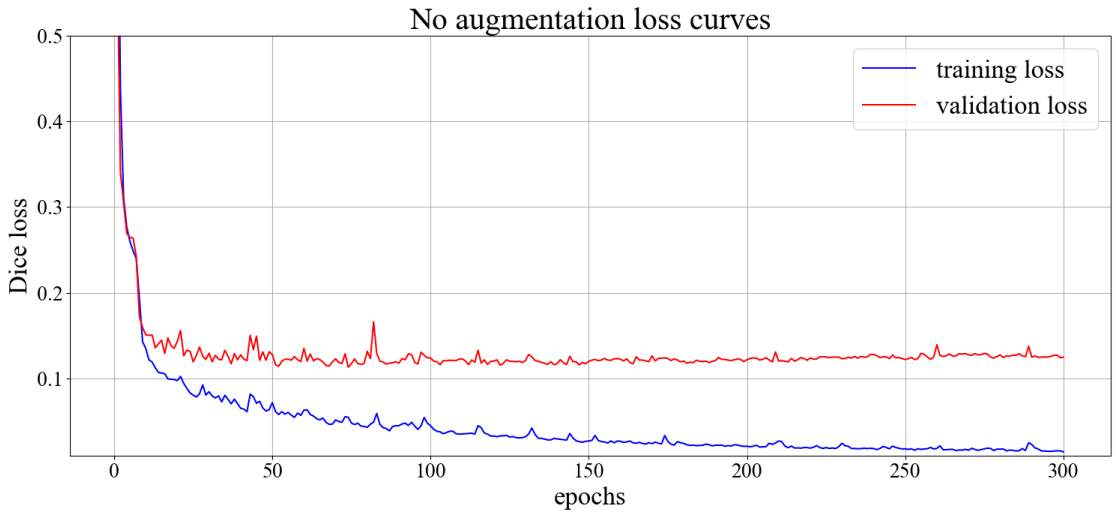

Figure 6.22: TK training and validation loss development of the U-net without data augmentation.

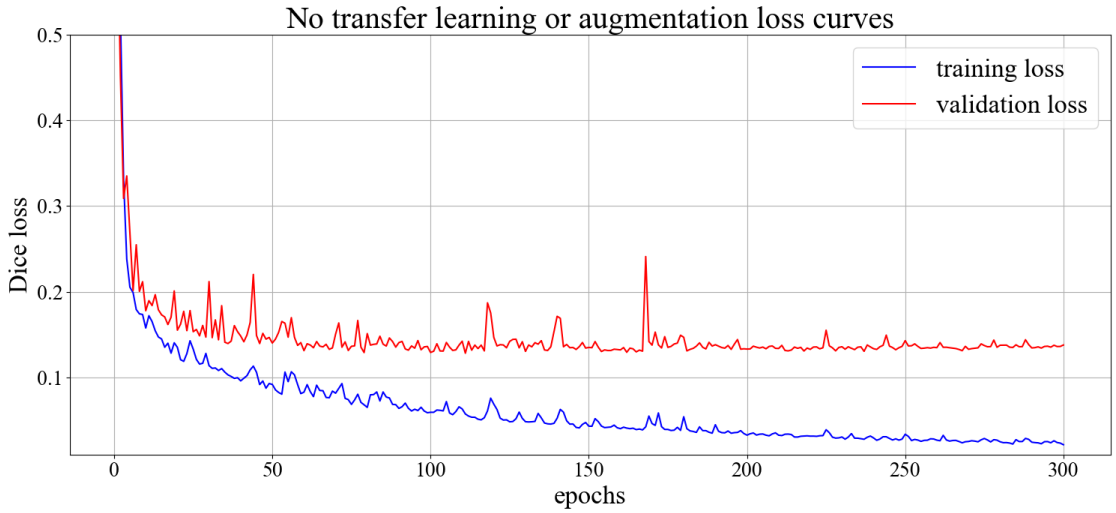

Figure 6.23: TK training and validation loss development of the U-net without data augmentation and transfer learning.

6.4 The grading system

The goal of this section is to design a system capable of using the segmentation maps of the deep learning model to make rejection decisions according to a predefined set of rules. The system designed here will implement the TK standard [7], [61] but in principle a set of rules can be implemented to adhere to any standard imaginable. The second purpose of this section is to demonstrate that a segmentation model can be used as a basis for quality control system for wood panels.

The base pipeline of this system consists of the segmentation stage and the contour analysis stage. A contour is a continuous curve enclosing an area of pixels sharing a common intensity or color. A contour can also be described as the boundary of said area [63]. First the images of size 384x384 are segmented with the deep learning segmentation model. Then the resulting segmentation maps are resized back to the original size of the input image. The maps are then joined together to form the map of the entire panel. The images are processed in a vertical orientation such that the longest side of the panel is in vertical orientation.

After the segmentation for the panel has been obtained the segmentations of different classes are separated and then analyzed with OpenCV's [63] contour analysis tools. At this stage the TK standard is implemented into a decision-making system by extracting relevant information from the contours of different classes.

6.4.1 Knot analysis

In the knot analysis phase, the segmentation maps for both dry and live knots are joined and then the knot contours are extracted. The TK standard states that a leaf/horn knot can be at most 50% of the panel's width but other knots may only have a width of one third of the panel's. Therefore, these types of knots must be differentiated from each other to apply different criteria to them. This is done by observing the shape of the contour. For leaf and horn knots, their respective

contours appear much more elongated than their round counter parts. To exploit this characteristic of the knot contours we first fit a minimum area enclosing rectangle to the contour. Then the ratio of the longer and shorter side of the rectangle is calculated. Then the knot contour is assigned to either leaf/horn or round knots based on the aspect ratio of the rectangle. Here the threshold value used is 2.5 but any value deemed suitable for this distinction task may be used here. The width of the knot is calculated as the difference in x-coordinate between the left and right most extreme points of the contour.

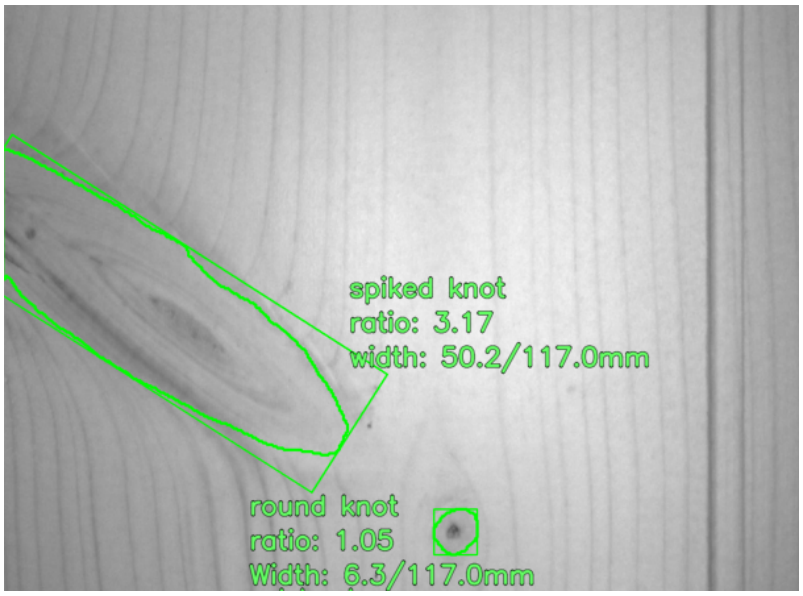

Figure 6.24: Visualization of the knot analysis step.

6.4.2 Resin pocket analysis

The requirements for resin pockets define a maximum length for singular pockets of 60 mm and a summed length of 120 mm on a 1-meter span. The checking process for the aforementioned requirement can be implemented by taking the top and bottom most extreme points of the contour since the length of the pocket is measured in the direction parallel to the panel's length. The pocket's length is calculated as the subtraction of the top and bottom most y-coordinates of the contour. The length in

millimeters is obtained by multiplying the length in pixels by the size of the pixel in millimeters (0.083 mm). If the pocket's length exceeds that of 60 mm the panel gets rejected. The summed length on a specified span is calculated by first forming an array of zeros with a length of the panel in pixels. Then the resin pocket contours are iterated over and their top and bottom most y-coordinates are extracted. Then the values in the array are raised by one between the top and bottom y-coordinate indices. This essentially forms an array that tells how many resin pockets have pixels in the y-coordinate equal to the index in the array.

After the first array has been formed a second array of ones is formed. This array on the other hand is length of the span in pixels we wish to check the presence of pockets on. The second array is convolved over the first array. This way we get a density array describing the sum of resin pockets lengths when the span is placed in different positions on the panel. The check is performed by taking the maximum value of the convolution and then multiplying it with the pixel size in millimeters. If the product exceeds that of the 120 mm threshold, then the panel is rejected.

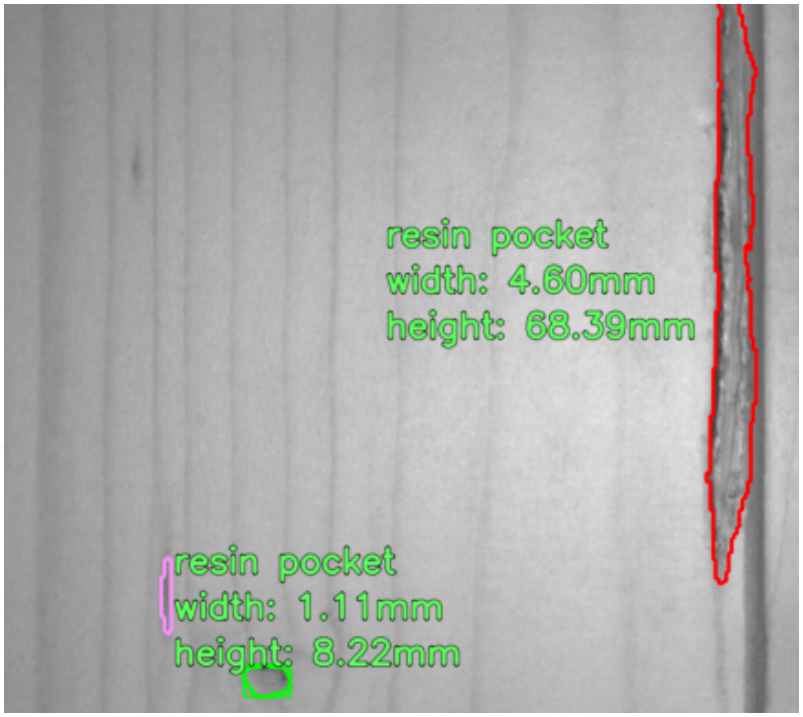

Figure 6.25: Visualization of the individual resin pocket grading. On the right the resin pocket exceeds the 60 mm maximum height requirement and on the left the pocket passes.

6.4.3 Knot hole and cleft analysis

Since knot holes and clefts are similar in cause and appearance it was beneficial to combine their respective classes in the segmentation stage. However, now we face the situation where we need to be able to differentiate between the two since they have different criteria for rejection. This can be done by extracting the left and right most extreme points of the contour. If the x-coordinate of the left extreme point is equal to zero or the x-coordinate right extreme point is equal to the width of the image-1 then it can be deduced that the contour is a cleft and not a hole since the contour touches the left or right boundary of the image. In the case of the contour being a knot hole, the panel is rejected.

The clefts are iterated over and assessed by their width. If a cleft's width or size

in x-axis exceeds that of 8mm then it is added to a list of critical clefts. When this is done the list of critical clefts are sorted by their middle points' y-coordinate. Then the list is iterated over and if the distance between two neighboring critical clefts on the list is less than 2 meters the panel gets rejected as the TK-standard states that on two-meter span there may exist at most one knot cleft exceeding 8 mm in width.

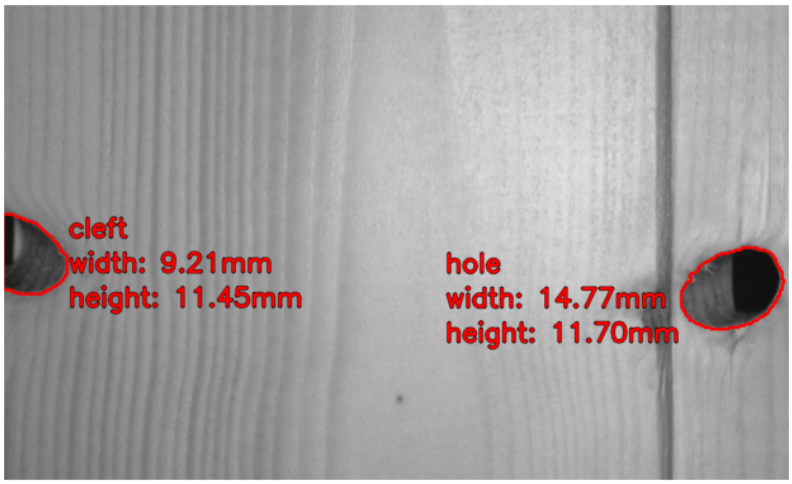

Figure 6.26: Knot hole analysis. On the left the hole is attached to the image boundary so it is processed as a cleft. The hole on the right however is detached from the image boundary making it a knot hole.

6.4.4 Pith streak analysis

The quality requirement for visible pith streaks is quite simple: a single pith streak may not exceed half of the panel's length. This can be checked by extracting the pith streak contours from the segmentation map and checking their length in y-axis. If the difference in y-coordinate of the top and bottom most extreme points of the contour exceed that of half of the panel's length the panel is rejected.

Figure 6.27: Pith streak analysis visualization.

6.4.5 Crack analysis

The requirements for cracks are quite strict. Hair cracks may only have a width at most 0.5 mm and the summed length of these hair cracks may not exceed 25% of the panel's length. Cracks are allowed if they reside at either end of the panel. These panel end cracks may not exceed the width of the panel if the panel doesn't have the tongue and groove at the ends and $\frac{1}{2}$ if it does. To check the width of the crack the difference in x-coordinate of the left and right most extreme points cannot be used since cracks can exist in bowlike shapes and diagonal orientations. To solve this problem, we will use distance transform. Distance transform is an algorithm that calculates the shortest distance to a pixel with value zero for all non-zero pixels within a binary image. The maximum width of a contour is therefore two times the maximum value of the distance transform of the contour's binary mask. The maximum width of the contour in millimeters is therefore:

$$\text{width}_{mm} = max(\text{distance\_transform}(\text{binary\_mask})) \times 2 \times \text{pixel\_size}_{mm} \quad (6.1)$$

The cracks narrower than 0.5 millimeters are gathered and their length in the direction of the y-axis summed. If the sum exceeds that of 25% of the panel's length the panel is rejected. If the crack resides at either end of the panel, then the crack is allowed to reach up to half or 100% of the panel's width in length depending on whether the panel has a tongue and groove at its ends or not. Whether or not the crack resides at the end of the panel can be checked by comparing the crack contour's top and bottom most extreme points to the top and bottom y-coordinates of the panel respectively. If the coordinates match, then the crack can be exempted if its length in y-axis doesn't exceed the requirements. A potential limitation of detecting cracks by segmenting downsized images is that the accuracy needed to measure the cracks' width is quite high. The horizontal field of view of the camera 19cm away from the panel was appr. 160mm. When the image is down sampled to a 384×384 resolution a lot of information is lost since at this resolution there are only $\frac{384px}{160mm} = 2.4\frac{px}{mm}$ and a pixel size of $\frac{160mm}{384px} = 0.41mm$. This is not enough to measure an object's width with an accuracy of 0.1 mm.

Also, when inspecting the original segmentations and the model predictions we can see that the masks marking the cracks are slightly wider than the cracks themselves. This introduces a small upward error in the width measurement.

A way to remedy these issues would be to segment the images in a higher resolution but this would have a high computational cost and could potentially compromise the real-time capability of the system.

Another way to solve these issues would be to use adaptive thresholding to the areas under the segmentation masks. With this method the threshold function is used for refining the crack's silhouette and should give us better accuracy when

measuring the width. In this method the rectangular area of the crack is extracted from the original image. Then adaptive gaussian inverted binary threshold function is applied to the image. The block size of the kernel is 21 which equates to appr. 1.8mm. Distance transform is then applied to the output of the threshold function. Finally, the largest value of the distance transform residing inside the segmentation mask is taken and multiplied by two like described previously to get the corrected maximum width of the crack. The steps of this process are depicted in figure 6.28.

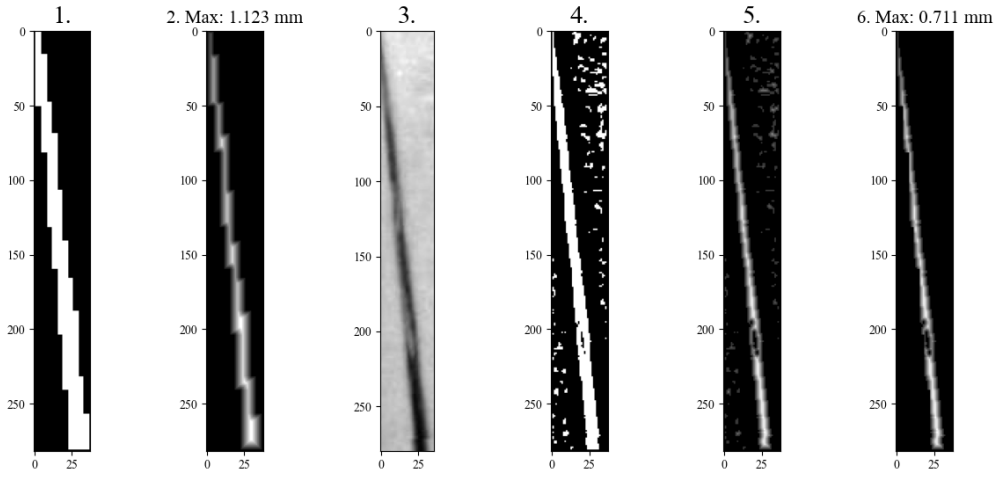

Figure 6.28: The crack width correction process.

1. Predicted segmentation mask produced by the model
2. Distance transform of the predicted mask
3. Crack region of the image
4. Output of the adaptive threshold on the crack region
5. Distance transform of the threshold output
6. Bitwise and on the segmentation mask (1) and second distance transform (5)

This postprocessing method is only applied to cracks with an initial width less than 1.5 mm. Cracks with an initial width higher than 1.5mm are significantly wider than 0.5 mm and might no longer be dark in the middle. This is depicted in figure 6.29.

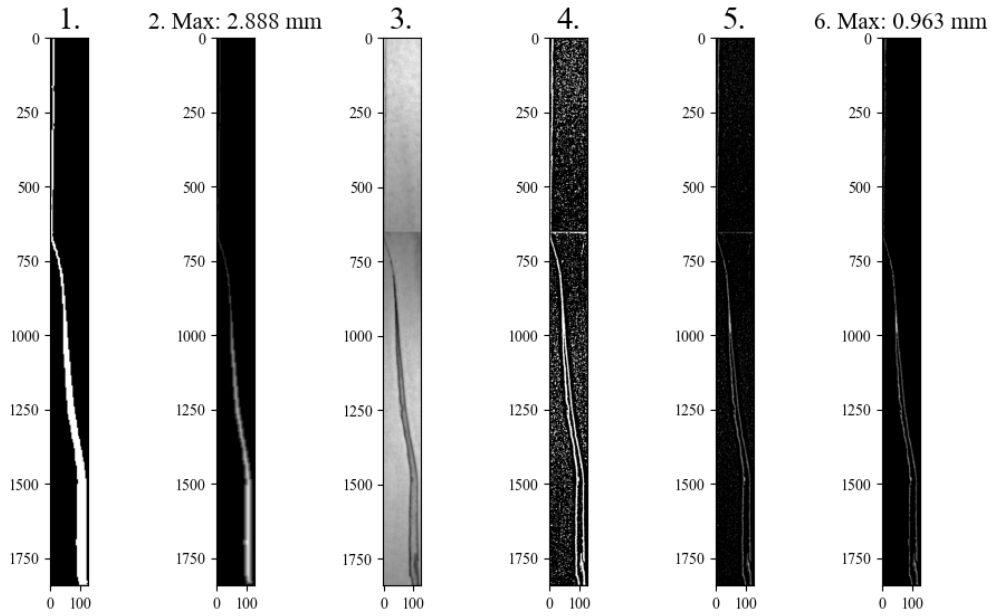

Figure 6.29: A wide crack. The crack has a light area in the middle, so the threshold output gets split.

6.4.6 Grading system results

To answer the research question: "Can a semantic segmentation model be used for automatic wood panel quality inspection?", we need to test how the trained segmentation model works with the programmed grading system. In this test I compared both overall gradings as well as grading results from each grading tool. Since the data doesn't include whole panels or actual human-made gradings from the production line, this test does not measure the effectiveness of the system compared to a human grader. Rather this test compares the performance of the model to a human annotator's segmentations when coupled with the rule-based grading tools. Here a positive result corresponds with a rejected grading result and negative with not rejected.

In summary we can conclude that the segmentation model produces similar grading results as the annotations. I will examine the combined results first and then

examine the results of each tool separately. I used accuracy, recall, precision and f1 to evaluate the grading results. The results of these metrics are shown in table 6.4.

Table 6.4: The grading system's performance metrics on the TK test set.

Class	Accuracy	Recall	Precision	F1
Combined	0.969	0.962	0.949	0.955
Knots	0.998	1.0	0.667	0.8
Pith streaks	0.989	0.945	0.929	0.937
Resin pockets	0.997	0.75	0.75	0.75
Cracks	0.989	1.0	0.926	0.962
Knot holes	0.99	0.938	0.968	0.953

The combined grading results suggest that the annotated and predicted segmentation maps interact very similarly with the grading tools. However, some recall, precision, and f1 results for the individual grading tools are not very impressive. We can analyze the results further by examining the confusion matrices in figure 6.30.

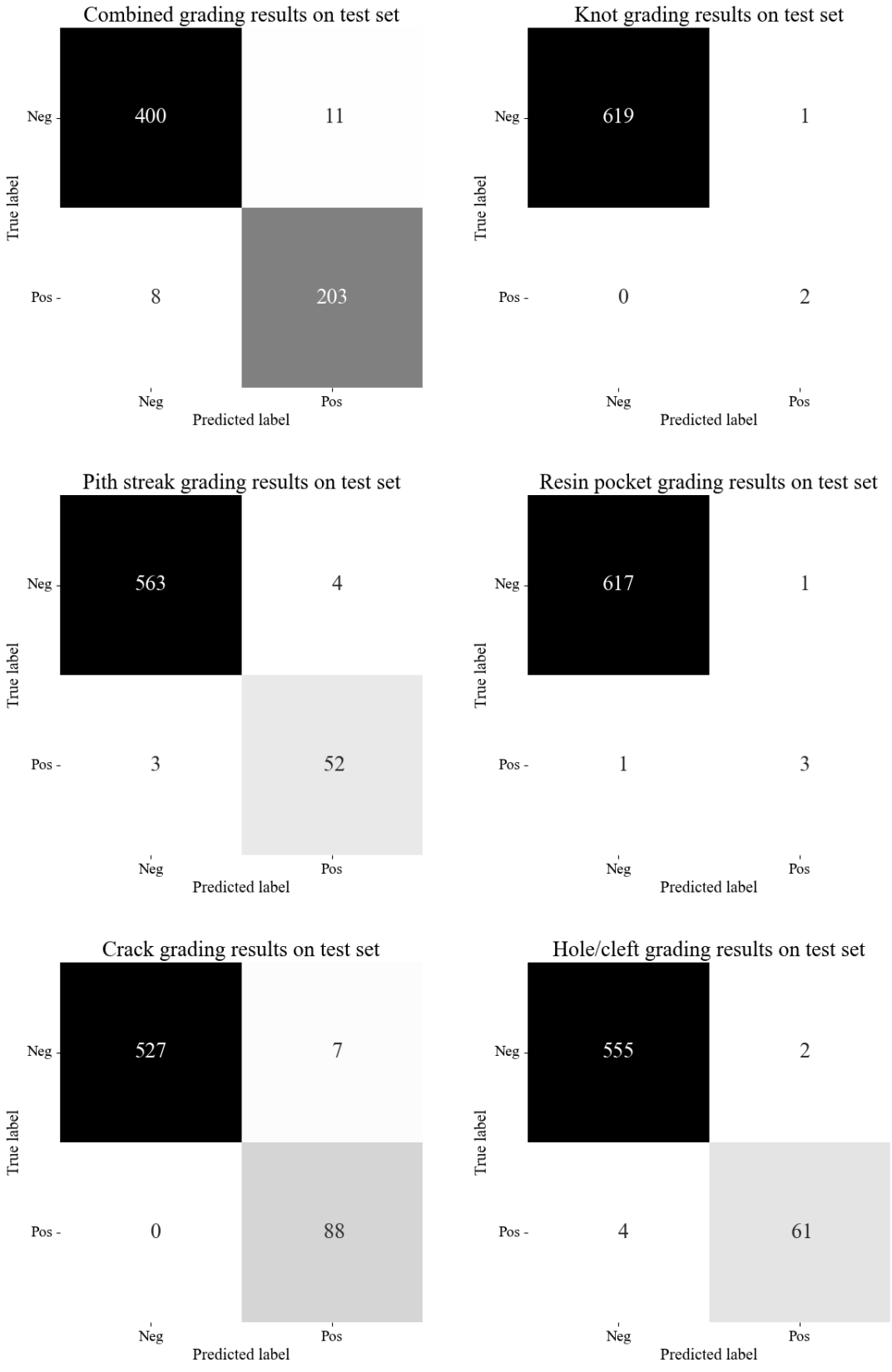

Figure 6.30: TK grading system's confusion matrices.

By first glance we can see that the grading results are all imbalanced. Even the combined matrix is heavily skewed towards the negative class. The skewness is greatly amplified in the individual grading results. Especially in the knot and resin pocket analysis tools. This is due to the overall scarcity of examples that should produce positive grading results.

The knot analysis tool produced one false positive and one false negative. When I inspected these examples, the false positive result was produced by a correctly detected leaf knot that had a predicted width of 57.4mm and an annotated width of 56.4mm while the width of the panel was 113.7mm. These measurements give us ratios of 0.504 and 0.496 respectively. While the width difference is quite small at 1 mm the one false positive result has an enormous effect on the precision and f1 scores.

The resin pocket grading tool's results suffer from the same reason. To produce a positive result from the resin pocket grading tool a photo would have to contain a single region with a vertical length exceeding 60 mm or a sum of lengths exceeding 120mm. These cases are quite rare in cases where only one image is used. The false positive result was produced by an example where the predicted region was slightly taller than the annotated one. The respective lengths were 63.1 and 59.3 mm. The false negative result on the other hand was produced by an example where the annotated region was 83.4 mm in length, but the prediction was divided into multiple regions that had a summed length of 66.6 mm.

The pith streak analysis tool on the other hand produces a decent amount of positive gradings. This is because many of the pith streak regions present in the test set are tall enough to trigger the rejection criteria. Two of the false negatives were produced by examples where the pith streak regions were slightly shorter than the annotated ones. One false negative was produced by a prediction where the pith streak was not detected. The false positives were produced by regions where the

predictions were slightly taller than the annotations, regions got fused together or the regions were falsely segmented.

For the evaluation of the crack grading tool, I disabled the panel end functionality. Allowing the presence of cracks at the ends of the panel would dismiss most of the cracks in both the annotated and predicted segmentations because the photos were short in comparison to actual panels. The crack grading tool didn't produce any false negatives. This means that the segmentation model could be too sensitive to detect cracks. When examining the false positives, they were mostly produced by falsely detected regions. Two of the seven false positives were produced by examples where the summed length of detected hair cracks was higher than in the annotations. The rest were produced by falsely detected cracks that exceeded the 0.5mm width restriction.

The length of the photos persists as a problem for the cleft evaluation tool. The photos that contain one or less edge cleft pass automatically because the grading rule looks for a pair of two clefts that are within 2 meters of each other. Here the tool mainly flags photos with a knot hole as positive. The grading tool produced four false negatives and two false positives. The false positives are produced by cases where the segmentation result can be considered either correct or inconsequential. In the former a region was predicted as a knot hole that was annotated as a dry knot. However, the knot had started to fall out of the hole, so this region was likely to become a knot hole in the immediate future. The latter case where the segmented region had no consequence for the overall grading result was in a photo with a very wide gap. This case is depicted in a previous figure. The hole region was predicted in the middle of a correctly detected wide crack. The false negatives on the other hand were the result of two causes: false negative segmentations and knot hole proximity to panel edge. One partially detached knot in the middle of the panel was falsely predicted as a resin pocket. In another false negative case, an

edge cleft was not detected where the photo contained two annotated clefts. The Other two false negatives were caused by the predicted knot hole regions becoming attached to the panel edge, so they were classified as edge clefts.

In conclusion, the annotations and the segmentation model produce similar grading results when paired with rule based grading tools. According to my results there is enough evidence to claim that final quality evaluation on the part of wood quality can be performed both after painting and planing the panel. Many of the misclassifications produced by the grading tools were due to small differences between the annotated and predicted segmentations. Additionally, since the segmentation model can be deemed oversensitive to detecting cracks the segmentation model can also be used as a separator to flag possibly suspicious regions for human inspection while rejecting the cases with a high probability of bad quality. At the very least a system such as this can be used to alleviate the quality inspection burden of the human operators by reducing the needed effort for production.

7 Discussion

7.1 Conclusions

In this thesis I studied the use of medical image segmentation methods in medical imaging and the wood processing industry. I did this by first surveying the most prominent supervised training methods used in the semantic segmentation of tumors from MR-images and then applying them to the detection of various features and defects of wooden interior panels.

While these techniques are heavily researched in the medical field their use in the wood industry isn't researched very much in comparison. However, since the segmentation applications are very similar in nature the research results achieved in the medical field transfer very well to the wood industry as I have demonstrated in this paper.

The results of the hyperparameter survey of the 4th chapter proved important for training the wood feature segmentation models in the 6th chapter. The purpose of the hyperparameter survey was to find popular choices for the loss function, optimizer, transfer learning method, and augmentation methods. The most popular choice for the loss function was Dice loss. The most popular optimizer was Adam. The most popular transfer learning method was replacing the encoder component of the segmentation network with a pretrained convolutional backbone such as ResNet. Popular augmentations included methods such as flips, rotations, elastic

transformation and Gaussian noise.

The popular hyperparameter choices uncovered in the hyperparameter survey produced good results in training a U-net, a DeeplabV3, and a Segformer model to segment various features and defects from wooden panels. Of the trained models the U-net performed slightly better than the others. The Dice scores exceeded 0.8 for each defect and feature type making it possible to design a quality inspection system based on a U-net segmentation model.

I also performed some additional examination on why the U-net model was able to learn the segmentation task to the observed degree. First, I constrained the amount of data the model had access during the training phase. Then I trained the U-net model without transfer learning and augmentation. I discovered that the model was able to learn quite well even with significant constraints to the amount of available data. The model was able to achieve good segmentation performance even trained on only 25% of the available training data. In the latter experiment I discovered that transfer learning contributed only a small increase in segmentation performance. The augmentation pipeline on the other hand played a crucial role in achieving good performance. When trained without augmentation the model started to overfit to the training data. Therefore, It can also be said that the reason why the decrease of available training data didn't affect performance significantly is because the augmentation pipeline successfully introduces randomness into the training pipeline.

In the process of designing a grading system for the wood panel quality inspection task, contours proved useful since they provide a great variety of tools for the analysis of the defect and feature regions. The TK-spruce quality class was implemented for the panel grading stage to demonstrate the potential of the designed processing pipeline. The only problematic defect type proved to be cracks since they required additional processing to measure their width. The processing pipeline was tested

by first extracting the quality decisions produced with the annotations of the test set and then with the segmentation model. The quality decisions produced by the annotations were compared to the ones produced by the segmentation model. This comparison provided proof for the usability of the segmentation model. Overall, the segmentation model's quality decisions were similar to the ones produced with the annotations showing the potential of deep learning in this domain.

Overall, this thesis successfully demonstrates how deep learning-based segmentation models can be used for quality inspection in the wood industry. At the very least the system could be used for alleviating the workload of the production line employees in the quality inspection task. In the first stage of production line integration the system could be used in a supporting or assisting role where the system flags potentially bad or difficult cases for the human inspector to verify.

7.2 Future work

For future reference this thesis can be expanded upon in at least three ways. Firstly, this thesis only considered the use of three base architectures for the segmentation task, U-net, DeeplabV3, and Segformer. They were chosen because they have very different network architectures. These architectures could be optimized further by experimenting with different encoder architectures and other settings. The use of other segmentation models should also be considered in future work, both other convolutional and transformer-based models.

Another way would be to include more finishing types for the study. In this study the finish applied to the spruce panels was a white varnish. In the future, the scope of the research could be expanded by collecting and annotating images of other finishing types. Since the semantic segmentation stage acts as a simple feature detector, the expert system can be expanded to perform quality inspection on other panel types as well. Also, in figure 6.16 we can see that some examples in the test

data produce higher loss figures. These examples could be analyzed further and add similar examples to the dataset so the model would learn to perform better on this type of data.

Finally, since the goal of developing the grading system is to replace human graders, it would be useful to know how these systems perform in comparison. in this thesis I was able to test the segmentation model against human annotated segmentations with both segmentation performance metrics and coupled with the panel grading tools. The deep learning segmentation model produced good results in both tests. However, to test this system against a human grader on the production line, a system would have to be built that records the decisions made by both the automated as well as the production line worker.

References

[1] X. Zheng, S. Zheng, Y. Kong, and J. Chen, "Recent advances in surface defect inspection of industrial products using deep learning techniques", *The International Journal of Advanced Manufacturing Technology*, vol. 113, Mar. 2021. DOI: 10.1007/s00170-021-06592-8.

[2] S. V. Kulkarni, A. S. Wani, S. N. Gohel, M. N. Javheri, and N. M. Pagare, "Visual quality inspection using deep learning", *International Journal of Multidisciplinary Research and Growth Evaluation*, vol. 2, pp. 103–106, 4 Jun. 2021.

[3] Y. Yang, L. Pan, J. Ma, *et al.*, "A high-performance deep learning algorithm for the automated optical inspection of laser welding", *Applied Sciences*, vol. 10, no. 3, 2020, ISSN: 2076-3417. DOI: 10.3390/app10030933.

[4] Y. Shu, B. Li, and H. Lin, "Quality safety monitoring of led chips using deep learning-based vision inspection methods", *Measurement*, vol. 168, p. 108 123, Sep. 2020. DOI: 10.1016/j.measurement.2020.108123.

[5] T. Wang, Y. Chen, M. Qiao, and H. Snoussi, "A fast and robust convolutional neural network-based defect detection model in product quality control", *The International Journal of Advanced Manufacturing Technology*, vol. 94, Feb. 2018. DOI: 10.1007/s00170-017-0882-0.

[6] N. Ismail and O. Malik, "Real-time visual inspection system for grading fruits using computer vision and deep learning techniques", *Information Processing in Agriculture*, vol. 9, Feb. 2021. DOI: 10.1016/j.inpa.2021.01.005.

[7] "Laatukortti-tk-kuusipaneeli", [Online]. Available: https://maler.fi/wp-content/uploads/2021/10/laatukortti-tk-kuusipaneeli.pdf (visited on 02/2024).

[8] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning", *Nature*, vol. 521, pp. 436–44, May 2015. DOI: 10.1038/nature14539.

[9] J. D. Kelleher, *Deep Learning*. The MIT Press, Sep. 2019, ISBN: 9780262354899. DOI: 10.7551/mitpress/11171.001.0001.

[10] S. Neuroscience, *Brain Facts*. Jan. 2017.

[11] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*, 1st ed. Springer, 2007, ISBN: 0387310738.

[12] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators", *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989, ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(89)90020-8.

[13] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org.

[14] G. James, D. Witten, T. Hastie, and R. Tibshirani, *An Introduction to Statistical Learning: with Applications in R*. Springer, 2013. [Online]. Available: https://faculty.marshall.usc.edu/gareth-james/ISL/.

[15] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 3rd ed. Prentice Hall, 2010.

[16] P. Celard, E. L. Iglesias, J. M. Sorribes-Fdez, R. Romero, A. S. Vieira, and L. Borrajo, "A survey on deep learning applied to medical images: From simple artificial neural networks to generative models", *Neural Comput. Appl.*, vol. 35, no. 3, pp. 2291–2323, Nov. 2022, ISSN: 0941-0643. DOI: 10.1007/s00521-022-07953-4.

[17] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors", *Nature*, vol. 323, pp. 533–536, 1986.

[18] N. Qian, "On the momentum term in gradient descent learning algorithms", *Neural networks : the official journal of the International Neural Network Society*, vol. 12, pp. 145–151, Feb. 1999. DOI: 10.1016/S0893-6080(98)00116-6.

[19] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization", *J. Mach. Learn. Res.*, vol. 12, no. null, pp. 2121–2159, Jul. 2011, ISSN: 1532-4435.

[20] S. Ruder, *An overview of gradient descent optimization algorithms*, 2017. arXiv: 1609.04747 [cs.LG].

[21] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2017. arXiv: 1412.6980 [cs.LG].

[22] S. Ioffe and C. Szegedy, *Batch normalization: Accelerating deep network training by reducing internal covariate shift*, 2015. arXiv: 1502.03167 [cs.LG].

[23] S. Bozinovski, "Reminder of the first paper on transfer learning in neural networks, 1976", *Informatica*, vol. 44, Sep. 2020. DOI: 10.31449/inf.v44i3.2828.

[24] C. Bishop, "Regularization and complexity control in feed-forward networks", English, pp. 141–148, 1995, International Conference on Artificial Neural Networks ICANN'95.

[25] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks", in *Advances in Neural Information Processing Systems*, vol. 25, Curran Associates, Inc., 2012. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[26] P. Simard, D. Steinkraus, and J. Platt, "Best practices for convolutional neural networks applied to visual document analysis", pp. 958–963, 2003. DOI: 10.1109/ICDAR.2003.1227801.

[27] Y. LeCun, B. Boser, J. S. Denker, *et al.*, "Backpropagation applied to handwritten zip code recognition", *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989. DOI: 10.1162/neco.1989.1.4.541.

[28] A. Dosovitskiy, L. Beyer, A. Kolesnikov, *et al.*, "An image is worth 16x16 words: Transformers for image recognition at scale", 2021. arXiv: 2010.11929 [cs.CV].

[29] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection", 2016. arXiv: 1506.02640 [cs.CV].

[30] M. Thoma, "A survey of semantic segmentation", 2016. arXiv: 1602.06541 [cs.CV].

[31] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation", 2015. arXiv: 1505.04597 [cs.CV].

[32] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation", 2015. arXiv: 1411.4038 [cs.CV].

[33] B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik, "Simultaneous detection and segmentation", 2014. arXiv: 1407.1808 [cs.CV].

[34] D. Ciresan, A. Giusti, L. Gambardella, and J. Schmidhuber, "Deep neural networks segment neuronal membranes in electron microscopy images", vol. 25, F. Pereira, C. Burges, L. Bottou, and K. Weinberger, Eds., 2012.

[35] L.-C. Chen, G. Papandreou, F. Schroff, and H. Adam, "Rethinking atrous convolution for semantic image segmentation", 2017. arXiv: 1706.05587 [cs.CV].

[36] J. Dong, X.-J. Mao, C. Shen, and Y.-B. Yang, "Learning deep representations using convolutional auto-encoders with symmetric skip connections", 2017. arXiv: 1611.09119 [cs.CV].

[37] V. Iglovikov and A. Shvets, "Ternausnet: U-net with vgg11 encoder pre-trained on imagenet for image segmentation", 2018. arXiv: 1801.05746 [cs.CV].

[38] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition", pp. 346–361, 2014, ISSN: 1611-3349. DOI: 10.1007/978-3-319-10578-9_23.

[39] E. Xie, W. Wang, Z. Yu, A. Anandkumar, J. M. Alvarez, and P. Luo, "Segformer: Simple and efficient design for semantic segmentation with transformers", 2021. arXiv: 2105.15203 [cs.CV].

[40] D. W. McRobbie, E. A. Moore, M. J. Graves, and M. R. Prince, *MRI from Picture to Proton*, 2nd ed. Cambridge University Press, 2006.

[41] *What is cancer?* National cancer institute. [Online]. Available: https://www.cancer.gov/about-cancer/understanding/what-is-cancer (visited on 02/2023).

[42] S. Maurya, S. Tiwari, M. C. Mothukuri, C. M. Tangeda, R. N. S. Nandigam, and D. C. Addagiri, "A review on recent developments in cancer detection using machine learning and deep learning models", *Biomedical Signal Processing and Control*, vol. 80, p. 104 398, 2023, ISSN: 1746-8094. DOI: https://doi.org/10.1016/j.bspc.2022.104398.

[43] *Cancer today*. International Agency for Research on Cancer. [Online]. Available: https://gco.iarc.who.int/media/globocan/factsheets/populations/900-world-fact-sheet.pdf (visited on 08/2024).

[44] B. Menze, A. Jakab, S. Bauer, *et al.*, "The multimodal brain tumor image segmentation benchmark (brats)", *IEEE Transactions on Medical Imaging*, vol. 99, Dec. 2014. DOI: 10.1109/TMI.2014.2377694.

[45] S. Bakas, H. Akbari, A. Sotiras, *et al.*, "Advancing the cancer genome atlas glioma mri collections with expert segmentation labels and radiomic features", *Scientific Data*, vol. 4, Sep. 2017. DOI: 10.1038/sdata.2017.117.

[46] S. Bakas, M. Reyes, A. Jakab, *et al.*, "Identifying the best machine learning algorithms for brain tumor segmentation, progression assessment, and overall survival prediction in the brats challenge", 2019. arXiv: 1811.02629 [cs.CV].

[47] S. Bakas, H. Akbari, A. Sotiras, *et al.*, "Segmentation labels and radiomic features for the pre-operative scans of the tcga-gbm collection", Jul. 2017. DOI: 10.7937/K9/TCIA.2017.KLXWJJ1Q.

[48] S. Bakas, H. Akbari, A. Sotiras, *et al.*, "Segmentation labels and radiomic features for the pre-operative scans of the tcga-lgg collection", Jul. 2017. DOI: 10.7937/K9/TCIA.2017.GJQ7R0EF.

[49] A. Buslaev, V. I. Iglovikov, E. Khvedchenya, A. Parinov, M. Druzhinin, and A. A. Kalinin, "Albumentations: Fast and flexible image augmentations", *Information*, vol. 11, no. 2, p. 125, Feb. 2020, ISSN: 2078-2489. DOI: 10.3390/info11020125.

[50] W. Falcon and The PyTorch Lightning team, *PyTorch Lightning*, version 1.4, Mar. 2019. DOI: 10.5281/zenodo.3828935.

[51] P. Iakubovskii, "Segmentation models pytorch", *GitHub repository*, 2019. [Online]. Available: https://github.com/qubvel/segmentation_models.pytorch.

[52] J. Shi, L. Zhenye, T. Zhu, D. Wang, and C. Ni, "Defect detection of industry wood veneer based on nas and multi-channel mask r-cnn", *Sensors*, vol. 20, p. 4398, Aug. 2020. DOI: 10.3390/s20164398.

[53] K. Hu, B. Wang, Y. Shen, J. Guan, and Y. Cai, "Defect identification method for poplar veneer based on progressive growing generated adversarial network and mask r-cnn model", *BioResources*, vol. 15, pp. 3041–3052, Mar. 2020. DOI: 10.15376/biores.15.2.3041-3052.

[54] E. Urtans, K. Bumanis, V. Vecins, *et al.*, "Detection of knots in oak wood planks: Instance versus semantic segmentation", pp. 163–168, 2022. DOI: 10.1109/BDAI56143.2022.9862633.

[55] A. Ziadi, F. Ntawiniga, and X. Maldague, "Neural networks for color image segmentation: Application to sapwood assessment", pp. 417–420, 2007. DOI: 10.1109/CCECE.2007.110.

[56] J. HU, X. YU, Y. ZHAO, K. WANG, and W. LU, "Research on bamboo defect segmentation and classification based on improved u-net network", *Wood Research*, vol. 67, pp. 109–122, Jan. 2022. DOI: 10.37763/wr.1336-4561/67.1.109122.

[57] Y. Lin, Z. Xu, D. Chen, Z. Ai, Y. Qiu, and Y. Yuan, "Wood crack detection based on data-driven semantic segmentation network", *IEEE/CAA Journal of Automatica Sinica*, vol. 10, no. 6, pp. 1510–1512, 2023. DOI: 10.1109/JAS.2023.123357.

[58] D. Li, W. Xie, B. Wang, W. Zhong, and H. Wang, "Data augmentation and layered deformable mask r-cnn-based detection of wood defects", *IEEE Access*, vol. 9, pp. 108 162–108 174, 2021. DOI: 10.1109/ACCESS.2021.3101247.

[59] T. He, Y. Liu, C. Xu, X. Zhou, Z. Hu, and J. Fan, "A fully convolutional neural network for wood defect location and identification", *IEEE Access*, vol. 7, pp. 123 453–123 462, 2019. DOI: 10.1109/ACCESS.2019.2937461.

[60] G. Ruz, P. Estevez, and C. Perez, "A neurofuzzy color image segmentation method for wood surface defect detection", *Forest Products Journal*, vol. 55, pp. 52–58, Apr. 2005.

[61] "Sisäverhous- ja lattialautojen laatu ml. listat", [Online]. Available: https://puuinfo.fi/puutieto/sahatavara-ja-sen-jalosteet/sisaverhous-ja-lattialautojen-laatu/ (visited on 02/2024).

[62] B. Russell, A. Torralba, K. Murphy, and W. Freeman, "Labelme: A database and web-based tool for image annotation", *International Journal of Computer Vision*, vol. 77, May 2008. DOI: 10.1007/s11263-007-0090-8.

[63] G. Bradski, "The OpenCV Library", *Dr. Dobb's Journal of Software Tools*, 2000.