UiT The Arctic University of Norway

Faculty of Science and Technology
Department of Physics and Technology

ConvMixerSeg: Weakly Supervised Semantic Segmentation for CT Liver Images

Harald Lykke Joakimsen

FYS-3941: Master's Thesis in Applied Physics and Mathematics
December 2021

“Overcoming naive impressions to figure out how things really work is one of humanity’s highest callings.”
–Steven Pinker

“Any sufficiently advanced technology is indistinguishable from magic.”
–Clark’s third law

Abstract

The predictive power of modern deep learning approaches is posed to revolutionize the medical imaging field, however, their usefulness and applicability are severely limited by the lack of well annotated data. Liver segmentation in CT images is an application that could benefit particularly well from less data hungry methods and potentially lead to better liver volume estimation and tumor detection. To this end, we propose a new semantic segmentation model called ConvMixerSeg and experimentally show that it outperforms an FCN with a ResNet-50 backbone when trained to segment livers on a subset of the Liver Tumor Segmentation Benchmark data set (LiTS). We have further developed a novel Class Activation Map (CAM) based method to train semantic segmentation models with image level labels without adding parameters. The proposed CAM method includes a Neighborhood Correlation Enforcement module using Gaussian smoothing that reduces part domination and prediction noise. Additionally, our experiments show that the proposed CAM method outperforms the original CAM method for both classification and segmentation with high statistical significance given the same ConvMixerSeg backbone.

Acknowledgements

I want to express my great appreciation to Assoc. Prof. Michael Kampffmeyer for his guidance, constructive suggestions and illuminating questions during this thesis. I am very grateful that he patiently has guided me away from digressive rabbit holes that could have significantly prolonged the time it would take to finish this thesis. I would also like to express my gratefulness to Kristoffer Knutsen Wickstrøm for his generous help and advice as well as good humor during my work. Additionally, I want to thank my class and especially Jonathan Edward Berezowski, Joel Burman and Naphat Amundsen for numerous discussions and difficult questions during my study. Their reflections have helped me to learn, but also to develop deeper philosophical views on Bayesian statistics, the opportunities and limitations of deep learning in the modern world as well as on many other topics. I would also like to thank Juha Vierinen for showing me that science, despite being hard, is an activity that kindles interest and can be fun. Finally I would like to thank my beloved Tuva, for her patience, support and love during this project.

Contents

List of Figures

List of Tables

1 Introduction

1.1 Liver Cancer

The liver is one of the essential organs of the human body performing numerous vital functions. Among these are metabolism of carbohydrates for energy storage, of proteins and amino acids that aid the bloods coagulating functions and of lipids for absorption and digestion of fats and fat soluble vitamins. Sadly, this essential and detoxifying organ is the focus of many unfortunate cancer patients world wide.

Globally, liver cancer is the type of cancer with the 3-rd highest mortality with an estimated number of deaths of over 800,000[77]. One of the most common tools to aid in diagnosis of liver cancer is CT imaging. This is the case not only for primary liver tumor (hepatocellular carcinoma), but for many types of cancer as it is common for many other types of cancer to spread to the liver (hepatic metastases). Examples include colorectal, breast, lung, esophageal, stomach, kidney and pancreatic cancer.

Because many types of cancer spread to the liver, it is a good place to look for tumors in screening procedures where the hope is to discover the cancer at an early stage. Such efforts could help to delay or stop the progress of the cancer to improve the patients' health and quality of life.

1.2 Automated Semantic Image Segmentation using Deep Learning

Automated image analysis is a field that has experienced great progress in the last decade. There has been a paradigm shift for image analysis in parallel with incredible advancements in the field of deep learning, strongly aided by increasingly accessible low-priced and powerfull computers. While traditional image analysis methods typically relied on manually defined features [28], modern methods are often end-to-end trained, i.e. the methods mainly discover relevant features of input images in an automated fashion based on given data. Although traditional methods are still used for many applications, modern deep learning methods can learn to perform well on much more complex problems[1].

One of the most significant deep learning architecture types used for image analysis is the Convolutional Neural Network (CNN). It applies the basic structure of artificial neural networks (ANNs) combined with the convolution operation, which can be viewed as a matching operation between an input and a template, to automatically create relevant templates that extract information from a given input image. This is done in several consecutive layers to analyze and reanalyze the provided information. CNNs have been used for different tasks such as image classification where the model determines which out of a given set of groups the image belongs to, image detection where the model produces a bounding box around a desired object type (e.g. a horse, a face, a person, a car, etc.) and many others. Among the tasks where CNNs have made a profound contribution is semantic segmentation where the goal is to classify every pixel of an image reflecting the semantic meaning of that pixel.

1.3 Weakly Supervised Learning

Despite the progress of deep learning and image analysis models that have expanded our view of what is possible, deep learning methods are not as ubiquitous today as one might expect given their performance. Crossing many old thresholds, new problems have taken over the role as the main bottleneck of progress. Today, one of the main bottlenecks preventing the use of recent deep learning models in many domains is the lack of well annotated data. Data represents what the deep learning models are supposed to learn. So models trained on inadequate amounts of data are unreliable and generalize badly.

1. See `https://paperswithcode.com/sota` for examples.

Attaining enough high quality annotations demands large amounts of expensive human labor. Therefore, the community of deep learning researchers are exploring new solutions that make current models less data hungry.

Weakly Supervised Learning is a field of machine learning that attempts to utilize features that are indirectly learned by a model, such that the model can make predictions of higher quality than the data it was trained on. Because such methods are less picky about their data quality, they could be much more applicable in the real world to help solve problems.

In this project we will focus on both fully and weakly supervised models that perform semantic image segmentation of Computed Tomography(CT) images of livers. The choice of data to analyze is motivated by the potential utility of a successful model which could help reduce the cost of CT screenings and thus help people.

For weak supervision, we experiment with the use of only image level labels, i.e. if an image contains liver or not. This is a very weak type of label, making the task at hand difficult but also valuable as such labels are much cheaper to obtain than more detailed alternatives.

1.4 Key Challenges

The advancements of deep learning model architectures have been fast in the last decade and accelerating in the last few years. Not long ago, the state-of-the-art models for many fields of image analysis were governed by classic CNN architectures such as VGG and ResNet [68, 33] that both originated for the use of image classification. But these models, although revolutionary when presented, still require a lot of computational power and are still hard to train. As a response to the omnipresent limitation of computational power, the branches of development stretch in many directions. Classical CNNs are now being challenged and surpassed by novel architectures with fewer parameters and better learning capabilities.

However, the developments do not happen everywhere at once. A revolutionary model architecture will typically first enter the stage to solve one specific problem. As an example, the Transformer model has made a sweeping impact on the field of natural language processing [76]. After some time the architecture made its way into image classification with e.g. the Vision Transformer [24] and DeiT [75] where they are making a significant impact. But there is typically a lag from a model is developed to solve one problem, until someone translates or generalizes the model to solve a different problems. That lag is a roadblock

for progress and should be minimized. It it therefore essential to be attentive and avid of developments and lead the way in the evolution of progress or else developments will stagnate.

A few months ago, a new image classification model called ConvMixer was presented [5]. The model is built with an deep architecture very different from both transformer models and classic CNNs and has a remarkably simple structure (which we will discuss in detail in Section 6). Surpassing both ResNet and DeiT in classification performance with fewer parameters, the model implies a potentially strong network for image analysis which warrants further investigation.

As thoroughly pointed out by Steven Pinker, progress is driven my hypothesis testing and deliberate trial and error [59]. But there are good and bad ways to do trial and error as some errors or successes are more informative than others. Having modular components on systems that are disentangled is a very effective way to isolate variables and increase the informativeness of trials. It is no coincidence that the word "independent" is a particularly familiar word for statisticians. Independent entities are disentangled and therefore much easier to analyze to extract valuable information.

When model architectures successfully have been applied to solve problems in several settings, one will often see references to the common parts in those architectures as a *backbone*. When the backbones are modular and methods of using them allow for that modularity we are able to make much more significant progress with much less data than when these components are still entangled. In the setting of weakly supervised semantic segmentation, some of the most influential and high performing models are not fully modular making it hard to test the significance of detail in labels independently of model architecture and parameter count.

Aside from estimating the advantage of level of detail, there are other significant limitations of modern weakly supervised semantic segmentation (WSSS) methods. One is quite general in the sense that the performance is just not as high as for fully supervised models and not high enough or reliable enough to be used for most applications. One specific problem that keeps the performance low in these models is *Part domination*.

Part domination is the tendency of a model to only focus on the most discriminative parts of an image rather than the whole object of interest. To the best of our knowledge, this was first noted by the creator of the Class Activation Map (CAM) method [85], one of the most influential and successful methods overall in weakly supervised image analysis. Part domination is a problem that has remained a significant issue in WSSS [40, 12, 35] as well as in many neighboring

fields such as weakly supervised object localization (WSOL) [53, 67, 18, 6] and weakly supervised object detection (WSOD)[9, 10]. Many of the cited papers in this paragraph attempt to address the problem while few have improved the overall performance of CAM [19].

1.5 Contributions

In this project, we propose a new semantic segmentation model with an architecture developed on the promising and relatively untested structure of the ConvMixer classification model. We further analyze the proposed model in comparison with a Fully Convolutional Network (FCN) with a ResNet-50 backbone, a previous state-of-the-art model with a classical CNN backbone with a large footprint in the world of image analysis [47, 33].

Additionally, we propose a novel method for weakly supervised semantic segmentation that is completely modular and allows semantic segmentation backbones to be trained with image level labels without adding parameters. This allows us to measure the value of label detail independently of the model architecture. We apply our method to train the proposed semantic segmentation model with image level labels on the Liver Tumor Segmentation Benchmark data set and show experimentally that our modular weakly supervised semantic segmentation model outperforms the Class Activation Map method [85] with high statistical significance.

Finally, as part of the novel weakly supervised semantic segmentation method, we address the problem of part domination by adding a neighborhood correlation enforcement module and show experimentally that the proposed method, with statistical significance, reduces the problem.

1.6 Outline

This thesis starts with a dive into the basics of deep learning, fully connected neural networks. We consider it important to understand the structure and logic that governs the world of artificial neural networks such that they can learn from experience. From fully connected neural networks we will proceed with convolutional neural networks and how and why they have become dominating for image analysis.

Then we will move into the field of weak supervision. Weakly supervised semantic segmentation models will be investigated and discussed before we

present our proposed segmentation model and developmental contributions for weakly supervised semantic segmentation. We explain our motivation and the hypotheses to test in our experiments.

This leads to the experimental design where we describe how we train the models and how we measure their performance. The results of the experiments are then presented along with our quantitative and qualitative analyses. Finally we present some reflections about future developments or further relevant directions of research.

2

What is an Artificial Neural Network

Artificial neural networks are very general statistical models. They can be adapted to replicate a wide variety of different systems that take an input and return an output. To be more specific we need to make some definitions.

We will define everything a system receives to make a decision or action as its inputs, which can be arranged as a vector we denote by $\boldsymbol{x}$. Further, we will define everything the system does or returns based on the inputs as its outputs, a vector we denote by $\boldsymbol{y}$. The mapping of interest is defined by a function $\boldsymbol{f_\theta(x)}$ which transforms the inputs $\boldsymbol{x}$ to the outputs $\boldsymbol{y}$, by rules defined by a set of parameters $\boldsymbol{\theta}$. In this framework, an attempt at a replication of any system where a set of decisions are made based on initial information can be represented by:

$$\boldsymbol{y} = \boldsymbol{f_\theta(x)}$$

2.1 The Structure of Fully Connected Neural Networks

A basic artificial neural network, here represented by the function $\boldsymbol{f_\theta}$, is built up in a repetitive pattern made up of subparts called *layers*. One can think of each layer as a separate smaller function that performs some small analysis and outputs a new, more analyzed representation of the input. All representations of the input are called *feature vectors*. The first layer of $\boldsymbol{f_\theta}$ is formulated in Equation (2.1) where the input vector $\boldsymbol{x}$ is mapped to a hidden[1] feature vector $\boldsymbol{h}_1$.

$$\boldsymbol{h}_1 = \sigma(\boldsymbol{w}_1\boldsymbol{x} + b_1) \tag{2.1}$$

$\boldsymbol{w}_1$ is a vector filled with parameters called the *weights* and b_1 is a constant called the *bias*. The weights decide in what way to linearly combine the features of $\boldsymbol{x}$, before the bias is added. σ is a function called an *activation* or a *non-linearity* and is the component that allows neural networks to model complex non-linear dependencies.

As a side note, letting σ be a linear function turns a neural network (with an arbitrary number of layers) into a linear transformation[2], turning the model into an over-parameterized linear regression model.

A full artificial neural network consists of many stacked layers. A formulation of an arbitrary deep network is shown below.

$$\begin{aligned}
\boldsymbol{h}_1 &= \sigma(\boldsymbol{w}_1\boldsymbol{x} + b_1) \\
\boldsymbol{h}_2 &= \sigma(\boldsymbol{w}_2\boldsymbol{h}_1 + b_2) \\
&\vdots \\
\boldsymbol{y} &= \sigma(\boldsymbol{w}_n\boldsymbol{h}_n + b_n)
\end{aligned}$$

n is here the number of layers. Each layer transforms one feature vector to another and performs some analysis along the way based on the weights and

1. Since there are other layers that follow, $\boldsymbol{h}_1$ is only a temporary representation and will rarely be viewed directly, hence the word *hidden*.
2. Consecutive linear transformations collectively make a linear transformation, but with composite parameters, i.e. several parameters act linearly on the same input values to form the outputs.

biases. Every feature in one layer directly affects every feature in the next. This type of network is therefore called a *fully connected* neural network. One can interpret the structure as letting every feature in one layer influence every feature in the next.

The stack of layers just described, together make up the function shown below.

$$\begin{aligned} \boldsymbol{y} &= \boldsymbol{f}_{\boldsymbol{w}_1,\ldots,\boldsymbol{w}_n,b_1,b_2,\ldots,b_n}(\boldsymbol{x}) \\ \boldsymbol{y} &= \boldsymbol{f}_{\boldsymbol{\theta}}(\boldsymbol{x}), \qquad\qquad \boldsymbol{\theta} = [\boldsymbol{w}_1,\ldots,\boldsymbol{w}_n,b_1,b_2,\ldots,b_n] \end{aligned} \tag{2.2}$$

In this section we have talked about the technical structure of fully connected artificial neural networks. We have introduced the idea of a composite function that, with a suitable choice of weights and biases, is able to model a wide variety of systems.

Choosing the weights and biases that make the function we actually desire is not trivial. To do so we first need a measure of performance.

2.2 Objective Function

To be able to learn by trial and error, we humans need a concept of performance. Some way to understand what needs to change in order to do better. Sometimes this can be provided by a teacher, but in many cases we also have an intuitive understanding ourselves which serves as a good teacher and lets us teach ourselves. When learning to bike, we intuitively know that falling means we did something wrong. Regardless of how we attain our measure of performance, that measure is essential to be able to successfully improve our behaviour.

The tool used to measure performance of neural networks is called an *Objective function*. It is a function that takes a prediction of a network and returns some number that indicates how good the prediction was. The weights and biases of a network can then be optimized or *learned* to give the best possible score by the objective function.

The name *Objective function* is one of many used for the performance measuring function. Among others are *cost function*, *criterion* and *error function*. The scope of each term depends on what the function measures and if a higher or lower score is better, but they all provide the same type of numerical performance measure from which to adjust a network[29].

Objective functions come in different forms depending on how we want a network to behave. In Chapter 5 we will define a task to solve, and show a few examples of objective functions for that task.

We have talked about how a sequence of layers made up of weights and biases forms a flexible function that makes predictions in an attempt to replicate a system. The prediction performance is measured by an objective function which guides adjustments of the network's weights and biases. Gradient descent is an optimization method that can use the objective functions judgments to improve our network's weights and biases.

2.3 Gradient Descent

Gradient descent is a first-order function minimization procedure done by repeatedly taking small steps in the opposite direction of the gradient of a function. The gradient points in the direction of steepest increase of the function at any point, thus by stepping in the opposite direction one will descend along the function hyper surface towards a local minimum.

The method can be visualized as a search for the bottom of a valley by a person wandering around in dense fog. The person is attempting to find the lowest point, but cannot see beyond the ground at her feet. The information she has to work with is the slope of the ground beneath her. By always walking in the direction of steepest descent, she will eventually arrive at some flat plane which hopefully is the bottom of the valley.

In more mathematical terms, Algorithm 1 describes how gradient descent can be applied to minimize a general function f with respect to a parameter θ.

Algorithm 1: Apply gradient descent to minimize a function f with respect to a parameter θ.

Input : An initial parameter guess θ_0 and a step size $\gamma \in \mathbb{R}_+$.
Result: An optimized parameter θ_{min}.

$i \leftarrow 0$;
while *minimum is not reached* **do**
  $\theta_{i+1} \leftarrow \theta_i - \gamma \nabla f(\theta_i)$;
end
$\theta_{min} \leftarrow \theta_{i+1}$;

For an artificial neural network, the parameters to optimize are the weights

and biases of the network and the function to optimize is the objective function. We will here assume the objective function is to be minimized[3].

The activation function and the objective function are in modern networks differentiable by design[4]. This is essential for training as it allows us to compute the gradient of the objective function's outputs with respect to the weights and biases of the network. i.e. The direction of change to the weights and biases that (locally) will give the greatest change to the computed results of the objective function. For a simplified supervised situation, assume we have one data point $\boldsymbol{x}$ with a corresponding label $\boldsymbol{y}$. The algorithm for optimizing a neural network to accurately predict our single data point is formulated below.

Algorithm 2: Optimize a neural network using gradient descent.

Input : An initial parameter vector guess $\boldsymbol{\theta}_0$.
Data: A data vector with a corresponding label vector $\{\boldsymbol{x}, \boldsymbol{y}\}$.
Result: An optimized parameter vector $\hat{\boldsymbol{\theta}}_{min}$.

1 Let the function $\boldsymbol{f}_{\boldsymbol{\theta}}$ be the subject of the minimization with respect to the parameter vector $\boldsymbol{\theta}$. Assume $\boldsymbol{f}_{\boldsymbol{\theta}}$ is locally defined and differentiable at every step of evaluation between $\boldsymbol{\theta}_0$ and the local minimum $\hat{\boldsymbol{\theta}}_{min}$ at which the algorithm will eventually arrive. Let the step size (also called a learning rate) be $\gamma \in \mathbb{R}_+$. The objective function, denoted by $\mathcal{L}$, takes the predictions $\hat{\boldsymbol{y}}$ and the corresponding labels $\boldsymbol{y}$.
2 **while** *minimum is not reached* **do**
3 $\quad \hat{\boldsymbol{y}} \leftarrow \boldsymbol{f}_{\boldsymbol{\theta}_{i-1}}(\boldsymbol{x})$;
4 $\quad \epsilon(\boldsymbol{\theta}) \leftarrow \mathcal{L}(\hat{\boldsymbol{y}}, \boldsymbol{y}; \boldsymbol{\theta})$;
5 $\quad \boldsymbol{\theta}_i \leftarrow \boldsymbol{\theta}_{i-1} - \gamma \nabla_{\boldsymbol{\theta}} \epsilon(\boldsymbol{\theta})$;
6 **end**

Note the vague condition for stopping the while-loop. As the function to optimize is so complex, the parameter landscape is very hard to navigate and understand. Usually, one will never actually know if one has reached the global minimum. Rather, one will monitor how the network performs on data it was not trained on to see if it has learned what we want it to learn.

Generalizing the algorithm to learn from a data set of more than one data point

3. If this is not the case by default, one can easily change a maximization scheme to a minimization scheme by moving in the direction of the gradient rather than in the opposite direction.
4. Note that this is for modern artificial neural networks. Their precursor, the Multilayer Perceptron (MLP) proposed by Frank Rosenblatt [61] used a step function as an activation function. The step function is is non-differentiable and is generally not well suited for gradient descent as the gradient is either 0 or undefined. However, the original MLP was not trained with gradient descent, but by more manual approaches.

can be done in many ways. The simplest method is called Stochastic Gradient Descent and is applied by making an update to the parameter vector $\boldsymbol{\theta}$ for every data point individually before going back to the first data point. This is done repeatedly until one believes a good minimum has been reached. There are other methods, such as *Batch Gradient Descent* and *Mini-Batch Gradient Descent* which we will revisit in Section 2.6.

2.4 Back Propagation

When applying gradient descent to train neural networks, every weight and bias needs to be adjusted. In Algorithm 2 the process is compressed into a single line (line 5), which might give the impression that the process is uncomplicated. This is not the case as the size of modern neural networks makes the task of finding the gradient with respect to all weights and biases infeasible unless a very efficient method is used. Typically when using gradient descent, one would differentiate with respect to every weight and bias and make an update to each weight and bias independently. Back propagation is a much more efficient method which utilizes the layer structure of artificial neural networks to compute the gradient of the weights for each layer sequentially. By use of the chain rule for differentiation, one can update layer for layer of weights in an efficient and orderly manner.

In order to do proper dissection of the back propagation scheme, let us revisit the formulation of an arbitrarily deep fully connected neural network first seen in Section 2.1. However, note that we have changed $\boldsymbol{y}$ to $\hat{\boldsymbol{y}}$ to emphasize that the network only makes a prediction.

$$\boldsymbol{h}_1 = \sigma(\boldsymbol{w}_1\boldsymbol{x} + b_1) \tag{2.3}$$
$$\boldsymbol{h}_2 = \sigma(\boldsymbol{w}_2\boldsymbol{h}_1 + b_2) \tag{2.4}$$
$$\vdots \tag{2.5}$$
$$\hat{\boldsymbol{y}} = \sigma(\boldsymbol{w}_n\boldsymbol{h}_n + b_n) \tag{2.6}$$

Assume we have been given a data point $\boldsymbol{x}$ with a corresponding label $\boldsymbol{y}$ to train on. First we compute a prediction $\hat{\boldsymbol{y}}$. Then we compute the measure of performance $\epsilon = \mathcal{L}(\hat{\boldsymbol{y}}, \boldsymbol{y})$. The next step is to find the gradient of ϵ with respect to all weights and biases.

If we want to understand the effects on the outputs caused by changes in the weights of the first layer, we have to take into account the effects the weights

of the first layer has on all the other weights in the network. The causality implies that a reasonable approach is to begin by focusing on the weights and biases in the last layer as shown below.

$$\begin{aligned}
\frac{\partial \epsilon}{\partial \boldsymbol{w}_n} &= \frac{\partial \epsilon}{\partial \hat{\boldsymbol{y}}} \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{w}_n} \\
&= \frac{\partial \epsilon}{\partial \hat{\boldsymbol{y}}} \frac{\partial \sigma}{\partial (\boldsymbol{w}_n \boldsymbol{h}_n + b_n)} \frac{\partial (\boldsymbol{w}_n \boldsymbol{h}_n + b_n)}{\partial \boldsymbol{w}_n} \\
\frac{\partial \epsilon}{\partial \boldsymbol{w}_n} &= \frac{\partial \epsilon}{\partial \hat{\boldsymbol{y}}} \sigma'(\boldsymbol{w}_n \boldsymbol{h}_n + b_n) \boldsymbol{h}_n
\end{aligned} \tag{2.7}$$

$$\begin{aligned}
\frac{\partial \epsilon}{\partial b_n} &= \frac{\partial \epsilon}{\partial \hat{\boldsymbol{y}}} \frac{\partial \hat{\boldsymbol{y}}}{\partial b_n} \\
\frac{\partial \epsilon}{\partial b_n} &= \frac{\partial \epsilon}{\partial \hat{\boldsymbol{y}}} \sigma'(\boldsymbol{w}_n \boldsymbol{h}_n + b_n)
\end{aligned} \tag{2.8}$$

$(\boldsymbol{w}_n \boldsymbol{h}_n + b_n)$ is often referred to as the *potential* or the *un-activated* output of the n-th layer. Note that the weights $\boldsymbol{w}_n$ and the bias b_n act on the same step of the equation, namely the n-th activation. Therefore the gradient propagating backwards from ϵ towards $\boldsymbol{w}_n$ and b_n is the same for the two until the very last step. This explains the short derivation of the bias in Equation (2.8).

The common parts of the equations (2.7) and (2.8) show why the objective function and activation function are differentiable by design, as their derivatives are used to compute the gradients. $\boldsymbol{h}_n$ and the n-th potential are both computed as part of the computation of making a prediction $\hat{\boldsymbol{y}}$, thus by temporary storing intermediate values when computing the prediction, little extra computation is needed to find the gradient wrt. $\boldsymbol{w}_n$ and b_n.

We have shown how to find the gradients wrt. the weights and biases of the last layer. To access weights and biases of earlier layers let us think of the first $n-1$ layers as its own self-standing neural network and the last layer as part of the objective function. From this perspective the prediction of the networks $n-1$ layers would be $\boldsymbol{h}_n$, and the measure of performance of that prediction would be $\frac{\partial \epsilon}{\partial \boldsymbol{h}_n}$ shown in Equation (2.9).

$$\begin{aligned}
\frac{\partial \epsilon}{\partial \boldsymbol{h}_n} &= \frac{\partial \epsilon}{\partial \hat{\boldsymbol{y}}} \frac{\partial \hat{\boldsymbol{y}}}{\partial \boldsymbol{h}_n} \\
\frac{\partial \epsilon}{\partial \boldsymbol{h}_n} &= \frac{\partial \epsilon}{\partial \hat{\boldsymbol{y}}} \sigma'(\boldsymbol{w}_n \boldsymbol{h}_n + b_n) \boldsymbol{w}_n
\end{aligned} \tag{2.9}$$

Notice that the dependence between $\hat{\boldsymbol{y}}$ and $\boldsymbol{h}_n$ is very similar to the dependence between $\boldsymbol{h}_n$ and $\boldsymbol{h}_{n-1}$. By use of the same method as above to find the gradient with respect to the newly assigned last layer of our network, we get the following:

$$\begin{aligned}
\frac{\partial \epsilon}{\partial \boldsymbol{w}_{n-1}} &= \frac{\partial \epsilon}{\partial \boldsymbol{h}_n}\frac{\partial \boldsymbol{h}_n}{\partial \boldsymbol{w}_{n-1}} \\
\frac{\partial \epsilon}{\partial \boldsymbol{w}_{n-1}} &= \frac{\partial \epsilon}{\partial \boldsymbol{h}_n}\sigma'(\boldsymbol{w}_{n-1}\boldsymbol{h}_{n-1} + b_{n-1})\boldsymbol{h}_{n-1} \\
\frac{\partial \epsilon}{\partial \boldsymbol{w}_{n-1}} &= \frac{\partial \epsilon}{\partial \hat{\boldsymbol{y}}}\sigma'(\boldsymbol{w}_n\boldsymbol{h}_n + b_n)\boldsymbol{w}_n\sigma'(\boldsymbol{w}_{n-1}\boldsymbol{h}_{n-1} + b_{n-1})\boldsymbol{h}_{n-1}
\end{aligned} \tag{2.10}$$

$$\begin{aligned}
\frac{\partial \epsilon}{\partial b_{n-1}} &= \frac{\partial \epsilon}{\partial \boldsymbol{h}_n}\frac{\partial \boldsymbol{h}_n}{\partial b_{n-1}} \\
\frac{\partial \epsilon}{\partial b_{n-1}} &= \frac{\partial \epsilon}{\partial \boldsymbol{h}_n}\sigma'(\boldsymbol{w}_{n-1}\boldsymbol{h}_{n-1} + b_{n-1}) \\
\frac{\partial \epsilon}{\partial b_{n-1}} &= \frac{\partial \epsilon}{\partial \hat{\boldsymbol{y}}}\sigma'(\boldsymbol{w}_n\boldsymbol{h}_n + b_n)\boldsymbol{w}_n\sigma'(\boldsymbol{w}_{n-1}\boldsymbol{h}_{n-1} + b_{n-1})
\end{aligned} \tag{2.11}$$

The pattern has been generalized to formulas for the gradients with respect to wrt. all weights and biases shown below.

$$\frac{\partial \epsilon}{\partial \boldsymbol{w}_{i-1}} = \frac{\partial \epsilon}{\partial \hat{\boldsymbol{y}}}\prod_{j=0}^{i-1}\sigma'(\boldsymbol{w}_{n-j}\boldsymbol{h}_{n-j} + b_{n-j})\boldsymbol{h}_{n-1} \tag{2.12}$$

$$\frac{\partial \epsilon}{\partial b_{i-1}} = \frac{\partial \epsilon}{\partial \hat{\boldsymbol{y}}}\prod_{j=0}^{i-1}\sigma'(\boldsymbol{w}_{n-j}\boldsymbol{h}_{n-j} + b_{n-j}) \tag{2.13}$$

The equations (2.12) and (2.13) can be used to compute each and every gradient independently. But the products also suggest that if you have already found the gradient wrt. a layer, little extra work is needed to compute the gradient wrt. the layer before it; a testament to the efficiency of back propagation.

2.5 Activations

Activations or activation functions of an artificial neural network allow it to learn complex dependencies. But they also influence a networks training properties

that heavily depend on the activations' derivatives. The most used activation functions are *Sigmoid*, *Tanh*, *Softmax*, *Rectified Linear Unit* (ReLU) [5]. There are two main relevant properties of these functions.

Firstly, the range of the functions. All of these functions have a domain that covers the real line, but their range of output can be actively used to serve a purpose. A probability ranges between 0 and 1, so does the output of a Sigmoid. If we want to train a network to output a probability we could choose to use the sigmoid as the last activation as the output would naturally be limited to the sensible range.

Secondly, their derivative. The formulas (2.12) and (2.13) show that the weight gradients used to adjust the network are made up of large products of activation derivatives; especially those of early layers. To understand why this causes a problem let us investigate the Sigmoid function.

Vanishing gradients

The Sigmoid maps the real line to the interval $(0, 1)$. As the real line is infinite and the interval $(0, 1)$ is not, the function has to compress some parts of the real line into an infinitesimal small size. Figure 2.1 illustrates the function along with its derivative. Notice that any large x-value is mapped very close to 1 and any very low x-value is mapped close to 0. For input values far from 0, changes to the input has little affect on the output, and the gradient diminishes. For training a neural network, this means that any gradient step to improve its weights and biases will be extremely small if the prediction is confident. The matter is made exponentially worse since a prediction might be confidently predicted through several layers, thus imposing the mitigation of a weak activation derivative numerous times. This issue is called *vanishing gradients*. The result is that for neural networks using the sigmoid activation at every layer, an increase in number of layers will cause significant mitigation of gradients. This will prevent such networks from successfully training their early layers.

Exploing gradients

The opposite problem called *exploding gradients* arises if one attempts to remedy vanishing gradients by scaling up the derivative of the activations. By imposing a scaling factor of, say 10, the vanishing gradient problem would be reduced as much more extreme values of activation inputs would be needed to severely

5. There are many variations of ReLU, but we will only mention the main one here.

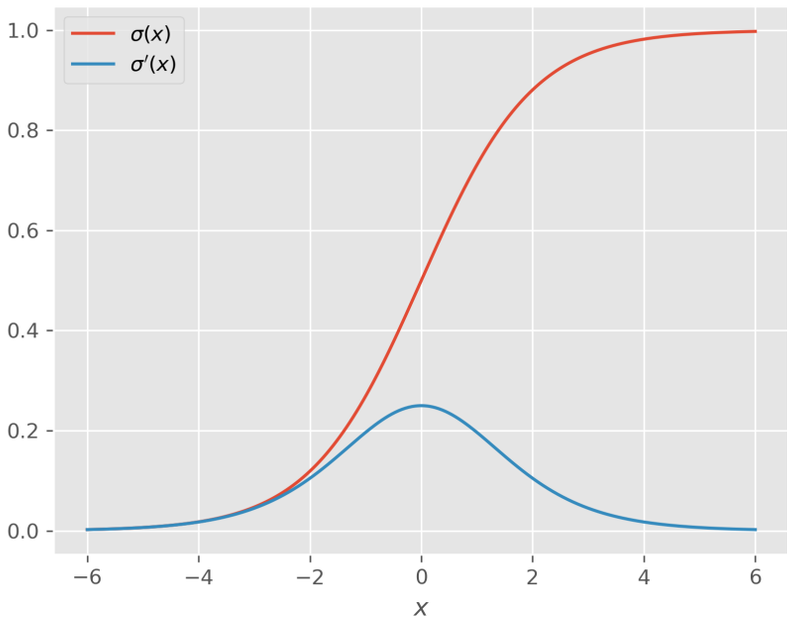


Figure 2.1: The Sigmoid activation function with its derivative.

diminish the gradients. However, imagine that a prediction of an input data point resulted in a bad prediction, but with every activation input close to 0 (view the derivative in Figure 2.1 and multiply by 10.). The prediction error would be significant since the prediction is wrong, and the scaling of activation derivatives (view the equations (2.12) and (2.13)) would make the gradients grow exponentially as they propagate back through the network. Because of the exponential growth, that one erroneous prediction would cause havoc to the early layers, effectively resetting weights and biases with significant relevance to the prediction. The careful coordinated nudging of the weights and biases prior to that prediction would be wasted and the network would have to be trained slowly again to recover.

Activations that suffer from vanishing gradients and therefore cannot support very deep networks are the sigmoid and tanh. Tanh is not shown here, but visually it looks very similar to the sigmoid but with a range of $(-1, 1)$. Its derivative is also similar, but with a peak around $x = 0$ of 1 and with a lobe that is a little narrower than for the sigmoid.

The Rectified Linear Unit(ReLU), or variations of it, is the most used activation function in modern CNNs. This is mainly due to its very advantageous gradient properties. The function is equal to its input values if they are positive, and

0 otherwise. This makes activation gradients that are either 0 or 1. If they are 1, the scaling of gradients propagation through a network is much easier to control than those of the sigmoid or the tanh. Most variations of ReLU are modifications that avoid the case of a 0 gradient for negative activation inputs as this causes no learning with difficult recovery; an issue often referred to as *dead nodes*.

One of the descendants of the ReLU is the Gaussian Error Linear Unit (GELU) activation function. It addresses the problem of dead nodes by allowing for a smoother transition into the negative part of the input domain to only taper off towards zero asymptotically. The function can be formulated as $\text{GELU}(x) = x\phi(x)$, where ϕ is the cumulative distribution function of a standard normal random variable.

Softmax is an activation that acts only on multidimensional inputs and its outputs in one dimension depends on the input values in the other dimensions. It has the convenient feature that the sum across all dimensions of its outputs is 1. Thus one can interpret the outputs of the softmax as a discrete probability distribution. A property making it ideal as a final mapping in a network used for classification. The softmax can be seen as a generalization of the sigmoid[29] and the two bear similarities in gradient behavior. The value of all other dimensions held constant, the gradient with respect to one dimension of the softmax will look very similar to the derivative of the sigmoid, but with the lobe centered at the maximum value of the other dimensions. The softmax is rarely used inside deep neural networks though there are exceptions (attention modules).

A mathematical formulation of the functions mentioned here are shown below.

$$S(x) = \frac{1}{1 + e^{-x}} \tag{2.14}$$

$$S'(x) = S(x)(1 - S(x)) \tag{2.15}$$

$$tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.16}$$

$$tanh'(x) = 1 - tanh^{2}(x) \tag{2.17}$$

$$\text{softmax}(x_i) = \frac{e^{x_i}}{\sum_{i=1}^{k} e^{x_i}} \tag{2.18}$$

$$\frac{\partial \text{softmax}(x_i)}{\partial x_j} = \begin{cases} \text{softmax}(x_i)(1 - \text{softmax}(x_j)) & , \ i = j \\ -\text{softmax}(x_i)\text{softmax}(x_j) & , \ i \neq j \end{cases} \tag{2.19}$$

$$\text{ReLU}(x) = \max(x, 0) \tag{2.20}$$

$$\text{ReLU}'(x) = \begin{cases} 0 & , \ x < 0 \\ 1 & , \ x > 0 \end{cases} \tag{2.21}$$

$$\text{GELU}(x) = xP(X < x), \qquad X \sim N(0, 1) \tag{2.22}$$

2.6 Optimizations

There are three main problems that will probably present themselves if no optimizations are applied to the training scheme of artificial neural networks we have talked about thus far.

The first is that the training is slow or stagnant. The second is training that converges to a bad result. The third problem is overfitting which in essence means that our network has learned too much about the training data. When a model makes predictions based on idiosyncratic details rather than a general system, it will work badly when presented with new data.

Assuming a convex, and sufficiently smooth function $\boldsymbol{f}_\theta(\boldsymbol{x})$, gradient descent as described in Section 2.3 can be shown to always reach the global minimum[31]. However, for neural networks, the parameter landscape is not convex, nor particularly smooth[36]. Consequently, training neural networks have traditionally been notoriously difficult [27]. Several decades of research on the subject has fortunately resulted in a forest of optimizations and modifications that have improved the method in different ways.

2.6.1 Batch Size Control

Batch size control refers to controlling the number of data points to use for each update of the weights and biases. The true parameter landscape we are interested in finding the bottom of is based on the true distribution of our data; a distribution which will be forever out of our reach. We might thus assume our best approach would be to compute the mean gradient based on all our available data before doing an update of the weights and biases. This approach is called *Batch gradient descent*. However, this has not proved the best solution for a different reason.

Batch size control addresses the second issue of finishing training at a bad solution. Gradient descent looks for minima, but does not distinguish between the bottom of a crater or a tiny bump in the parameter landscape. To avoid the bumps one could try to search around an approached solution in every direction to see if it makes sense to keep searching. But the parameter landscape of neural networks typically have millions of dimensions. The curse of dimensionality thus makes this seemingly small task enormous. The best we can do is introduce some random movement which might cause the search to jump out of small local minima. Notice that if we introduce too much randomness, we might just end up jumping around leaving any minimum we encounter regardless of how successful the solution might have been.

The method of batch size control that introduces the most randomness is called *Stochastic gradient descent* and is performed by doing an update of the weights and biases for every individual data point.

Between the two extreme methods we have *Mini-batch gradient descent* which makes an parameter update with the mean gradient based on a subpart of the given data. This is the most used approach which strikes a balance between searching a representative parameter landscape while introducing enough randomness to avoid the worst local minima.

Note that it has been shown that even if one never reaches the global minimum, there are a huge number of comparably good local minima. Thus if one escapes the worst local minima, one will probably approach a good solution even though one might not arrive at the same solution upon repetition of the training.

2.6.2 Variants of Gradient Descent

The following section is based on an excellent review of gradient descent methods by Sebastian Ruder [63].

Imagine that you were the person walking down the slope of a valley in the dense fog and you found yourself walking in an oscillating pattern downwards. By remembering the slope at the prior few steps, you might understand that you are in fact traversing slowly down a declining gulch. To make the descent more effective you reasonably adjust your next step so that you will probably stop moving across the gulch and more in direction of its descent. Another method that might be effective is to move around the current spot a little to figure out the geometry of the local environment before deciding where to move next. By viewing the slope at the current location in context with all the information you already have about the landscape, you can make your descent more effective. In the context of a gradients, by taking already acquired information into account or doing small acts of scouting, we can address the problem of slow training. This is common of methods such as *Momentum*, *Nesterov accelerated gradients*, *Adagrad*, *Adadelta*, *RMSprop* and *Adam* and *AngularGrad*.

In Algorithm 2, the updating step of the weights and biases was compressed into the formulation shown in Equation (2.23).

$$\boldsymbol{\theta}_i \leftarrow \boldsymbol{\theta}_{i-1} - \gamma \nabla_{\boldsymbol{\theta}} \epsilon \tag{2.23}$$

ϵ is the objective, γ is a constant learning rate and $\boldsymbol{\theta}$ is the vector of all weights and biases of the network. By use of the same notation we will now go through the family of improvements.

Momentum

Gradient descent with momentum introduces a positive autocorrelation with the prior gradient estimate. This causes less jagged movements and makes the parameter changes more gradual and smooth. One can visualize it as having a heavy ball rolling around in a landscape rather than a person walking.

$$\begin{aligned} \boldsymbol{v}_i &\leftarrow \alpha \boldsymbol{v}_{i-1} + \beta \nabla_{\boldsymbol{\theta}} \epsilon(\boldsymbol{\theta}) \\ \boldsymbol{\theta}_i &\leftarrow \boldsymbol{\theta}_{i-1} - \boldsymbol{v}_i \end{aligned} \tag{2.24}$$

α and β are constants that both serve as learning rate and distribute weight on the current and prior computed gradient.

Nesterov Momentum

Nesterov momentum or Nesterov accelerated gradient is very similar to the method of momentum, but with a small modification. One will first move forward one step based on the current gradient to scout for the gradient there. Then one will go back and make the move as a weighted combination of the current and the scouted gradient. The method is formulated in Equation (2.25).

$$\begin{aligned} \boldsymbol{v}_i &\leftarrow \alpha \boldsymbol{v}_{i-1} + \beta \nabla_{\boldsymbol{\theta}} \epsilon(\boldsymbol{\theta} - \alpha v_{i-1}) \\ \boldsymbol{\theta}_i &\leftarrow \boldsymbol{\theta}_{i-1} - \boldsymbol{v}_i \end{aligned} \tag{2.25}$$

Adagrad

Each weight and bias will affect the decision making of the network differently, thus having one constant learning rate that applies to all weights and biases might not be the best choice. Adagrad introduces a learning rate for each weight and bias that changes dynamically during training. The method is formulated below.

$$\begin{aligned} \boldsymbol{g}_{i-1} &\leftarrow \nabla_{\boldsymbol{\theta}} \epsilon(\boldsymbol{\theta}_{i-1}) \\ \boldsymbol{G}_{i-1} &\leftarrow \operatorname{diag}\left(\sum_{t=1}^{i-1} \boldsymbol{g}_t \boldsymbol{g}_t^T\right) \\ \boldsymbol{\theta}_i &\leftarrow \boldsymbol{\theta}_{i-1} - \frac{\gamma}{\sqrt{\boldsymbol{G}_{i-1} + \xi}} \circ \boldsymbol{g}_{i-1} \end{aligned} \tag{2.26}$$

ξ is a small number to avoid extremely high values, $\circ$ is used to denote elementwise multiplication and $\boldsymbol{G}_{i-1}$ is the sum of the elementwise squared gradients of all the recorded gradients up until the current time step. A frequent issue with this method is that the learning rate will monotonically decrease quite aggressively which will stop the learning prematurely.

RMSprop

RMSprop is a variation of Adagrad that seeks to mitigate its aggressive shrinking of gradients. The method switches out the total sum of squared gradients $\boldsymbol{G}_{i-1}$, with a decaying recursive mean square of past gradients. The method is formulated below.

$$\begin{aligned} \boldsymbol{g}_{i-1} &\leftarrow \nabla_{\boldsymbol{\theta}} \epsilon(\boldsymbol{\theta}_{i-1}) \\ E[\boldsymbol{g}^2]_{i-1} &\leftarrow \alpha E[\boldsymbol{g}^2]_{i-2} + (1-\alpha)\boldsymbol{g}_{i-1}^2 \\ \boldsymbol{\theta}_i &\leftarrow \boldsymbol{\theta}_{i-1} - \frac{\gamma}{\sqrt{E[\boldsymbol{g}^2]_{i-1} + \xi}} \circ \boldsymbol{g}_{i-1} \end{aligned} \tag{2.27}$$

Note that $\boldsymbol{g}^2$ means the elementwise squared vector of $\boldsymbol{g}$, not the 2-norm.

Adadelta

Adadelta was developed in parallell with RMSprop to address Adagrads problem of diminishing parameter changes. The method is almost identical to RMSprop, but removes the hyperparameter γ. The replacement of γ is rather involved as it uses prior parameter changes in addition to prior gradients to to the updating. Let us define the current change to the parameters as $\Delta\boldsymbol{\theta}_i$. The method is thus formulated below.

$$\begin{aligned} E[\Delta\boldsymbol{\theta}^2]_{i-1} &\leftarrow \alpha E[\Delta\boldsymbol{\theta}^2]_{i-2} + (1-\alpha)\Delta\boldsymbol{\theta}_{i-1}^2 \\ RMS[\Delta\boldsymbol{\theta}]_{i-1} &\leftarrow \sqrt{E[\Delta\boldsymbol{\theta}^2]_{i-1} + \xi} \\ \\ \boldsymbol{g}_{i-1} &\leftarrow \nabla_{\boldsymbol{\theta}} \epsilon(\boldsymbol{\theta}_{i-1}) \\ E[\boldsymbol{g}^2]_{i-1} &\leftarrow \alpha E[\boldsymbol{g}^2]_{i-2} + (1-\alpha)\boldsymbol{g}_{i-1}^2 \\ RMS[\boldsymbol{g}]_{i-1} &\leftarrow \sqrt{E[\boldsymbol{g}^2]_{i-1} + \xi} \\ \\ \Delta\boldsymbol{\theta}_{i-1} &\leftarrow -\frac{RMS[\Delta\boldsymbol{\theta}]_{i-1}}{RMS[\boldsymbol{g}]_{i-1}} \circ \boldsymbol{g}_{i-1} \\ \boldsymbol{\theta}_i &\leftarrow \boldsymbol{\theta}_{i-1} + \Delta\boldsymbol{\theta}_{i-1} \end{aligned} \tag{2.28}$$

Note that $RMS[\Delta\boldsymbol{\theta}]_{i-1}$ takes the place of γ.

Adam

Adam is short for Adaptive Moment Estimation and is considered a good default optimization algorithm for neural networks. As we have just seen, in Adadelta we keep track of an autoregressive estimate of the squared gradients. With Adam, we do the same for both the gradients and the squared gradients. Or rather, we keep track of weighted estimates of the first and second moments of

the gradients where recent gradients count more than old ones. The method is formulated below.

$$
\begin{aligned}
\boldsymbol{g}_{i-1} &\leftarrow \nabla_{\boldsymbol{\theta}} \epsilon(\boldsymbol{\theta}_{i-1}) \\
m_{i-1} &\leftarrow \beta_1 m_{i-2} + (1-\beta_1)\boldsymbol{g}_{i-1} \\
v_{i-1} &\leftarrow \beta_2 v_{i-2} + (1-\beta_2)\boldsymbol{g}_{i-1}^2 \\
\hat{m}_{i-1} &\leftarrow \frac{m_{i-1}}{1-\beta_1} \\
\hat{v}_{i-1} &\leftarrow \frac{v_{i-1}}{1-\beta_2} \\
\boldsymbol{\theta}_i &\leftarrow \boldsymbol{\theta}_{i-1} - \frac{\gamma}{\sqrt{\hat{v}_{i-1}} + \xi}\hat{m}_{i-1}
\end{aligned}
\tag{2.29}
$$

$\hat{m}_{i-1}$ and $\hat{v}_{i-1}$ are adjusted as m_{i-1} and v_{i-1} are biased estimators of the gradient moments. β_1, β_2, ξ and γ are hyper parameters. The authors proposed the values $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\xi = 10^{-8}$, while the learning rate γ must be adapted to the specific case.

AngularGrad

AngularGrad is a gradient optimization method that not only records the gradients and parameter steps, but also the changes in parameter movement angle [62]. It can be seen as a generalization of Adam with one additional factor reducing the oscillatory movements in the parameter space. The methods is formulated below.

$$
\begin{aligned}
\boldsymbol{g}_{i-1} &\leftarrow \nabla_{\boldsymbol{\theta}} \epsilon(\boldsymbol{\theta}_{i-1}) \\
A_{i-1} &\leftarrow \tan^{-1} \left| \frac{\boldsymbol{g}_{i-1} - \boldsymbol{g}_{i-2}}{1 + \boldsymbol{g}_{i-1} \cdot \boldsymbol{g}_{i-2}} \right| \\
A_{\min} &\leftarrow \min(A_{i-2}, A_{i-1}) \\
\phi_{i-1} &\leftarrow \tanh(|\angle(A_{\min})|) \cdot \lambda_1 + \lambda_2 \\
m_{i-1} &\leftarrow \beta_1 m_{i-2} + (1 - \beta_1) \boldsymbol{g}_{i-1} \\
v_{i-1} &\leftarrow \beta_2 v_{i-2} + (1 - \beta_2) \boldsymbol{g}_{i-1}^2 \\
\hat{m}_{i-1} &\leftarrow \frac{m_{i-1}}{1 - \beta_1} \\
\hat{v}_{i-1} &\leftarrow \frac{v_{i-1}}{1 - \beta_2} \\
\boldsymbol{\theta}_i &\leftarrow \boldsymbol{\theta}_{i-1} - \frac{\gamma \cdot \phi_{i-1}}{\sqrt{\hat{v}_{i-1}} + \xi} \hat{m}_{i-1}
\end{aligned}
\tag{2.30}
$$

$\angle(\cdot)$ is either $\cos(\cdot)$ or $\tan(\cdot)$ and $\lambda_1, \lambda_2 \in \mathbb{R}$ are hyper parameters, both proposed to have a default value of $1/2$ based on experimental results [62]. β_1, β_2 are hyper parameters adopted from Adam.

2.6.3 Batch Normalization

Batch normalization is a procedure used between successive layers in artificial neural networks to improve their training properties. The process is done by normalizing every batch going into a layer to have mean 0 and standard deviation 1, before introducing trainable parameters to scale and shift the standardized batch. The advantageous effects of batch normalization are well established and the technique is widely used in modern networks. It has been believed that the success of the procedure was a consequence of mitigating *internal covariate shift* (ICS)[36], but this has been shown to be incorrect [65]. Rather, batch normalization smooths the parameter landscape making it easier for optimizers to work with.

The formulation of the method shown below is taken from the original proposal of the method [36].

The input $\boldsymbol{x}$ is the un-activated output of a layer and $\boldsymbol{y}$ are the values passed on to an activation function. The parameters $\boldsymbol{\gamma}$ and β are parameters to be learned by the network. $\mathcal{B}$ is a batch of m data samples. ϵ is a nonzero (but very small) constant to avoid zero division.

Algorithm 3: Batch normalization for a batch of unactivated input data $\boldsymbol{x}$.

Input: A mini-batch $\mathcal{B} = \{\boldsymbol{x}\}_{i=1}^{m}\}$. Parameters to be learned: $\boldsymbol{\gamma}, \beta$.
Output: $\{\boldsymbol{y}_i = \mathbf{BN}_{\boldsymbol{\gamma},\beta}(\boldsymbol{x}_i)\}$

1

$$\boldsymbol{\mu}_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}\boldsymbol{x}_i \tag{2.31}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(\boldsymbol{x}_i - \boldsymbol{\mu}_{\mathcal{B}})^2 \tag{2.32}$$

$$\hat{x}_i \leftarrow \frac{\boldsymbol{x}_i - \boldsymbol{\mu}_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \tag{2.33}$$

$$\boldsymbol{y}_i \leftarrow \boldsymbol{\gamma}\hat{x}_i + \beta \equiv \mathbf{BN}_{\boldsymbol{\gamma},\beta}(\boldsymbol{x}_i) \tag{2.34}$$

2.6.4 Initialization

The gradient descent algorithm need a starting point, a first candidate parameter vector to work with. A naive approach would simply be to set the weights randomly. A somewhat more careful, though arbitrary, approach would be to choose them randomly, but from a distribution of low values such that no initial parameter value dominates a prediction.

Recall the issue of vanishing gradients. Inputs to the sigmoid or tanh that are too big or too small will cause slow or stagnant training. If not being careful about the initialization, almost all inputs to activations could be extreme values and would consequently prevent networks from learning. To avoid this problem Xavier Glorot et al. proposed an initialization method that would make the distribution of the back propagating gradients stay constant through the network[27]. The proposed approach was specific for the sigmoid or tanh activations and is formulated below.

$$W \sim U\left(-\frac{1}{\sqrt{n}}, \frac{1}{\sqrt{n}}\right) \tag{2.35}$$

Every weight in a layer of n features would be an observation of the stochastic variable W.

Kaiming He et al. had a similar approach for finding a good initialization when using the rectified linear unit activation. They found the variance of weights

that would let the variance of the activated features stay constant through a network. Consequently, a distribution was proposed to draw initial weights from given a layer with a number of nodes n (shown below).

$$W \sim N\left(0, \frac{2}{n}\right) \tag{2.36}$$

2.7 Evaluation

Earlier we discussed the role of the objective function as a teacher lecturing the neural network to improve. However, there is a distinction between two types of evaluation we have not made clear. The one we have discussed is an evaluation of a prediction done by an objective function with the main goal of providing the feedback that most effectively lets a network improve its performance. The other type, done with a function often referred to as an *evaluation metric* or just a metric, gives a measure of performance meant for users of a model to know the capabilities of the network.

A function that provides very good feedback for the network to train is a function that has a gradient that increases in magnitude as a prediction gets worse. This property lets the training go faster as very bad predictions will cause gradients of large magnitude encouraging the optimization scheme to take bigger steps. When approaching a minimum, the gradients will be smaller and the optimization scheme will naturally slow down and take small steps.

Gradient properties such as those just described are not very desirable for a function which is meant to clearly indicate the performance of a network. Small differences between two bad models will seem much more significant than small differences between two good models. To better be able to distinguish usable models, we might even desire the opposite gradient behavior, i.e. large magnitude gradients for good predictions. This would make small variations in performance seem more significant for a very good model than for a bad one which intuitively makes more sense. Note that functions used for evaluation do not need to be differentiable.

It should also be mentioned that even though we use the term *metric*, it is not always the case that the function used satisfies the demands of being a mathematical metric function. It is typically a function that compares a prediction and a label yielding a number that depends on their difference. However, evaluation metrics might yield negative outputs and will not typically be 0 for perfect predictions. One example is Matthews correlation coefficient

(MCC) ranging between −1 and 1, where a perfect prediction yields a value of 1.

Occasionally the two evaluation types can be performed with the same tool, i.e. the objective function and the evaluation metric are the same function, but this is generally not the case.

3

Convolutional Neural Networks

Convolutional neural networks (CNNs) make up a class of deep neural networks that has had great impact, particularly for image analysis. From self-driving cars to measurements of the size of the pole ice sheets, CNNs are tools with a currant and future footprint on our world.

In the last chapter we talked about fully connected neural networks (FCNNs) that connect every feature in one layer with every feature in the next. A convolutional layer on the other hand, will only let a feature in one layer influence a neighborhood of features in the next. Such restrictions significantly limit the number of possible model configurations. Such restrictions cause model limitations, but also advantages.

If the model restrictions remove connections that would prove relevant in the network, it will not be able to learn a good parameter configuration and no amount of data or training time could lead to a good model. However, a smaller, less complex model is easier to train. So if the restrictions reflect valid assumptions about the data generating process, the model restrictions help the training by limiting the number of parameters to adjust.

3.1 Convolutions

In this section we will go through how discrete 2 dimensional convolutions are used in context of Convolutional neural networks(CNNs).

A convolution as used in a CNN is an operation that works with grids of numbers. The operation takes an image $\boldsymbol{x}$ and a filter $\boldsymbol{\omega}$ and produces an output image $\boldsymbol{y}$.

The sizes of the grids used in most real applications can get quite large and are thus difficult to illustrate. For clarity, we will therefore assume both the image and the filter are monochromatic, meaning there is only one value per pixel. Generalizations will be shortly explained. Let us begin with some definitions:

The image, denoted by $\boldsymbol{x}$, is a rectangular grid of pixels with a height of $h_{\boldsymbol{x}}$ pixels and a width of $w_{\boldsymbol{x}}$ pixels. The sizes of the filter and the output image are correspondingly $(h_{\boldsymbol{\omega}}, w_{\boldsymbol{\omega}})$ and $(h_{\boldsymbol{y}}, w_{\boldsymbol{y}})$ respectively. The pixel on row i and column j of the image is denoted by $x(i, j)$ for $i = 1, \ldots, h_{\boldsymbol{x}}$ for $j = 1, \ldots, w_{\boldsymbol{x}}$. References to single pixels in the filter and the output image follow the same pattern.

3.1.1 The Basic Convolution

Computing one pixel in the output image can be divided into 3 steps:

1. Place the filter on top of the image. The placement of the filter relative to the image is what defines which pixel of the output image is being computed.

2. In the region where the filter and image overlap, compute an elementwise product between each number in the filter and the corresponding number in the image.

3. Sum the resulting products.

To visualize the operation, view Figure 3.1. In the figure, an image of size $(5, 5)$ is convolved with a filter of size $(3, 3)$ resulting in an output image of size $(3, 3)$[1]. In in each part of the figure, the image is placed at the bottom with the filter placed over it to compute one pixel of the output image shown on top.

1. The figure shows the result of the first two and the last step of the convolution.

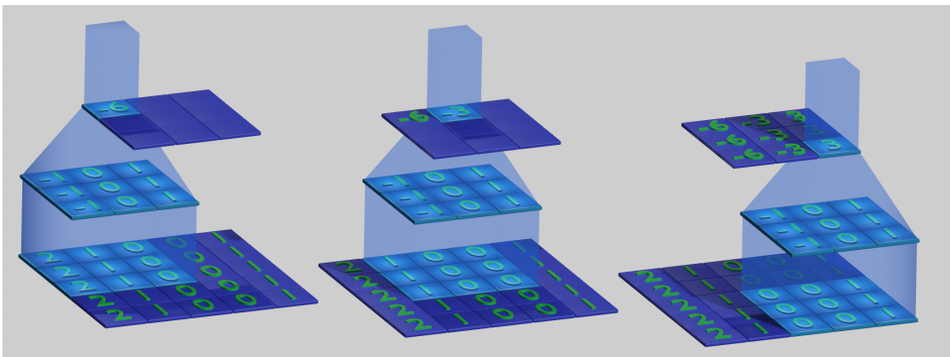


Figure 3.1: Visualization of the first, second and last step of a convolution.

It should be mentioned that before the 3 listed steps, a convolution as defined in mathematics would begin by flipping the filter both vertically and horizontally. However, as the filters in CNNs are to be learned during training, the flipping has no practical effect on the network and is hence omitted.

A mathematical formulation of a convolution[2] applied to the image $\boldsymbol{x}$ and filter $\boldsymbol{\omega}$ is given in Equation 3.1.

$$\boldsymbol{y} = \boldsymbol{x} * \boldsymbol{\omega}$$

$$y(a,b) = \sum_{i=0}^{h_{\boldsymbol{\omega}}-1} \sum_{j=0}^{w_{\boldsymbol{\omega}}-1} x(a+i, b+j)\omega(h_{\boldsymbol{\omega}}+i, w_{\boldsymbol{\omega}}+j) \tag{3.1}$$

$$a = 1, \ldots, h_{\boldsymbol{y}}, \quad b = 1, \ldots, w_{\boldsymbol{y}} \tag{3.2}$$

A way to understand the operation is to think of the filter as a template for a pattern of interest. The template is then passed all over the image to get a measure of match between the the image and our pattern of interest. The resulting output will have bright spots at locations where the input image matches well with the template.

If the image is not monochromatic, we will have several values for every pixel. Thus our image would consist of several monochromatic images in a stack creating a 3D cubeiod of numbers rather than a flat grid. The thickness of such a cuboid is usually referred to as the number of *channels*. Since the image is now

2. Note that this is only in context of CNNs. In a normal convolution as defined in mathematics and signal processing we would use $f(h_{\boldsymbol{\omega}} - i, w_{\boldsymbol{\omega}} - j)$ instead of $f(h_{\boldsymbol{\omega}} + i, w_{\boldsymbol{\omega}} + j)$ which corresponds to flipping the filter in both vertically and horizontally.

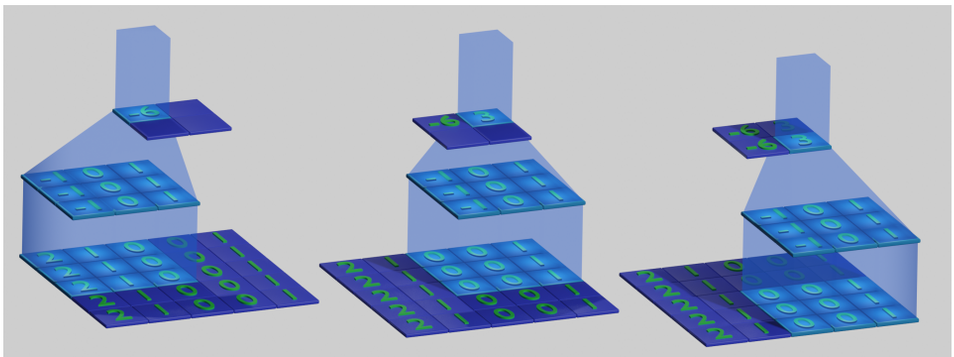


Figure 3.2: Visualization of a convolution with a stride of 2.

3 dimensional, a convolution including it must also be 3 dimensional. Every step of the convolution will still be to compute a dot product of an overlapping region between an image and a filter, but the region will now be a cuboid, not a rectangle.

3.1.2 Tailoring the Convolution

There are a few parameters that can be adjusted to change the behaviour of the convolution operation.

The *stride* is the distance of movement of the filter with respect to the image for each computed value. A stride of 2 would thus produce the same image as by performing the full convolution and then removing every second row and column of the output image. View Figure 3.2 for a visualization of the operation.

Padding is to concatenate new rows and columns around the input image before preforming the convolution. Which numbers to put in the new rows and columns is a design choice. Most frequently used is *zero padding*, where all new values are set to zero. The main motivation for using padding in CNNs is to control the size of the output image by changing the number of locations the filter can take and still fully overlap with the image. Figure 3.3 shows a regular convolution with zero padding of 1.

Dilation is a variable that determines by how much the filter should be expanded or inflated before the convolution. This is equivalent to inserting rows and columns of numbers between the original values of the filter. As with padding the most usual value to insert is zero. A convolution with a dilation of 2 is shown in Figure 3.4.

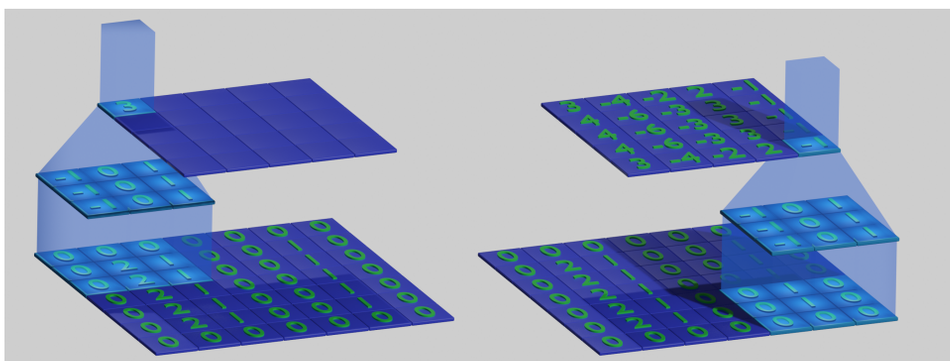


Figure 3.3: Visualization of a convolution with zero padding of 1.

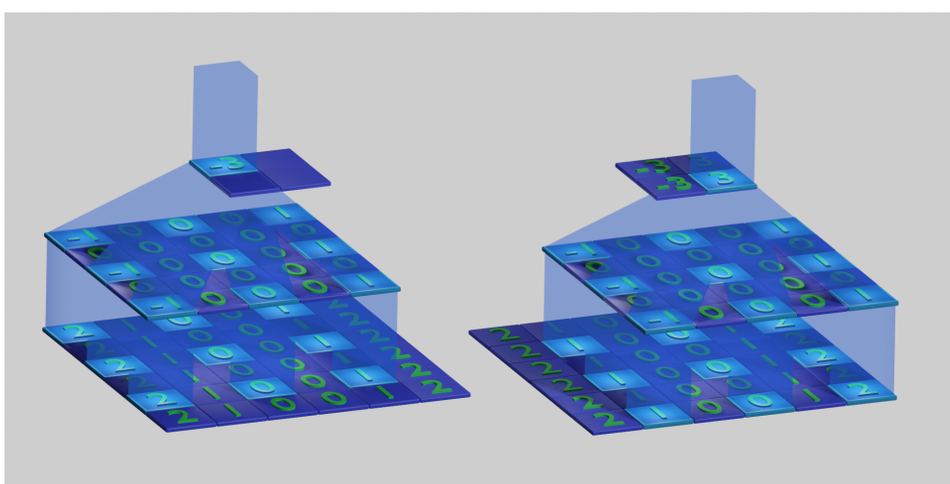


Figure 3.4: Visualization of a convolution with a dilation of 2.

All of the mentioned modifications to the original convolution changes the size of the output image. The formulas (3.3) and (3.4) calculate the size of the output image $\boldsymbol{y}$ resulting from convolving $\boldsymbol{x}$ with $\boldsymbol{\omega}$.

$$h_{\boldsymbol{y}} = \left\lfloor \frac{h_{\boldsymbol{x}} + 2 \cdot p - d(h_{\boldsymbol{\omega}} - 1) - 1}{s} + 1 \right\rfloor \tag{3.3}$$

$$w_{\boldsymbol{y}} = \left\lfloor \frac{w_{\boldsymbol{x}} + 2 \cdot p - d(w_{\boldsymbol{\omega}} - 1) - 1}{s} + 1 \right\rfloor \tag{3.4}$$

$$\tag{3.5}$$

p is the padding, s the stride and d the filter dilation.

3.2 Pooling

As convolutions, pooling inputs and outputs grids of numbers, but there is no filter. Rather, a pooling operation focusses on one region in an input image at a time, and computes a mapping from that region to one pixel value in the output image. The most frequently used pooling operation is *max pooling* for which the output value is the highest value from the selected region in the input image. Other pooling types include *average pooling*, *min pooling* and *rank-based pooling*.

The example below illustrates the use of max pooling with a 2 by 2 kernel and a stride of 2 in both directions.

3.3 The Convolutional Layer

The phrase *layer* was in the last chapter used in reference to a subunit of a fully connected neural network. But the phase is used more generally in reference to any subpart of a network that maps a set of features to another and they exist in many forms. The component that defines a convolutional neural network is the *convolutional layer*.

The mapping defining a convolutional layer is formulated in Equation (3.6).

$$\boldsymbol{Y} = \sigma(\Omega * \boldsymbol{X} + \boldsymbol{B}) \tag{3.6}$$

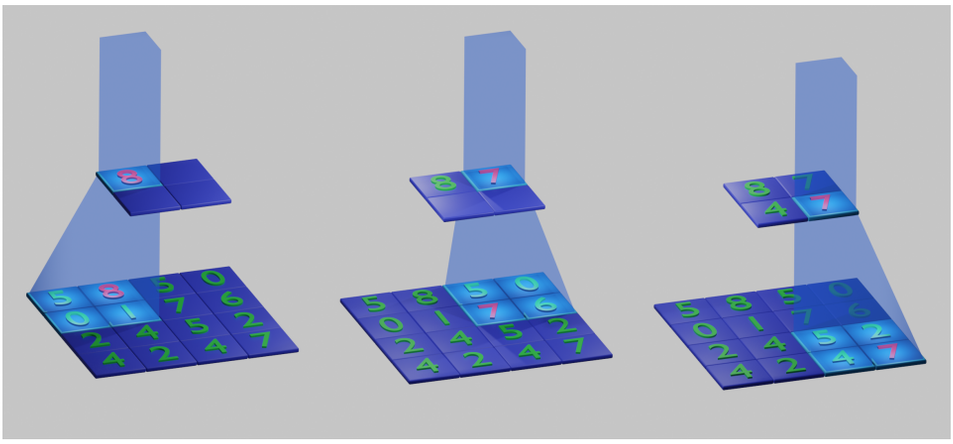


Figure 3.5: Illustration of max pooling with a kernel-size of 2, a stride of 2 and dilation of 1. The bright parts highlight the focus of each step, while the red number emphasized the basis for the final results.

Note that with convoltional layers, more often than not, one will be working with multidimensional grids rather than vectors; hence the use of capital letters.

A note on vocabulary: When the features are structured in matrices or higher dimensional volumes, they are often called *feature maps*. Thus the input image $\boldsymbol{X}$ above is mapped to the feature map $\boldsymbol{Y}$.

Recall that the convolution could be described as a search for locations in an image where it matches a filter. Here, the filter is subject to learning. The convolutional layer is thus learning to design a small template[3] to extract meaningful information about the image. A template that, when well trained, will capture some interactions of relevance between pixels in small neighborhoods of the image such as an edge.

3.3.1 Benefits of convolutions in Neural Networks

There are three defining features of convolutions that make them ideal for analysis of data where features mostly interact with features in their neighborhood: *Sparse interaction*, *parameter sharing* and *equivalent representations*.

Convolutional layers do not attempt to find interactions between pixels outside the scope of one filter-size. Very few interactions are ever attempted found

3. With several channels there will be more than one filter, but as the concept is in focus here we will avoid the complicating generalization for now.

compared to for fully connected layers, a defining feature of convolutional layers called *sparse interactions*[29]. The builtin assumption is that a pixel is only significant in context of its neighborhood, regardless of values on the other side of the image. If the true dependencies of interest in an image are grouped into neighborhoods, this is a powerful advantage as resources are not waisted trying to model a vast number of insignificant interactions.

In an image, one might reasonably assume that there are patterns that look similar at different locations. An edge at the bottom is not much different from an edge at the top. A convolutional layer is built to appreciate that fact as the same filter is used to find matches all over the input image. A filter that has learned to recognize an edge will for convolutional layers find similar edges where ever they are in an image. The concept is reasonably called *parameter sharing* [29]. Note the contrast to the fully connected analog where one weight will only connect the same two feature values, one in the input to one in the output.

If we apply a convolution, then move something in an image, the result would be identical if we switched the order of operations. This is called *equivariant representations*. More generally we can say that if $g(f(x)) = f(g(x))$, then g and f are equivariant transformations. A filter that matches well with a specific feature (e.g. an edge) will give the same response for equal edges independent of location. Thus spatial displacements and convolutions are equivariant transformations. The concept fits well with how we humans see images. A pattern in an image (e.g. a face) is generally the same pattern if it is moved to the other side of the image.

3.4 Back Propagation

We will only discuss back propagation for convolutional layers in general terms as the main ideas have already been discussed in the last chapter. The main difference between a fully connected layer and a convolutional one is the convolution operation itself. Recall that the gradients of interest are divided into three main parts. Gradients wrt. the weights, wrt. biases and finally wrt. the inputs. It can be shown that all of these can in fact be computed by convolutions. To get an indication of why, view Figure 3.6.

y_1 is the prediction of one pixel in the output, and the filter of the convolution is filled with weights to be adjusted. After the prediction is made, the objective function will return a matrix of gradients of the same size as the output to propagate backwards. As the receptive field of y_1 is the region in the input image that influences the output, it is the same region that corresponds to non-

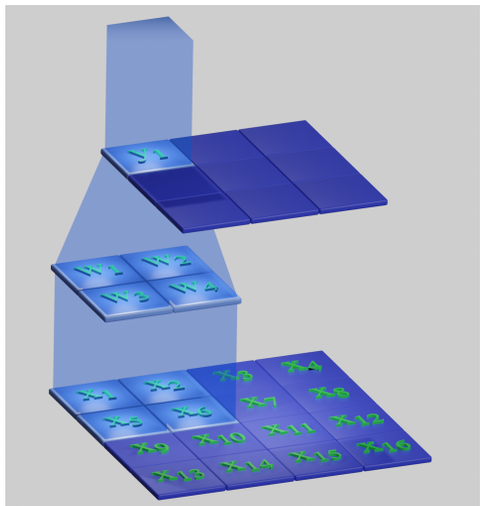


Figure 3.6: Visualization of a convolution in a CNN.

zero gradients in the back propagation from y_1. Thus the back propagation is a weighted broadcasting from each output, an operation called a *deconvolution* or a *transposed convolution*. It is the inverse of a convolution, but the operation can be reconstructed into a regular convolution by flipping the filter, padding the input, interpreting the stride as an inflation factor of the inputs and interpreting the given image padding as a removal of the newly padded image frame. Note that the deconvolution is not just used in back propagation, but can also be used as its own type of feed forward network layer. The operation is useful as it will neatly do an up-sampling with learnable parameters by using a stride higher than 1.

3.5 Optimizations

3.5.1 Filter sizes

The choice of filter size is one of the hyper parameters of convolutional layers. The filter size affects the number of parameters in a layer, but also the size of the region with influence on a feature in the next layer, often referred to as the *receptive field* of the feature. Intuitively, one might believe that small filters would prevent the network from learning larger patterns. However, this is only the case if ignoring the fact that layers can be stacked. A stack of layers will allow for small structure to be found in the first layer which can be put into bigger patterns in subsequent layers even if all filters are small. To see why modern network generally use small filters, we will do a small comparison

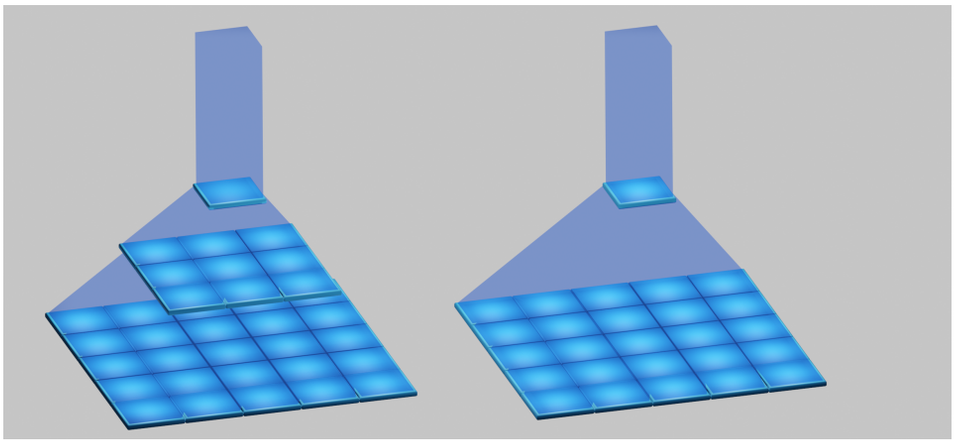


Figure 3.7: Comparison of filter sizes.

between two networks. The first is a stack of two convolutional layers, where the filter size in each layer is 3×3. The second is a single layered network with a filter size of 5×5. The receptive fields of the networks are the same as can be seen in the illustration in Figure 3.7.

But the two differ in parameter count as the first has $3 \times 3 \times 2 = 18$ parameters compared to $5 \times 5 = 25$ in the second. The reduction of parameters needed to cover the same receptive fields for hidden features is advantageous as fewer parameters need to be trained. In addition, as more activation functions are used, more complex interactions between pixels can be modeled.

But the change also comes at a cost of increasing network depth. A cost that was considerably higher before optimizations such as skip connections, initialization schemes and developments in activation functions.

3.5.2 Skip Connections and Residual Connections

To further reduce the cost of network depth, skip connections and residual connections have provided instrumental tools for further developments of the field. Both types of connections are paths in a network where a set of feature from a layer is moved past several layers to be directly reintroduced into a layer deeper into the network; by concatenation for skip connections and by addition for residual connections. It can be proven that residual connections is a special case of skip connections with parameter sharing in the connection receiving layers, so we will continue in this section by referring to both by the name: Skip connections.

The process effectively makes several parallel paths through a network, some of which will be shorter than the total network depth as the skipping applies both ways. The effects during training is a smoother gradient surface to maneuver with less likelihood of encountering singularities[4] [54] allowing for deeper networks.

To understand the effect, an explanatory model is that a skip connection simplifies the job of the layers being surpassed by the connection. A stack of layers collectively will try to learn how the input can be understood to match a given output. Sometimes the best analysis is done when a layer views the information processed by some hidden layers together with information from early in the network. With no skip connections, such an analysis can only happen if several layers perform a selective identity transformation, thus learning a lot of parameters to do nothing. With the skip connection however, a layer only has to learn the differences between its inputs and outputs and does not have to choose what should be kept as is from the previous layer.

Other significant improvements have been made such as analyzing feature maps on several scales in the same layer (e.g. Inception blocks[72] or Spatial pyramid pooling [32]) or making systems of skip connections (e.g. ResNet[33] og DenseNet[34]).

3.6 Models

The figures of the networks shown here were made with open source software created by Alexander Lenail [44].

3.6.1 AlexNet

AlexNet is a CNN model design proposed by Alex Krizhevsky, Ilya Sutskever and Geoffrey E. Hinton. It was first used in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) of 2012. With a top-5 test error rate of 15.3%, the model set a mile stone for the field of image recognition. The construction of the model is shown in Figure 3.8. The network had 60 million parameters and was one of the first models to be trained on a GPU.

4. Singularities in this context refer to situations where network capacity is reduced because of issues in the parameter landscape. This includes hidden features that become consistently non-active (dead nodes, a known problem of ReLU) or features that become linearly dependent of others, making them completely redundant.

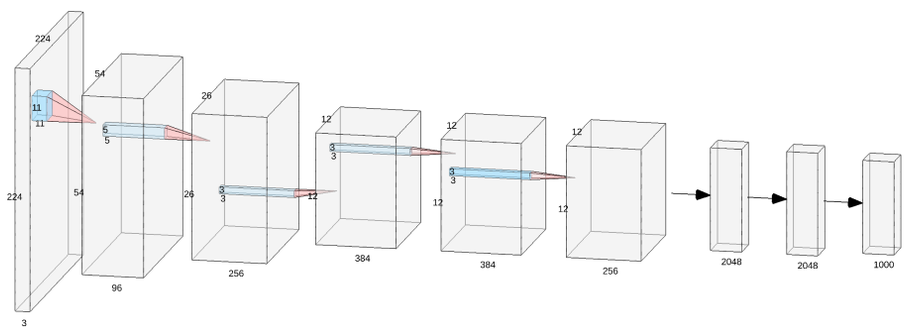


Figure 3.8: Illustration of the architecture of AlexNet. The clear blocks indicate layers ending with an activation. The thicknesses is the number of channels and the filters are shown as blue boxes. Max-pooling has implicitly been used for every down sampling (where the feature map size decreases).

The figure is visually informative on a large scale and is therefore used here. However, some details have been abstracted away to prevent cluttering. Each clear block indicates a layer including an activation function. The small blue boxes indicate the filter size of their respective layers. The first 6 layers are convolutional and the last 3 are fully connected. Softmax is the activation used for the last layer, while ReLU us used for all the others.

3.6.2 VGG-16

When VGG-16 was announced by Karen Simonyan and Andrew Zisserman in 2015 [68], the model was considered very big. With its approximately 138 million parameters, it is more than double the size of AlexNet. Despite the major change in size, the change in structure was less striking. Replacing convolutional layers with large filters by stacks of layers with smaller filters was perhaps the most significant architectural change.

On the big data set of natural images ImageNet [41], the model achieved a top-1 error score of 8.1[5]. The model has been one of the most influential to this day both because of its performance, but also because of the architectural simplicity.

5. To achieve the score, the training images were rescaled to two different sizes during training. But the performance was 8.7 without the rescaling.

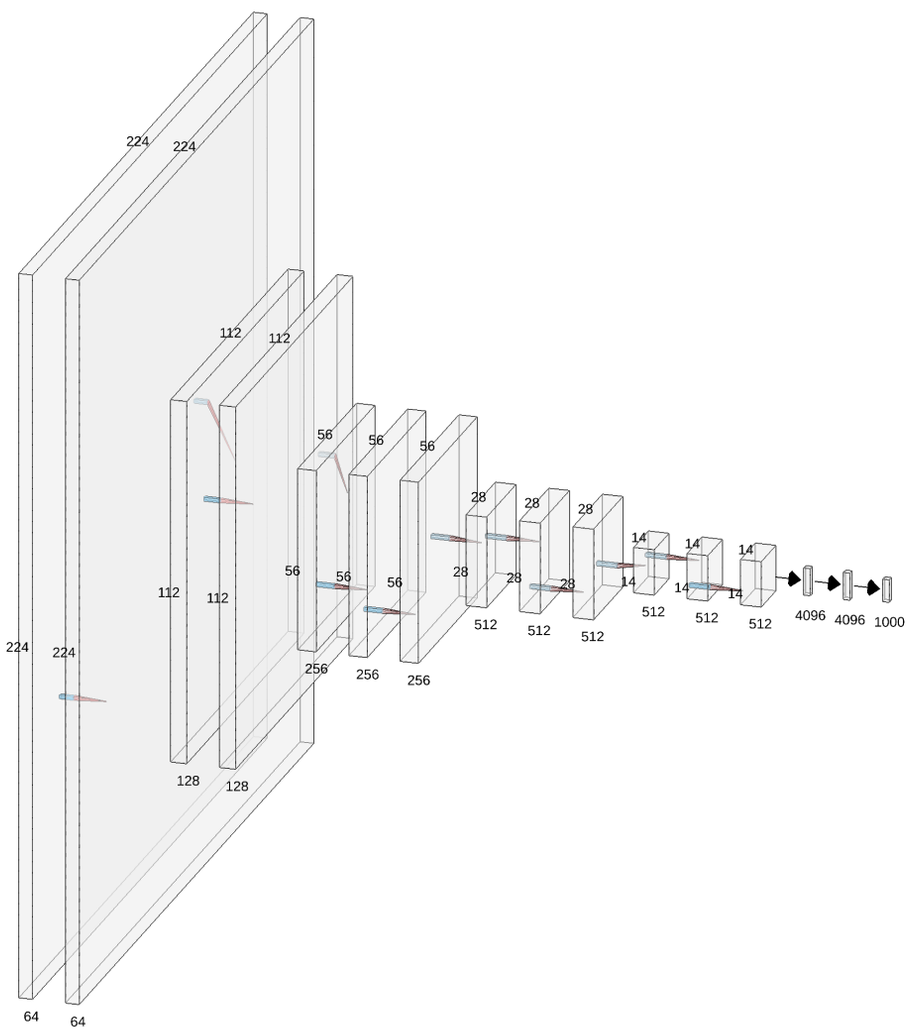


Figure 3.9: Illustration of the architecture of VGG-16. The clear blocks indicate layers ending with an activation. The thicknesses is the number of channels(visually on the log-scale as the figure would otherwise be very large) and the filters are all of size 3×3. Max-pooling has implicitly been used for every down sampling (where the feature map size decreases).

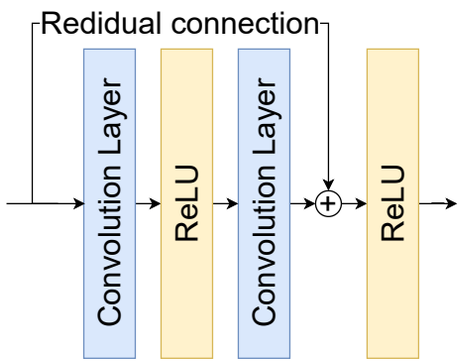


Figure 3.10: Illustration of a block of the ResNet architecture.

3.6.3 ResNet

The ResNet architecture is the deep CNN architecture with the largest footprint. The original paper written by Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun was ranked the most influential paper of 2019 with its citation score of 25, 256. The current citation count is (as of November 29. 2021) is 97, 694. The proposed network architecture forms a chain of convolutional blocks that are all bypassed with a residual connection. Figure 3.10 shows a block of the ResNet architecture. In the original paper several versions of different size were proposed, but they all followed the same pattern of blocks similar to the illustration. The largest and best performing version of the ResNet was made up of 152 such blocks.

3.6.4 Vision Transformer (ViT)

Less than a year ago(2020), Alexey Dosovitskiy et al. presented results of applying a transformer model on images[6][24].

The Transformer is a deep neural network design built on attention modules. The, when presented, radical design outperformed all competition in the field of natural language processing([76, 23]) and has made its way into neighboring fields such as audio classification[50] and recently also to image analysis.

Dosovitskiy et al. adapted the initial transformer model to images by first splitting the input image into patches that were flattened into vectors. The patch vectors were then simply treated as a sequence of words to be directly fed into the transformer model. The design and its variations now lead the

6. Others have applied transformers on images before this, but not with such combined simplicity and impressive results.

score boards of several benchmark image analysis tasks.

Despite its relatively simple structure, a variation of the ViT (ViT-H/14) is the currently best rated model on the CIFAR-10 image classification benchmark as well as being rated number 14 on the ImageNet benchmark.

It should be mentioned that there have been other successful attempts to adapt transformer models to be used with image data. The currently (November 28. 2021) 4 best performing models on the ImageNet classification benchmark as well as 6 out of 7 of the top performing models on the CIFAR-10 classification benchmark are all at least partially based on the Transformer model[7]. Only a few of these are ViT models or descendants thereof.

3.6.5 ConvMixer

The ConvMixer is a model design recently proposed in the wake of the very impressive results shown from the use of the ViT. The authors motivate their experiments by questioning which part of the ViT design most significantly caused the model's success, the transformer architecture or the patch encoding. If the success of the ViT was mostly a result of the patching one could potentially improve a CNN based model by adding a patch encoding module.

Thus they design a CNN starting with a convolutional encoding using a large kernel and an equally large stride (suggested to be 7×7). This can be viewed as reading patch for patch of the input with the same set of encoding filters for every patch. A visualization of the proposed model design is shown in Figure 3.11.

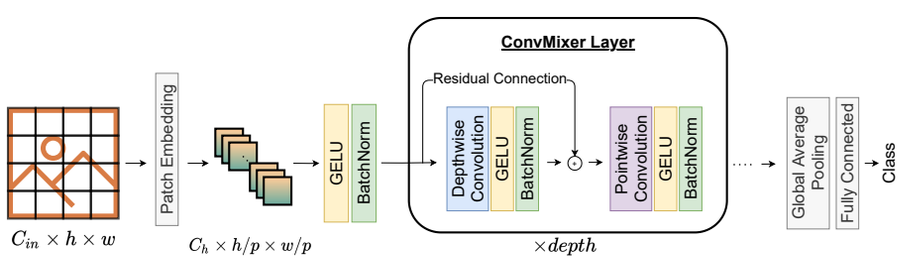


Figure 3.11: Model illustration of the ConvMixer model.

C_{in} is the number of input channels which in this case was 3, one channel for each input color (RGB). C_h is the number of hidden channels that is constant for all hidden representations between the patch encoding and the final global average pooling.

7. https://paperswithcode.com/sota/image-classification-on-imagenet

The hidden ConvMixer Layer contains two parts, both revolving a convolutional operation. The convolutional operation of the first part is called the *depthwise* convolution. It is a convolution where each channel is applied a single-channel filter separately. The operation thus analyses spatial structures of each hidden channel without viewing the whole context provided by the other channels. In the second part of the ConvMixer Layer the context between channels is in focus rather than the spatial structures. The *pointwise* convolution consists of 1×1 filters that analyses one pixel at a time, but with all channels of that pixel.

Following each convolutional operation there is a GELU activation function and a batch normalization layer (discussed in Section 2.5 and 2.6.3 respectively).

The model was design for classification and surpassed the performance of ResNet on the ImageNet Classification Benshmark with a Top-1 accuracy of 81.37% despite a lower parameter count[8]. Despite the measured performance, the model formulated in code was very short and simple. Although with a somewhat cryptic formulation, the model could be formulated in 4 lines of python code (imports excluded) using the pytorch framework. Their formulation is shown below.

```
1 def ConvMixer(h, depth, kernel_size=9, patch_size=7, n_classes
      =1000):
2     Seq, ActBn = nn.Sequential, lambda x: Seq(x, nn.GELU(), nn.
          BatchNorm2d(h))
3     Residual = type('Residual', (Seq,), {'forward': lambda self
          , x: self[0](x) + x})
4     return Seq(ActBn(nn.Conv2d(3, h, patch_size, stride=
          patch_size)), *[Seq(Residual(ActBn(nn.Conv2d(h, h,
          kernel_size, groups=h, padding="same"))),ActBn(nn.
          Conv2d(h, h, 1))) for i in range(depth)], nn.
          AdaptiveAvgPool2d((1,1)), nn.Flatten(), nn.Linear(h,
          n_classes))
```

Note that although the authors compares their model with a ResNet, they did not ablate their model to see which part of the their design had the greatest effect on the performance. Despite the somewhat unclear effects of the patch embedding itself, the ConvMixer model shows convincing results in classification and should be considered a novel CNN architecture worthy of further experimenting and research.

8. In open revision, this result was put in question because the chosen training scheme was known to be suboptimal for the ResNet architecture. However, the authors defend their result by claiming that no hyperparameter optimization was done with respect to the ConvMixer either. Thus the comparison is in some sense more fair than using a ResNet optimized training scheme for comparison.

As the paper is still under open review, the authors are currently anonymous and the publication not yet peer reviewed. However, we still find it reasonable to mention the model here for two reasons. The first is its claimed merit of 81.37% on ImageNet; a very strong result deserving of attention. The second is that we in this project have experimented with the model and some variations of it. The authors suggest the model could be adapted to semantic segmentation and we have taken them up on the challenge.

3.6.6 GoogLeNet/Inception (v1-v4)

GoogLeNet[72, 36, 73, 74], also called the Inception architecture, is a CNN architecture that bears similarities to the ResNet in that it consists of convolutional modules in a chain. However, it does not use residual connections and the blocks are build differently. Each block in the GoogLeNet architecture outputs a concatenations of results of applying different sequences of convolutional layers on the features from the last layer. The authors argue that the design can be though of as doing analysis of each set of hidden features at several scales in every layer. In spite of the reasonable hypothesis observations show that features in early layers of CNNs (including the Inception v1 architecture) typically captures small features or details of the input image while later layers typically capture larger structures [81, 52]. Each new version of the Inception architecture has competed among state-of-the-art models in several image analysis benchmark tasks [72, 36, 73, 74].

3.6.7 Other Models

Our list of models is not exclusive and there are several state-of-the-art models that we have not discussed, but most are modified versions of the models mentioned (e.g. EfficientNet which uses known models, but determines the hyper parameters with a search algorithm.).

4

Semantic Image Segmentation

The phrase *Semantic Image Segmentation* is made up of two parts, *Image Segmentation* and *Semantic*. Image segmentation is the task of partitioning an image into non-overlapping regions and then classify the regions[28, chap. 10][1]. The word *Semantic* implies that the partitioned regions have meaning that corresponds to their labelled class.

An illustrated example is shown in Figure 4.1. The image is taken from the publicly available PASCAL Visual Object Classification (VOC) dataset from 2012, used for training and evaluating semantic image segmentation models [25, 26]. In the example we can see that most of the colored regions in the image to the right corresponds to objects in the image to the left s.a. the person, horse and the dining tables. The black regions represent the background which is made up of all objects that are not included in the list of classes the dataset has labels for. Note that the edges are also given a separate class. Some datasets provide this option for detecting region boundaries independently of the region classes.

1. Gonzalez et al. [28] imposes the restriction that no two neighboring regions can have the same class, but this restriction can always be satisfied by merging neighboring regions of the same class in post processing.

(a) **(b)**

Figure 4.1: Sample from the PASCAL VOC 2012 dataset [25, 26]. *Left:* The sample image. *Right:* The segmentation label.

Upon close inspection one might notice that the details of the labeling is far from perfect in Figure 4.1. The image was chosen not to be a perfect example of how semantic segmentation can be done, but rather to indicate the goal of our efforts while also indicating what makes it hard.

4.1 Applications

Semantic segmentation is a type of image analysis that provides very detailed information about an image. A fact that generally makes it more desirable than most other types of image analysis, if it can be provided.

The main reason for the rare use in practice is the costs and efforts needed to make and validate a successful semantic segmentation model. Following are applications where semantic segmentation models are either currently being used or are in the development process and have shown convincing potential.

Self-driving vehicles need powerful tools to analyze their surroundings to be able to drive safely. Among other things, they need to be able to distinguish between objects and accurately determine the boundaries of objects. Thus, the detail provided by semantic segmentation makes it possible for self-driving vehicles to navigate complex and crowded scenes [38][2].

In the medical domain there are various uses for semantic segmentation. In

2. Not all self-driving vehicle models rely on semantic segmentation. Some use other types of information such as object detection with bounding boxes and some bypass the segmentation stage by training end-to-end from sensor inputs to appropriate actions, assuming the necessary scene and context understanding is implicitly learned by the model.

medical volumetric images s.a. those taken by a nuclear magnetic resonance machine, semantic segmentation can be applied to estimate the volume and location of organs and tumors. Such measures are vital for making the appropriate choices such as determining the right dosage of medicines for effective and accurate medical treatment.

4.2 Evaluation Metrics

Semantic image segmentation is a classification task and in the domain of classification, a set of predictions can be aggregated into a matrix called a *confusion matrix* that categorizes every kind of true and false prediction[3]. We will first consider the case of two classes and then generalize to the multiclass case.

4.2.1 Binary Evaluation Metrics

When considering the case of two classes the resulting confusion matrix might look something like shown in Table 4.1.

Predicted \ True	Positive	Negative
Positive	True Positives(TP)	False Positives(FP)
Negative	False Negatives(FN)	True Negatives(TN)

Table 4.1: Confusion matrix for 2 classes.

In itself, the confusion matrix provides valuable information about the classification performance of a model, but it is still not obvious how to directly compare two models by use of their confusion matrices. One can immediately see that a confusion matrix with high values along the diagonal and low values everywhere else is a sign of good performance. All the measures we shall view have in common that their best values are reached when the matrix is diagonal. However, the measures tell different things about the models performance and carry drawbacks in that there are special cases where the measures value will seem misleading. Such behavior will be exemplified.

We shall now view a few functions based on the confusion matrix that return informative performance measures that more easily can be applied to directly

3. To the best of the authors knowledge there is not a consensus on the correct orientation of the matrix. That is if the rows should represent the true values or the predicted once. We have here arbitrarily chosen to use the latter.

compare the performance of models. Note that there is a plethora of such functions based on the confusion matrix and we will only mention some of those is frequent use in for the subject of semantic segmentation. Below we have shown one formulation of the measures Accuracy (Acc), Precision / Positive Predictive Value (Pre), Recall / Sensitivity (Rec), Dice Similarity Coefficient / F1-score (Dice), Intersection over Union / Jaccard Index (IoU) and Matthews Correlation Coefficient (MCC) respectively. Note that there are typically several names for the same measures stemming from their use in different domains. Although the MCC has a different range than the rest, higher is better for all.

$$\text{Acc} = \frac{TP + TN}{TP + TN + FP + FN} \qquad \text{Acc} \in [0, 1] \quad (4.1)$$

$$\text{Pre} = \frac{TP}{TP + FP} \qquad \text{Pre} \in [0, 1] \quad (4.2)$$

$$\text{Rec} = \frac{TP}{TP + FN} \qquad \text{Rec} \in [0, 1] \quad (4.3)$$

$$\text{Dice} = \frac{2TP}{2TP + FN + FP} \qquad \text{Dice} \in [0, 1] \quad (4.4)$$

$$\text{IoU} = \frac{TP}{TP + FP + FN} \qquad \text{IoU} \in [0, 1] \quad (4.5)$$

$$\text{MCC} = \frac{TP \times TN - FP \times FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}} \qquad \text{MCC} \in [-1, 1] \quad (4.6)$$

Some of these measures have clear interpretations. The accuracy shown in Equation (4.1) is the ratio of the number of correct predictions to the total number of predictions and can thus be interpreted as an estimate for the probability of making a correct classification given a random sample. Precision or positive predictive value formulated in Equation (4.2) can be interpreted as an estimate for the probability of a sample being positive given that it was predicted positive (hence the name positive predictive value). In segmentation, it might help to visualize it as the amount of the model's prediction that was actually true. The measure gives information about how much one can actually trust a positive prediction. Conversely, one can interpret recall as an estimate for the probability of a random true positive being predicted positive. This gives information about how many positives will go undetected.

Typically, one has to balance between high values of precision and high values of recall. Imagine a model that has correctly predicted some samples with a high degree of certainty, but where for some samples, the model is very uncertain. Assume these uncertain samples consists of a mix of positives and

negatives. If the model predicts all the uncertain cases to be positive, the overall recall becomes higher as fewer positive samples go undetected. However, the precision decreases because not all those positive predictions are correct. The ideal balance between precision and recall depends on the application.

IoU can be seen as the probability of a sample being a correct prediction given that it is either predicted positive or is a true positive or both. It is not as intuitive a measure as precision and recall, but can be seen as an aggregation of the two as it takes both false positives and false negatives into account.

The dice score is the harmonic mean of precision and recall and is often viewed as almost analogous to IoU as it is very similar. However, IoU penalizes false predictions more aggressively.

To the best of the best of the authors knowledge, MCC does not directly correspond to some visual or probabilistic property of the data.

All measures mentioned except MCC suffer from being misleading when the classes are imbalanced (i.e. when the true samples of one class heavily outnumber the samples of the other class). To see an example of this, we will compare the accuracy of the two confusion matrices below (assume they represent the testing of two models to compare).

$$A = \begin{bmatrix} 48 & 4 \\ 2 & 46 \end{bmatrix} \qquad B = \begin{bmatrix} 94 & 5 \\ 1 & 0 \end{bmatrix} \tag{4.7}$$

Both confusion matrix A and B yield an accuracy of 0.94 which would imply both models did quite well. Despite the impression, matrix B shows the second model predicts one class for almost all samples and the only prediction of the second class is false. The high accuracy score is a consequence of the fact that the true samples of class 1 are 20 times more numerous than those of class 2. For reference, note that all measures produce high scores for matrix A, while the results for B are as follows: $\text{Pre} = 0.96, \text{Rec} = 0.99, \text{Dice} = 0.97, \text{IoU} = 0.95, \text{MCC} = -0.02$. Among the measures mentioned, MCC is the only one that successfully reveals the problem in matrix B. Note an MCC value of 0 is interpreted as "as good as random".

We have only shown one special case where it is very apparent that not all our measures of performance always reflect the true performance of a tested model. Chicco et al. have thoroughly compared MCC to accuracy and Dice score[16] as well as to other measures that are designed to handle imbalances such as balanced accuracy, bookmaker informedness and Kohen's Kappa[17]. Their

conclusion is that MCC for most cases reflects the true performance of a model used for binary classification compared to the alternatives and they advise MCC to be used as the default evaluation metric for binary classification.

4.2.2 Multiclass Evaluation Metrics

All the measures in the last section can be generalized to the multi class case. But this is more easily done when viewing a confusion matrix as a matrix of indexed rows and columns rather than just a collection of elements. Let the confusion matrix be denoted by CM and let $\text{CM}_{i,j}$ denote the element of the confusion matrix on row i and column j starting at the index 1 (e.g. the number of samples predicted to be of class 1 when they in fact belonged to class 3 is denoted by $\text{CM}_{1,3}$). Some of the measures mentioned are not class agnostic in that they have one particular class they focus on rather than all of them together[4]. In these cases let the class of interest be denoted by κ.

The equations (4.8 - 4.13) are generalizations of the equations (4.1 - 4.6) where κ is the class of main interest and N is the total number of samples used. By letting the class of interest be $\kappa = 1$, and the number of classes be $K = 2$, there is exact equivalence. But these expressions are directly applicable for an arbitrary number of classes.

$$\text{Accuracy} = \frac{\text{trace(CM)}}{\sum_{i,j} \text{CM}_{i,j}} \tag{4.8}$$

$$\text{Precision} = \frac{\text{CM}_{\kappa,\kappa}}{\sum_{j} \text{CM}_{\kappa,j}} \tag{4.9}$$

$$\text{Recall} = \frac{\text{CM}_{\kappa,\kappa}}{\sum_{i} \text{CM}_{i,\kappa}} \tag{4.10}$$

$$\text{Dice} = \frac{2\text{CM}_{\kappa,\kappa}}{\sum_{i} \text{CM}_{i,\kappa} + \sum_{j} \text{CM}_{\kappa,j}} \tag{4.11}$$

$$\text{IoU} = \frac{\text{CM}_{\kappa,\kappa}}{\sum_{i} \text{CM}_{i,\kappa} + \sum_{j} \text{CM}_{\kappa,j} - \text{CM}_{\kappa,\kappa}} \tag{4.12}$$

$$\text{MCC} = \frac{N\ \text{trace(CM)} - \sum_{i,j} CM * CM}{\sqrt{N^2 - \sum_{i,j} CM^T * CM}\sqrt{N^2 - \sum_{i,j} CM * CM^T}} \tag{4.13}$$

The symbol "$*$" denotes matrix multiplication and the $(\cdot)^T$ denotes the trans-

4. Note that there are at least two ways to generalize these specific expressions, "one vs. all" or "all vs. all". We have chosen to only view the "one vs. all" perspective here.

posed.

For a deeper understanding of how MCC was generalizes to the expression shown in (4.13) view the work of Gorodkin et al. [30].

4.3 Objective Functions

In the first chapter we discovered that the objective function provided feedback for a neural network to evaluate its predictions. Some are quite general and can be applied in many different situations, while others are designed to make networks learn very specific behavior.

Cross Entropy

The most used objective function for classification in general is cross entropy loss. Cross entropy can be seen as a very strict difference measure between two probability distributions, and has convenient properties as an error measuring tool comparing predictions to labels. When using cross entropy loss we generally assume the binary case where the labels are either 0 or 1 and the predictions lie in the range $(0, 1)$. With a prediction $\hat{p}$ and a corresponding label p, the cross entropy of that prediction is shown below.

$$CE = -\left(p \log(\hat{p}) + (1 - p) \log(1 - \hat{p})\right) \tag{4.14}$$

With a small reformulation of the problem, the binary cross entropy shown above can be adapted to the more general case of several classes. Let one output feature only represents one class such that a prediction is a vector of probabilities $\hat{\boldsymbol{p}} = [\hat{p}_1, \ldots, \hat{p}_n]$. The label could be changed to a one-hot vector (a label of class 2 would be changed to $\boldsymbol{p} = [p_1, p_2, p_3, \ldots] = [0, 1, 0, \ldots]$). The cross entropy of a new vector prediction $\hat{\boldsymbol{p}}$ with a one-hot vector label $\boldsymbol{p}$ is thus shown below.

$$CE = -\sum_{i=1}^{n} p_i \log(\hat{p}_i) \tag{4.15}$$

Notice that even if the function is a sum, the computed loss value for each prediction will only contain one part of that sum as $\boldsymbol{p}$ only has a single

non-zero value. If the prediction is good, i.e. close to 1 for the true class, the computed value will reside close to 0. If the prediction is bad, i.e. close to 0 for the true class, the computed cross entropy will be very large as $\lim_{x \to 0} -\log(x) = \infty$.

Generalization of the function to apply to more than one dimension (of several classes) is direct, in which case the sum in Equation (4.15) sums over dimensions as well as classes.

A known issue with cross entropy loss function arises when there is class imbalance, i.e. data points from one class outnumber those of another by a large margin. Because the normal algorithm thus will distribute equal significance to every data point, much more learning will occur to improve the networks predictions of one class, compared to another. One way to remedy the problem is to introduce weights that can be tuned to adjust for the imbalance. Note that the weights can also be used to generally prioritize classes if that is desirable. The formulation with weights is shown below.

$$\mathcal{L}_{\text{CE}} = -\sum_{i=1}^{n} w_i p_i \log(\hat{p}_i) \tag{4.16}$$

Cross entropy and its variations are generally considered effective classification loss functions that work well in symbiosis with the softmax output activation.

Confusion Matrix based Losses

In Section 4.2, we discussed different functions of the confusion matrix that could be used to measure the performance of a model. As these are our best tools for measuring good performance, it seems reasonable to want to utilize the same functions as objective functions (i.e. that a model could be directly optimized with respect to the outputs of these functions). But there is one problem, the confusion matrix is not differentiable. To compute the confusion matrix, one first has to discretize the predictions to the available classes, a non differentiable process.

However, one can circumvent this problem by doing an approximation. Rather than simply counting the number of similar and dissimilar predictions, one can allow for soft predictions. This can be done by representing the labels as one-hot vectors (vectors of all zeros except a 1 at the index of the true class)

and not discretizing the prediction vector. Thus, we may reapply all confusion matrix based evaluation metrics as objective functions. This has been shown to work quite well for some cases such as for Dice loss or small variations of it [64, 71, 37]. This, despite that the gradient properties of the confusion matrix based objective functions are less ideal than e.g. those of cross entropy loss.

Dice loss, based on the dice coefficient, is probably the most well known confusion based loss function [37]. Its relative, Tversky loss is very similar but adds parameters weighting the false predictions (equivalent to element wise multiplying the confusion matrix by some weight matrix where the diagonal of the weight matrix are all ones) which has also shown good results [37]. The IoU loss is similar to Dice loss, but has steeper gradients for good predictions and flatter gradients for bad once, which implies it should not work as well as dice loss while otherwise having very similar properties. However, we do not know of any experimental results on the use of IoU based loss to confirm or deny our reason based suspicions of its hypothetical properties.

MCC loss, recently proposed by Kumar Abhishek and Ghassan Hamarneh [1], was shown to work quite well in comparison to dice loss. They argue that the fact that MCC takes all elements of the confusion matrix into account and is class agnostic could be advantageous for the classification problem in general and especially when the data is unbalanced(the setting of their focus). Note that Abhishek and Hamarneh did not include any versions of cross entropy loss in their experiments arguing that they typically do not handle imbalance well. Noting the informative and well behaving evaluation properties of MCC for both balanced and imbalanced data, it would be interesting to see how MCC loss compares to other loss functions in general. If its gradients properties prove to be manageable by optimizers in general one could imagine MCC loss potentially becoming a default objective function for the case of classification.

4.4 Models

In this section we will mostly discuss models in broad terms to keep focus on the major developments and general architectural ideas rather than focussing on the specific details of each. We consider it a good approach for seeking broad understanding.

The Fully Convolutional Network is a VGG-16 network without the last fully connected layer, but with an appended deconvolutional up-sampling at the end resulting in image predictions rather than vector predictions [47]. The model varies in a few different forms depending on the number of steps used in the

up-sampling. Common to all variations was the use of the deconvolution for learnable non-linear up-scaling. The fully connected layers were also replaced with 1×1 convolutions, a computationally equivalent sibling that will allow variable input sizes. The model was ground breaking in performance when it was proposed and has been used as a baseline for many methods since. Note that the method of adapting an upsampling step at the end of a classification network has been done for other classification networks and this is referred to as FCN with a different backbone.

The UNet was proposed for the analysis of medical images [60]. Visually, it looks like two small VGG networks linked head to tail, where the last one is reversed. The first half of the network, the encoder, has convolutional blocks ended with max-pooling layers performing a stepwise down-sampling similar to the VGG. The other half, the decoder, performs up-sampling in the same number of steps as the down-sampling. For each up-sampling step of the decoder, a skip connection (as described in Section 3.5.2) was added from the encoder step with corresponding resolution. Note that in this case, the motivation of the skip connections was to let the network more easily analyze the images at different resolutions.

There have been several versions of a model called Deeplab that has performed very well [14, 15]. An essential part of their architecture is the *atrous spatial pyramid pooling* (ASPP) operation which bears similarities to Inception blocks but for up-sampling. These models have been used with several base-networks including the ResNet and VGG.

OCNet [80] and DANet [79] are similar networks to those mentioned, but with the introduction of attention modules to better learn scene contexts. There are many other networks and architectures, but the most important general architectures are to the best of our knowledge mentioned here.

From segmentation models trained with full supervision we will now move to the harder problem of weak supervision.

5

Weakly Supervised Learning

Weakly supervised learning is a branch of deep learning where one assumes the training data is imperfect. The field is motivated mainly by cost and effort. When reading the leader boards over the best performing methods on publicly available data sets[1], one might get the impression that modern methods could solve almost any task. But when comparing publicly available data sets with how real data collection works, one will quickly realize that it takes a lot of work to get from a well defined problem, to having data of adequate size and quality to successfully train and test a model. The cost of data collection is intuitive, but also well documented by facilitators of labeling of data with the specific purpose of model development [46].

Imperfection of data labelling is a rather vague description that has been elaborated by Zhi-Hua Zhou et al. who divide the subject into 3 categories based on types of imperfection: *incomplete*, *incomplete* and *inaccurate* supervision[86].

Incomplete labels can be further divided into the two categories *active learning* and *semi-supervised learning*. Active learning is a query-based situation, where a human is asked by the model to verify or label during training of the model. This might be an advantage as it lets the network choose the data points that

1. `https://paperswithcode.com/sota`

will the most informative to get labels for, but it also makes the process of labeling more demanding. Semi-supervised learning revolves situations where only a subset of the given training data is labeled. This is by some considered the most productive type of weakly supervised learning as it does not rely on a networks ability to learn important details implicitly [19].

Inexact labels are labels that give less detailed information than the model is assumed to predict. The potential for the type of models that would solve this problem is hard to overstate. If the models that perform well with full supervision today could be trained only on weak labels, they would be much more accessible and therefore useful for a range of applications. In industry and medicine the collection of information is vast, but usually with a degree of detail and consistency s.t. an advanced understanding of context is needed for correct interpretation (e.g. Notes with subject specific jargon and abbreviations). There is little information that is detailed enough, consistent enough or stored in a way that makes it convenient to make adequate labels for a neural network to replicate the labeling process. In such cases, a good weakly supervised model might automate processes that today seem repetitive to humans, but demands too much to automate today. It should also be mentioned that the level of detail of all kinds of labels for data are limited; either by our absolute performance as humans, or by the nature of the problem (e.g. maximal resolution of images taken with optical light which is limited by the wavelength of the light used.). In such cases, models developed for weak supervision could potentially combine analysis and synthesis to produce results surpassing assumed performance boundaries[2].

Inaccurate labels are the hardest to tackle as they will occasionally give faulty information. Neural network models are flexible functions that will be misled if trained with misleading guidance. Thus most ways to solve the problem of inaccurate labels are attempts to circumvent the problem by identifying the data points with faulty labels and remove them [86].

5.1 Weakly Supervised Semantic Segmentation (WSSS)

In the context of semantic segmentation, the phrase "weakly supervised" usually only refers to inexact labels such as points [8], scribbles [45], bounding boxes

2. This is not as far fetched as it might seem. Generative adversarial networks (GANs) are methods that explicitly combine analysis and synthesis to generate samples that seem convincing, but are not deterministic results based on the provided data.

[22] or image level labels [4, 43, 35, 56, 42]. The types of labels just listed was arranged from the most to the least informative. Most methods in this domain focus on utilizing image level labels, i.e. labels that indicate what is in an image, but not where. Thus models trained on image level labels will be the main focus of this section.

The motivation for the development of many of the models we will go though came from a series of observations about what CNNs learned when trained for classification. Karen Simonyan, Andrea Vedaldi and Andrew Zisserman[69] thoroughly inspected the activations of intermediate layers of a VGG network trained for classification. When images containing an object of a given class was viewed, the location of that object would show up brightly in the intermediate activations through the network before being aggregated into a prediction vector.

Their discovery showed that CNNs propagate information about local features through deep convolutional layers. Based on their discovery, others have made attempts to utilize the implicitly learned information for generation of segmentation masks. We will thus dive into a set of segmentation models trained with image level labels.

5.1.1 Models

In this section we will focus on the most defining models or representative examples of modern weakly supervised semantic segmentation approaches, but will not investigate all current models in depth. This was considered a reasonable approach as it keeps the focus on the common ideas rather than the idiosyncratic details.

Class Activation Map (CAM)

One of the most influential contributions to the field of weakly supervised segmentation as well as for many other closely related fields (weakly supervised object localization and detection) is the *Class activation map*[85]. The method uses a regular segmentation model such a FCN and replaces the final softmax with two layers. The first new layer is a global average pooling layer, where the predicted segmentation map for each class is aggregated by an average into an output feature. Following the global average pooling is a fully connected layer finalized with a softmax activation. The model is trained as any other classification model. However, after making a prediction of one class, one can attain a heat map from two layers earlier in the network that reflects relevance or salience for the prediction. The results of the approach were convincing

and later methods have shown that the precision of discriminative cues are, to some degree, reliable although noisy [19, 53, 40, 7]. The model has been used as a baseline in many weakly supervised experiments because of its combined simplicity and success; among others [40, 35, 13, 83, 18, 84]. But while the positional cues typically do cover relevant location, they are vague, not respective of natural boundaries in the input images and prone to only focus on the most discriminative features of images.

To compute a CAM, the segmentation maps from one prediction are weighted according to the weights connecting the global average pooling layer to the output layer, before being summed together. This can be viewed as a partial back propagation stopping at the segmentation layer. But viewing the computation of the CAMs as a partial back-propagations can be generalized to focus on other layers than just the last. Grad-CAM is a generalized version of CAM that can produce saliency maps from any convolutional layer [66], based on the change of perspective. It should be mentioned that only positive contributions for a class, that is positive gradients that correspond with positive activations, were used(; the equivalent of using a ReLU activation in both propagation directions).

To improve CAMs tendency to focus only on the most discriminative parts, many have tried to hide selected features during training to force the network to focus on larger parts more of the class objects. Junsuk Choe et al. have investigated such attempts for the task of object localization, i.e. finding the tightest bounding box around an image class object (assuming only one object of any class in each image). This is not the same task, but it is a task that struggles with the same issues of proposals not covering the objects of interest. The methods analyzed covered a range of attempts from randomly selecting patches of the input images to hide [70], to specifically hiding the parts that were considered salient by the network[83]. The conclusion of Choe et al. was that carefully thersholded CAMs will usually produce results at least as good as any of the compared erasing methods [49, 9, 70, 83, 84]. Note that the thresholded CAMs were used to find bounding boxes from which the results were evaluated. However, the similar feature dropout method have also been used for segmentation [7] with results that support the conclusion of Choe et al. as they seem to produce very similar results for segmentation as CAM.

Seed, Expand and Constrain (SEC)

Seed, Expand and Constrain is a model that was proposed by Alexander Kolesnikov and Christoph H. Lampert in an attempt to address mainly two problems of the CAM method [40]. First, the tendency for CAM to only focus on the most discriminative parts of objects rather than the full objects. Second,

the tendency of CAM predictions to not respect the boundaries of objects in the input image. The method is a segmentation CNN model trained with a three part loss; a seeding loss, an expansion loss and a constraining loss. The seeding loss addresses the location of object predictions and the expansion loss tries to improve the prediction sizes to better match those of objects in the input images. Finally, the constraining loss adapts the network to make predictions that better respects natural boundaries in the input images.

The seeding loss that is only concerned with a small part of the predicted output image. It begins with a set of proposed seeding segmentation maps generated by thresholding heat maps found with a visual drop out method with results equivalent to those of CAM[7]. These are segmentation maps, but with class labels only covering a very small portion of the objects. The seeding loss is used to train a CNN in a fully supervised manner to make predictions that cover the given seeds with the right class. The loss encourages segmentation maps that cover the given seeds, but since we expect the segmentation regions should be larger than the seeds, the loss does not penalize false positives.

The next part of the loss is the expansion loss. It is very similar to CAM by introducing a global pooling layers ending in a cross entropy classification loss. But there are a few differences. In experiments, the authors observed that the pooling method used in CAM changes the size of the predicted regions. Global max pooling only focuses on the highest prediction activation from each segmentation map during training which results in smaller regions of high activation; thus causing segmentation maps that are small (similar to the proposals used in the seeding loss). Using global mean pooling on results in the opposite problem since every activation aggregated into the pooling layer counts equally for the prediction; thus overestimating the size of objects[3]. The proposed solution is a global weighted rank pooling(GWRP) layer which ranks the activated pixel class scores based on activation before aggregating them into a weighted sum. The pooling strategy can be viewed as a generalization of both global max pooling and global average pooling. The way to distribute weight on pixels based on the ranks could thus be adjusted to control the size of the segmentation predictions.

The last part of the loss function is the constraining loss. It was designed to penalize bad correspondence between predicted segmentation maps and the spatial and colory information in the image. This was done by using a fully connected CRF on a predicted segmentation maps resulting in an image used as a pseudo label. The KL-divergence was then computed between the CRF

3. This depends on the choice of thresholding value for turning the activation maps into segmentation maps as pointed out by Choe et al. [19]. The subject was however not thoroughly investigated at the time (to the authors knowledge).

input and output to form the constraining loss.

SEC is today not among the best models, but it applied many important concepts that are significant parts of better models. Growing small, reliable regions to better cover the objects and using pseudo labels have lead to many improvements in the field.

Following the SEC are two similar methods that have improved upon the design. Deep Seeded Region Growing (DSRG) proposed by Zilong Huang et al. is very similar to SEC, but with one main difference. The region growing is done with an older algorithm made for that purpose called Seeded Region Growing (SRG) [2]. The CRF loss is in large part the same, and the seeds are found by thresholding CAM predictions.

Related Approaches

Other methods such as AffinityNet [48] and IRNet [3] also descend from SEC, but have further improved on the method for region growing and boundary constraining. AffinityNet uses the L1 difference scores between neighboring pixels of an input image to estimate pixel affinities. These are used as transition probabilities in a random walk process starting with the seeds. The process can be compared to an MCMC method for estimating a posterior distribution based on a prior computed from the natural structures in the input image.

A third descendant of SEC is the Saliency and Segmentation Network (SSNet) [82]. It combines the seed making network and the segmentation network trained with the pseudo labels such that the seeds might improve during training.

Different Approaches

Expectation maximization(EM) is a different type of method than those described above. Assuming the segmentation maps are latent variables of the model, a first segmentation prediction can serve as the expected value of those latent variables. The expected values are then used in a maximization step performed by training the model in a fully supervised manner with the earlier predictions as labels, or pseudo labels. The two steps are then alternatingly repeated until convergence. Pathak et al. have used this method even without the seeds, by only imposing class specific biases and object size priors during training [55].

Multiple instance learning (MIL) refers to cases where one starts with clusters

(called *bags*) of data points (called *words*), where the clusters are known to contain data points of a certain class, but without the knowledge of which or how many of the data points belong to the class. The task is to assign classes to the data points. In context of weakly supervised segmentation the bags are images and the pixels words. Methods such as [57] and [58] use an alternating method similar to approaches presented as EM. The similarity has also been noted earlier by [55].

5.1.2 Juxtaposed Objectives

When observing that locations of objects seem to be implicitly learned in models trained for classification, attempts to exploit this property for localization, detection or segmentation seems a natural progression. However, one must be careful when assessing what situations the property of interest applies to as any objective function in WSSS cannot really measure what we desire.

Not all configurations of a network that produces accurate classifications must necessarily produce accurate localization or segmentation masks and vice versa. There is clearly some overlap shown in the observations of Simonyan et al.. However, as noted by Choe Junsuk et al.[19] the problem we are trying to solve will in many cases be ill-posed because of the nature of the data[4].

A supervised model tries to associate features of given inputs with provided labels. If the model is perfect, it will pick up the strongest associations and model those. The features of the inputs we intend to be informative might in some cases provide less conclusive or more complicated evidence to predict the correct labels than some alternative features. In such cases, even the perfect model will thus never reach a desired configuration in the sense that even though measured performance is high, the model does in fact model the system we are interested in well.

As an example, imagine a case where we want to train a network to classify if a given image would contain a dog or a cat. The training set in this hypothetical project are images which happen to be framed. Images with dogs are framed with blue, those with cats with red. To determine the connection between the images and the animals, the model might try to learn general features in the images such as the shape of the ears or nose of the animals. However, a potentially much easier alternative would be to learn to look at the colored frames. If we attempted to make segmentation masks of the activations inside

4. Note that the problem assessed by Choe Junsuk et al. was weakly supervised object localization. But the argument for ill-posedness applies as much for the case of segmentation as for localization.

our hypothetical network we would expect to encounter high activation regions around the features of the input image most significant for the classification predictions, i.e. the frames. Thus we could have a network that performed very well according to our classification measures(i.e. the prediction of the correct season), but which would completely fail on our segmentation measure(i.e. having high activations near the animals on the images). In this case the informativeness of the frames are obvious, but such shortcuts to good answers for the objective function might not be so obvious in all cases.

It should be mentioned that the problem here is a common one in deep learning in general and it is strongly related to the topics of intepretability and explainability of deep learning models. Understanding what models actually learn, why they make the decisions they make and how certain they can be about their predictions are important for reliability. But the issue is particularly pronounced in WSL as the lack of strong labels force us to train with an objective that is further away from our true goal.

The models discussed must therefore be used with caution and must be well tested on data that well represents the distribution of the data the model will be used on in practice.

6 Method

We propose a novel semantic segmentation framework entitled ConvMixerSeg and a new training scheme for weakly supervised semantic segmentation using image level labels. The architecture of the proposed model is based on the ConvMixer classification model discussed in Section 3.6.5, and is motivated by the ConvMixer's high performance with few parameters. To the best of our knowledge, this is the first ConvMixer-based architecture for weakly supervised semantic segmentation. Moreover, we propose a new weakly supervised CAM method that does not introduce additional parameters to the segmentation framework while also mitigating the problem of part domination and prediction noise.

6.1 ConvMixerSeg

The simplicity and effectiveness of the ConvMixer model discussed in Section 3.6.5 implies the architecture might have merit in neighboring fields. To the best of our knowledge, neither the patch encoding nor the alternating structure of the hidden layers in the ConvMixer have previously been used for semantic segmentation. The results presented in the original paper of the ConvMixer could imply that the deep architecture learns more with fewer parameters compared to previous state-of-the-art architectures like the CNN based ResNet [33] as well as recent transformer based models like DeiT [75]. The results motivates further development and investigation of the potential of the ar-

chitecture. Thus we propose a redevelopment of the ConvMixer model and present with that a novel architecture for semantic segmentation.

View Figure 6.1 for an illustration of the model. We have adopted the patch encoding with both kernel size and stride of 7×7 and the hidden structure of alternating depthwise and pointwise convolutional blocks[1]. Recall that we in Section 3.5.1 Surpassing each of the convolutional blocks we have added a residual connection.

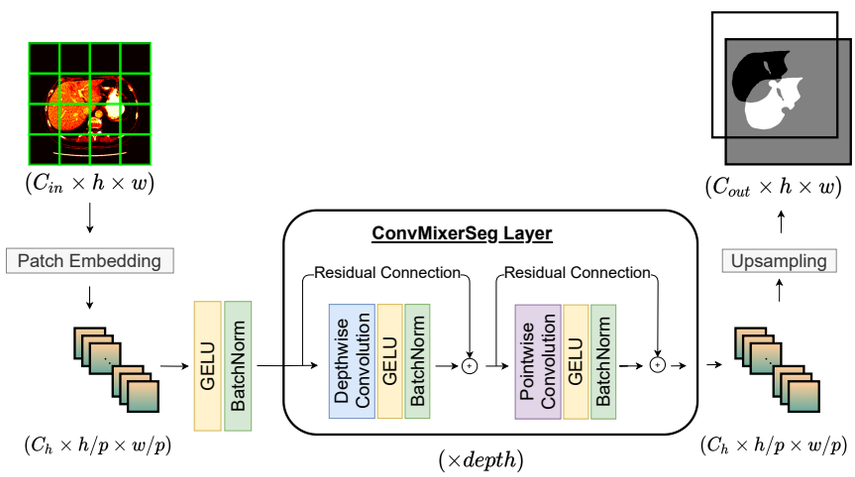


Figure 6.1: Illustration of the proposed segmentation model design. The input is an $h \times w$ image with C_{in} color channels. The output has the same spatial dimensions as the input, but with a segmentation map for each data class. In the illustration the nearest layer represents liver and the other background.

The upsampling step (shown as a gray block on the right side of Figure 6.1) is shown in more detail in Figure 6.2. The feature maps of shape $C_h \times h/p \times w/p$ are the outputs of the last ConvMixerSeg layer. The transposed convolutional block maps the feature maps to the resolution of the inputs but with a channel count of C_t where t marks that the channel count is a transitional step from the depth of the hidden layers to the very few channels in the output. We then apply an unactivated convolutional layer with filter size 1×1 before applying Gaussian smoothing which finally outputs the prediction feature maps for each class.

1. "Block" is here used in reference to the convolutional operation in combination with the following activation layer and batch normalization layer.

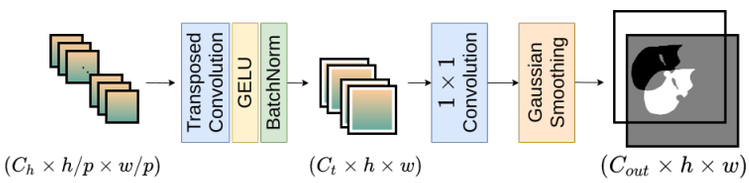


Figure 6.2: Illustration of the upsampling of the proposed segmentation model. Output feature maps of the last ConvMixerSeg layer are processed through the upsampling to return the final segmentation predictions for each channel (here shown for the classes "background" and "liver").

6.1.1 Symmetric Patch Encoding and Decoding

The first step of the upsampling is inspired by the upsampling structure used by Long et al. in the Fully Convolutional Network [47], as well as by many later segmentation models. It performs an upsampling step, but based on learnable parameters that can model a more complex upsampling scheme compared to non-learable upsampling methods such as bilinear interpolation. We use a patch size and stride equal to that of the patch encoding motivated by symmetry. The original idea that led to the discovery of the CAM method was that spatial information was seen to be propagated through a CNN, i.e. contents at a location in the inputs to the network were seen to correspond with feature activations at the same location later in the network. Thus for best correspondence between input and output features based on the assumed locational correspondence in the hidden layers, we find it reasonable to try an upsampling or patch decoding that is symmetric with the patch encoding.

6.1.2 CAM Modification

In this section we will address the transition from a $(C_t \times h \times w)$ feature map to the final segmentation prediction output shown in Figure 6.2. The motivation for the proposed setup is two fold. First, the proposed design allows a segmentation model to be trained with image level labels without adding learnable parameters. Second, by using Gaussian smoothing to enforce locational correlation in the final feature map, we hypothesize the problem of part domination is mitigated while also reducing noise in prediction segmentation maps.

To explain the setup, we will first investigate the original CAM method and the proposed change a little more deeply. View Figure 6.3 where we assume $\boldsymbol{x}$ is the unactivated output feature map of a backbone segmentation network. The original CAM model is shown on the left and the modification is shown on the right.

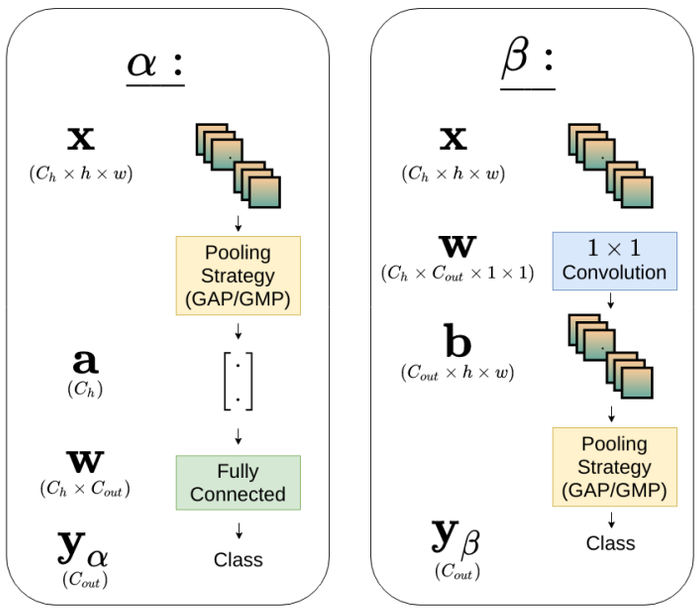


Figure 6.3: Illustration of the original CAM model vs. an alternative structure to train a segmentation model with weak supervision using a classification loss. α illustrated the structure of the final layers in the original CAM model. β illustrated the alternative.

Junsuk Choe et al. showed that the CAM_β architecture shown on the right is computationally equivalent to the CAM_α on the left of the figure when using global average pooling (GAP) as pooling strategy [19].

This lead us to the idea that the convolutional layers before the segmentation maps can be defined as part of the base model to be extended by the CAM-adaptation. An observation that naturally implies a simplifying modification to the CAM method is to have a base segmentation model with a global pooling layer directly followed by a softmax activation. With such a model, each channel in the segmentation map would represent positional information about their corresponding output class predictions without needing further weighting or aggregation. This architecture allows us to apply a weakly supervised training scheme using image level labels on a model designed for fully supervised semantic segmentation without addition of parameters as long as the final layer in the segmentation model is a 1×1 convolution.

This stands in contrast to the original CAM method, where a segmentation model is modified to be trained with classification labels by adding a global pooling layer followed by a fully connected layer. The fully connected layer adds parameters that are not part of the original segmentation model, thus creating a change in architectural design. This architectural design difference thus

introduces an additional difference to take into account besides the training information itself when comparing the weakly supervised model to its fully supervised counter part.

Although the architectures are computationally equivalent when using GAP as shown by Choe et al. [19], this is not the case when using GMP. We show the computational difference below where $y_{p,\alpha}$ and $y_{p,\beta}$ denote the class prediction for the p-th class of model α and β respectively.

$$
\begin{aligned}
y_{p,\alpha} &= \sum_{k=1}^{C_h} a_k \mathrm{w}_{kp} && y_{p,\beta} = \operatorname*{GMP}_{i,j}(b_{ijp}) \\
y_{p,\alpha} &= \sum_{k=1}^{C_h} \operatorname*{GMP}_{i,j}(x_{ijk}) \mathrm{w}_{kp} && y_{p,\beta} = \max_{i,j}(b_{ijp}) \\
y_{p,\alpha} &= \sum_{k=1}^{C_h} \max_{i,j}(x_{ijk}) \mathrm{w}_{kp} \quad \neq && y_{p,\beta} = \max_{i,j}\left(\sum_{k=1}^{C_h} x_{ijk} \mathrm{w}_{kp}\right), \qquad p = 1, \ldots, C_{out}
\end{aligned}
\tag{6.1}
$$

Note that $C_h = 100$ and $C_{out} = 2$ in our experiments and that generally one would typically expect $C_h > C_{out}$ as C_h represents the knowledge capacity of the hidden layers, while C_{out} are only the number of output classes. The analysis of the creators of the original CAM method, which we will refer to as CAM_α from here on, was that the final hidden feature map (here of shape $C_h \times h \times w$) would learn to view isolated objects in the image which would be aggregated in a weighted sum in the final fully connected layer producing the class prediction. Using max pooling on each of the hidden feature maps with C_h channels would pick out the most confident pixel for each hidden channel producing a confidence score for each of the assumed C_h objects or regions of significance in the image. A vector of these confidence scores would then be aggregated in a weighted sum for each class prediction to decide which object or region of significance contributed positively and negatively for each class prediction.

The modified architecture which we denote CAM_β does not use the max operation before on the final segmentation feature map of C_{out} channels (here 2). Therefore, only one pixel from each channel in the final feature map will be used for the class prediction. By inspection of the computation, that would cause each back propagation step to provide non-zeros gradients for a $C_h \times 1 \times 1$ vector of values in the last hidden feature map corresponding to the pixel position of highest confidence in the final segmentation feature map.

We hypothesize that this could cause easier training by encouraging even more locational correlation between class predictions and the objects or scalient features at that location in the input image.

However, note that our modification puts even more weight on the activation scores at single locations than using the CAM_α method with GMP. Thus we hypothesize the problem of part domination will be even more severe for CAM_β than for CAM_α and we also hypothesize the predictions will be quite noisy as only one pixel location in both the final segmentation feature map and the final hidden layer feature map will get direct feed back by the loss function from each image prediction.

6.1.3 Neighborhood Correlation Enforcement

To address the problems of part domination and noisy predictions, we propose to apply a Gaussian smoothing layer at the end of the final segmentation feature map before aggregation with GMP.

Kolesnikov et al. [40] wanted to improve upon the results of CAM_α by finding a pooling strategy that would produce appropriately sized objects in the final segmentation predictions [85]. As discussed in Section 5.1.1, they proposed global weighted rank pooling (GWRP). Their contribution was an attempt to find a sweet spot between the overestimation of object sizes when using GAP and underestimation when using GMP. One can visualize the difference as an adjustment of how many pixel locations in the final feature maps can have a say in the classification prediction and will therefore also get feed back for a prediction. When all pixels are equally weighted (GAP), a lot of pixels need high activations for a positive class prediction. GWRP caused a few more pixels for each feature map to have a significant influence on the final feature map compared to GMP. However, their method did not take location correlation into account. As an alternative attempt to find such a sweet spot, we propose the addition of the Gaussian smoothing layer to the CAM feature maps. The smoothing imposes a greater correlation between neighboring pixels both during forward and backward propagation.

With the CAM_β scheme explained in Section 6.1.2, the addition of a Gaussian smoothing layer would make the gradients flow through the most confidently predicted neighborhood (not pixel) for each class for each training image. This could considerably mitigate part domination as the gradients flowing back from one predicted pixel in the final segmentation map is distributed to a full neighborhood of locations in all channels of the final hidden feature map.

Additionally, the Gaussian smoothing moderates extreme pixel values in the

final segmentation feature map by a weighted sum of the neighborhood which should reduce the noise in the final predictions. Also note that this is different from adding smoothing only during testing as the network will be trained to adjust for the introduced error of smoothing which will be particularly pronounced around sharp corners and regions of detail. To be sure, such adjustments have limits as the smoothing does prevent any strong contrast in small neighborhoods despite anything the network can do. However, the error introduced by the smoothing is very small in comparison to the error caused by part domination with our parameter choice for the smoothing kernel (described in the following chapter).

In the next chapter we describe how we have designed our experiments to address the hypotheses posed.

7

Experimental Setup

In this section we describe how we trained and tested a set of models to test the hypotheses posed in Chapter 6. The data set used in these experiments is the Liver Tumor Segmentation Benchmark data set [11] while we only have had the computational power to use a small portion of it. The setup is designed with informativeness in mind and we use a range of metrics to that end while following the advice of Davide Chicco et al. [16, 17] to make compare overall performance of binary classification with the use of Matthews Correlation Coefficient (MCC) discussed in Section 4.2.

In the fully supervised setting, we compare our proposed ConvMixerSeg model described in Chapter 6 to a Fully Convolutional Network described in Chapter **??** with a ResNet-50 backbone[1], which we refer to as *FCN-ResNet-50*. The FCN was considered a good candidate for a comparison as it has been one of the most influential segmentation models in the last 5 years with a current cite count of more than 27000. The ResNet-50 backbone was chosen as the ResNet based FCN has been shown to outperform the original FCN for semantic segmentation [78]. We thus find the FCN-ResNet-50 a strong candidate both for its footprint and performance while also representing a classical CNN framework. We hypothesize that the ConvMixerSeg network will train faster and produce better overall performance than the FCN-ResNet-50.

1. The FCN model used in our experiments was accessed through pytorch and is described in the documentation: `https://pytorch.org/hub/pytorch_vision_fcn_resnet101/`

For the case of weakly supervised semantic segmentation, we have several hypotheses. First, that the overall performance of the proposed CAM_β method in combination with Gaussian smoothing will outperform the CAM_α method using the same ConvMixerSeg backbone. Second, that the problem of part domination will be more pronounced in our CAM_β model than the CAM_α model when not using the Gaussian smoothing layer. Third, that the Gaussian smoothing will help specifically for the issue of part domination by on average being more sensitive and predicting larger object sizes. Forth and finally, we hypothesize that both the proposed addition of a residual connection and the use of intensity augmentation during training improves performance.

Note that the order of hypotheses have changed as they are now arranged in an order corresponding to how we have run our experiments.

To get results that address our hypotheses, we first evaluate how the proposed architecture performs against the FCN-ResNet-50 in the fully supervised setting. Then we investigate its performance in the weakly supervised setting and compare it to the original CAM design. Finally an ablation study is performed to investigate significance of the effect of each discussed model component.

7.1 Comparisons

In the fully supervised setting, we trained both the proposed ConvMixerSeg and an FCN-ResNet-50 with weighted cross entropy loss (CE-loss) where the weights were the inverse frequency of pixels of each class in the training data.

For the case of weak supervision we first compare the CAM_α to our proposed CAM_β method, both described in detail in Section 6. After some optimization of the threshold value to convert heat maps into segmentation maps for the CAM_α method, we settled on 0.5. Note that according to Choe et al. this optimization step could potentially make a significant difference which is why we attempt to address it here [19]. The optimization was done using a small subset of the training data. For Both CAM methods, we use global max pooling (GMP) as pooling strategy and binary cross entropy loss (BCE-loss) as the classification objective function.

7.2 Ablation Study

We investigate the effects of each part of the proposed CAM$_\beta$ method by comparing our model to alternatives where portions of the design were removed. The baseline model was the ConvMixerSeg trained with CAM$_\beta$ using GMP and trained with the use of intensity augmentation (ConvMixerSeg-CAM$_\beta$). The alternatives are listed below.

- **Baseline-GS:** The Baseline less the Gaussian smoothing layer.
- **Baseline-GS+BU:** The Baseline less the Gaussian smoothing layer where the transposed convolutional upsampling layer is replaced with bilinear upsampling.
- **Baseline-Res:** Baseline less the added residual connection over the point-wise convolutional block in the hidden ConvMixerSeg layers.
- **Baseline-Aug:** The Baseline model less the intensity augmentation.

Note that we have added an experiment that might seem surprising. Baseline-GS+BlnrUp is a design to simply make sure our reasoning makes sense. A worry was posed by a colleague during this project that following a learnable upsampling with Gaussian smoothing might negate the effects of having learnable upsampling at all. Therefore we also experiment with an alternative where we replace the upsampling and Gaussian smoothing layers with a single non-learnable bilinear upsampling layer.

For each alternative, we trained at least 40 models independently to get representative estimates. For each, we perform a Mood's median test to estimate the probably of the median performance. We initially use the Mood's test rather than the t-test as it is a non-parametric test that does not assume normally distributed groups. However, as it is not a very powerful test when groups can be assumed to be normally distributed, we will also use the t-test to compare means in a few cases when it is reasonable to assume normality.

7.3 Data

All experiments were conducted on the Liver Tumor Segmentation benchmark data set (LiTS) [20]. The data set consists of 131 volumetric computed tomography (CT) images taken at a variety of different institutions with different machines (including Ludwig Maxmilian University of Munich, Radboud University Medical Center of Nijmegen, Poly-technique & CHUM Research Center

Montral, Tel Aviv University, Sheba Medical Center and IRCAD Institute Strasbourg and Hebrew University of Jerusalem). The resolution of all are 512×512 in the horizontal/axial plane while the resolution in the vertical direction varies between 74 and 987 with a mean of approximately 450. The resolution in the vertical direction is also affected by the fact that the distance between voxels (3 dimentional pixels) also varies. We have in all our experiments chosen to only focus on segmentation on liver and we leave lesion segmentation to future work. The liver lesion pixels were thus in our experiments labels as part of the liver.

Because of computational limitations we divide every volumetric image into slices in the vertical direction and use only a subset of those slices for both training and testing. To get realistic estimations of performance of unseen data, we divide the volumetric images into groups used for training, validation and testing before dividing them up into slices. When divided into slices, we randomly choose a subset of slices from each volumetric image. Each subset was designed such that the number of slices containing liver is equal to the number containing only background. Unless stated otherwise this number was set to 10 for each volume which produced a total of $2 \cdot 10 \cdot 131 = 2620$ 2D images. All experiments were run with the same proportions of training, validation and testing data of 0.6, 0.2 and 0.2 respectively. (i.e. when training withe the 2620 2D images, 1572 were used for training, 524 for validation and 524 for testing).

As our interest was mainly in the properties of the model architectures, we settled for the simple preprocessing of mapping the pixel intensities to the range $[-1, 1]$.

7.4 Optimization, Augmentation and Hyperparameters

For all experiments we use the Adam optimizer with the default $\beta_1 = 0.9$, $\beta_2 = 0.999$). The learning rate was set to 0.01 for all experiments with the ConvMixerSeg model and to 0.001 for the experiments with the FCN based on a few initial experiments on a small portion of the training data. (Note that this ever before viewing the results implies the ConvMixerSeg trains more easily).

Intensity augmentation was applied during training for most models. We used a gamma transformation with randomly chosen gamma values in the range $(0.5, 1.5)$ which was applied to all inputs during training. The effects of applying the augmentation was investigated in the ablation study.

For hyperparameters in the ConvMixerSeg architecture, we have used a depth of 20 ConvMixerSeg layers, all with a channel number of $C_h = 256$. The transition channel count in the upsampling was set to $C_t = 100$.

For the Gaussian smoothing, we use the parameters $\sigma = 7$ which defined the spread and a kernel size of 50×50.

7.5 Evaluation

For the quantitative evaluation, several measures have been used. As discussed in Section 4.2, different measures provide different types of information and some are more prone to being misinterpreted when the data in imbalanced.

In our experiments, we have chosen to follow the advice of Davide Chicco and Giuseppe Jurman in that we use Metthews correlation coefficient (MCC) as our main source of evidence for evaluating and comparing models [16, 17]. However, we include other measures for reference and for deeper interpretations as e.g. the recall score contributes with additional information related to the discussed problem of part domination, that the MCC does not provide. Thus, for segmentation, we present the result of all methods using their mean score of Accuracy (Acc), Matthews Correlation Coefficient (MCC), intersection over union (IoU), precision (Pre) and recall (Rec) as quantitative measures. The precision, recall and intersection over union scores were only calculated for the liver class while the accuracy and the MCC scores were global estimates. For every measure we provide estimates of mean, standard deviation and median to aid in analyzing the distributions of each measure.

For classification performance of the weakly supervised methods, we use the same metrics. Estimators for classification scores are marked with a leading "Cls-" to clearly distinguish classification and segmentation results. On occasion we also include maximum values or show full histograms for additional information and analysis.

A note of caution: The qualitative evaluation of the experiments should be considered less informative and more prone to erroneous conclusions than the quantitative analysis. This is because we do not have the capacity to view enough samples from each independently trained model for every configuration, to be sure our analyses are actually representative. However, we still deem it valuable to attempt to view as many examples as possible and analyze them in context of measured results to see if we can make sense of the numbers.

7.6 Uncertainty of Estimates

Despite powerful initialization schemes such as those discussed in Section 2.6, random variations in initialization converges on different solutions. It is thus hard to know if a single result reflects what one can expect to achieve upon reproducing the experiment with a different random seed.

In an attempt to provide as informative results as possible about the distribution of potential results rather than a single sample, we run every experiment at least 25 times with different random seeds and give standard deviation estimates for quantitative estimates.

8

Results and Discussion

In this section we analyze the measured semantic segmentation performance of the proposed ConvMixerSeg model and compare it to an FCN with a ResNet-50 backbone. For the weakly supervised case, we compare our proposed development of CAM to the original before investigating the effects of model components more thoroughly with an ablation study. We inquire relevant comparisons using statistical tests and evaluate the experimental results both quantitatively and qualitatively. Finally we give a summary to address the hypotheses posed in Chapter 6. Results of all experiments are shown in extended form in the Appendix 10.1.

8.1 Fully supervised

In this section we show our investigation of the proposed model as well as the model chosen for comparison before using statistical tools to arrive at a conclusion about which model performs best. This is done by comparing the group of trained models using the ConvMixerSeg model and the group using the FCN-ResNet-50 model. We conclude our results based on estimates of most representative MCC scores from each group, either the mean or the median depending on the skewness of group distributions.

8.1.1 ConvMixerSeg trained with CE-loss

The segmentation performance of the ConvMixerSeg trained with CE-loss is quantitatively summarized in Table 10.1.

Table 8.1: Measured performance of the ConvMixerSeg model trained with CE-loss for 25 epochs. Each element is given on the format: $mean \pm std(median)$.

	Training	Validation	Testing
Acc	$0.981 \pm 0.098(0.995)$	$0.897 \pm 0.223(0.975)$	$0.893 \pm 0.233(0.983)$
MCC	$0.906 \pm 0.126(0.927)$	$0.669 \pm 0.251(0.756)$	$0.682 \pm 0.27(0.782)$
IoU	$0.842 \pm 0.118(0.864)$	$0.537 \pm 0.236(0.593)$	$0.562 \pm 0.255(0.644)$
Prec	$0.845 \pm 0.118(0.867)$	$0.556 \pm 0.247(0.61)$	$0.582 \pm 0.266(0.694)$
Rec	$0.992 \pm 0.03(0.996)$	$0.939 \pm 0.043(0.953)$	$0.94 \pm 0.049(0.959)$

There is a drop in the mean scores going from the training data, to the validation and testing data. This is a sign of overfitting, but the initial impression may be somewhat incomplete when we see the means in context of the very high spread of the data as well as the large difference between the means and medians. The recall varies much less than the rest of the metrics, but standard deviations of $3-5\%$ are still quite high considering that state-of-the-art models often are separated by less than two percentage points.

Overall, the values imply negatively skewed distributions. If the scores were all normally distributed, we would expect that $\approx (95/2)\% = 47.5\%$ of the samples would lie within two standard deviations above the mean values and the means and medians would have been very similar. However, the median value is larger then the mean for all metrics measured for all portions of data and for some, by ≥ 10 percentage points (e.g. Validation MCC). Observe also that all of the mean scores in the table would become greater than 1 (the range limit) if they were added two standard deviations, and some of them by adding only one standard deviation. We thus induce that the distributions of all scores in the table are negatively skewed, some largely so.

Note that the accuracy here is quite misleading as the background pixels outnumber the liver pixels by more than an order of magnitude.

To address the question of overfitting more closely, view Figure 8.1 which shows a histogram of the MCC scores. The colors correspond with the portion of the data analyzed; blue for training data, orange for validation data and red for testing data.

The histogram shows that almost all models score in the upper 80-s or lower 90-s on the training data (The corresponding IoU scores are in the middle

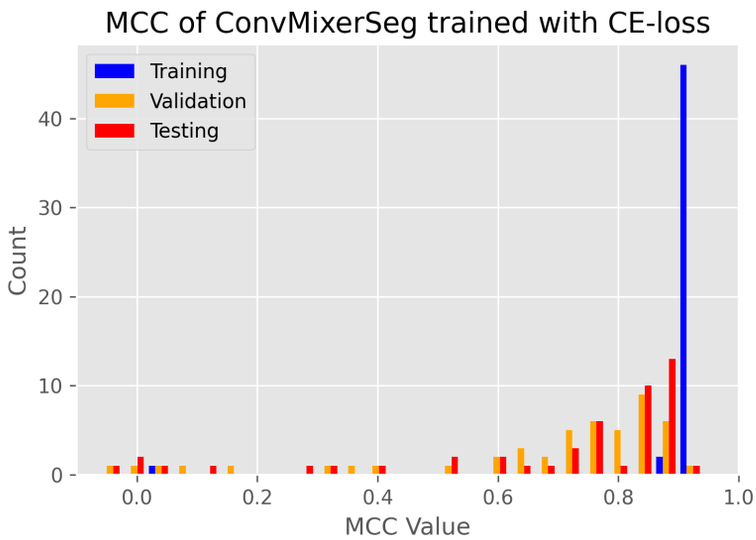


Figure 8.1: Histogram of the measured MCC score for ConvMixerSeg models trained with CE-loss. The performance on the training, validation and testing data are shown in blue, orange and red respectively.

80-s for all). It also shows that approximately one third of the models scored comparably well or only slightly lower on the testing data. The best model had an MCC score of 94%, 91% and 92% on the training, validation and testing data respectively (88%, 84% and 85% IoU).

Note the small blue bar very close to 0 in the histogram. Among all the models trained in this setting, approximately 2% seems to not have trained at all with an MCC score implying predictions were as good as random on the training set[1]. We have here only shown the histogram of MCC scores, but the histograms of IoU and precision scores look similar.

Note that even though the generalization of the models vary greatly, most models have a high recall value with only a few exceptions being lower than 90%. The implication is that the models typically lets few true liver pixels go undetected. It also means that the majority of the errors made my the models are false predictions of liver when the true class is background. A description that seems to describe examples images well in general. See a few examples from the testing data in Figure 8.2. The first row are the input images, the second and third rows are the corresponding ground truths and prediction heat maps[2] respectively.

1. Recall that MCC ranges from -1 to 1 where 0 is as good as random.
2. We choose to show heat maps rather than segmentation predictions as they are somewhat

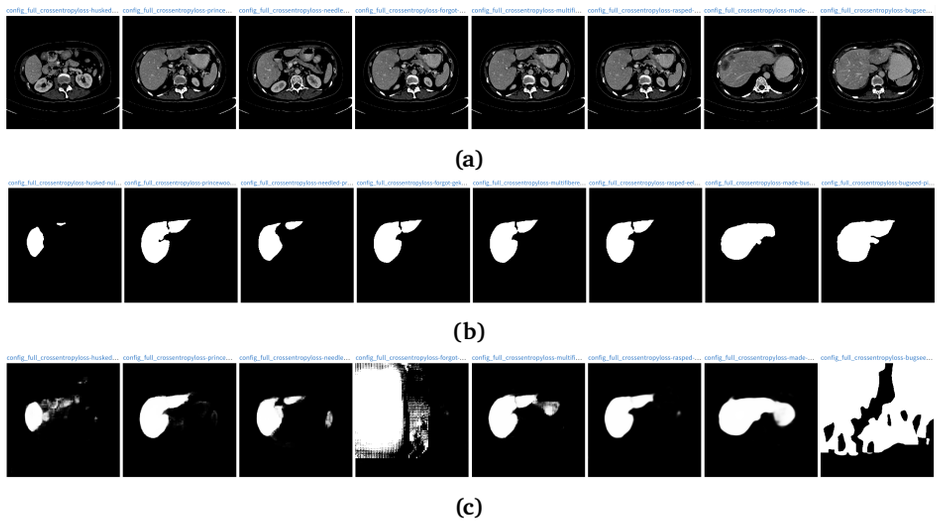


Figure 8.2: Example images from the testing data shown in (a) with corresponding labels or ground truths shown in (b) and heat map predictions of arbitrarily picket ConvMixerSeg models trained with CE-loss in (c).

Some of the predictions are very good such as those in the second, third and sixth column. This represents well the best third of the models that measurably does well on all portions of the data. The first, fifth and seventh columns show predictions that cover the ground truth well, but are somewhat over sensitive as we would expect with the observed recall scores. Column four and eight show models that perform very badly.

To summarize the performance of the ConvMixerSeg model trained with CE-loss, almost all models successfully train to do correct predictions on the training set, most generalize to some degree to unseen data, while about a third do comparatively well on training, validation and testing data. Overall, the measured performance is a mean MCC score of 0.694 ± 0.271 and IoU score of 0.694 ± 0.271 and respective median scores of 0.782 and 0.644.

8.1.2 FCN trained with CE-loss

The segmentation performance of the FCN with a ResNet-50 backbone is quantitatively summarized in Table 10.2.

We will point out three things that are strikingly different here compared to the results of the ConvMixerSeg. First, the larger similarity between the training

more informative as intensity can be interpreted as the level of confidence by the network.

Table 8.2: Measured performance of the FCN-ResNet-50 model trained with CE-loss for 25 epochs (marked with R). Each element is given on the format: $mean \pm std(median)$. We have here also repeated the table for the ConvMixerSeg model for reference (marked with C). The highest mean test performances are shown in bold.

		Training	Validation	Testing
R	Acc	0.962 ± 0.01(0.96)	0.936 ± 0.129(0.964)	0.939 ± 0.129(0.965)
	MCC	0.656 ± 0.072(0.649)	0.637 ± 0.147(0.677)	0.655 ± 0.146(0.686)
	IoU	0.464 ± 0.094(0.45)	0.459 ± 0.149(0.49)	0.48 ± 0.152(0.504)
	Prec	0.472 ± 0.094(0.458)	0.492 ± 0.173(0.504)	0.508 ± 0.174(0.515)
	Rec	0.96 ± 0.013(0.959)	0.923 ± 0.123(0.962)	0.934 ± 0.098(0.966)
C	Acc	0.981 ± 0.098(0.995)	0.897 ± 0.223(0.975)	0.893 ± 0.233(0.983)
	MCC	0.906 ± 0.126(0.927)	0.669 ± 0.251(0.756)	0.682 ± 0.27(0.782)
	IoU	0.842 ± 0.118(0.864)	0.537 ± 0.236(0.593)	0.562 ± 0.255(0.644)
	Prec	0.845 ± 0.118(0.867)	0.556 ± 0.247(0.61)	0.582 ± 0.266(0.694)
	Rec	0.992 ± 0.03(0.996)	0.939 ± 0.043(0.953)	0.94 ± 0.049(0.959)

performance and the validation and testing performance. This could imply that the model is more robust, but it might also be the result of the ResNet training slower on average. Second, the mean scores are overall lower than for the ConvMixerSeg model with one exception. The test accuracy is higher, but as discussed in Section 4.2, the accuracy metric is misguiding when the data is imbalanced. Taking all measures into account, the high accuracy is a symptom of severe undersegmentation of the liver. We will make appropriate comparisons of distribution representative estimates in the next section. Third, notice the much more similar values between means and medians. These distributions are not as skewed as those for the ConvMixerSeg.

The mentioned observations are supported by the histogram of the MCC scores for the FCN in Figure 8.3.

The distribution of the MCC scores on the training data are more spread out for the FCN model than for the ConvMixerSeg. The highest scoring models lie in the lower 80-s. Compared to the ConvMixerSeg, the spread in performance is generally smaller for the validation and testing data, but wider for the training data. Approximately 60% of the models score equal or better on the testing data than the training data indicating very good generalization.

8.1.3 Comparison

To find out if there is a significant difference in means, we could apply a student t-test. However, the t-test assumes normality in each group which we

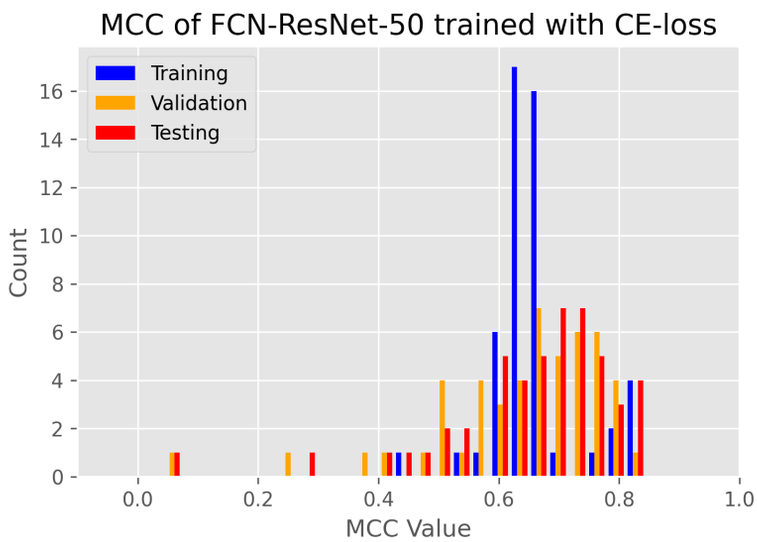


Figure 8.3: Histogram of measured MCC score for FCN-ResNet-50 models trained with CE-loss. The performance on the training, validation and testing data are shown in blue, orange and red respectively.

know to be far from reasonable for the ConvMixerSeg results making a t-test result potentially misguiding and thus less informative. As a more reasonable alternative, we have used the non-parametric Mood's median test witch puts no assumptions on the distributions of the groups. Also note that the median of the distribution might be considered more representative of the ConvMixerSeg models than the mean because of the skewness of the distributions.

Applying Mood's median test to compare the MCC scores on the testing data results in a p-value$\approx$ 0.006 with the ConvMixerSeg median being the highest. The p-value for the IoU scores is in this case identical [3]. We can thus, with a high degree of confidence, assume the true median of the distribution of MCC scores (as well as IoU scores) on the testing data is higher for the ConvMixerSeg model trained with CE-loss than for the FCN-ResNet-50 trained with the same objective function. The proposed model outperforms the FCN-ResNet-50 model in the median with high probability.

3. Note that Mood's Median test uses ranks of values rather than the actual values to compute the test result.

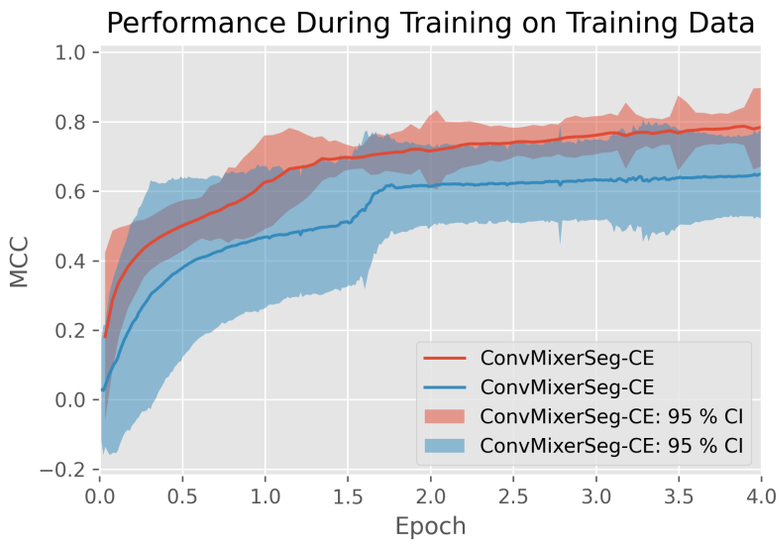


Figure 8.4: MCC performance on training data during the first few epochs of training for the ConvMixerSeg and FCN-ResNet-50 models, both trained with CE-loss. The plot shows the estimated means and 95% confidence intervals for each log step.

8.1.4 Further Discussion

To be sure, we cannot ignore the fact that the ConvMixerSeg model overfitted more often than the FCN. This could imply that the ConvMixerSeg model trains faster and thus overfits earlier. By observation of Figure 8.4, this seems to be the case.

One might question if the number of epochs should have been higher for the FCN model as the slower learning would imply it would take longer for the model to reach its full potential. We cannot reject the possibility, but we can get an indication by seeing if some of the FCN models have overfitted. Typically, we assume that models learn the relevant features before overfitting, i.e. the performance on the validation data improves in parallel with the performance on the training data for as long as the appropriate training lasts. After this, the training performance and validation performance typically diverge. If many FCN models overfitted, it could indicate that training further would not improve the overall result by much. Note however, it might also only imply that the ideal training length varies greatly.

Approximately 14% of the FCN models had a drop in measured MCC score of more than 10 percentage points going from training to testing data, a sign of strong overfitting. This implies a significant proportion of the FCN models have

overfitted. We therefore assume that one of two cases is true. Either the ideal length of training for the FCN-ResNet-50 varies significantly or 25 epochs is near adequate for the FCN-ResNet-50 with the dataset we have used. In both cases, to our best judgment, it seems reasonable to accept the conclusion of the comparison as valid.

8.2 Weakly Supervised

For the weakly supervised setting we first analyze our proposed weakly supervised segmentation model, i.e. the ConvMixerSeg trained with CAM_β with Gaussian smoothing using GMP as pooling strategy which we will reference as *ConvMixerSeg-CAM*$_\beta$. Then we investigate the results of using the original CAM_α method discussed in Section 5.1.1 using the same backbone and thus denoted by *ConvMixerSeg-CAM*$_\alpha$.

8.2.1 ConvMixerSeg-CAM$_\beta$

The measured classification and segmentation performance of our experiments with the ConvMixerSeg-CAM_β model are shown in Table 8.3 and Table 8.4 respectively.

Table 8.3: Measured classification performance of the ConvMixerSeg model trained with the CAM_β method for 3 epochs. Each element is given on the format: $mean \pm std(median)$.

	Training	Validation	Testing
Cls-Acc	0.913 ± 0.039(0.92)	0.885 ± 0.08(0.93)	0.888 ± 0.084(0.936)
Cls-MCC	0.795 ± 0.06(0.805)	0.7 ± 0.271(0.828)	0.707 ± 0.279(0.842)
Cls-IoU	0.887 ± 0.052(0.896)	0.858 ± 0.094(0.908)	0.861 ± 0.099(0.915)
Cls-Prec	0.975 ± 0.013(0.976)	0.954 ± 0.078(0.983)	0.957 ± 0.079(0.988)
Cls-Rec	0.909 ± 0.056(0.921)	0.902 ± 0.1(0.936)	0.903 ± 0.103(0.936)

For classification, the precision of the ConvMixerSeg-CAM_β models is typically in the high 90-s, while the recall is somewhat lower, i.e. the models will typically let a few images containing liver go undetected. However, when they predict that an image contain liver, that is typically correct. As discussed in section 4.2, there is typically a balance between precision and recall where the highest overall performance is reached by allowing some error in both directions. In this case the balance is more on the under sensitive side, but the imbalance is not very large.

As with the case of full supervision, several of the mean scores are lower for the validation and testing data than the training data. Yet, this is not reflected by the medians. Further more, the standard deviations for the unseen data are much higher than for the training data, particularily for the precision and MCC scores. The overall picture seems to be that for classification, most models perform quite well on both training and testing data, while a few models overfit. The overfitted models produce low scores on the unseen data and thus reduces the means. But as these models are outnumbered by the models than generalize well, the medians remains high.

Table 8.4: Measured segmentation performance of the ConvMixerSeg model trained with the CAM$_\beta$ method for 3 epochs. Each element is given on the format: $mean \pm std(median)$.

	Training	Validation	Testing
Acc	0.971 ± 0.004(0.972)	0.932 ± 0.13(0.967)	0.93 ± 0.143(0.968)
MCC	0.375 ± 0.169(0.409)	0.279 ± 0.222(0.365)	0.301 ± 0.234(0.396)
IoU	0.212 ± 0.105(0.218)	0.161 ± 0.136(0.171)	0.178 ± 0.148(0.203)
Prec	0.684 ± 0.238(0.747)	0.561 ± 0.355(0.719)	0.59 ± 0.352(0.765)
Rec	0.23 ± 0.114(0.24)	0.213 ± 0.184(0.209)	0.239 ± 0.202(0.223)

The segmentation performance of the ConvMixerSeg-CAM$_\alpha$ is divided. The difference between means and medians indicate skewed distributions, especially prominent in e.g. MCC score for the testing data. But upon closer inspection, there are really two groups of models. The once that produce class activation maps that are more or less indicative of the location of the liver and the once that seem to learn the classification problem in some other way. This can be more clearly seen in the distribution of the MCC-scores shown in Figure 8.5.

The figure shows what can be understood as a mixture of two distributions. First, a bell shaped distribution of models with a mean between 0.4 and 0.5. Second, a high and narrow spike around 0. The first group are models that perform relatively well. We have inspected a few of these models and have seen that they consistently produce results that are convincing and typically have precision scores higher than 0.7. The best models produce MCC scores in the low 60-s (and IoU scores in the low 50-s) while such results are untypical. The models in the first group also generalize well as they often produce as good or ever better results on the validation and testing data than on the training data.

However, the recall scores lag behind. Upon closer inspection, we find this to be caused by two problems. First, that some livers go completely undetected. This is also reflected in the classification recall values. Second is the problem

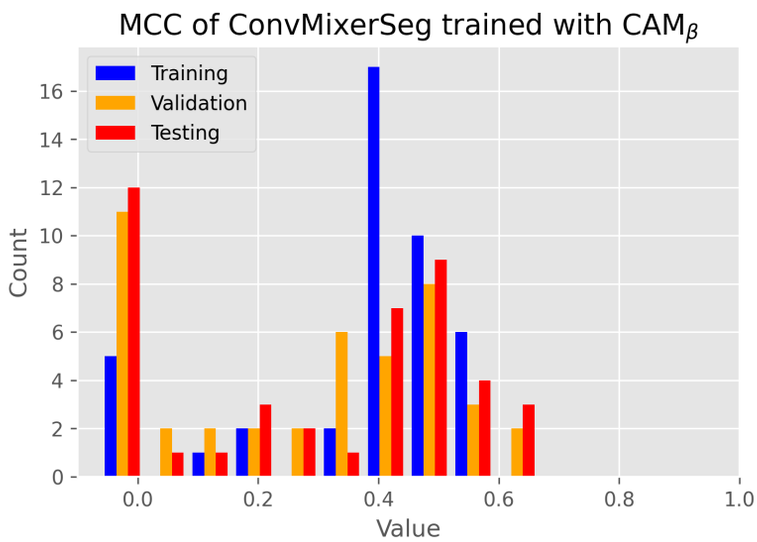


Figure 8.5: Histogram of the measured MCC scores for ConvMixerSeg models trained with the CAM$_\beta$ method. The performance on the training, validation and testing data are shown in blue, orange and red respectively.

of part domination that, although reduced, have not completely solved.

The spike around 0 shows that there is a small group of models are have badly overfitted and only produce segmentation maps better than random on the training data while generalizing poorly to validation and testing data. It also shows another small group that on average produce nothing better than random on any data, thus seeming to have learned the classification problem in a different way.

While some models do not seem to do well, most models produce segmentation maps with a high correlation between predictions and labels on unseen data and some reach IoU scores in the lower 50-s. See some examples images with corresponding prediction heat maps and labels in Figure 8.6.

The figure shows a random subset of the testing data shown on the first row, the corresponding ground truths or labels on the second row and the prediction heat maps of arbitrarily picked ConvMixerSeg-CAM$_\beta$ models on the third row.

In the images we see that the prediction in the second column seems to cover the true prediction quite well although it also predicts a neighboring organ to be part of the liver. The more representative situation for the average performance can be seen in column 1, 4 and 8 where the precision seems high, but some

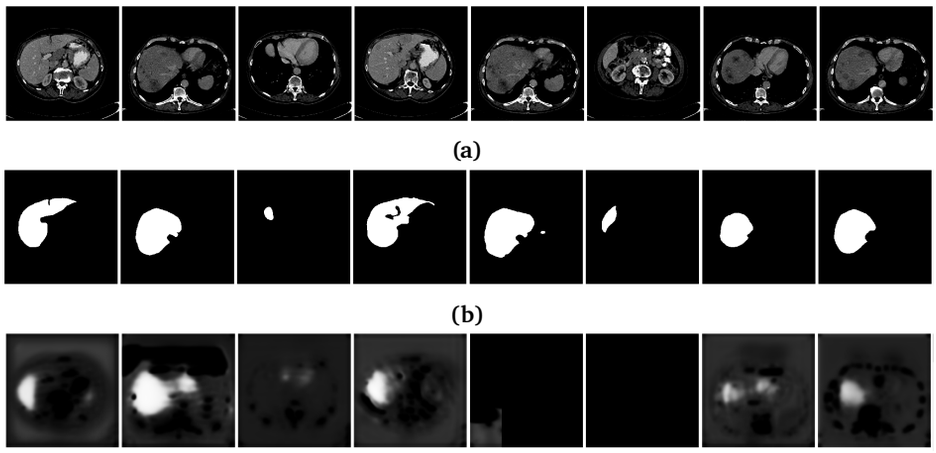


Figure 8.6: Example images from the testing data shown in (a) with corresponding labels or ground truths shown in (b) and heat map predictions of arbitrarily picket ConvMixerSeg-CAM$_\beta$ models in (c).

part domination is more prominent.

8.2.2 ConvMixerSeg-CAM$_\alpha$

The classification and segmentation results of our experiments with the ConvMixerSeg-CAM$_\alpha$ model are shown in Table 8.5 and Table 8.6 respectively. Note that we have repeated the results of the ConvMixerSeg-CAM$_\alpha$ model in the corresponding tables for convenience.

The classification results using ConvMixerSeg-CAM$_\alpha$ are more consistent than those of ConvMixerSeg-CAM$_\beta$ in that the standard deviations are all lower. The ConvMixerSeg-CAM$_\alpha$ results also seem to generalize well in that the scores are similar for training, validation and testing data. Notice that classification precision is perfect, i.e. all training, validation and testing images containing liver were correctly predicted to have liver.

But the recall is very low in comparison to the precision, indicating the model is significantly under sensitive. With the exception of the precision, the overall scores do not seem to reach up to those of the ConvMixerSeg-CAM$_\beta$ and we will show that the difference in medians is statistically significant for the MCC.

In our experiments, the performance of ConvMixerSeg-CAM$_\alpha$ for segmentation was often close to as good as random despite a consistently much better than

Table 8.5: Measured classification performance of the ConvMixerSeg model trained with the CAM_α method for 10 epochs (marked with α). Each element is given on the format: $mean \pm std(median)$. For reference we also repeat the classification results of the CAM_β method (marked with β).

		Training	Validation	Testing
α	Cls-Acc	0.732 ± 0.025(0.742)	0.739 ± 0.026(0.75)	0.74 ± 0.022(0.75)
	Cls-MCC	0.557 ± 0.027(0.568)	0.565 ± 0.027(0.577)	0.566 ± 0.024(0.577)
	Cls-IoU	0.643 ± 0.033(0.656)	0.652 ± 0.034(0.667)	0.653 ± 0.03(0.667)
	Cls-Prec	1.0 ± 0.0(1.0)	1.0 ± 0.0(1.0)	1.0 ± 0.0(1.0)
	Cls-Rec	0.643 ± 0.033(0.656)	0.652 ± 0.034(0.667)	0.653 ± 0.03(0.667)
β	Cls-Acc	0.913 ± 0.039(0.92)	0.885 ± 0.08(0.93)	0.888 ± 0.084(0.936)
	Cls-MCC	0.795 ± 0.06(0.805)	0.7 ± 0.271(0.828)	0.707 ± 0.279(0.842)
	Cls-IoU	0.887 ± 0.052(0.896)	0.858 ± 0.094(0.908)	0.861 ± 0.099(0.915)
	Cls-Prec	0.975 ± 0.013(0.976)	0.954 ± 0.078(0.983)	0.957 ± 0.079(0.988)
	Cls-Rec	0.909 ± 0.056(0.921)	0.902 ± 0.1(0.936)	0.903 ± 0.103(0.936)

Table 8.6: Measured segmentation performance the ConvMixerSeg model trained with the CAM_α method for 10 epochs. Each element is given on the format: $mean \pm std(median)$.

Testing		Training	Validation
Acc	0.32 ± 0.079(0.308)	0.394 ± 0.145(0.363)	0.406 ± 0.147(0.378)
MCC	0.132 ± 0.109(0.122)	0.119 ± 0.119(0.132)	0.107 ± 0.126(0.101)
IoU	0.017 ± 0.013(0.017)	0.018 ± 0.017(0.015)	0.02 ± 0.018(0.018)
Prec	0.018 ± 0.013(0.018)	0.019 ± 0.017(0.016)	0.022 ± 0.018(0.021)
Rec	0.345 ± 0.241(0.331)	0.3 ± 0.274(0.272)	0.32 ± 0.281(0.319)

random for classification scores. Some models produce segmentation maps that in some regions are correlated with the labels, but typically also contain large regions that are falsely predicted.

It should also be mentioned that some of the ConvMixerSeg-CAM_α models among the best for classification actually produced segmentation maps with negative MCC scores. That is, the predictions did worse than random. Upon inspection, the predictions of such models typically contained large regions outside the body predicted liver.

View Figure 8.7 for some examples. The first column shows a small region with relatively high confidence that is well placed. However, the upper part of the image is predicted with similar confidence causing very low precision. In the second column the model correctly predicts a region approximately near the correct spot and with almost the correct size. But the region is only slightly

brighter than the bright frame around the image. Thus a very carefully chosen threshold values is needed to make a good prediction out of the heat map[4] which well reflect the significant effect of chosing the right threshold discussed by Choe et al. [19] when using the same method for object localization(, although with a different backbone and data set).

The last column seems to overall do worse than random as the highest confidence region mostly does not overlap with the ground truth.

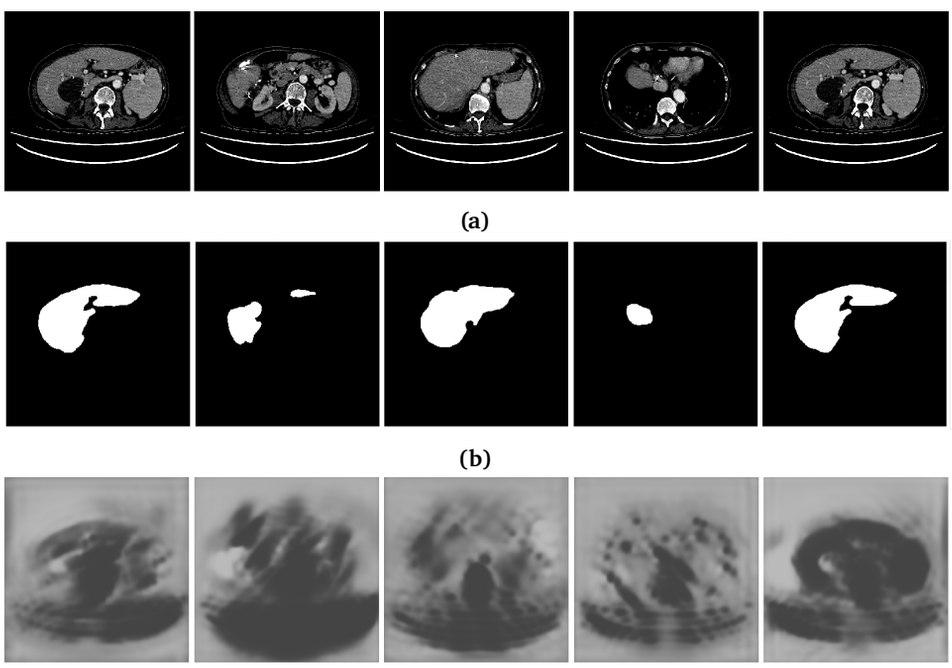


Figure 8.7: Example images from the testing data shown in (a) with corresponding labels or ground truths shown in (b) and heat map predictions of arbitrarily picket ConvMixerSeg-CAM$_\alpha$ models in (c).

8.2.3 Comparison

Mood's median test was used to find out if the difference in medians, for both the Cls-MCC score and the MCC score, was significant. The p-value for Cls-MCC and MCC were $\approx 1.3 \cdot 10^{-8}$ and $\approx 3.6 \cdot 10^{-5}$. We therefore confidently conclude that when using the ConvMixerSeg as backbone, the proposed CAM$_\beta$ training scheme outperformed CAM$_\alpha$.

4. We did attempt to optimize this using a small subset of the training data, but the best threshold value varied from image to image so the overall.

8.2.4 Further Discussion

We have here used a different training length for the two models. This was a decision made based on a few (>5) experiments with a small portion of the training set to see at which training length the models would perform best on segmentation. Note that for classification, both models typically did a few Cls-MCC percentage points better if we continued the training for more epochs.

We chose 3 epochs as we observed in early experiments on a small subset of the training data that the ConvMixerSeg-CAM$_\beta$ method typically learned the classification problem quite fast (if it learned at all). Also, when the training was effective, the segmentation performance often improved in parallel with the classification performance before quickly deteriorating when the classification performance started converging.

It should be mentioned that although we have made attempts to improve the threshold value for the CAM$_\alpha$ model, further experiments to optimize it might improve the final result. However, this could be considered an unfair extra optimization of the CAM$_\alpha$ in comparison to the CAM$_\beta$ method for which we have not optimized the threshold value. Although optimizing the threshold might improve both methods, it seems the CAM$_\beta$ method is more robust to the hyper parameter as the heat maps are.

8.3 Ablation Study

To build an understanding of which parts of our proposed model contributed to the overall performance, we experimented with the alternative models described in Section 7.2.

For the quantitative analysis, we are mainly concerned with the question of if each component did in fact contribute positively to the overall performance. Therefore, we focus on the probability of the observed results occurring assuming equal distribution medians or means depending on what could be safely assumed about performance distributions. For details about each model's performance, view Appendix 10.1.

View Table 8.7. For each Alternative Model setup, we perform a Mood's median test between scores produced by the proposed ConvMixerSeg-CAM$_\beta$ baseline model. A p-value lower than 0.05 in each test implies the true population medians are probably unequal. In all of our experiments, the measured median values were higher (i.e. better) for the baseline. Thus we can interpret low

p-values in every cell of the table to imply a greater loss in model performance caused by the corresponding model modification. The second and third columns address the classification and segmentation performance respectively.

Table 8.7: Ablation study results. Mood's median test has been used to test for median equality between the results of the Baseline and the given Alternative Model. Values marked with "(*)" indicates the result is significant and it is reasonable to assume the baseline model is better in the median.

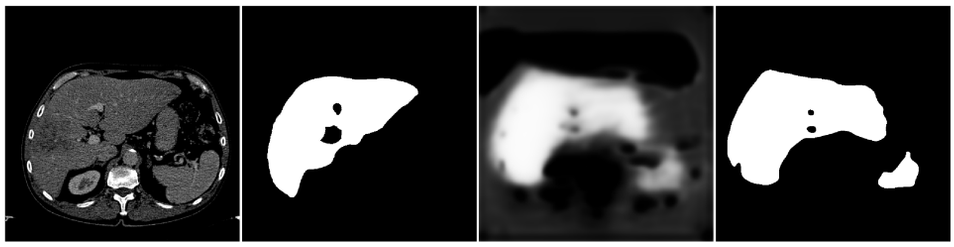
Alternative Model	Cls-MCC: p-value	MCC: p-value
Baseline-GS	0.002 (*)	0.01 (*)
Baseline-GS+BlnrUp	0.03 (*)	0.03 (*)
Baseline-Res	$2.3 \cdot 10^{-7}$ (*)	$5.6 \cdot 10^{-4}$ (*)
Baseline-Aug	0.67	0.39

The first thing to notice in Table 8.7 is that all model changes that have a significantly effect on segmentation performance also cause a significant effect on the classification performance. This might be intuitive, but it is not obvious as some models learned to classify (at least much better than random) while producing class activation maps that were as good as random or even worse.

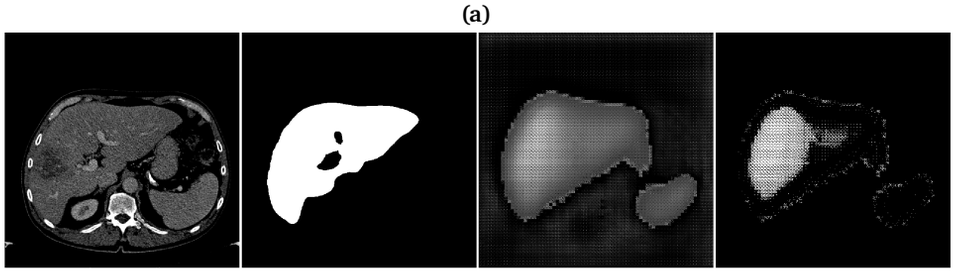
Before continuing the discussion, keep in mind that the p-values do not say anything about distance between the medians. That is to say, a low p-value does not necessarily imply a large median difference, just that there is a difference.

Note that the intensity augmentation contribution was not significant. However, the difference between median and mean imply less skewed distributions of segmentation MCC scores than for the fully supervised setting. For symmetric distributions, the Mood's test has relatively low statistical power compared to the t-test. However, as the t-test assumes normality we should test if that is a reasonable assumption. Thus we applied the Anderson-Darling test to each group [51] which concluded none were significantly different from normality confirming that a t-test result would probably be valid. Finally applying the t-test resulted in a p-value of 0.026. We therefore conclude a significant contribution of the augmentation to the performance of the final model.

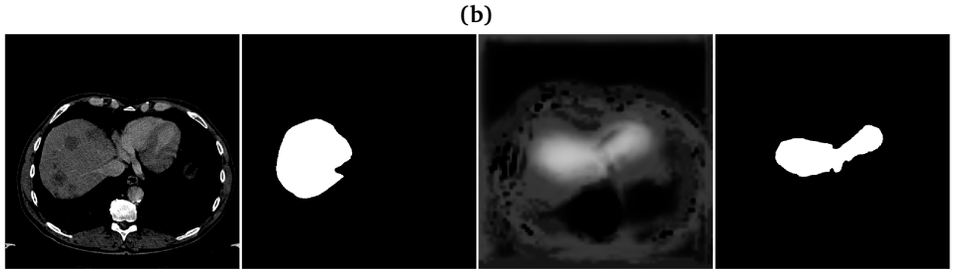
So we have shown all parts contribute significantly. However, for a more complete picture we add some points. We hypothesized that the adding Gaussian smoothing would mitigate the problem of part domination. A typical symptom of part domination is a low recall value. Testing for a difference in medians recall scores (with Mood's median test) results in a p-value of 0.0002 where the median of the baseline models is better.



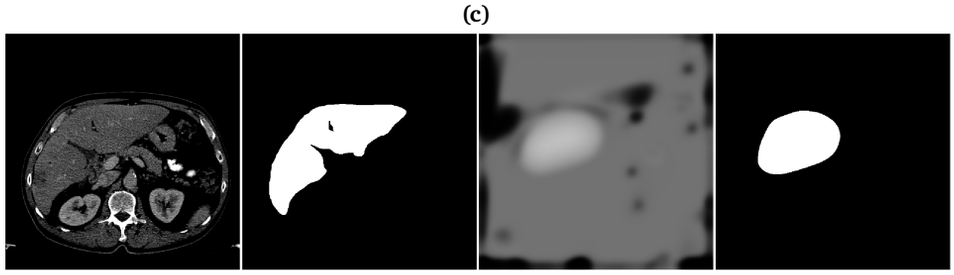
(a)



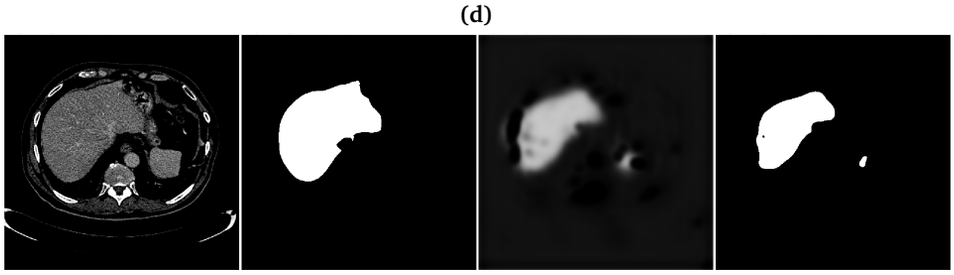
(b)



(c)



(d)



(e)

Figure 8.8: From left to right: Image, ground truth, prediction heat map and thresholded prediction respectively taken from the testing data. The rows correspond to Baseline (a), Baseline-GS (b), Baseline-GS+BlnrUp (c), Baseline-Res (d) and finally for Baseline-Aug (d).

Note that the baseline method also performed better overall so one might question if the improvement was just for the recall or similarly for the precision. If both improvements were equally significant, that would imply the problem of part domination was technically not addressed specifically, any more so than the precision error. Testing for a difference in medians between precision scores from the baseline to the baseline less the Gaussian smoothing results in a p-value of 0.12 (with Mood's medain test). When testing for a difference in means using the t-test the resulting p-value was 0.45 (both groups passed the Anderson-Darling test). The precision improvement is thus not significant although both the mean and median values are higher for the Baseline. Both the precision and the recall measurements thus support our hypothesis that adding the Gaussian smoothing improves results by mitigating the problem of part domination.

For a qualitative evaluation, view the examples shown in Figure 8.8. Each row corresponds to one model design. The example input images are shown on the left followed by the ground truths, the prediction heat maps and finally the thresholded predictions on the right. Note that we have picked examples from relatively successful models for each model design. But to our best judgment, the examples illustrate representative patterns we have observed among the various example images for each model design in general.

The first and final rows represent the baseline and the baseline less the augmentation, which both quantitatively performed yielded high performance scores overall.

The second row illustrates the issue of both noisy predictions and part domination (only predicting small discriminative portions of the true object) which we addressed with the Gaussian smoothing layer. The baseline predicts a larger portion of the true label and also is much less noisy than the Baseline-GS prediction. However, the Baseline-GS model is more sensitive to the choice of thershold as the prediction heat map corresponds much better to the ground truth than the final prediction. Nevertheless, comparing the heat maps rather than the predictions we still see that the level of detail correspondence between input image and heat map prediction seems better for the baseline model.

For the Baseline-GS+BlnrUp experiments were done to test if the Gaussian smoothing would negate or neutralize the effects of the learnable upsampling. This hypothesis is neither supported by the quantitative nor the qualitative evidence as the difference is median of both precision and recall (as well as MCC) are highly significant. More specifically with p-values of 0.002 and 0.0002 for precision and recall medians respectively.

The examples shown from the Baseline-Res model are hard to interpret, but might indicate a network that is a little less developed than the baseline as the predictions are less detailed, there are larger regions of positive predictions with low confidence and the overall performance is significantly worse than for the baseline.

We conclude our analysis with the following: The proposed ConvMixerSeg model outperforms the FCN-ResNet-50 for fully supervised semantic segmentation in the median with high statistical significance. For both classification and segmentation, the results imply that the performance of ConvMixerSeg-CAM$_\beta$ is better than that of CAM$_\alpha$ in the median with high statistical significance.

Further, the results of the ablation study prove a significant positive effect on both classification and segmentation from using the gaussian smoothing layer the added residual connection in the hidden model architecture. Additionally, the results support the hypothesis that this is caused, at least partly, by the gaussian layer mitigating the issue of part domination.

8.4 Full vs. Weak Supervision

On a direct comparison of the value of full supervision, the conclusion is clear. Our results suggest, the gap between full and weak supervision is large. The performance measurements show that the ConvMixerSeg model trained with full supervision performs much better on all metrics used than its weakly supervised counter part. The difference in the median MCC score on the testing data was $0.782 - 0.301 = 0.481$ (for IoU the difference in medians was $0.644 - 0.203 = 0.441$). The p-value testing for a difference is median precision scores is $4.8 \cdot 10^{-15}$, i.e. profoundly significant. If assuming independent normally distributed groups, the estimate of difference in mean MCC score would be: $(0.682 - 0.301) \pm \sqrt{(0.27^2 + 0.234^2)} = 0.381 \pm 0.357$. The estimate is probably a strong underestimate of the true difference as the distribution of the fully supervised MCC scores are very negatively skewed, but the estimate still illustrates a clear difference although not statistically significant as the uncertainties are very high.

Despite the overall picture, there is a significantly higher mean precision score for the weakly supervised model than for its fully supervised equivalent with a p-value of 0.045 (using the t-test as both distributions pass the Anderson-Darling test for normality). This means that a pixel predicted as liver by the average weakly supervised model is more trustworthy to be a true positive prediction than that of the fully supervised model.

Reliability is another question that is important to address, especially for weakly supervised models as one typically cannot test the segmentation performance in an actual application.

The observed reliability was better for the proposed model than the CAM_α alternative as the bulk of the distribution of the CAM_β models did in fact learn both classification and segmentation to some extent, while the bulk of the CAM_α models did not.

That being said, the standard deviations of all weakly supervised models tested here are telling.

We have observed very large spread of results on almost all metrics for both classification and segmentation in the setting of weak supervision. To our best judgment, this provides evidence that the weakly supervised methods investigated in this project have not been reliable as a whole with the small amount of data we have used. We would thus not advise the weakly supervised models tested in this project to be used for any real segmentation application without further investigation of how they perform with more data and/or testing with at least some fully supervised data.

8.4.1 Further Discussion

In our experiments using image level labels, the training data used is very small. Both in number of classes and number of samples. With 1572 images used for training, our data is almost an order of magnitude smaller than the PASCAL Visual Object Classes (VOC) data set of 2012 often used to benchmark weakly supervised segmentation or object localization models ([39, 70, 83, 84, 35]). Additionally, PASCAL VOC 2012 contains 21 classes including the background, while we have used 2. Thus the information provided for each sample is markedly weaker. This could partially be one of the reasons why we see a much worse IoU performance using CAM_α with our data then what has been reported when the same model has been applied on PASCAL VOC 2012 [19] as well as what has been qualitatively observed when trained on ImageNet [85].

8.5 Reflection and Future Directions

In this project we have experimented with both fully and weakly supervised semantic segmentation. We have presented a new model and tested it. The tests we have performed have been done in an attempt to steer the direction

of development towards solving problems that could help a lot of people, thus we find it valuable to experiment with a data set with real CT images of people with real liver problems. However, the drawback of this choice is that our contributions are, at this stage, not easy to rank among other modern models with the exception of the once we have included in our tests.

Based on our experiments, ConvMixerSeg is expected to lead to improvements beyond the liver segmentation task. Albeit we cannot know this less a more thorough study performed on additional datasets. A natural progression of this project would therefore be to test the proposed methods using both fully and weakly supervised segmentation on traditional benchmark data sets such as PASCAL VOC 2012 [25] and Cityscapes [21] to more easily compare with other models. If yielding convincing results the model could be made publically available and in that sense become a tool to help users potentially produce useful results more easily to solve problems.

Testing the model more extensively can provide information about its performance, but will more importantly lead to more elaborate understanding of its limitations which can be targeted for further improvements.

In the weakly supervised setting, we have already seen that part domination, although reduced, still is a significant source of error in the proposed method. And while we have showed that the Gaussian smoothing did improve the problem, we have not yet experimented with adjustments of the σ-parameter which would adjust the size of the neighborhoods in the Neighborhood Correlation Enforcement. The experiments of Kolesnikov et al. [40] indicated object size estimates on the PASCAL VOC 2012 would typically be over estimated when using GAP while being underestimated when using GMP. Although their method was somewhat different to ours, the size estimation is based on similar principles and should to our best understanding also apply here (note however, this should be tested). Notice that a very small value of σ would correspond to using GMP while a very large one would correspond to using GAP during training (although the smoothing would also affect the segmentation maps in a way it would not using GAP). By carefully adjusting the σ parameter, one could potentially further improve on the problem of part domination in a controlled manner. By recording precision and recall, the adjustment could be carefully increased until a quantitative balance was reached between precision and recall or qualitatively until the smoothing became too much of an issue.

Assuming the smoothing would cause problems, another developmental proposal would be to use the Gaussian smoothing during training, but remove it during testing as the advantages of more distributed gradients would be kept during training to help with noisy predictions and part domination, while the details in predictions would be preserved during testing. This could allow for

free adjustment of the σ parameter for size adjustment of object predictions while not affecting the level of detail in the predictions for testing. We also encourage others to explore this.

9

Conclusion

In this project we have presented a novel semantic segmentation model called ConvMixerSeg developed for liver segmentation. The model has been explained and extensively tested in comparison to an FCN-ResNet-50 model. We have shown that the proposed model outperforms the contender with high statistical significance on a small subset of the Liver Tumor Segmentation Benchmark data set and thereby prove its potential.

Moreover, we have developed an effective method for training semantic segmentation models with image level labels without adding parameters, allowing for modularity and therefore improved experiment informativeness in comparative studies. We have then trained the proposed segmentation model with only image level labels using the proposed CAM_β method and compared it to the original CAM method [85]. The results show that the proposed method outperforms the original CAM on the data set used.

The presented CAM_β method has finally been tested along side with several ablated alternatives to find out which parts of our design contributed with a significant effect to the performance. The results support our hypotheses that the Gaussian smoothing reduces prediction noise and mitigates part domination during testing and thus significantly improves overall performance.

Bibliography

[1] Kumar Abhishek and Ghassan Hamarneh. Matthews Correlation Coefficient Loss For Deep Convolutional Networks: Application To Skin Lesion Segmentation. In *2021 IEEE 18th International Symposium on Biomedical Imaging (ISBI)*, pages 225–229, 2021. doi: 10.1109/ISBI48211.2021.9433782.

[2] R. Adams and L. Bischof. Seeded region growing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(6):641–647, June 1994. ISSN 1939-3539. doi: 10.1109/34.295913. Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence.

[3] Jiwoon Ahn, Sunghyun Cho, and Suha Kwak. Weakly Supervised Learning of Instance Segmentation With Inter-Pixel Relations. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2204–2213, Long Beach, CA, USA, June 2019. IEEE. ISBN 978-1-72813-293-8. doi: 10.1109/CVPR.2019.00231. URL https://ieeexplore.ieee.org/document/8953768/.

[4] Valentin Anklin, Pushpak Pati, Guillaume Jaume, Behzad Bozorgtabar, Antonio Foncubierta-Rodríguez, Jean-Philippe Thiran, Mathilde Sibony, Maria Gabrani, and Orcun Goksel. Learning Whole-Slide Segmentation from Inexact and Incomplete Labels using Tissue Graphs. *arXiv:2103.03129 [cs]*, March 2021. URL http://arxiv.org/abs/2103.03129. arXiv: 2103.03129.

[5] Anonymous. Patches Are All You Need? In *Submitted to The Tenth International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=TVHS5Y4dNvM.

[6] Wonho Bae, Junhyug Noh, and Gunhee Kim. Rethinking Class Activation Mapping for Weakly Supervised Object Localization. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision – ECCV 2020*, Lecture Notes in Computer Science, pages 618–634, Cham, 2020. Springer International Publishing. ISBN 978-3-030-58555-6. doi: 10.1007/978-3-030-58555-6_37.

[7] Loris Bazzani, Alessandra Bergamo, Dragomir Anguelov, and Lorenzo Torresani. Self-taught object localization with deep networks. In *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1–9, 2016. doi: 10.1109/WACV.2016.7477688.

[8] Amy Bearman, Olga Russakovsky, Vittorio Ferrari, and Li Fei-Fei. *What's the Point: Semantic Segmentation with Point Supervision*. 2016. _eprint: 1506.02106.

[9] Hakan Bilen and Andrea Vedaldi. Weakly Supervised Deep Detection Networks. pages 2846–2854, 2016. URL https://www.cv-foundation.org/openaccess/content_cvpr_2016/html/Bilen_Weakly_Supervised_Deep_CVPR_2016_paper.html.

[10] Hakan Bilen, Marco Pedersoli, and Tinne Tuytelaars. Weakly Supervised Object Detection With Convex Clustering. pages 1081–1089, 2015. URL https://openaccess.thecvf.com/content_cvpr_2015/html/Bilen_Weakly_Supervised_Object_2015_CVPR_paper.html.

[11] Patrick Bilic, Patrick Ferdinand Christ, Eugene Vorontsov, Grzegorz Chlebus, Hao Chen, Qi Dou, Chi-Wing Fu, Xiao Han, Pheng-Ann Heng, Jürgen Hesser, Samuel Kadoury, Tomasz Konopczynski, Miao Le, Chunming Li, Xiaomeng Li, Jana Lipkovà, John Lowengrub, Hans Meine, Jan Hendrik Moltz, Chris Pal, Marie Piraud, Xiaojuan Qi, Jin Qi, Markus Rempfler, Karsten Roth, Andrea Schenk, Anjany Sekuboyina, Eugene Vorontsov, Ping Zhou, Christian Hülsemeyer, Marcel Beetz, Florian Ettlinger, Felix Gruen, Georgios Kaissis, Fabian Lohöfer, Rickmer Braren, Julian Holch, Felix Hofmann, Wieland Sommer, Volker Heinemann, Colin Jacobs, Gabriel Efrain Humpire Mamani, Bram van Ginneken, Gabriel Chartrand, An Tang, Michal Drozdzal, Avi Ben-Cohen, Eyal Klang, Marianne M. Amitai, Eli Konen, Hayit Greenspan, Johan Moreau, Alexandre Hostettler, Luc Soler, Refael Vivanti, Adi Szeskin, Naama Lev-Cohain, Jacob Sosna, Leo Joskowicz, and Bjoern H. Menze. The Liver Tumor Segmentation Benchmark (LiTS). *arXiv:1901.04056 [cs]*, January 2019. URL http://arxiv.org/abs/1901.04056. arXiv: 1901.04056.

[12] Lyndon Chan, Mahdi S. Hosseini, and Konstantinos N. Plataniotis. A Comprehensive Analysis of Weakly-Supervised Semantic Segmentation in Different Image Domains. *International Journal of Computer Vision*, 129(2): 361–384, February 2021. ISSN 0920-5691, 1573-1405. doi: 10.1007/s11263-020-01373-4. URL http://arxiv.org/abs/1912.11186. arXiv: 1912.11186.

[13] Jie Chen, Fen He, Yi Zhang, Geng Sun, and Min Deng. SPMF-Net: Weakly Supervised Building Segmentation by Combining Superpixel Pooling and

Multi-Scale Feature Fusion. *Remote Sensing*, 12(6):1049, January 2020. doi: 10.3390/rs12061049. URL https://www.mdpi.com/2072-4292/12/6/1049. Number: 6 Publisher: Multidisciplinary Digital Publishing Institute.

[14] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs. *arXiv:1606.00915 [cs]*, May 2017. URL http://arxiv.org/abs/1606.00915. arXiv: 1606.00915.

[15] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking Atrous Convolution for Semantic Image Segmentation. *arXiv:1706.05587 [cs]*, December 2017. URL http://arxiv.org/abs/1706.05587. arXiv: 1706.05587.

[16] Davide Chicco and Giuseppe Jurman. The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation. *BMC Genomics*, 21(1):6, January 2020. ISSN 1471-2164. doi: 10.1186/s12864-019-6413-7. URL https://doi.org/10.1186/s12864-019-6413-7.

[17] Davide Chicco, Niklas Tötsch, and Giuseppe Jurman. The Matthews correlation coefficient (MCC) is more reliable than balanced accuracy, bookmaker informedness, and markedness in two-class confusion matrix evaluation. *BioData Mining*, 14(1):13, February 2021. ISSN 1756-0381. doi: 10.1186/s13040-021-00244-z. URL https://doi.org/10.1186/s13040-021-00244-z.

[18] Junsuk Choe and Hyunjung Shim. Attention-Based Dropout Layer for Weakly Supervised Object Localization. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2214–2223, Long Beach, CA, USA, June 2019. IEEE. ISBN 978-1-72813-293-8. doi: 10.1109/CVPR.2019.00232. URL https://ieeexplore.ieee.org/document/8954302/.

[19] Junsuk Choe, Seong Joon Oh, Seungho Lee, Sanghyuk Chun, Zeynep Akata, and Hyunjung Shim. Evaluating Weakly Supervised Object Localization Methods Right. *arXiv:2001.07437 [cs]*, April 2020. URL http://arxiv.org/abs/2001.07437. arXiv: 2001.07437.

[20] Patrick Ferdinand Christ. LiTS - Liver Tumor Segmentation Challenge, June 2017. URL https://competitions.codalab.org/competitions/17094#results.

[21] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The Cityscapes Dataset for Semantic Urban Scene Understanding. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[22] Jifeng Dai, Kaiming He, and Jian Sun. BoxSup: Exploiting Bounding Boxes to Supervise Convolutional Networks for Semantic Segmentation. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1635–1643, Santiago, Chile, December 2015. IEEE. ISBN 978-1-4673-8391-2. doi: 10.1109/ICCV.2015.191. URL http://ieeexplore.ieee.org/document/7410548/.

[23] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-Training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-HLT*, 2019. URL https://www.aclweb.org/anthology/N19-1423.

[24] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image Is Worth 16x16 Words: Transformers for Image Recognition at Scale. In *ICLR*, 2021. URL https://openreview.net/forum?id=YicbFdNTTy.

[25] Mark Everingham and John Winn. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Development Kit. page 32. URL http://host.robots.ox.ac.uk/pascal/VOC/voc2012/index.html#devkit.

[26] Mark Everingham, S. M. Ali Eslami, Luc Van Gool, Christopher K. I. Williams, John Winn, and Andrew Zisserman. The Pascal Visual Object Classes Challenge: A Retrospective. *International Journal of Computer Vision*, 111(1):98–136, January 2015. ISSN 1573-1405. doi: 10.1007/s11263-014-0733-5. URL https://link.springer.com/article/10.1007/s11263-014-0733-5. Company: Springer Distributor: Springer Institution: Springer Label: Springer Number: 1 Publisher: Springer US.

[27] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. page 8, 2010.

[28] Rafael C. Gonzalez and Richard E. Woods. *Digital Image Processing, Forth Edition*. Pearson, England, 4 edition, 2018. ISBN 978-1-292-22304-9. URL https://www.books-by-isbn.com/1-292/1292223049-Digital-Image-Processing-Global-Edition-1-292-22304-9.html.

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

[30] J. Gorodkin. Comparing two K-category assignments by a K-category correlation coefficient. *Computational Biology and Chemistry*, 28(5):367–374, December 2004. ISSN 1476-9271. doi: 10.1016/j.compbiolchem.2004.09.006. URL https://www.sciencedirect.com/science/article/pii/S1476927104000799.

[31] Robert M Gower. Convergence Theorems for Gradient Descent. page 12.

[32] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. *arXiv:1406.4729 [cs]*, 8691:346–361, 2014. doi: 10.1007/978-3-319-10578-9_23. URL http://arxiv.org/abs/1406.4729. arXiv: 1406.4729.

[33] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. pages 770–778, 2016. URL http://openaccess.thecvf.com/content_cvpr_2016/html/He_Deep_Residual_Learning_CVPR_2016_paper.html.

[34] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely Connected Convolutional Networks. *arXiv:1608.06993 [cs]*, January 2018. URL http://arxiv.org/abs/1608.06993. arXiv: 1608.06993.

[35] Zilong Huang, Xinggang Wang, Jiasi Wang, Wenyu Liu, and Jingdong Wang. Weakly-Supervised Semantic Segmentation Network with Deep Seeded Region Growing. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 7014–7023, Salt Lake City, UT, USA, June 2018. IEEE. ISBN 978-1-5386-6420-9. doi: 10.1109/CVPR.2018.00733. URL https://ieeexplore.ieee.org/document/8578831/.

[36] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv:1502.03167 [cs]*, March 2015. URL http://arxiv.org/abs/1502.03167. arXiv: 1502.03167.

[37] Shruti Jadon. A survey of loss functions for semantic segmentation. *2020 IEEE Conference on Computational Intelligence in Bioinformatics and Computational Biology (CIBCB)*, pages 1–7, October 2020. doi: 10.1109/CIBCB48159.2020.9277638. URL http://arxiv.org/abs/2006.14822. arXiv: 2006.14822.

[38] Joel Janai, Fatma Güney, Aseem Behl, and Andreas Geiger. Computer Vision for Autonomous Vehicles: Problems, Datasets and State of the Art. *arXiv:1704.05519 [cs]*, December 2019. URL http://arxiv.org/abs/1704.05519. arXiv: 1704.05519.

[39] Sanghyun Jo and In-Jae Yu. Puzzle-CAM: Improved localization via matching partial and full features. *arXiv:2101.11253 [cs]*, February 2021. URL http://arxiv.org/abs/2101.11253. arXiv: 2101.11253 version: 3.

[40] Alexander Kolesnikov and Christoph H. Lampert. Seed, Expand and Constrain: Three Principles for Weakly-Supervised Image Segmentation. *arXiv:1603.06098 [cs]*, August 2016. URL http://arxiv.org/abs/1603.06098. arXiv: 1603.06098.

[41] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems*, volume 25, pages 1097–1105. Curran Associates, Inc., 2012. URL https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf.

[42] Suha Kwak, Seunghoon Hong, and Bohyung Han. Weakly Supervised Semantic Segmentation Using Superpixel Pooling Network. *AAAI*, 1(2017): 7, 2017.

[43] Jungbeom Lee, Eunji Kim, Sungmin Lee, Jangho Lee, and Sungroh Yoon. FickleNet: Weakly and Semi-Supervised Semantic Image Segmentation Using Stochastic Inference. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5262–5271, Long Beach, CA, USA, June 2019. IEEE. ISBN 978-1-72813-293-8. doi: 10.1109/CVPR.2019.00541. URL https://ieeexplore.ieee.org/document/8953687/.

[44] Alexander LeNail. NN-SVG: Publication-Ready Neural Network Architecture Schematics. *Journal of Open Source Software*, 4(33):747, 2019. doi: 10.21105/joss.00747. URL https://doi.org/10.21105/joss.00747. Publisher: The Open Journal.

[45] Di Lin, Jifeng Dai, Jiaya Jia, Kaiming He, and Jian Sun. ScribbleSup: Scribble-Supervised Convolutional Networks for Semantic Segmentation. *arXiv:1604.05144 [cs]*, 2012. URL http://arxiv.org/abs/1604.05144. arXiv: 1604.05144.

[46] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO:

Common Objects in Context. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 740–755, Cham, 2014. Springer International Publishing. ISBN 978-3-319-10602-1.

[47] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully Convolutional Networks for Semantic Segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[48] Tianle Ma and Aidong Zhang. AffinityNet: semi-supervised few-shot learning for disease type prediction. *arXiv:1805.08905 [cs, stat]*, September 2018. URL http://arxiv.org/abs/1805.08905. arXiv: 1805.08905.

[49] Jinjie Mai, Meng Yang, and Wenfeng Luo. Erasing Integrated Learning: A Simple Yet Effective Approach for Weakly Supervised Object Localization. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 8763–8772, Seattle, WA, USA, June 2020. IEEE. ISBN 978-1-72817-168-5. doi: 10.1109/CVPR42600.2020.00879. URL https://ieeexplore.ieee.org/document/9156910/.

[50] Arsha Nagrani, Shan Yang, Anurag Arnab, Aren Jansen, Cordelia Schmid, and Chen Sun. Attention Bottlenecks for Multimodal Fusion. *Advances in Neural Information Processing Systems*, 34, June 2021. URL http://arxiv.org/abs/2107.00135. arXiv: 2107.00135 version: 1.

[51] Lloyd S Nelson. The Anderson-Darling test for normality. *Journal of Quality Technology*, 30(3):298–299, July 1998. ISSN 00224065. URL https://www.proquest.com/scholarly-journals/anderson-darling-test-normality/docview/214483507/se-2?accountid=17260. Place: Milwaukee Publisher: Taylor & Francis Ltd.

[52] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature Visualization. *Distill*, 2017. doi: 10.23915/distill.00007.

[53] Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic. Is Object Localization for Free? - Weakly-Supervised Learning With Convolutional Neural Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 685–694, 2015. URL https://openaccess.thecvf.com/content_cvpr_2015/html/Oquab_Is_Object_Localization_2015_CVPR_paper.html.

[54] A. Emin Orhan and Xaq Pitkow. Skip Connections Eliminate Singularities. *arXiv:1701.09175 [cs]*, March 2018. URL http://arxiv.org/abs/1701.09175. arXiv: 1701.09175.

[55] George Papandreou, Liang-Chieh Chen, Kevin P. Murphy, and Alan L. Yuille. Weakly-and Semi-Supervised Learning of a Deep Convolutional Network for Semantic Image Segmentation. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1742–1750, Santiago, Chile, December 2015. IEEE. ISBN 978-1-4673-8391-2. doi: 10.1109/ICCV.2015.203. URL http://ieeexplore.ieee.org/document/7410560/.

[56] Deepak Pathak, Philipp Krahenbuhl, and Trevor Darrell. Constrained Convolutional Neural Networks for Weakly Supervised Segmentation. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1796–1804, Santiago, Chile, December 2015. IEEE. ISBN 978-1-4673-8391-2. doi: 10.1109/ICCV.2015.209. URL http://ieeexplore.ieee.org/document/7410566/.

[57] Deepak Pathak, Evan Shelhamer, Jonathan Long, and Trevor Darrell. Fully Convolutional Multi-Class Multiple Instance Learning. *arXiv:1412.7144 [cs]*, April 2015. URL http://arxiv.org/abs/1412.7144. arXiv: 1412.7144.

[58] Pedro O. Pinheiro and Ronan Collobert. From Image-level to Pixel-level Labeling with Convolutional Networks. *arXiv:1411.6228 [cs]*, April 2015. URL http://arxiv.org/abs/1411.6228. arXiv: 1411.6228.

[59] Steven Pinker. *Enlightenment now: The case for reason, science, humanism, and progress*. Penguin Books, Harlow, England, 2019. ISBN 978-0-14-197909-0.

[60] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. *arXiv:1505.04597 [cs]*, May 2015. URL http://arxiv.org/abs/1505.04597. arXiv: 1505.04597.

[61] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386, May 1959. ISSN 1939-1471. doi: 10.1037/h0042519. URL https://psycnet.apa.org/fulltext/1959-09865-001.pdf. Publisher: US: American Psychological Association.

[62] S. K. Roy, M. E. Paoletti, J. M. Haut, S. R. Dubey, P. Kar, A. Plaza, and B. B. Chaudhuri. AngularGrad: A New Optimization Technique for Angular Convergence of Convolutional Neural Networks. *arXiv:2105.10190 [cs, stat]*, May 2021. URL http://arxiv.org/abs/2105.10190. arXiv: 2105.10190.

[63] Sebastian Ruder. An overview of gradient descent optimization algo-

rithms. *arXiv:1609.04747 [cs]*, June 2017. URL http://arxiv.org/abs/1609.04747. arXiv: 1609.04747.

[64] Seyed Sadegh Mohseni Salehi, Deniz Erdogmus, and Ali Gholipour. Tversky loss function for image segmentation using 3D fully convolutional deep networks. *arXiv:1706.05721 [cs]*, June 2017. URL http://arxiv.org/abs/1706.05721. arXiv: 1706.05721.

[65] Shibani Santurkar, Dimitris Tsipras, Andrew Ilyas, and Aleksander Madry. How Does Batch Normalization Help Optimization? *arXiv:1805.11604 [cs, stat]*, April 2019. URL http://arxiv.org/abs/1805.11604. arXiv: 1805.11604.

[66] Ramprasaath R. Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-CAM: Visual Explanations from Deep Networks via Gradient-based Localization. *International Journal of Computer Vision*, 128(2):336–359, February 2020. ISSN 0920-5691, 1573-1405. doi: 10.1007/s11263-019-01228-7. URL http://arxiv.org/abs/1610.02391. arXiv: 1610.02391.

[67] Miaojing Shi and Vittorio Ferrari. Weakly Supervised Object Localization Using Size Estimates. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, volume 9909, pages 105–121. Springer International Publishing, Cham, 2016. ISBN 978-3-319-46453-4 978-3-319-46454-1. doi: 10.1007/978-3-319-46454-1_7. URL http://link.springer.com/10.1007/978-3-319-46454-1_7. Series Title: Lecture Notes in Computer Science.

[68] Karen Simonyan and Andrew Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv:1409.1556 [cs]*, April 2015. URL http://arxiv.org/abs/1409.1556. arXiv: 1409.1556.

[69] Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman. Deep Inside Convolutional Networks: Visualising Image Classification Models and Saliency Maps. *arXiv:1312.6034 [cs]*, April 2014. URL http://arxiv.org/abs/1312.6034. arXiv: 1312.6034.

[70] Krishna Kumar Singh and Yong Jae Lee. Hide-and-Seek: Forcing a Network to be Meticulous for Weakly-supervised Object and Action Localization. *arXiv:1704.04232 [cs]*, December 2017. URL http://arxiv.org/abs/1704.04232. arXiv: 1704.04232.

[71] Toufique A. Soomro, Ahmed J. Afifi, Junbin Gao, Olaf Hellwich, Manoranjan Paul, and Lihong Zheng. Strided U-Net Model: Retinal Vessels Segmen-

tation using Dice Loss. In *2018 Digital Image Computing: Techniques and Applications (DICTA)*, pages 1–8, 2018. doi: 10.1109/DICTA.2018.8615770.

[72] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going Deeper With Convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015. URL https://www.cv-foundation.org/openaccess/content_cvpr_2015/html/Szegedy_Going_Deeper_With_2015_CVPR_paper.html.

[73] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. Rethinking the Inception Architecture for Computer Vision. *arXiv:1512.00567 [cs]*, December 2015. URL http://arxiv.org/abs/1512.00567. arXiv: 1512.00567.

[74] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-first AAAI conference on artificial intelligence*, 2017.

[75] Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. *arXiv:2012.12877 [cs]*, January 2021. URL http://arxiv.org/abs/2012.12877. arXiv: 2012.12877.

[76] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. *arXiv:1706.03762 [cs]*, December 2017. URL http://arxiv.org/abs/1706.03762. arXiv: 1706.03762.

[77] WORLD HEALTH ORGANIZATION: REGIONAL OFFICE FOR EUROPE. *WORLD CANCER REPORT: cancer research for cancer development.* IARC, Place of publication not identified, 2020. ISBN 978-92-832-0447-3. OCLC: 1145902769.

[78] Zifeng Wu, Chunhua Shen, and Anton van den Hengel. High-performance Semantic Segmentation Using Very Deep Fully Convolutional Networks. *arXiv:1604.04339 [cs]*, April 2016. URL http://arxiv.org/abs/1604.04339. arXiv: 1604.04339.

[79] Haolan Xue, Chang Liu, Fang Wan, Jianbin Jiao, Xiangyang Ji, and Qixiang Ye. DANet: Divergent Activation for Weakly Supervised Object Localization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2019.

[80] Yuhui Yuan, Lang Huang, Jianyuan Guo, Chao Zhang, Xilin Chen, and Jingdong Wang. OCNet: Object Context Network for Scene Parsing. *arXiv:1809.00916 [cs]*, March 2021. URL http://arxiv.org/abs/1809.00916. arXiv: 1809.00916.

[81] Matthew D. Zeiler and Rob Fergus. Visualizing and Understanding Convolutional Networks. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, volume 8689, pages 818–833. Springer International Publishing, Cham, 2014. ISBN 978-3-319-10589-5 978-3-319-10590-1. doi: 10.1007/978-3-319-10590-1_53. URL http://link.springer.com/10.1007/978-3-319-10590-1_53.

[82] Yu Zeng, Yunzhi Zhuge, Huchuan Lu, and Lihe Zhang. Joint Learning of Saliency Detection and Weakly Supervised Semantic Segmentation. *arXiv:1909.04161 [cs]*, September 2019. URL http://arxiv.org/abs/1909.04161. arXiv: 1909.04161.

[83] Xiaolin Zhang, Yunchao Wei, Jiashi Feng, Yi Yang, and Thomas Huang. Adversarial Complementary Learning for Weakly Supervised Object Localization. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1325–1334, Salt Lake City, UT, USA, June 2018. IEEE. ISBN 978-1-5386-6420-9. doi: 10.1109/CVPR.2018.00144. URL https://ieeexplore.ieee.org/document/8578242/.

[84] Xiaolin Zhang, Yunchao Wei, Guoliang Kang, Yi Yang, and Thomas Huang. Self-produced Guidance for Weakly-Supervised Object Localization. In Vittorio Ferrari, Martial Hebert, Cristian Sminchisescu, and Yair Weiss, editors, *Computer Vision – ECCV 2018*, volume 11216, pages 610–625. Springer International Publishing, Cham, 2018. ISBN 978-3-030-01257-1 978-3-030-01258-8. doi: 10.1007/978-3-030-01258-8_37. URL http://link.springer.com/10.1007/978-3-030-01258-8_37. Series Title: Lecture Notes in Computer Science.

[85] Bolei Zhou, Aditya Khosla, Agata Lapedriza, Aude Oliva, and Antonio Torralba. Learning Deep Features for Discriminative Localization. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2921–2929, Las Vegas, NV, USA, June 2016. IEEE. ISBN 978-1-4673-8851-1. doi: 10.1109/CVPR.2016.319. URL http://ieeexplore.ieee.org/document/7780688/.

[86] Zhi-Hua Zhou. A brief introduction to weakly supervised learning. *National Science Review*, 5(2018):44–53, August 2017. URL https://academic.oup.com/nsr/article/5/1/44/4093912.

10 Appendix

10.1 All Results

Table 10.1: Measured performance of 52 ConvMixerSeg models trained independently with CE-loss for 25 epochs. Each element is given on the format: $mean \pm std(median)$.

	Training	Validation	Testing
Acc	0.981 ± 0.098(0.995)	0.897 ± 0.223(0.975)	0.893 ± 0.233(0.983)
MCC	0.906 ± 0.126(0.927)	0.669 ± 0.251(0.756)	0.682 ± 0.27(0.782)
IoU	0.842 ± 0.118(0.864)	0.537 ± 0.236(0.593)	0.562 ± 0.255(0.644)
Prec	0.845 ± 0.118(0.867)	0.556 ± 0.247(0.61)	0.582 ± 0.266(0.694)
Rec	0.992 ± 0.03(0.996)	0.939 ± 0.043(0.953)	0.94 ± 0.049(0.959)

Table 10.2: Measured performance of 49 FCN-ResNet-50 models trained independently with CE-loss for 25 epochs. Each element is given on the format: $mean \pm std(median)$.

	Training	Validation	Testing
Acc	0.962 ± 0.01(0.96)	0.936 ± 0.129(0.964)	0.939 ± 0.129(0.965)
MCC	0.656 ± 0.072(0.649)	0.637 ± 0.147(0.677)	0.655 ± 0.146(0.686)
IoU	0.464 ± 0.094(0.45)	0.459 ± 0.149(0.49)	0.48 ± 0.152(0.504)
Prec	0.472 ± 0.094(0.458)	0.492 ± 0.173(0.504)	0.508 ± 0.174(0.515)
Rec	0.96 ± 0.013(0.959)	0.923 ± 0.123(0.962)	0.934 ± 0.098(0.966)

Table 10.3: Measured classification performance the ConvMixerSeg model trained with the CAM_{α} method for 10 epochs[1]. Each element is given on the format: $mean \pm std(median)$.

	Training	Validation	Testing
Cls-Acc	0.732 ± 0.025(0.742)	0.739 ± 0.026(0.75)	0.74 ± 0.022(0.75)
Cls-MCC	0.557 ± 0.027(0.568)	0.565 ± 0.027(0.577)	0.566 ± 0.024(0.577)
Cls-IoU	0.643 ± 0.033(0.656)	0.652 ± 0.034(0.667)	0.653 ± 0.03(0.667)
Cls-Prec	1.0 ± 0.0(1.0)	1.0 ± 0.0(1.0)	1.0 ± 0.0(1.0)
Cls-Rec	0.643 ± 0.033(0.656)	0.652 ± 0.034(0.667)	0.653 ± 0.03(0.667)

Table 10.4: Measured segmentation performance of the ConvMixerSeg model trained with the CAM_{α} method for 10 epochs. Each element is given on the format: $mean \pm std(median)$.

	Training	Validation	Testing
Acc	0.32 ± 0.079(0.308)	0.394 ± 0.145(0.363)	0.406 ± 0.147(0.378)
MCC	0.132 ± 0.109(0.122)	0.119 ± 0.119(0.132)	0.107 ± 0.126(0.101)
IoU	0.017 ± 0.013(0.017)	0.018 ± 0.017(0.015)	0.02 ± 0.018(0.018)
Prec	0.018 ± 0.013(0.018)	0.019 ± 0.017(0.016)	0.022 ± 0.018(0.021)
Rec	0.345 ± 0.241(0.331)	0.3 ± 0.274(0.272)	0.32 ± 0.281(0.319)

Table 10.5: Measured classification performance of the ConvMixerSeg model trained with the CAM_{β} method for 3 epochs. Each element is given on the format: $mean \pm std(median)$.

	Training	Validation	Testing
Cls-Acc	0.913 ± 0.039(0.92)	0.885 ± 0.08(0.93)	0.888 ± 0.084(0.936)
Cls-MCC	0.795 ± 0.06(0.805)	0.7 ± 0.271(0.828)	0.707 ± 0.279(0.842)
Cls-IoU	0.887 ± 0.052(0.896)	0.858 ± 0.094(0.908)	0.861 ± 0.099(0.915)
Cls-Prec	0.975 ± 0.013(0.976)	0.954 ± 0.078(0.983)	0.957 ± 0.079(0.988)
Cls-Rec	0.909 ± 0.056(0.921)	0.902 ± 0.1(0.936)	0.903 ± 0.103(0.936)

Table 10.6: Measured segmentation performance of the ConvMixerSeg model trained with the CAM_{β} method for 3 epochs. Each element is given on the format: $mean \pm std(median)$.

	Training	Validation	Testing
Acc	0.971 ± 0.004(0.972)	0.932 ± 0.13(0.967)	0.93 ± 0.143(0.968)
MCC	0.375 ± 0.169(0.409)	0.279 ± 0.222(0.365)	0.301 ± 0.234(0.396)
IoU	0.212 ± 0.105(0.218)	0.161 ± 0.136(0.171)	0.178 ± 0.148(0.203)
Prec	0.684 ± 0.238(0.747)	0.561 ± 0.355(0.719)	0.59 ± 0.352(0.765)
Rec	0.23 ± 0.114(0.24)	0.213 ± 0.184(0.209)	0.239 ± 0.202(0.223)

Table 10.7: Measured classification performance of the Baseline-Res model trained for 3 epochs. Each element is given on the format: $mean \pm std(median)$.

	Training	Validation	Testing
Cls-Acc	0.774 ± 0.048(0.751)	0.788 ± 0.07(0.75)	0.787 ± 0.07(0.75)
Cls-MCC	0.238 ± 0.26(0.14)	0.253 ± 0.344(0.0)	0.249 ± 0.343(0.0)
Cls-IoU	0.757 ± 0.048(0.751)	0.773 ± 0.072(0.75)	0.772 ± 0.072(0.75)
Cls-Prec	0.812 ± 0.082(0.764)	0.825 ± 0.106(0.75)	0.824 ± 0.105(0.75)
Cls-Rec	0.934 ± 0.092(0.976)	0.945 ± 0.107(1.0)	0.947 ± 0.106(1.0)

Table 10.8: Measured segmentation performance of the Baseline-Res model trained for 3 epochs. Each element is given on the format: $mean \pm std(median)$.

	Training	Validation	Testing
Acc	0.663 ± 0.243(0.731)	0.705 ± 0.261(0.79)	0.708 ± 0.258(0.788)
MCC	0.051 ± 0.126(0.027)	0.094 ± 0.183(0.049)	0.094 ± 0.175(0.038)
IoU	0.054 ± 0.067(0.037)	0.071 ± 0.109(0.042)	0.072 ± 0.106(0.041)
Prec	0.109 ± 0.177(0.04)	0.133 ± 0.202(0.051)	0.131 ± 0.202(0.051)
Rec	0.389 ± 0.269(0.376)	0.398 ± 0.348(0.418)	0.403 ± 0.349(0.345)

Table 10.9: Measured classification performance of the Baseline-GS model trained for 3 epochs. Each element is given on the format: $mean \pm std(median)$.

	Training	Validation	Testing
Cls-Acc	0.815 ± 0.061(0.808)	0.836 ± 0.078(0.862)	0.838 ± 0.08(0.871)
Cls-MCC	0.4 ± 0.288(0.41)	0.459 ± 0.368(0.638)	0.466 ± 0.37(0.682)
Cls-IoU	0.798 ± 0.052(0.769)	0.818 ± 0.074(0.824)	0.819 ± 0.076(0.831)
Cls-Prec	0.844 ± 0.091(0.809)	0.874 ± 0.106(0.899)	0.876 ± 0.105(0.91)
Cls-Rec	0.948 ± 0.061(0.974)	0.941 ± 0.083(0.976)	0.942 ± 0.085(0.979)

Table 10.10: Measured segmentation performance of the Baseline-GS model trained for 3 epochs. Each element is given on the format: $mean \pm std(median)$.

	Training	Validation	Testing
Acc	0.942 ± 0.024(0.946)	0.93 ± 0.073(0.962)	0.93 ± 0.076(0.961)
MCC	0.128 ± 0.129(0.079)	0.154 ± 0.175(0.067)	0.164 ± 0.179(0.091)
IoU	0.072 ± 0.079(0.034)	0.082 ± 0.106(0.029)	0.086 ± 0.11(0.03)
Prec	0.291 ± 0.272(0.202)	0.411 ± 0.325(0.341)	0.435 ± 0.335(0.396)
Rec	0.168 ± 0.215(0.043)	0.19 ± 0.275(0.053)	0.195 ± 0.279(0.057)

Table 10.11: Measured classification performance of the Baseline-GS+BlnrUp model trained for 3 epochs. Each element is given on the format: $mean \pm std(median)$.

	Training	Validation	Testing
Cls-Acc	0.892 ± 0.03(0.901)	0.873 ± 0.076(0.911)	0.878 ± 0.076(0.913)
Cls-MCC	0.719 ± 0.084(0.735)	0.651 ± 0.275(0.779)	0.66 ± 0.278(0.793)
Cls-IoU	0.865 ± 0.035(0.877)	0.848 ± 0.084(0.884)	0.853 ± 0.084(0.887)
Cls-Prec	0.937 ± 0.04(0.953)	0.929 ± 0.086(0.969)	0.93 ± 0.084(0.96)
Cls-Rec	0.921 ± 0.043(0.929)	0.916 ± 0.093(0.942)	0.921 ± 0.09(0.942)

Table 10.12: Measured segmentation performance of the Baseline-GS+BlnrUp model trained models for 3 epochs. Each element is given on the format: $mean \pm std(median)$.

	Training	Validation	Testing
Acc	0.966 ± 0.006(0.965)	0.949 ± 0.046(0.959)	0.95 ± 0.048(0.96)
MCC	0.251 ± 0.2(0.266)	0.192 ± 0.244(0.06)	0.202 ± 0.243(0.073)
IoU	0.154 ± 0.137(0.158)	0.127 ± 0.171(0.03)	0.133 ± 0.169(0.044)
Prec	0.391 ± 0.238(0.401)	0.286 ± 0.305(0.132)	0.308 ± 0.3(0.217)
Rec	0.196 ± 0.179(0.199)	0.195 ± 0.267(0.07)	0.203 ± 0.263(0.077)

Table 10.13: Measured classification performance of the Baseline-Aug setup trained for 3 epochs. Each element is given on the format: $mean \pm std(median)$.

	Training	Validation	Testing
Cls-Acc	0.913 ± 0.056(0.935)	0.898 ± 0.072(0.932)	0.9 ± 0.072(0.931)
Cls-MCC	0.795 ± 0.123(0.834)	0.748 ± 0.226(0.835)	0.751 ± 0.226(0.835)
Cls-IoU	0.888 ± 0.07(0.915)	0.871 ± 0.086(0.911)	0.873 ± 0.086(0.909)
Cls-Prec	0.974 ± 0.032(0.977)	0.97 ± 0.062(0.99)	0.97 ± 0.063(0.99)
Cls-Rec	0.911 ± 0.071(0.936)	0.9 ± 0.09(0.923)	0.902 ± 0.089(0.933)

Table 10.14: Measured segmentation performance of the Baseline-Aug model trained for 3 epochs. Each element is given on the format: $mean \pm std(median)$.

	Training	Validation	Testing
Acc	0.963 ± 0.039(0.969)	0.961 ± 0.024(0.964)	0.962 ± 0.027(0.966)
MCC	0.322 ± 0.178(0.365)	0.252 ± 0.186(0.212)	0.259 ± 0.183(0.254)
IoU	0.188 ± 0.116(0.197)	0.135 ± 0.124(0.085)	0.139 ± 0.119(0.116)
Prec	0.555 ± 0.224(0.621)	0.509 ± 0.27(0.535)	0.548 ± 0.269(0.61)
Rec	0.215 ± 0.132(0.234)	0.156 ± 0.144(0.118)	0.158 ± 0.135(0.139)