Faculty of Science and Technology, Department of Mathematics and Statistics

Deep learning applied to fish otolith images

—

Iver Martinsen

STA-3900 - Master's Thesis in Statistics

Abstract

Otoliths are calcified structures in the inner ears of fish, and are used extensively in discrimination between stocks and in age determination of fish. The analysis of otoliths is a time consuming task which requires education of human experts, which in turn provides otolith analysis that are often inconsistent. To resolve for the inconsistencies resulting from analysis by human experts, an automated procedure based on Convolutional neural networks (CNNs), a class of deep learning models efficient in image classification, is investigated. CNNs are applied on two separate problems in this thesis; discrimination between Northeast Arctic Cod (NEAC) and Norwegian Coastal Cod (NCC), and age estimation of Greenland halibuts. Using deep learning, we obtain classification results on cod otoliths that are comparable to previously published results, and we show that deep learning in combination with other classification methods results in a significant increase in classification accuracy. We also show that deep learning regression on Greenland halibut otolith images gives good results, and that the age distribution of predicted samples closely resembles the distribution provided by the labels. By using k*l-fold cross-validation we are able to test all available samples, and we show that test results are subject to large variations, and that cross-validation procedures should be conducted to verify the model performance. Finally, we apply interpretability techniques for analyzing the decision process of deep learning models, and we produce heatmaps of input images for identifying the input features that are the most important in the computation of the output.

Acknowledgements

I would like to thank my supervisor, Filippo Maria Bianchi at UiT, for his guidance in the writing of this thesis.

I would also like to thank my co-supervisor, Alf Harbitz at the Institute of Marine Research (IMR), for his guidance, and for providing me with interesting problems which created the foundation for this thesis. He also provided all the input data and otolith images, on behalf of the institute.

Iver Martinsen

Table of Contents

List of Figures

List of Tables

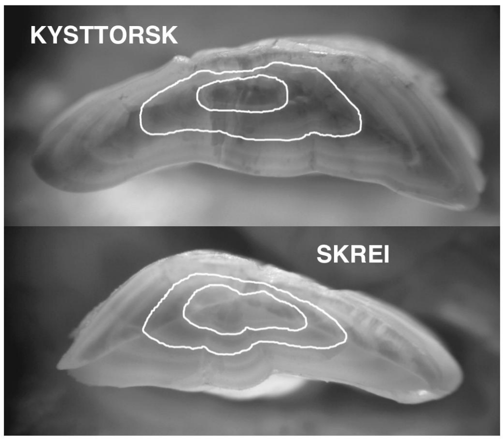
1. Introduction

The otolith is a calcified structure present in the inner ear of vertebrates. In fish, the otolith has several areas of application, two of which will be discussed here. They concern discrimination betwen cod stocks, and age determination of Greenland halibuts.

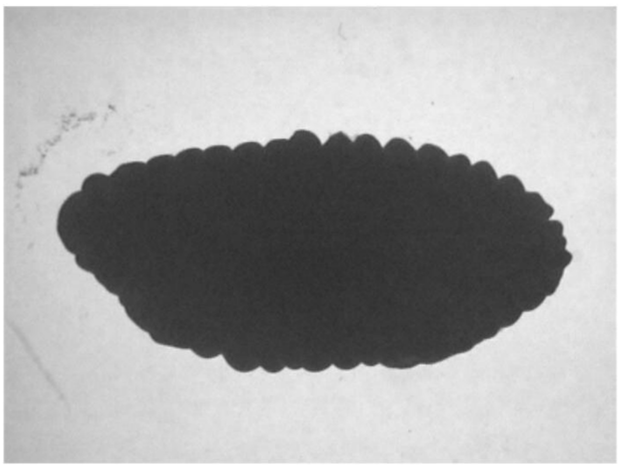
1.1. Classification of fish stocks based on otoliths

Atlantic cod (*Gadus morhua L.*) consists of several stocks that are assessed and managed as separate units (Berg et al., 2005). The Northeast Arctic Cod (NEAC) is the largest stock in the Northeast Arctic. They migrate over long distances, from their feeding areas in the Barents Sea, to spawning grounds off and along the Norwegian coast (Bergstad et al., 1987). The much less numerous Norwegian Coastal Cod (NCC) are found along the coast of Norway, where they spawn at numerous locations inside fjords, in addition to the same coastal areas used by NEAC (Jakobsen, 1987). During spawning, mixed catches are taking place which threatens the existence of the endangered NCC. Stock discrimination between NEAC and NCC are done by analyzing the otolith of the fish, which is used to determine the stock affiliations.

A cod otolith (earstone) is a 3-dimensional calcified structure, where the difference between the first two annual zones in a vertical cross section of the otolith (figure 1a) is interpreted by expert human readers to determine stock affiliation. This official stock classification method goes back to 1933 (Rollefsen, 1933). In the last decades, alternative discrimination methods have been developed, where Fourier analysis of the closed 2D contour of the whole otolith has been particularly successful. Stransky et al. (2008) obtained a classification accuracy of 89 % for NCC and 90 % for NEAC using Elliptical Fourier Descriptors (Kuhl & Giardina, 1982) applied to the contour of the whole otolith automatically segmented from otolith images (figure 1b).



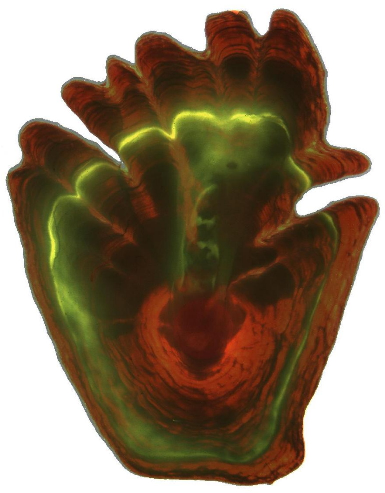
(a) *Cod otolith annual zones*


(b) *Cod otolith contour*

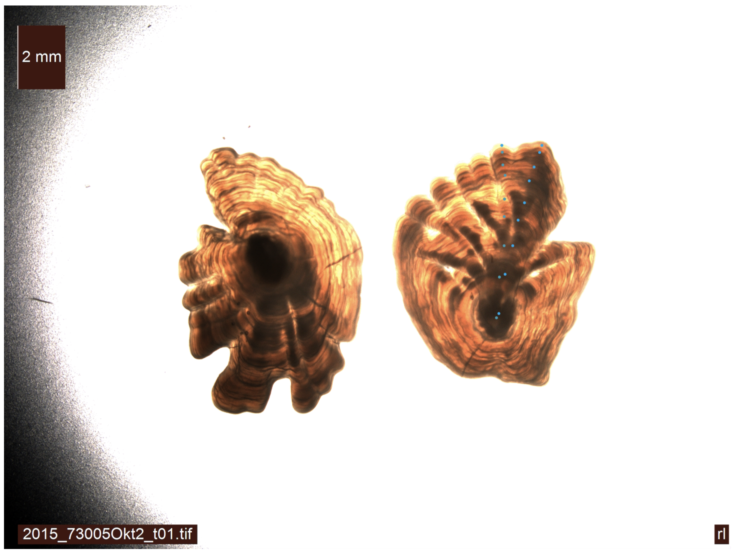
Besides from classification, there are other areas of applications where the analysis of otoliths is important, one of which is age determination.

1.2. Age prediction based on otoliths

A reliable estimate of the age distribution of a fish stock is important in stock assessment and stock development modeling, for example to detect recruitment to the stock by increased recruitment of young fish. The age of fish can be estimated by several methods, where the counting of annual zones in the otolith is the dominant one. This is done by direct investigation of the physical otolith, or by image analysis. For Greenland halibut, the age is read by analysis of the whole otolith images by translucent light (figure 2), or by image analysis of the cross section. In this thesis, images of the first category (translucent light) are analyzed. These otolith images have shown to be suitable for automatic age estimation using deep learning methods, which is shown bt Moen et al. (2018) and by Ordoñez et al. (2020). Moen et al. predicted 57 % of test samples to be at most one year off the read age, while Ordoñez et al. (2020) furthermore proved the validity of deep learning by analyzing pixel relevance to the output.



(a) *Greenland halibut growth revealed by the re-catch of a fish with injection marks*



(b) *Backlight through the transucent Greenland halibut otolith reveals otolith annual rings*

The core of the Greenland halibut otolith is approximately equal in size for all individuals, but the otolith itself grows as the fish grows older, also after the fish itself has stopped growing in length. Because of physical obstacles in the inner ear, the otolith growth will eventually by limited to the upwards direction with regard to figure 2a.

1.3. Deep learning and Convolutional Neural Networks

Artificial Neural Networks have proven to be effective in modeling complex relationships between high-dimensional inputs and labels in classification and regression problems. Deep learning models include multilayer neural networks, and provides a flexibility that enables model fitting using complex inputs, with images being an input of particular interest. The ImageNet project (Deng et al., 2009) has held an annual contest in image recognition since 2010, with the emphasis on image classification, object detection and object localization in images. Convolutional Neural Networks have developed as the benchmark framework in image classification, a position achieved by excellent performances on the ImageNet database. The GPU implementation of CNNs has resulted in a rapid development in the last decade, a development that is demonstrated by the evolution of CNN architectures with ever increasing performance on the ImageNet database. Architectures such as AlexNet (Krizhevsky et al., 2012), VGG-16 (Simonyan et al., 2014) and Inception (Szegedy et al., 2014) has proven the learning ability of CNNs.

1.4. Label uncertainty

The models in this thesis are fitted under the objective of minimizing the discrepancy between the model output and the provided labels. This supervised tale in which the labels are assumed to provide the ground truth, does not tell the whole story as the labels used in this thesis are estimated by human experts. There is uncertainty regarding the accuracy and the precision of the human readers, which concerns the difference between the empirical labels and the truth, and to what extent repeated readings by the same reader of the same otolith will give varying results. It is reasonable to assume that the precision in otolith age estimation will be worse with increasing age, but there is also a range of other possible sources of variations. Covariates that may influence readings includes image quality and seasonal differences among other things. There exists ways to estimate the reader uncertainty:

- Accuracy can be assessed by a catch-re-catch procedure where an injected material is applied to the fish, setting detectable marks on the otolith contour (figure 2a). When re-caught some years later, the true age difference between the mark on the otolith image and the outer contour is revealed.

- Precision can be assessed by comparing empirical labels given by several readers, as well as comparing repeated readings by the same reader. In cases where age estimation is performed by several readers, a typical coefficient of variation of about 12-14 % is reported, but this measure will inherently include bias between readers and is not directly comparable to the CV reported in this thesis.

The estimation of reader accuracy is generally a fundamental challenge for many species.

1.5. Motivation and problem statement

Reliable estimates of fish age distribution and stock affiliation are vital factors in the achievement of good management of wild fish resources. It turns out that determining those features is a non-trivial process, and that simple measures such as fish length and fish weight are insufficient for that task. It has been shown that for most fish species, counting of annual zones in otoliths is so far the most reliable method for ageing, and in several cases otolith analysis has proven to be useful in stock separation as well as in discrimination between fish species. The analysis of otoliths are done by human experts, a process which is time consuming and requires education and training. The analysis is particularly time consuming when otolith images are needed, which is the case for a range of species. On a global basis, about 1 million otoliths are read per year (Campana & Thorrold, 2001), so there is obviously a demand for more efficient ageing methods.

Machine learning methods, and deep learning in particular, have seen an enormous increase in popularity in the last decade due to an increase in computer power and an increased understanding of the models and optimization procedures themselves. Convolutional neural networks are especially suited for image interpretation, and make up a natural starting point for analyzing otolith images. Furthermore, analysis of deep learning models themselves are important in verifying the validity of the deep learning model. For input images, the visualization of pixel relevance is considered important in explaining model decisions. A good interpretation of which image features that trigger the deep learning decision is also a motivation, as this would help to communicate the validity of good results with the marine biology expertise, contrary to the report of results as scores from a "black box".

In this thesis we want to develop a method which is able to

1. discriminate between NEAC and NCC

2. estimate the age of Greenland halibuts

by using otolith images. We propose convolutional neural networks - a subclass of deep learning models - as a solution.

1.6. Thesis contribution

The contributions of this thesis are:

- Application of deep learning on fish otolith images for two different problems; discrimination between Northeast Arctic Cod and Norwegian Coastal Cod, and age prediction of Greenland halibuts.

- Evaluation of model performance in comparison to existing methods such as linear discrimination and linear regression.

- Evaluation of the uncertainty of the model by using exhaustive cross-validation procedures, thereby investigating the influence of test sets on the model performance.

- Application of different methods that aims at explaining decisions of deep learning models by highlighting important input features.

1.7. Paper structure

The rest of the paper is structured as follows:

- Section 2 covers background theory for deep learning methods and includes a brief review of approaches that have been devised for this problem. The section will provide necessary mathematical derivations, leaving the detailed mathematical explanations and examples to the appendix.

- Section 3 covers the dataset and includes a preliminary analysis of the data.

- Section 4 presents developed approaches that uses the methods of interest. The section will provide some propositions with regard to model choices, validation procedures and tools for visual analysis.

- Section 5 covers the experimental procedure and the results. Here we will go through a detailed discussion of the model fitting procedure itself, a discussion of the results obtained, and suggestions for further analysis.

- Lastly, final discussions and conclusions are given in Section 6. Auxiliary material will be left to the appendix.

All the deep learning code is written in Python using the Keras API (Chollet et al., 2015). The TensorFlow framework (Abadi et al., 2015) is used for model fitting, while R is used for analysis and figures. The source code and notebooks used in this thesis can be found at

`https://github.com/IverMartinsen/MastersThesis.git`

All data used in this thesis are provided by Alf Harbitz at the Institute of Marine Research (IMR).

2. Background Theory

This section covers briefly the general theory behind neural networks, in addition to fundamental concepts of feedforward networks in particular. We start off this section by stating some fundamental definitions and results. Subsection 2.2 and 2.3 provide an introduction to layers and operations used in deep learning models such as convolutional neural networks, while regularization methods are discussed in subsection 2.4. Validation methods, along with explanatory tools useful for visualizing feature relevance are introduced and explained in the final two subsections.

2.1. Fundamentals

Here, we will go through abbreviations and the notation used throughout this thesis. Fundamental results and definitions are listed, along with explanations of necessary technical terms.

2.1.1. Abbreviations

- CNN: Convolutional Neural Network
- CV: Coefficient of Variation
- EFD: Elliptical Fourier Descriptor
- GAP: Global Average Pooling
- IID: Independent and Identically distributed
- MLE: Maximum Likelihood Estimator
- MLP: Multilayer perceptron
- MSE: Mean Squared Error
- NCC: Norwegian Coastal Cod
- NEAC: Northeast Arctic Cod
- OLS: Ordinary Least Squares
- PDF: Probability Density Function
- PMF: Probability Mass Function
- RNN: Recurrent Neural Networks
- TPM: Total Probability of Misclassification

2.1.2. Notation

- Scalars will be written as lowercase letters, for instance x
- Vectors will be written with lowercase boldface letters, for instance $\mathbf{x}$
- Matrices will be written with uppercase letters, for instance X
- Tensors will be written with uppercase calligraphic letters, for instance $\mathcal{X}$
- A vector of ones is denoted by $\mathbf{1}$
- Element-wise operations are denoted by $\odot$
- Convolutions are denoted by $*$
- The gradient of f with regard to $\mathbf{x}$ is denoted by $\nabla_{\mathbf{x}} f$
- The identity matrix is denoted by I
- The indicator function is denoted by $I\{\text{condition}\}$, and returns 1 if condition is true, 0 otherwise.
- For deep learning operations the superscript $^{(l)}$ is read as *layer* l, the subscript $_j$ as *unit* j and the paranthesis (i) as *sample* i. E.g., $a_j^{(l)}(i)$ is concerning output j in layer l for sample i.

2.1.3. Fundamental definitions and theorems

This section will provide fundamental definitions and results used throughout this thesis.

2.1.3.1. Stratified Proportionate Allocation

Consider a population consisting of a finite number of disjoint homogeneous subgroups (strata). Stratified sampling is a sampling procedure where the sampling population consists of random samples from each stratum. Using proportionate allocation, the proportion in the population is preserved in the sample size drawn from each stratum. In the context of this thesis, proportionate allocation refers to each subset having equal proportions with regard to the values of some ordinal or categorical feature. An example is a dataset split into subsets, where each subset has an equal proportion of females.

2.1.3.2. The chain rule of calculus

Let $z = f(\mathbf{y})$ and $\mathbf{y} = g(\mathbf{x})$ denote two differentiable functions. The chain rule states (Adams & Essex, 2013) that

$$\frac{\partial z}{\partial x_i} = \sum_j \frac{\partial z}{\partial y_j} \frac{\partial y_j}{\partial x_i}$$

The chain rule can be generalized to vector form as well, where the gradient of z with regard to $\mathbf{x}$ is written as

$$\nabla_{\mathbf{x}} z = \left(\frac{\partial \mathbf{y}}{\partial \mathbf{x}}\right)^T \nabla_{\mathbf{y}} z$$

where $\left(\frac{\partial \mathbf{y}}{\partial \mathbf{x}}\right)$ is the Jacobian matrix of $\mathbf{y}$. Furthermore, if $\mathbf{f}$ is vector valued function such that $\mathbf{z} = \mathbf{f}(\mathbf{x})$ and $\mathbf{y} = \mathbf{g}(\mathbf{x})$, the chain rule can be expressed in matrix form by

$$\left(\frac{\partial \mathbf{z}}{\partial \mathbf{x}}\right) = \left(\frac{\partial \mathbf{z}}{\partial \mathbf{y}}\right)\left(\frac{\partial \mathbf{y}}{\partial \mathbf{x}}\right)$$

2.1.3.3. Taylor's theorem

Let $f(\mathbf{x})$ be a function that is k times differentiable at $\mathbf{a} \in \mathbb{R}^n$. Furthermore, let $\alpha = (\alpha_1, \ldots, \alpha_n)$ be defined using multi-index notation such that

$$\begin{aligned} |\alpha| &= \alpha_1 + \cdots \alpha_n \\ \alpha! &= \alpha_1! \cdots \alpha_n! \\ \mathbf{x}^\alpha &= x_1^{\alpha_1} \cdots x_n^{\alpha_n} \end{aligned}$$

and

$$\partial^\alpha = \partial_1^{\alpha_1} \cdots \partial_n^{\alpha_n}$$

Then there exists an $h : \mathbb{R}^n \to \mathbb{R}$ such that

$$f(\mathbf{x}) = \sum_{|\alpha| \le k} \frac{\partial^\alpha f(\mathbf{a})}{\alpha!} (\mathbf{x} - \mathbf{a})^\alpha + \sum_{|\alpha| \le k} h(\mathbf{x})(\mathbf{x} - \mathbf{a})^\alpha$$

and

$$\lim_{\mathbf{x} \to \mathbf{a}} h(\mathbf{x}) = 0$$

2.1.3.4. Mean Squared Error (MSE)

Let $\hat{\theta}$ be an estimator of a parameter θ. Using the definition in (Casella & Berger, 2001), the mean squared error of an estimator $\hat{\theta}$ is defined as

$$\mathrm{MSE}\left(\hat{\theta}\right) = E\left[\left(\hat{\theta} - \theta\right)^2\right]$$

2.1.3.5. Maximum Likelihood Estimation

Let $X_1, \ldots, X_n$ be a random sample from a population with pdf $f(x|\boldsymbol{\theta})$. The likelihood function is defined by

$$\mathcal{L}(\boldsymbol{\theta}|\mathbf{x}) = \prod_{i=1}^{n} f(x_i|\boldsymbol{\theta})$$

and the log-likelihood is defined by

$$l(\boldsymbol{\theta}|\mathbf{x}) = \log \mathcal{L}(\boldsymbol{\theta}|\mathbf{x}) = \sum_{i=1}^{n} \log\left(f(x_i|\boldsymbol{\theta})\right)$$

The maximum likelihood estimator for $\boldsymbol{\theta}$ is given by

$$\hat{\boldsymbol{\theta}} = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\ \log \mathcal{L}(\boldsymbol{\theta}|\mathbf{x}) = \underset{\boldsymbol{\theta}}{\operatorname{argmax}}\ l(\boldsymbol{\theta}|\mathbf{x})$$

2.1.3.6. The Bernoulli distribution

The pdf of a Bernoulli distributed random variable Y is given by

$$f(y) = p^y \cdot (1-p)^{1-y}, \quad y = 0, 1$$

2.1.3.7. The Categorical distribution

Let $Y \in \{1, ..., k\}$ be a random variable with a corresponding probability vector

$$\underset{(k\times 1)}{\mathbf{p}} = (p_i)$$

where $P(Y = i) = p_i$. Then the distribution of Y is called a categorical distribution and is denoted by $Y \sim Cat(\mathbf{p})$ with a probability mass function (pmf) defined by

$$P(Y = y) = p_y = \prod_{j=1}^{k} p_j^{[y=j]} \tag{1}$$

where $\sum_{j=1}^{k} p_j = 1$. The distribution is also known as the generalized Bernoulli distribution.

2.1.3.8. Cross-entropy and the Kullback-Leibler (KL) divergence

Following the definition by Goodfellow et al. (2016), let X be a random variable distributed according to a probability density (pdf) function P, and let Q be the probability function of a proposed distribution on the same set. The entropy of $P(x)$ is defined as

$$H(P) = -E_{X\sim P}\left[\log P(x)\right] \tag{2}$$

and the cross-entropy with respect to $P(x)$ is defined as

$$H(P, Q) = -E_{X\sim P}\left[\log Q(x)\right] \tag{3}$$

The Kullback-Leibler divergence of $Q(x)$ with respect to $P(x)$ is defined as

$$D_{KL}(P||Q) = E_{X\sim P}\left[\log \frac{P(x)}{Q(x)}\right] = E_{X\sim P}\left[\log P(x) - \log Q(x)\right] \tag{4}$$

2.1.3.9. Discrete convolutions and correlations

The one dimensional discrete convolution of $f(x)$ and $g(x)$ is defined by (Gonzalez & Woods, 2008)

$$(f * g)(x) = \sum_s f(x) \cdot g(x - s)$$

The two dimensional convolution is defined similarly by

$$(f * g)(x, y) = \sum_s \sum_t f(s, t) \cdot g(x - s, y - t)$$

Because the convolution operation is commutative, this can equivalently be written as

$$(f * g)(x, y) = \sum_s \sum_t f(x - s, y - t) \cdot g(s, t) \tag{5}$$

The discrete correlation is defined similarly be replacing subtraction with addition. Convolution implementations usually computes the correlation, however the convolution term usually encompasses both operations. Thus, when talking about a convolution between two matrices I and W, the actual operation is the correlation operation

$$(I * W)_{x,y} = \sum_i \sum_j I_{x+i,y+j} \cdot W_{i,j}$$

In the context of CNNs, the convolution is usually computed between two 3-dimensional tensors consisting of a set of feature maps and a kernel. The term filter and kernel are used interchangeably and denotes the 3-dimensional tensor of parameters that the feature maps are convolved with. The 2-dimensional convolution between two 3-dimensional tensors $\mathcal{I} \in \mathbb{R}^{h_I \times w_I \times c}$ and $\mathcal{W} \in \mathbb{R}^{h_W \times w_W \times c}$ consists of channel-wise 2D convolutions and a summation along the third axis. This is expressed by

$$(\mathcal{I} * \mathcal{W})_{x,y} = \sum_i \sum_j \sum_k \mathcal{I}_{x+i,y+j,k} \cdot \mathcal{W}_{i,j,k} \tag{6}$$

2.1.3.10. Pooling

A 2-dimensional pooling operation with pooling size (i, j) on a matrix I is defined as a single statistic on the set $\{I[s,t]; x \leq s \leq x + i, y \leq t \leq y + j\}$. Particularly the 2-dimensional max pooling operation is defined as

$$\mathcal{P}_{\max}\{I\}(x, y) = \max\{I[s,t]; x \leq s \leq x + i, y \leq t \leq y + j\}$$

Other operations such as average pooling, median pooling and min pooling are also possible. They are defined similarly. Pooling layers and subsampling layers are terms that are used interchangeably.

2.1.3.11. Receptive field

Consider a function $f : \mathbb{R}^{m \times n} \to \mathbb{R}^{p \times q}$. The receptive field of the single real number $f(X)_{ij}$ is the region of X with a nonzero contribution to the value of $f(X)_{ij}$. E.g., an element in the matrix resulting from a convolution (5) between a 5-by-5 matrix I and a 3-by-3 kernel K has a 3-by-3 region in I as its receptive field (see figure 6).

2.2. The fundamentals of Neural Networks

A neural network is a network of simple computational units which together forms a function with high computational capabilities. These units were first introduced as perceptrons, which form the foundation of artificial neural networks. We will here discuss the two most fundamental designs; the multilayer perceptron and the convolutional neural network. In addition, this section will cover topics that are fundamental in the model fitting procedure, such as gradient descent, backpropagation and loss functions. The term neural network is general, and intends to encompass the types of deep learning models relevant for this thesis, namely MLPs and CNNs. The term *deep learning* is often understood to refer to deep neural networks, i.e. neural networks that are deep with regard to the number of layers. As in the textbook by Goodfellow et al. (2016), I will treat the terms artificial neural networks, deep neural networks and deep learning as one and the same.

2.2.1. Multilayer perceptrons

We begin this discussion on multilayer perceptrons by introducing the single units of which the MLP is composed of; perceptrons. A perceptron is a mathematical function that maps a vector to a real number by summing up its weighted inputs and passing the sum through the unit step function (see appendix A.2.1) tp produce a 0 or 1 output (see figure 3). If we let the input vector to the perceptron be denoted by $\mathbf{x}$, the weights be denoted by $\mathbf{w}$, and the activation function by $f(\cdot)$, then the output a of the perceptron is given by

$$a = f\left(\mathbf{x}^T \mathbf{w}\right)$$

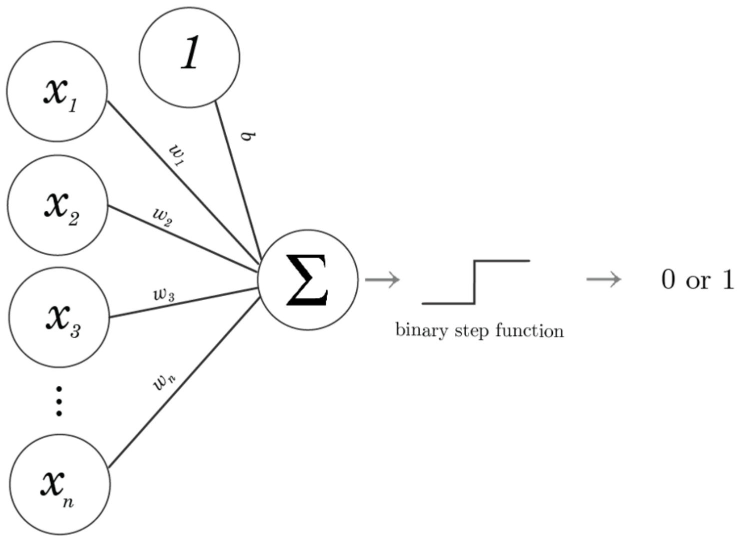


Figure 3: *A diagram of a single perceptron (Dhalla, 2021)*

An MLP is in essence a network of single units organized in layers, where each layer consists of several parallel units, and the input x to each unit consists of a vector **a** of all scalar outputs from the previous layer. The general equation describing the output of unit j in layer l, can be written as

$$a_j^{(l)} = f\left((\mathbf{a}^{(l-1)})^T \mathbf{w}_j^{(l)}\right)$$

where the vector of all outputs from layer l is given by

$$\mathbf{a}^{(l)} = f\left(W^{(l)}\mathbf{a}^{(l-1)} + \mathbf{b}^{(l)}\right) \tag{7}$$

In equation 7 the activation is applied element-wise, and the weights and biases are defined by

$$W^{(l)} = \begin{bmatrix} \mathbf{w}_1^{(l)T} \\ \vdots \\ \mathbf{w}_{n_l}^{(l)T} \end{bmatrix} \tag{8}$$

and

$$\mathbf{b}^{(l)} = \begin{bmatrix} b_1^{(l)} \\ \vdots \\ b_{n_l}^{(l)} \end{bmatrix} \tag{9}$$

Figure 4 shows an example of an MLP which displays one of the fundamental characteristics of MLPs, namely that every unit is connected to every unit in both the preceding and the following layer. The hidden layer in figure 4 goes under term fully connected layer, or dense layer.

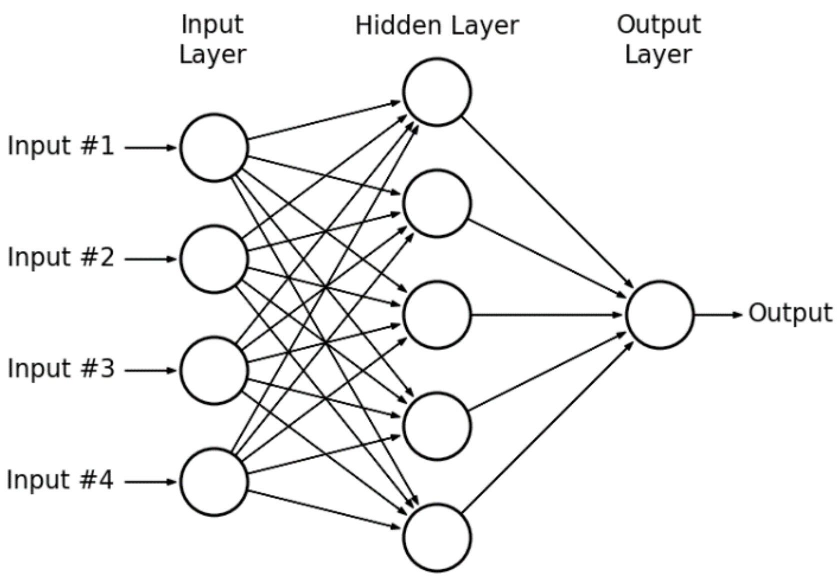


Figure 4: *An example of a multilayer perceptron (Hassan et al., 2015)*

Note that there are several possible activation functions that are applicable in addition to the unit step function, which is only included here for historical purposes. In modern networks the perceptron is usually replaced by the Rectified Linear Unit (ReLU) which differs from the perceptron in the use of the ReLU activation function (appendix A.2.4) in place of the unit step activation function. Appendix A.2 provides a walk-trough of well known activation functions.

2.2.2. Convolutional Neural Networks

Convolutional neural networks (CNNs) is a class of deep learning models specifically designed for image inputs. CNNs relies on discrete convolutions (section 2.1.3.9), which, in the field of image analysis, has several areas of application, including gradient computations, edge detection and noise removal (Gonzalez & Woods, 2008). The fundamental design of a CNN is a feedforward network of units similar to the MLP, with the key difference that one or more of the fully connected layers are replaced by convolutional layers. Where the hidden units in the fully connected layer takes in a vector and returns a scalar, the hidden units in the convolutional layer takes in a tensor and returns a matrix. The output of a convolutional unit is given by

$$A_j^{(l)} = f\left(\mathcal{A}^{(l-1)} * \mathcal{W}_j^{(l)}\right)$$

where the convolution is performed between a tensor of input feature maps and a kernel (see equation 6). As before, $f(\cdot)$ denotes the activation function which is applied elementwise on the input.

The convolution operation is often followed by a pooling/downsampling operation (section 2.1.3.10) that reduces the dimension of the output, and the output after convolution, activation and downsampling can be expressed by

$$O_j^{(l)} = \mathcal{P}\left\{ f\left(\mathcal{A}^{(l-1)} * \mathcal{W}_j^{(l)} \right) \right\}$$

Figure 5 shows an example of a CNN with 16 convolutional/pooling units, followed by an MLP.

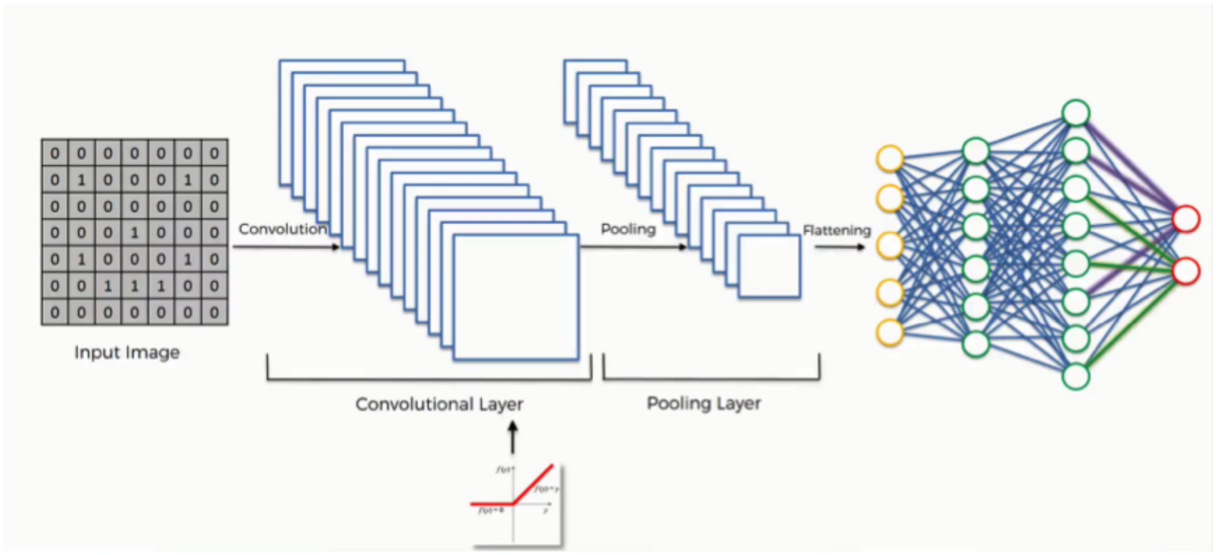


Figure 5: *Convolution and pooling operations followed by an MLP (SuperDataScience Team, 2018)*

Two beneficial results of using convolutions in place of fully connected layers, are sparse interactions and parameter sharing. Sparse interactions refer to the fact that the receptive field (section 2.1.3.11) of each feature is restricted by the kernel size, thus by using a 3-by-3 filter, the receptive field of a feature is restricted to a 3-by-3 region of the input feature map (see figure 6). Also note that the convolution operation applies the same filter repeatedly, i.e. the parameters are shared among the feature mappings.

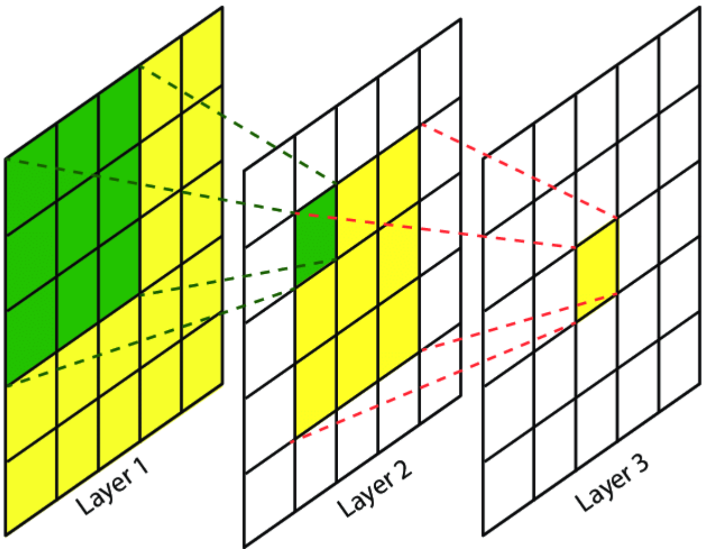


Figure 6: *The receptive field of a single feature is a 3-by-3 region of the input (Lin et al., 2017)*

The parameter reduction that results from replacing fully connected layers with convolutional layers can be demonstrated by the following calculation. Consider an input in

$\mathbb{R}^{h \times w \times c}$ and a 3-convolution kernel in $\mathbb{R}^{h_k \times w_k \times c}$ with n_l being the number of units in the layer of interest. If the layer is fully connected, the number of parameters is only dependent on the shape of the input and is given by $N = (h \cdot w \cdot c + 1) \cdot n_l$. This differs from the number of parameters in a convolutional layer which only depends on the kernel size and is given by $N = (h_k \cdot w_k \cdot c + 1) \cdot n_l$. Thus by choosing a kernel size much smaller than the size of the input we greatly reduce the number of parameters in the model. A layer with 8 units and an input of size (16, 16, 3) will have 6152 parameters if fully connected, vs 224 parameters if convolutional with a kernel of size 3-by-3.

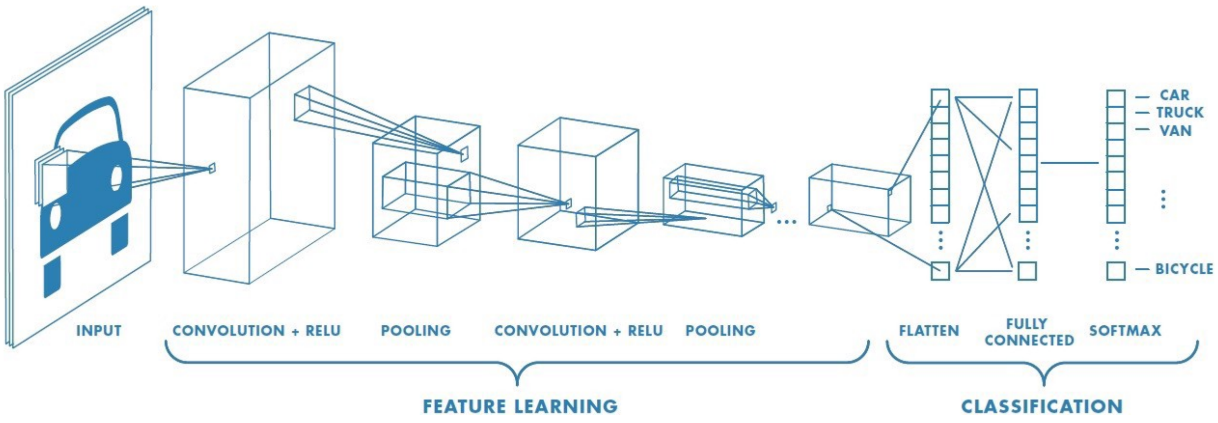


Figure 7: *The basic design of a CNN (Medium, 2018)*

The fundamental design of a CNN is illustrated in figure 7, which show a network with an arbitrary number of convolution/pooling operations followed by an MLP. The convolution network is considered the feature extraction part of the model, whereas the MLP is considered the classification part of the network. In the context of images, the convolutions are extracting features from the images, while the fully connected layers are weighting those features.

2.2.3. The role of activation functions

Activation functions are an integral part of neural networks, and are necessary in solving non-linear problems often exemplified by the well known XOR[1] problem described by Theodoridis and Koutroumbas (2009) amongst others. In an MLP, a non-linear activation function is in fact required if the neural network itself is to be non-linear. If we consider an L-layer perceptron with linear activations, and we use the definition in equation 8 and 9, the output for each layer given the input $\mathbf{x}$ can be written as

[1]The first proposed solution to the XOR problem is an artificial neuron with an output that is mapped to 0 or 1 using the unit step function. However, as we shall see in section 2.2.7, the backpropagation algorithm used for the optimization of the network parameters requires activation functions that are differentiable.

$$\begin{aligned}
\mathbf{y}^{(1)} &= W^{(1)}\mathbf{x} + \mathbf{b}^{(1)} \\
\mathbf{y}^{(2)} &= W^{(2)}\mathbf{y}^{(1)} + \mathbf{b}^{(2)} = W^{(2)} \cdot W^{(1)} \cdot \mathbf{x} + W^{(2)}\mathbf{b}^{(1)} + \mathbf{b}^{(2)} \\
&\vdots \\
\mathbf{y}^{(L)} &= \left(\prod_{j=1}^{L} W^{(j)}\right)\mathbf{x} + \sum_{j=1}^{L}\left(\prod_{k=j+1}^{L} W^{(k)}\right)\mathbf{b}^{(j)}
\end{aligned}$$

Thus, without a non-linear activation function the output is of the form

$$\mathbf{y} = \mathbf{x}^T W + \mathbf{b}$$

which indeed defines a linear function.

The activation function usually applied in CNNs is the ReLU activation function (Goodfellow et al., 2016), which is described along with a selection of activation functions in appendix A.2. All activation functions addressed here are monotonic. This is not a formal requirement, however a non-monotonic activation function would add ambiguity to the optimization process, since different values would produce equal derivatives. This could possibly result in divergence or slow convergence during training.

2.2.4. Activation at the output

The nature of the final output of a neural network depends on the problem at hand. Possible outputs include scalars, probabilities, vectors and matrices. The role of the output activation is to map the output to the function image, and I will address three situations that are relevant for this thesis; a binary classification problem, a k-class classification problem and a regression problem.

For a binary classification problem the activation function needs to be a monotonic function returning a value in the range 0 to 1. The sigmoid function (A.2.2) is a common and natural choice. Other possibilities exists, such as the standard normal cdf used in the probit model (Dobson, 2002).

For a k-class classification problem we need k outputs $\{\hat{y}_{ij} \in (0,1), j = 1, \ldots, k\}$ under the constraint that $\sum_{j=1}^{k} \hat{y}_{ij} = 1 \; \forall i$. The function most often used (Goodfellow et al., 2016) is the softmax function which is defined in appendix A.2.7. The softmax function can be seen as a generalization of the sigmoid function, and it is straightforward to show that it satisfies the conditions above.

In a situation where we are using a deep learning model as a regression model, the output is a real number as opposed to the estimated probability in the classification setting. If the domain of the response variable is the whole real line, there might be no need for any activation at the output. If the domain is the set of positive real numbers, a ReLU activation function (A.2.4) is a natural choice.

2.2.5. The role of loss functions

The aim of a neural network is to estimate some function that maps an input to an output. The output might be an estimated probability of class affiliation, or it could be an estimated mean response. In supervised learning we always have corresponding labels with our data, and the labels are treated as the ground truth. The loss function, also called the cost function or the objective function, is a summary statistic which provides an overall quantification of the model performance with regard to the labels. There exists a vast amount of variations regarding loss functions, however a common property is that the loss function sums up the performance in a single scalar, and that the loss function is differentiable with regard to its input. The loss function is closely related to the output activation function, and the loss function choice is problem dependent. In appendix A.3 we derive the loss functions for classification and regression, the result of which will be summarized here.

Classification

The loss function normally used in a binary classification problem is the binary cross-entropy loss function defined by

$$J = -\sum_{i=1}^{n} \left((1 - y_i) \log(1 - \hat{y}_i) + y_i \log(\hat{y}_i)\right)$$

For a k-class classification problem the loss function normally used is the cross-entropy loss function defined by

$$J = -\sum_{i=1}^{n} \sum_{j=1}^{k} [y_i = j] \log(\hat{y}_{ij})$$

As shown in appendix A.3.2, the functions have the interpretations that they are minimizing the Kullback-Leibler divergence of the categorical distribution of the output with regard to the prior categorical distribution of the labels.

Regression

For a regression problem the loss function normally used is the MSE loss function given by

$$J = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$$

As shown in appendix A.3.1, minimizing the MSE loss function is equivalent to maximizing the likelihood of the parameters under the assumption of normally distributed error terms with equal variances.

2.2.6. Gradient descent

The aim in the training of a neural network is to find a set of model parameters such that the loss function is minimized. A commonly used method is the gradient descent algorithm, where we are trying to find the global minima of the loss function by moving in the direction of steepest descent. Gradient descent is described in detail below.

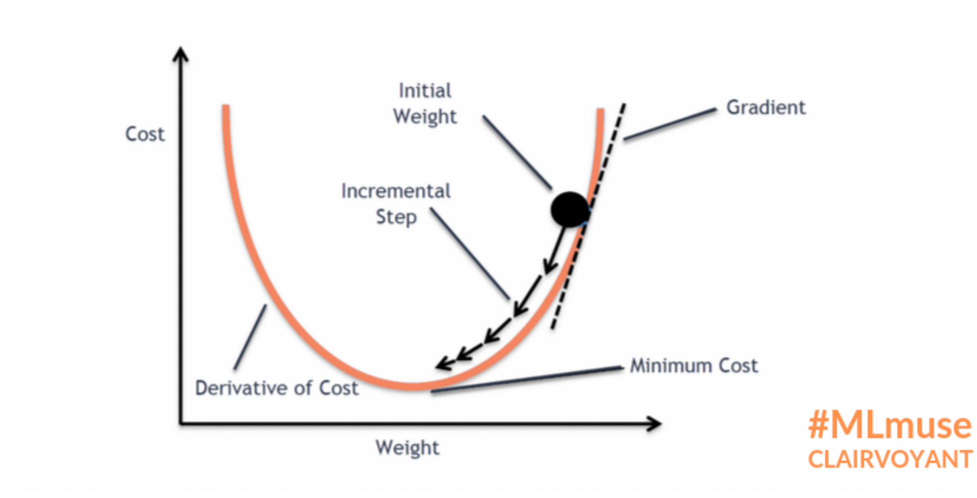


Figure 8: *Gradient descent algorithm on a single-parameter loss function (Clairvoyant Blog, 2019)*

Let $\boldsymbol{\theta}$ be a vector of model parameters and let $\mathbf{x}$ be the model input with $\mathbf{y}$ as the desired model output. Analogous to the derivative in univariate calculus, the direction of steepest descent for a vector valued function is given by the gradient $\nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}; \mathbf{x}, \mathbf{y})$, and the update equation for the gradient descent algorithm is written as

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^{t} - \alpha \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta}; \mathbf{x}, \mathbf{y}) \big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{t}} \tag{10}$$

where $J(\cdot)$ is the objective function to be minimized and α - the learning rate - is a hyperparameter which influences the speed of convergence. I.e., the weights at the next time step are obtained by executing a small movement in the direction of steepest descent from the weights at the current time step. The principle behind gradient descent is illustrated in figure 8, where convergence is achieved when the weight vector resulting in a zero gradient is found.

Note that the loss in equation (10) is computed using all available examples and labels. This variant of gradient descent is known as *Batch gradient descent*, and the gradients are updated using the whole dataset for each epoch. Batch gradient descent are guaranteed to converge to a local minima (Ruder, 2017), however large datasets could make the method inefficient.

A variant that is normally used in practice are *Mini-batch gradient descent*. Here the dataset are split up into mini-batches, and the gradients are updated one mini-batch at a time. This is the standard implementation used in the TensorFlow (Abadi et al., 2015) framework.

Stochastic gradient descent provides an alternative at the other extreme. The gradients are updated one sample at a time, resulting in a fast online iteration procedure with high variance of the loss. With a decreasing learning rate, this method is also shown to converge to a local minima (Ruder, 2017).

There are several different variations of the gradient descent update equation, all of which are aiming at improving the speed of convergence. Ruder (2017) provides a walk-trough of gradient descent variations, and a very brief summary of the gradient descent evolution are presented in appendix A.4.

2.2.7. The backpropagation algorithm

We have seen that a feedforward network produces an output by using the relation that each layer's output serves as the input to the next one. We have also addressed the role of the loss function, and we have also seen that an optimal set of weights can be obtained by gradient descent. Here we will address the actual gradient computations, which are carried out using the backpropagation algorithm.

Before expressing the algorithm, we first note that the output of a neural network, i.e. the output from layer L, can be written as a function of all outputs from layer $L-1$, i.e.

$$a^{(L)} = f_L\left(a_1^{(L-1)}, \cdots, a_{n_{L-1}}^{(L-1)}\right)$$

All the outputs from the layer $L-1$ are dependent on $L-2$ in the same manner, and the final output is obtained by layer-wise output computations, starting at the input. If we define the total loss by

$$J = \sum_{i=1}^{n} \text{loss}(\mathbf{a}^{(L)}(i), y(i))$$

and we let $\mathbf{a}^{(l)}(i)$ denote the output from layer l for sample i, the first part of the back-propagation algorithm - the forward pass - can be expressed as in algorithm 1, where $f_j^{(l)}$ is a function that includes all operations applied on the input $\mathbf{a}^{(l-1)}(i)$ given a set of weights $\mathbf{w}_j^{(l)}$.

The second part of the backpropagation algorithm is concerned with obtaining the gradients. The gradient of the loss with regard to the weights $\mathbf{w}^{(l)}$ can be found by recursively computing gradients starting at the loss. First we note that

$$\nabla_{\mathbf{w}^{(l)}} J = \sum_{i=1}^{n} \nabla_{\mathbf{w}^{(l)}} \text{loss}(\mathbf{a}^{(L)}(i), y(i))$$

Algorithm 1 The forward pass

```
$J = 0$
for $i$ in $1, \cdots, n$ do
    $\mathbf{a}^{(0)}(i) = \mathbf{x}(i)$
    for $l$ in $1, \cdots, L$ do
        $\mathbf{a}^{(l)}(i) = \begin{bmatrix} f_1^{(l)}\left(\mathbf{a}^{(l-1)}(i), \mathbf{w}_1^{(l)}\right) \\ \vdots \\ f_{n_l}^{(l)}\left(\mathbf{a}^{(l-1)}(i), \mathbf{w}_{n_l}^{(l)}\right) \end{bmatrix}$
    end for
    $J = J + \text{loss}\left(\mathbf{a}^{(L)}(i), y(i)\right)$
end for
```

which by the chain rule (2.1.3.2) can be written as

$$\nabla_{\mathbf{w}^{(l)}} J = \sum_{i=1}^{n} \left(\frac{\partial \mathbf{a}^{(L)}(i)}{\partial \mathbf{w}^{(l)}}\right)^T \nabla_{\mathbf{a}^{(L)}(i)} \text{loss}(\mathbf{a}^{(L)}(i), y(i))$$

Furthermore, by the chain rule for total derivatives, we have the relation

$$\left(\frac{\partial \mathbf{a}^{(L)}(i)}{\partial \mathbf{w}^{(l)}}\right) = \left(\frac{\partial \mathbf{a}^{(L)}(i)}{\partial \mathbf{a}^{(L-1)}(i)}\right)\left(\frac{\partial \mathbf{a}^{(L-1)}(i)}{\partial \mathbf{a}^{(L-2)}(i)}\right)\cdots\left(\frac{\partial \mathbf{a}^{(l)}(i)}{\partial \mathbf{w}^{(l)}}\right)$$

Thus we see that the gradient of the loss with regard to the weights $\mathbf{w}^{(l)}$ can be obtained by recursive multiplications of the intermediate derivatives. This part of the backpropagation algorithm is known as the backward pass which is described in algorithm 2.

Algorithm 2 The backward pass

```
for $l$ in $L, \cdots, 1$ do
    for $i$ in $1, \cdots, n$ do
        if $l = L$ then
            compute $\nabla_{\mathbf{a}^{(L)}(i)} J$
        else
            $\nabla_{\mathbf{a}^{(l)}(i)} J = \left(\frac{\partial \mathbf{a}^{(\mathbf{l+1})}(\mathbf{i})}{\partial \mathbf{a}^{(\mathbf{l})}(\mathbf{i})}\right)^T \nabla_{\mathbf{a}^{(l+1)}(i)} J$
        end if
        $\nabla_{\mathbf{w}_k^{(l)}} J = \nabla_{\mathbf{w}_k^{(l)}} J + \left(\frac{\partial \mathbf{a}^{(l)}(i)}{\partial \mathbf{w}_k^{(l)}}\right)^T \nabla_{\mathbf{a}^{(l)}(i)} J$
    end for
    update $\mathbf{w}_k^{(l)}$ by $\nabla_{\mathbf{w}_k^{(l)}} J$ using gradient descent
end for
```

The exact gradient expressions are dependent on the loss function, the activation functions and the layers in the network. Appendix A.1 shows the exact update rule for the special case of multilayer perceptrons.

2.3. Deep learning layers and operations used in CNNs

This section addresses some common techniques used in deep learning models which are applied along with the fundamental operations addressed earlier. This includes regularization methods such as Batch normalization and Dropout, along with Global Average Pooling and Depthwise Separable Convolutions, which serve as an alternative to the basic convolution operation.

2.3.1. Batch Normalization

Batch Normalization (Ioffe & Szegedy, 2015) is a technique used to improve optimization and to speed up convergence of deep neural networks. Since each layer serves as an input to the next, and the number of unit outputs is equal to the sample size, the layer outputs can be interpreted as a realization of a data distribution specific for that layer. However, since the layer output itself is a function of weights that are constantly being updated, the data distribution is constantly shifting. This phenomena is referred to as *Internal Covariate Shift*, and since we are optimizing the parameters under the assumption that the data distribution remains constant, it is natural to propose that this inhibits convergence.

The batch normalization procedure is analogous to the standardization of variables common in statistical analysis and it is defined as follows:

Let $a_j^{(l)}(i)$ be the output of unit j in layer l for observation i. The normalized output $\hat{a}_j^{(l)}(i)$ is computed by

$$\hat{a}_j^{(l)}(i) = \frac{a_j^{(l)}(i) - \overline{\mathbf{a}}_j^{(l)}}{\sqrt{\mathrm{Var}(\mathbf{a}_j^{(l)}) + \epsilon}}$$

where the ϵ term is a small number added to avoid zero division. The original paper uses the biased variance estimator with the batch size m in the denominator. In addition to being normalized, the data is also re-parametrized with a new mean and variance using the formula

$$o_j^{(l)}(i) = \alpha_j^{(l)} \hat{a}_j^{(l)}(i) + \beta_j^{(l)}$$

where $\alpha_j^{(l)}$ and $\beta_j^{(l)}$ are trainable parameters. The new parametrization makes the optimal parameters for the hidden distributions easier to learn, which in turn makes the network easier to train (Goodfellow et al., 2016). Note that during inference[2], the normalization

[2]In the Keras API (Chollet et al., 2015), a deep learning model has two set of modes which may alter the models behaviour. *Inference mode* refers to the set of specifications for a trained model applied on an independent set of data, whereas *training mode* refers to the set of model specifications used during the model fitting procedure.

is performed by using the pooled mean and variance over the training batches, resulting in a simple linear transformation of the test data.

Although the original motivation was to diminish the effect of internal covariate shift, it has been suggested by Shibani Santurkar and Madry (2019) that the real effect of batch normalization is an increased smoothness of the objective function. Ioffe and Szegedy (2015) suggested to apply batch normalization just before the activation, however there are no practical limitations for using batch normalization. Batch normalization has gained massive popularity, and is often included by default in conjunction with fully connected and convolutional layers in deep neural networks.

2.3.2. Dropout

Dropout (Srivastava et al., 2014) is a widely applied regularization technique used to counteract overfitting in deep learning models. During training, a random sample of individual units are dropped, where a hyperparameter - the dropout rate p - determines the dropout probability. The technique is closely related to bagging (Goodfellow et al., 2016), where a final model is computed as an arithmetic mean of an ensemble of fitted models. In dropout however, we start with a neural network consisting of several units of which a selection is relevant for dropout. The ensemble of models are now the set of all possible models given the units relevant for dropout.

If we let $a_i^{(l)}$ be the activation for unit i in layer l, the output from that unit when dropout is applied is written as $o_i^{(l)} = a_i^{(l)} \mu_i^{(l)}$ where $\mu_i^{(l)} \sim \text{Bernoulli}(p)$.

The dropout operation requires special attention during inference. During inference, we are neither sampling or averaging over the submodels as is the case in bagging. Instead we are using the expected output for each unit as our final model, not actually dropping any unit at all. Since the expected output of our example unit during training is given by $E\{o_i^{(l)}\} = a_i^{(l)} \cdot p$, the outputs are rescaled to $a_i^{(l)} \cdot p$ during inference to preserve the total magnitude of all layer activations.

Dropout is known to be successful, and the aim is to improve the robustness of the final model by forcing the nodes in a dropout layer to transfer generalized representations of the data, which effectively inhibits overfitting to the training data. A possible reason behind its success can be attributed to the problem of *co-adaptation* in neural networks (Ranjan, 2019). The principle of co-adaptation is an inherent part of the gradient update procedure; units that are already contributing significantly to the output will improve further, while units with small contributions will in comparison sustain negligible improvements.

A simple example illustrating the effect of dropout is derived here. Consider a simple model where 4 units are connected to a single neuron in the output layer. If we use a dropout rate of 0.25 this means that one out of four outputs in the second to last layer are set to zero. Since the output should be the same regardless of which neurons that are dropped (otherwise the model will exhibit inconsistent performance), the usage of dropout is equivalent to adding a constraint on the model, namely that

$$\sum_{j\neq 1} a_j^{(L-1)}(i)w_j^{(L)} \approx \sum_{j\neq 2} a_j^{(L-1)}(i)w_j^{(L)} \approx \cdots \approx \sum_{j\neq 4} a_j^{(L-1)}(i)w_j^{(L)}$$

This in turn forces the model to make sure that any permutation of the outputs represents the data equally well. Thus, an interpretation of dropout is that you drive the network to produce individual units that pass on general information about the data.

2.3.3. Global Average Pooling

Global Average Pooling (GAP) (Min Lin & Yan, 2014) is a pooling operation used in conjunction with CNNs, working either as a bridge between a convolutional layer and a fully connected layer, or as the final layer in the network. A GAP layer takes a set of feature maps as inputs, and returns a set of global averages, one for each feature map (see figure 9). Thus, if we have a single sample with c feature maps of size $h \times w$, the GAP layer converts an $w \times h \times c$ tensor input into a vector output of length c.

Formally, let $\overline{a}$ denote the mean of all elements in the matrix $\underset{(h\times w)}{A}$, e.g.

$$\overline{a} = \frac{1}{hw} \cdot \underset{(1\times h)}{\mathbf{1}^T} \cdot \underset{(h\times w)}{A} \cdot \underset{(w\times 1)}{\mathbf{1}}$$

and let $\mathcal{A} \in \mathbb{R}^{n\times h\times w\times c}$ be a 4-dimensional tensor. The Global Average Pooling operation is defined as

$$\text{GAP}\{\underset{(n\times h\times w\times c)}{\mathcal{A}}\} = (\overline{a}_{ij}) \in \mathbb{R}^{n\times c}$$

where $(a_{ij}) = \mathcal{A}_{i,:,:,j}$.

The GAP layer is used in the final classification part of a CNN, either as the final output layer itself, or as an input to a fully connected layer. The important thing to note is that the GAP output dimension is independent of image size, and that the GAP layer has zero parameters. This makes GAP applicable for situations where the input dimensions are not constant. When used in conjunction with a fully connected layer, GAP also reduces the number of parameters in the model by reducing the number of features used in classification. For instance will an input consisting of 64 feature maps of size 4×4 result in $1024 + 1$ parameters if fed directly into a single neuron, while the same input will result in $64 + 1$ parameters if a GAP operation were to be applied on the input before the fully connected layer. By reducing the number of parameters in the network, the GAP layer serve as a way to prevent overfitting.

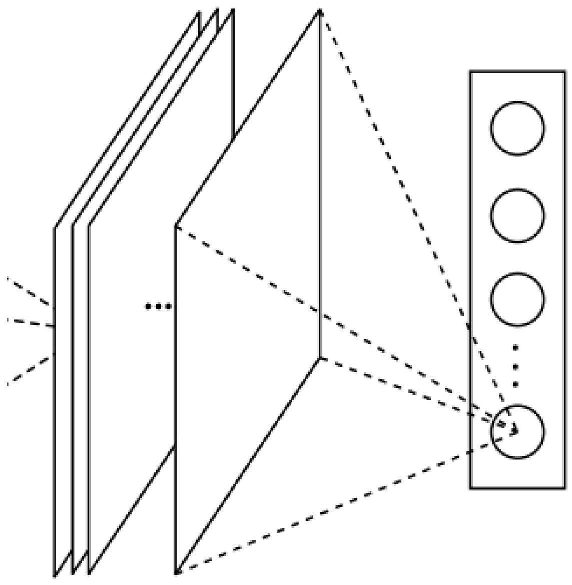


Figure 9: *The principle of global average pooling. Here the average of all elements in the final feature map constitutes as the final element in the output vector. (Min Lin & Yan, 2014)*

2.3.4. Depthwise Separable Convolutions

Depthwise separable convolutions was first introduced by Chollet (2017) under the model hypothesis that all spatial correlations and cross-channel correlations can be decoupled. Depthwise separable convolutions serve as an alternative to the normal convolution (2.1.3.9), and the process is a two step procedure where we are transforming an input of size $H_{in} \times W_{in} \times C_{in}$ into an output of size $H_{out} \times W_{out} \times C_{out}$. In a normal convolution, we are obtaining this by convolving the input C_{out} times, using C_{out} different $m \times n$ kernels. In a depthwise separable convolution we are instead using a two-step procedure described by the following:

- Depthwise convolution (figure 10): Convolve each channel in the input with a $m \times n \times 1$ kernel, resulting in an intermediate output of size $H_{out} \times W_{out} \times C_{in}$.

- Pointwise convolution (figure 11): Convolve the intermediate output with C_{out} kernels of size $1 \times 1 \times C_{in}$, resulting in a final output of size $H_{out} \times W_{out} \times C_{out}$.

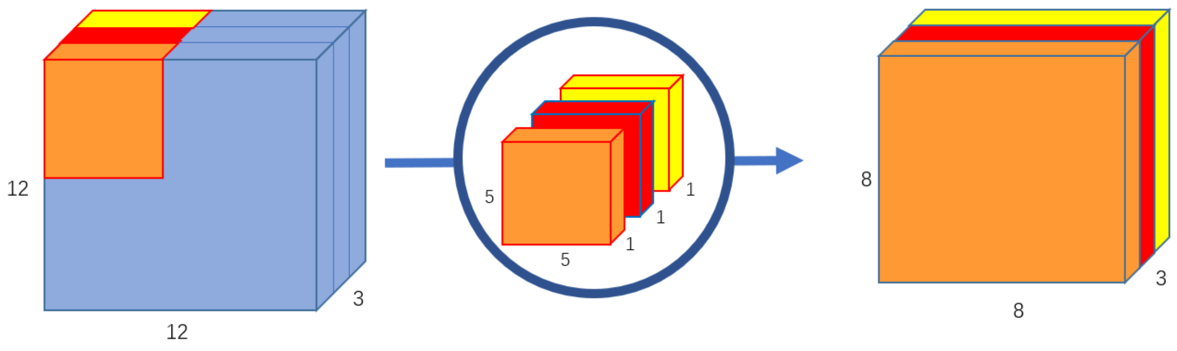


Figure 10: *Depthwise step (Wang, 2018)*

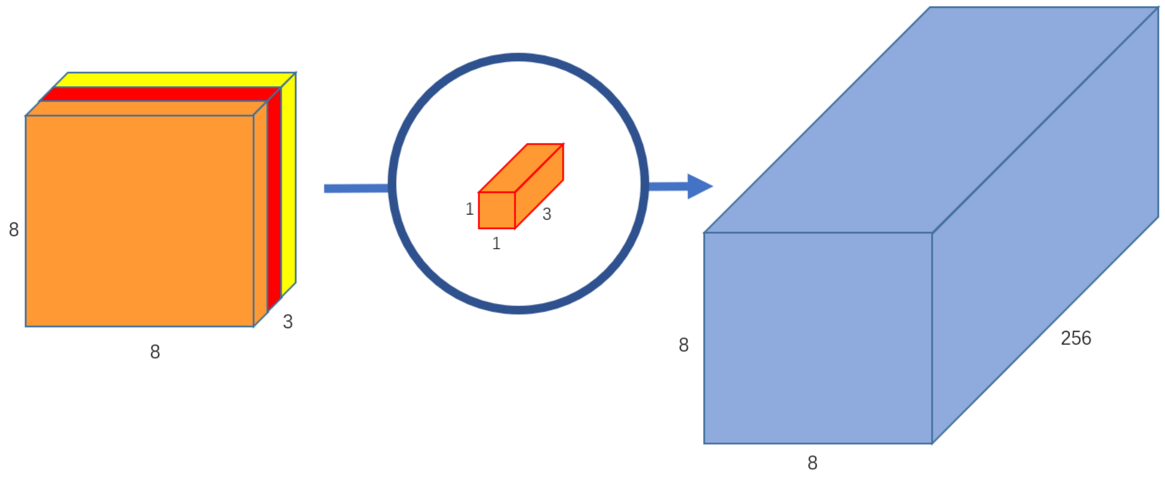


Figure 11: *Pointwise step (Wang, 2018)*

A key difference between a normal convolutional layer and a depthwise separable convolutional layer lies in the number of parameters. A convolutional layer with C_{out} units needs C_{out} kernels of size $m \times n \times C_{in}$, while a separable convolutional layer needs C_{out} kernels of size $1 \times 1 \times C_{in}$ and C_{in} kernels of size $m \times n$. Thus, the total number of parameters is $(m \cdot n \cdot C_{in} + 1) \cdot C_{out}$ for normal convolutions, versus $m \cdot n \cdot C_{in} + (C_{in} + 1) \cdot C_{out}$ parameters for separable convolutions. As an example, going from an input with 128 feature maps to an output of 256 feature maps using a filter size of 3×3, the number of parameters is reduced from approximately $300,000$ to approximately $30,000$ if convolutions are replaced by separable convolutions.

2.4. Regularization methods

Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not its training error.

Goodfellow et al. (2016)

This section is concerned with regularization methods and the task of preventing overfitting. Some methods have been discussed already, and I will not repeat the arguments here. Dropout (2.3.2), Global Average Pooling (2.3.3) and Separable Convolutions (2.3.4) are all techniques that can be applied with the aim of reducing model complexity. I will instead address weight decay and data augmentation as means to regularize neural networks, and I will start off with a brief discussion regarding generalization error, and a short discussion regarding input size.

The following list is a summary of possible regularization approaches along with specific methods discussed in this thesis:

- Restrictions on the functional relationship between samples and labels, e.g. by only considering a linear model:
  - Depthwise Separable Convolutions (2.3.4)
- Restrictions on the number of model parameters:
  - Depthwise Separable Convolutions (2.3.4)
  - Global Average Pooling (2.3.3)
  - Reduction of the number of input features (2.4.2)
- Restrictions on the domain of the model parameters:
  - Parameter norm penalties (2.4.3)
  - Early stopping (2.4.5)
- Bootstrap aggregation (Bagging) of an ensemble of models:
  - Dropout (2.3.2)
- Data augmentation (2.4.4)

2.4.1. Generalization error

As in all statistical analysis, the data that we are handed are sampled from an underlying population, and the deep learning model is fitted to the sample under the paradigm that we want our model to perform well on any sample from the true population of the data, not only on the training sample at hand. I.e., we are training a model under the directive of minimized training loss, while our primary objective is the minimization of the generalization loss. Figure 12 shows the typical development of the loss as the model complexity increases. We define an overfitted model as a model that has adapted to the training data to such an extent that the model is unable to perform on any other data from the underlying population, and we define an underfitted model as a model that fails to model the approximate relationship between the input data and the labels.

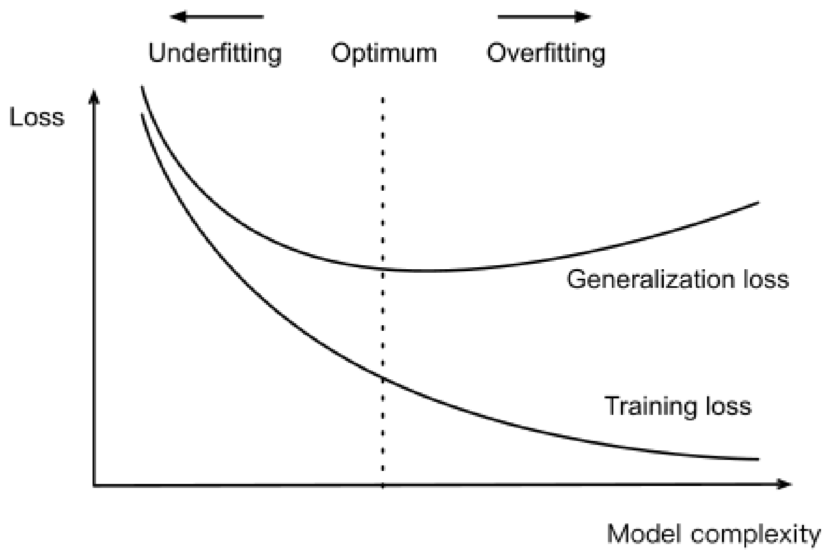


Figure 12: *Generalization error as a function of model complexity (Zhang et al., 2021)*

The fact that an increase in model complexity leads to a decrease in training loss, is evident from several examples:

- The Main Theorem of Polynomial Interpolation regarding Lagrange interpolation (Sauer, 2011), states that all y_i's in the set of points $(x_1, y_1), \ldots, (x_n, y_n)$ can be perfectly modeled by a $n - 1$ degree polynomial of the x_i's.

- In linear regression, where we are predicting the response vector $\mathbf{y}$ by the predictors in the design matrix $X \in \mathbb{R}^{n \times p}$, we know by the Rouché-Capelli theorem that the equation

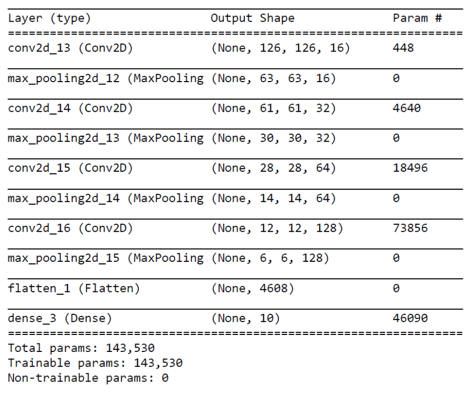
$$\mathbf{y} = X\mathbf{w}$$

  has a unique solution for $\mathbf{w}$ if $\text{rank}(X) = \text{rank}(X|\mathbf{y}) = p$.

Both these examples illustrate the fact that by increasing the number of independent variables, we will always be able to construct a model with a zero error on the training set. This way of fitting a model is problematic since it implies that the relationship between an observation and its label is truly deterministic, and that no randomness in the response exists. Thus, if the label values indeed are subject to an additional randomness, we would expect a poor performance in predictions on previously unseen independent data.

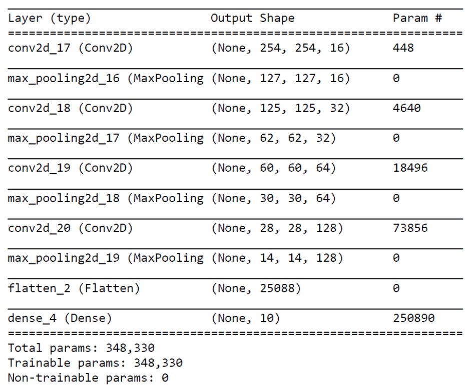
2.4.2. The role of input shape and network depth

Depending on the architecture, the size of the input images can greatly effect the complexity of the model. Consider a simplified deep learning model consisting of four convolutional/pooling blocks, and a fully connected layer at the end with 10 output neurons. Since the output of the last convolutional block is flattened, the number of parameters in the following fully connected layer will depend on the size of the input image. Figure 13 (a) and (b) shows a comparison of the number of parameters for input images of size $(128, 128, 3)$, versus the number of parameters for the same model using input images of size $(256, 256, 3)$. Note that while the number of convolutional parameters is independent of the input size, the number of parameters in the fully connected layer is not. In this case, the complexity of the model is reduced significantly by reducing the size of the input images[3]. Figure 13 (c) and (d) show the same comparison for a model with an additional convolutional block. The additional downsampling layer reduces the dimensions of the feature maps, which in turn reduces the dimension of the dense layer input. Thus, by increasing the depth of the network, we are able to further reduce the number of model parameters.



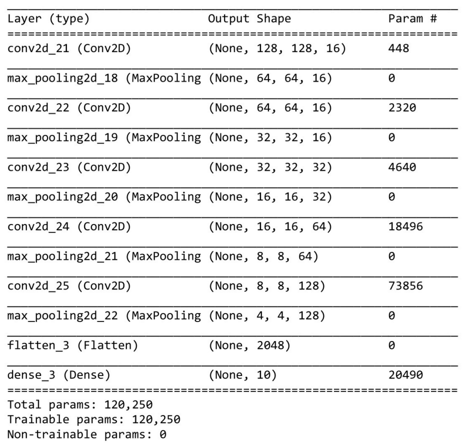
```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_13 (Conv2D)           (None, 126, 126, 16)      448
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 63, 63, 16)        0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 61, 61, 32)        4640
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 30, 30, 32)        0
_________________________________________________________________
conv2d_15 (Conv2D)           (None, 28, 28, 64)        18496
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 14, 14, 64)        0
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 12, 12, 128)       73856
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 6, 6, 128)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 4608)              0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                46090
=================================================================
Total params: 143,530
Trainable params: 143,530
Non-trainable params: 0
```

(a) *Model with* $(128, 128)$ *input images and 4 convolutional blocks*



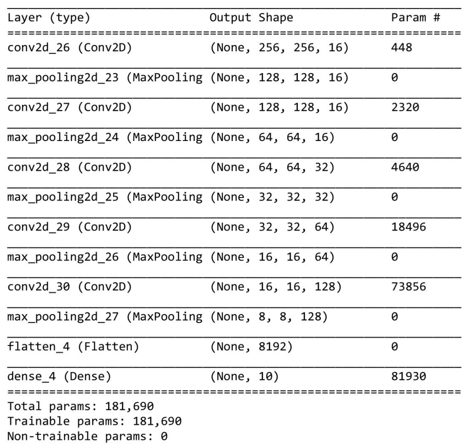
```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_17 (Conv2D)           (None, 254, 254, 16)      448
_________________________________________________________________
max_pooling2d_16 (MaxPooling (None, 127, 127, 16)      0
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 125, 125, 32)      4640
_________________________________________________________________
max_pooling2d_17 (MaxPooling (None, 62, 62, 32)        0
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 60, 60, 64)        18496
_________________________________________________________________
max_pooling2d_18 (MaxPooling (None, 30, 30, 64)        0
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 28, 28, 128)       73856
_________________________________________________________________
max_pooling2d_19 (MaxPooling (None, 14, 14, 128)       0
_________________________________________________________________
flatten_2 (Flatten)          (None, 25088)             0
_________________________________________________________________
dense_4 (Dense)              (None, 10)                250890
=================================================================
Total params: 348,330
Trainable params: 348,330
Non-trainable params: 0
```

(b) *Model with* $(256, 256)$ *input images and 4 convolutional blocks*



```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_21 (Conv2D)           (None, 128, 128, 16)      448
_________________________________________________________________
max_pooling2d_18 (MaxPooling (None, 64, 64, 16)        0
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 64, 64, 16)        2320
_________________________________________________________________
max_pooling2d_19 (MaxPooling (None, 32, 32, 16)        0
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 32, 32, 32)        4640
_________________________________________________________________
max_pooling2d_20 (MaxPooling (None, 16, 16, 32)        0
_________________________________________________________________
conv2d_24 (Conv2D)           (None, 16, 16, 64)        18496
_________________________________________________________________
max_pooling2d_21 (MaxPooling (None, 8, 8, 64)          0
_________________________________________________________________
conv2d_25 (Conv2D)           (None, 8, 8, 128)         73856
_________________________________________________________________
max_pooling2d_22 (MaxPooling (None, 4, 4, 128)         0
_________________________________________________________________
flatten_3 (Flatten)          (None, 2048)              0
_________________________________________________________________
dense_3 (Dense)              (None, 10)                20490
=================================================================
Total params: 120,250
Trainable params: 120,250
Non-trainable params: 0
```

(c) *Model with* $(128, 128)$ *input images and 5 convolutional blocks*



```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_26 (Conv2D)           (None, 256, 256, 16)      448
_________________________________________________________________
max_pooling2d_23 (MaxPooling (None, 128, 128, 16)      0
_________________________________________________________________
conv2d_27 (Conv2D)           (None, 128, 128, 16)      2320
_________________________________________________________________
max_pooling2d_24 (MaxPooling (None, 64, 64, 16)        0
_________________________________________________________________
conv2d_28 (Conv2D)           (None, 64, 64, 32)        4640
_________________________________________________________________
max_pooling2d_25 (MaxPooling (None, 32, 32, 32)        0
_________________________________________________________________
conv2d_29 (Conv2D)           (None, 32, 32, 64)        18496
_________________________________________________________________
max_pooling2d_26 (MaxPooling (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_30 (Conv2D)           (None, 16, 16, 128)       73856
_________________________________________________________________
max_pooling2d_27 (MaxPooling (None, 8, 8, 128)         0
_________________________________________________________________
flatten_4 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_4 (Dense)              (None, 10)                81930
=================================================================
Total params: 181,690
Trainable params: 181,690
Non-trainable params: 0
```

(d) *Model with* $(256, 256)$ *input images and 5 convolutional blocks*

Figure 13: *Example model showing the effect of image size and depth on the number of model parameters*

[3]Recall from section 2.3.3 that if we were to apply GAP on the output of the final convolutional block, the input size would not effect the number of parameters in the model.

2.4.3. Norm penalties and L2-regularization

Norm penalties are additional penalties that are added to the loss of the objective function by defining a new loss function

$$J(\boldsymbol{\theta})_{\text{new}} = J(\boldsymbol{\theta})_{\text{old}} + \gamma \cdot \boldsymbol{\Omega}(\boldsymbol{\theta})$$

where $\boldsymbol{\Omega}$ denotes the norm penalty, and γ is a hyperparameter that controls the magnitude of the regularization. The norm penalty puts a constraint on the model complexity by increasing the loss if the parameters gets too large in magnitude, giving rise to several interpretations. One interpretation is that it prevents the model in putting to much emphasis on a single feature by the fact that small evenly distributed weights will be contribute less to the loss than sparse weights with extreme values. Another point of view is that instead of restricting the number of parameters, we are putting a constraint on the domain of the parameters by noting that even if the true domain is the whole $\mathbb{R}^p$, a parameter norm penalty will direct the weights closer to the origin.

A special case of norm penalty regularization is L2-regularization which is a regularization method that stems from Ridge regression described in appendix B.1. In the deep learning literature this is commonly known as weight decay, and the method involves adding to the loss function the squared 2-norm of the weights. If we let $\mathbf{w}$ be a vector of model parameters chosen to be subject to regularization, the L^2 regularization consists of redefining the loss function by

$$J_{\text{new}} = J_{\text{old}} + \gamma \cdot \mathbf{w}^T\mathbf{w}$$

The gradient of the loss with regard to the weights are now

$$\nabla_{\mathbf{w}} J(\boldsymbol{\theta})_{\text{new}} = \nabla_{\mathbf{w}} J(\boldsymbol{\theta})_{\text{old}} + \gamma \cdot \mathbf{w}$$

resulting in the modification

$$\mathbf{w}^{(t+1)} = \mathbf{w}^{(t)} - \alpha\gamma \cdot \mathbf{w}^{(t)} - \alpha \nabla_{\mathbf{w}} J(\boldsymbol{\theta})_{\text{old}}\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{(t)}}$$

of the gradient descent update rule (2.2.6). Note that while the third term on the right hand side drives the weights in the direction of steepest descent, the second term actually drives the weight closer to the origin, thus the regularization puts a restriction on the domain of the parameters.

2.4.4. Data augmentation

The ideal way of ensuring good generalization capabilities for a model is by increasing the sample size of the dataset. If we could in some way increase the sample to whichever size necessary, we would always be able to obtain a training set representational of the underlying distribution. While additional sampling from the true distribution is not an option, modification of the data already obtained is a viable way to artificially increase the dataset. This is the key principle behind dataset augmentation, where the idea is to add modified versions of the data already available, thereby increasing the number of labeled examples. The goal is to make the model robust to variations in the input space, thereby improving the models generalization property.

For classification tasks where the inputs are images, data augmentation has proven to be very efficient (Goodfellow et al., 2016). Data augmentation on images can be obtained in several ways, all of which aims at modifying the images sufficiently, while at the same time preserving the fundamental features that the image is meant to represent. Examples of augmentation methods applied to images include rotation, translation, flipping, zooming and contrast adjustments, all of which directs the model to base its predictions on some general features of the input.

These methods can be applied during preprocessing; i.e. by increasing the dataset using added random variations of the original images, or during training where inputs are randomly transformed at runtime. In the last case, these methods are applied randomly per batch during the training process, thereby slightly changing the training set. With modifications at runtime, the exact same input are rarely encountered twice.

2.4.5. Early stopping

Early stopping is a regularization method motivated by a situation that is often encountered when fitting deep learning models, namely a model that is overfitting to the training data (Goodfellow et al., 2016). The overfitting occurs when training iterations continues past the point of optimal generalization loss, thereby overfitting to the training data which in turn causes the generalization loss to increase.

Figure 14 shows the principle of early stopping, and the procedure can be summarized by the following steps

- Split the available training samples into a training and a validation set.
- Fit the model on the training set, while at the same time monitoring the loss on the validation data.
- Set a *patience* parameter, and stop training after *patience* number of epochs without improvement in the validation loss.
- Restore the model parameters to the point of best validation performance.

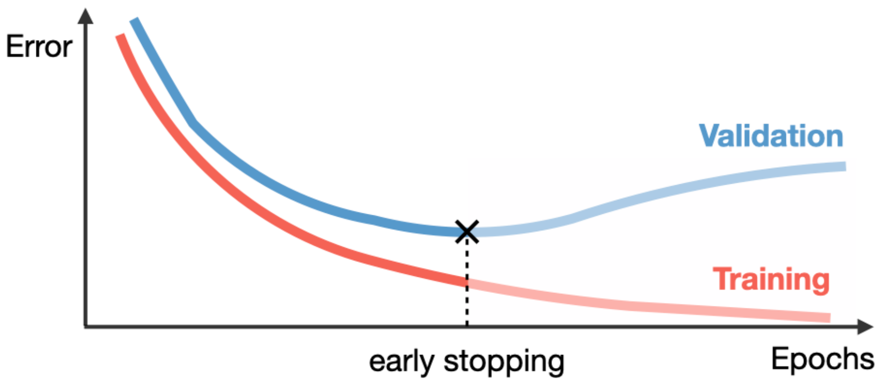


Figure 14: *Early stopping at the optimal validation loss during a training process (Amidi & Amidi, 2019)*

An interpretation of early stopping is that it puts a restriction on the distance between the parameters and the initial parameter values by halting the training procedure before the distance gets too large (Goodfellow et al., 2016). This is in close resemblance to the discussion regarding norm penalties in section 2.4.3, however a clear advantage of early stopping is that by monitoring the validation loss, we are able to stop the parameter updates at a particular good point in the parameter space (Goodfellow et al., 2016).

2.5. Validation of test results

The generalization capability of a model can only be measured on independent data not used in the model fitting procedure itself, thus we need to preserve a subset of the available samples for testing. In addition, hyperparameter search and early stopping requires the monitoring of the generalization loss, which requires us to preserve additional data for an independent validation set. The preservation of data for testing and validation is not a problem in situations where the amount of data is abundant, however this is rarely the case.

Both the test set and the validation set should be large enough to be representative of the population, which is unattainable for small datasets. In most situations, we have to make trade-offs between the sizes of the training, validation and test sets, which results in a test that consists of a small fraction of the available data. To validate the test results, it is therefore necessary to perform a cross validation procedure to gain confidence in that the model performance truly generalizes to independent data. I will discuss three such methods here, where the *k*l*-fold cross-validation with early stopping is most applicable for the problem at hand.

2.5.1. Leave-one-out cross-validation

Leave-one-out cross validation is an exhaustive cross validation procedure in which each sample is taking out of the dataset one time each and where the model is fitted on the remaining data and tested on the sample left out. For simple models this can be practical and easy to implement, and an advantage of leave-one-out cross-validation is that we are able to preserve a maximum number of samples for training. For deep learning models, however, this is not a viable option as the model fitting procedure is time consuming and computationally expensive.

2.5.2. *k*-fold cross-validation

k-fold cross validation is a validation method that implements exhaustive testing, while at the same time controlling the number of test procedures. k-fold can be considered in situations where only data sets for model fitting and testing are needed. In a deep learning situation, this applies when all hyperparameters are chosen prior to the model fitting procedure and the number of training epochs is preset. In k-fold cross-validation the dataset is split into k subsets, each subset is used as a test set once, and the model is fitted on the remaining $k - 1$ subsets. Algorithm 3 displays the procedure in detail. Note that the validation method is exhaustive in the sense that all available data are tested during validation.

Algorithm 3 k-fold cross-validation algorithm

```
$\mathbb{D} = \{\mathbb{D}_1, \ldots, \mathbb{D}_k\}$
for $i$ in 1 to $k$ do
    Obtain $f_i$: function fitted on $\mathbb{D} \setminus \mathbb{D}_i$
    Obtain $e_i$: $f_i$ performance evaluated on $\mathbb{D}_i$
end for
```

2.5.3. *k*l*-fold cross validation

*k*l*-fold cross-validation is an extension of the k-fold cross-validation procedure for situations where an additional independent validation set is needed. Early stopping (2.4.5) provides such an example, where we, in addition to selecting a test set prior to training, also reserve a subset for monitoring the loss.

In *k*l*-fold cross validation we divide the dataset into k subsets where each subset is used as a test set once. The data included in the $k - 1$ remaining subsets are further divided into l subsets, each of which is used as a validation set once. Finally, the model is fitted on the training set consisting of the remaining $l - 1$ subsets. This method results in $k \cdot l$ different training procedures with each example being tested l times.

Algorithm 4 below describes *k*l*-fold used with early stopping.

Algorithm 4 *k*l*-fold cross-validation algorithm with Early Stopping

```
$\mathbb{D} = \{\mathbb{D}_1, \cdots, \mathbb{D}_k\}$
for $i$ in 1 to $k$ do
    $\mathbb{D}^* = \mathbb{D} \setminus \mathbb{D}_i$
    $\mathbb{D}^* = \{\mathbb{D}^*_1, \ldots, \mathbb{D}^*_l\}$
    for $j$ in 1 to $l$ do
        Obtain $f_j$: function fitted on $\mathbb{D}^* \setminus \mathbb{D}^*_j$ using early stopping on $\mathbb{D}^*_j$
        Obtain $e_j$: $f_j$ performance evaluated on $\mathbb{D}_i$
    end for
end for
```

Note that by using $l = k - 1$, the *k*l*-fold cross-validation method is equivalent to dividing the dataset into k sets and trying out every possible combination for testing and validation.

2.6. Techniques for interpretability and explainability

Deep Learning models are often considered black boxes. To be confident in the models generalization capabilities, we must be able to analyze the relationship between the input features and the model output. The demonstrations by Lapuschkin et al. (2019) show several example of a type of model behavior dubbed as the Clever Hans syndrome, where a model makes its decision based on some image artifact not expected to be present in future images.

This section is concerned with techniques regarding the interpretation and explanation of a models decision. Using the definition by Biran and Cotton (2017), we define the model interpretability as the models ability to be understood by humans. Likewise, we define the models explainability as its ability to be explained by humans. In the search for explanations and interpretations, we are not only concerned with the models output, we are also concerned with the reasoning behind it.

A way of gaining insight into the models decision basis, is by analyzing the relevance of the input features to the model output. There exists several different methods for quantifying input feature relevance, all of which have in common that they aim at attributing scores to the features which contributes to the models decision. In this section I will cover four different methods used in visualization of pixel importance for deep learning image models. All these methods are based on the backpropagation of feature relevance. There exists other perturbation based methods that measures the effect of variations on the input, that will not be discussed here.

2.6.1. Gradient saliency maps

The simplest approach to evaluating pixel relevance is the gradient based saliency map introduced by Simonyan et al. (2014). Using Taylor's theorem (2.1.3.3), the first order approximation of $f(\mathbf{x})$ around $\mathbf{x}_0$ is given by

$$f(\mathbf{x}) = f(\mathbf{x}_0) + \left(\nabla_{\mathbf{x}} f(\mathbf{x})|_{\mathbf{x}=\mathbf{x}_0}\right)^T (\mathbf{x} - \mathbf{x}_0)$$

Given an input image $\mathcal{X}_0$, where $\mathbf{x}_0$ denotes a vector of all the pixel values, the neural network can be approximated by the linear function

$$f(\mathbf{x}) = \mathbf{w}^T \mathbf{x} + b$$

where the weights of the linear function is given by the gradient

$$\mathbf{w} = \nabla_{\mathbf{x}} f(\mathbf{x})|_{\mathbf{x}=\mathbf{x}_0}$$

and the bias is given by the constant

$$b = f(\mathbf{x}_0) - \left(\nabla_{\mathbf{x}} f(\mathbf{x})|_{\mathbf{x}=\mathbf{x}_0}\right)^T \mathbf{x}_0$$

Since the gradients constitute the weights of the function, the gradients in this sense represent the relative importance of each feature. I.e., the features with derivatives of the largest magnitudes, are the pixels that would change the output the most by being modified the least. In this approach the relevance attribution for input feature i is given by

$$R_i = \left.\frac{\partial f(\mathbf{x})}{\partial x_i}\right|_{\mathbf{x}=\mathbf{x}_0}$$

2.6.2. Guided Backpropagation

Guided backpropagation was introduced by Springenberg et al. (2015) in conjunction with a fully convolutional network using ReLU activations, and the method is identical to the procedure of computing gradients, except at the ReLU activations. In explaining the method we first note that the gradient of the output $a^{(L)}$ with regard to the input $\mathbf{x}$ is given by

$$\nabla_{\mathbf{x}} a^{(L)} = \left(\left(\frac{\partial \mathbf{a}^{(1)}}{\partial \mathbf{x}} \right) \left(\frac{\partial \mathbf{a}^{(2)}}{\partial \mathbf{a}^{(1)}} \right) \cdots \left(\frac{\partial \mathbf{a}^{(L-1)}}{\partial \mathbf{a}^{(L-2)}} \right) \right)^T \nabla_{\mathbf{a}^{(L-1)}} a^{(L)}$$

We also note that when using the ReLU activation function, each element in $\left(\frac{\partial \mathbf{a}^{(l+1)}}{\partial \mathbf{a}^{(l)}} \right)$ can be written as

$$\frac{\partial a_j^{(l+1)}}{\partial a_k^{(l)}} = w_{jk}^{(l+1)} \cdot f'(z_j^{(l+1)}) = \begin{cases} w_{jk}^{(l+1)}, & z_j^{(l+1)} > 0 \\ 0, & \text{otherwise} \end{cases}$$

Thus, ReLU activations ensures that only gradients corresponding to positive activations are propagated backwards. Note however that this does not require that the weights are positive, only that the sum

$$z_j^{(l+1)} = \sum_{k=1}^{n} w_{jk}^{(l+1)} a_k^{(l)} + b_j^{(l+1)}$$

is positive. Thus, it is still possible for activations contributing negative to the score to get attributed by backpropagation. In guided backpropagation however, this is avoided by the additional requirement

$$\frac{\partial a_j^{(L)}}{\partial a_k^{(l+1)}} > 0$$

resulting in the relevance attribution rule

$$R_j^{(l)} = \sum_k R_{jk}^{*(l+1)}$$

where

$$R_{jk}^{*(l+1)} = \begin{cases} w_{jk}^{(l+1)} \cdot R_k^{(l+1)}, & z_j^{(l+1)} > 0, \quad R_k^{(l+1)} > 0 \\ 0, & \text{otherwise} \end{cases}$$

Thus for a network with rectified linear units, guided backpropagation will give a zero attribution to any activation that at some point contributes negatively to the output score.

2.6.3. Layer-wise Relevance Propagation

Layer-wise Relevance Propagation (LRP) (Bach et al., 2015) is a relevance attribution method that is based on the principle of propagating relevance backwards in the network. The fundamental principle relies on the fact that each input feature is propagated forward layer by layer to the output, and by propagating a relevance score backwards, we can trace back the output contribution for each feature. The relevance backpropagation is not computed using gradients, instead the relevance is backpropagated according to a set of rules which aims at weighting the importance of each activation. The relevance of activation $a_j^{(l)}$, denoted by $R_j^{(l)}$, is assigned a proportion of the relevance $R_k^{(l+1)}$, such that the relevance is proportional to the influence $a_j^{(l)}$ has on the output $a_k^{(l+1)}$. The general LRP relevance rule is expressed by

$$R_j^{(l)} = \sum_{k=1}^{n_{l+1}} \frac{r_{jk}}{\sum_{j=1}^{n_l} r_{jk}} R_k^{(l+1)}$$

The basic LRP rule has a global and a layer-wise conservation property (Bach et al., 2015), $\sum_i R_i^{(0)} = \sum_i R_i^{(1)} = \cdots = f(\mathcal{X})$, in that the total relevance of the input are preserved throughout the model, summing up to equal the final output.

There exists several different rules that provides alternative expressions of the r_{jk} terms. I will address a selection of them here.

Basic rule (LRP-0)

Under the basic rule, the relevance of unit j to k is computed by its weighted contribution to unit k, normalized by the total output of unit k. The basic backpropagation rule is given by

$$R_j^{(l)} = \sum_{k=1}^{n_{l+1}} \frac{a_j^{(l)} w_{jk}^{(l+1)}}{z_k^{(l+1)}} R_k^{(l+1)}$$

Epsilon rule (LRP-ϵ)

The epsilon rule differs to the basic rule only in the addition of a small term, $\epsilon \geq 0$, in the denominator. This is a regulator, in that by increasing ϵ, we reduce the impact of neurons with small activations (Montavon et al., 2019). The epsilon rule is given by

$$R_j^{(l)} = \sum_{k=1}^{n_{l+1}} \frac{a_j^{(l)} w_{jk}^{(l+1)}}{\epsilon + z_k^{(l+1)}} R_k^{(l+1)}$$

Gamma rule (LRP-γ)

Under the gamma rule, positive contributions are favored by weighting positive contributions with a constant, $\gamma \geq 0$. The gamma rule is given by equation 11 where the $+$ sign notation indicates that the term is considered only if positive. Note that by letting $\gamma \to \infty$ we are disregarding negative contributions, making LRP-γ equivalent to another LRP-rule not addressed here, the LRP-$\alpha_1\beta_0$.

$$R_j^{(l-1)} = \sum_{k=1}^{n_l} \frac{a_j^{(l-1)} \left(w_{jk}^{(l)} + \gamma w_{jk}^{(l)+} \right)}{z_k^{(l)} + \gamma z_k^{(l)+}} R_k^{(l)} \tag{11}$$

An overview of LRP is provided by Montavon et al. (2019), where the basic rule is suggested for higher layers in the network, the epsilon rule for the middle layers in the network and the gamma rule for the lower layers in the network.

2.6.4. Integrated Gradients

Integrated gradients (Sundararajan et al., 2017) is a proposed feature relevance attribution method that is based on an axiomatic approach to determining input feature attributions. The idea is to compare the input at interest with a baseline input, and to determine the feature-wise contribution to the difference in model output for the two inputs. The authors propose that relevance attribution methods should comply with the following axioms:

- Sensitivity: A feature should be attributed if a change in that feature for the input results in a change in the output. If the network output doesn't depend on that feature, the attribution for that feature should be zero.
- Invariance: A feature relevance attribution method applied on different models with identical outputs for all identical inputs, should produce identical attributions for all models.
- Completeness: The sum of all feature attributions should equal the difference between the output of the model at the input, minus the output for the model at the baseline.
- Linearity: The attributions should preserve any linearity within the network. I.e., if the output is a weighted sum of two models, the attributions to the input should be a weighted sum of the attributions with respect to the two models.

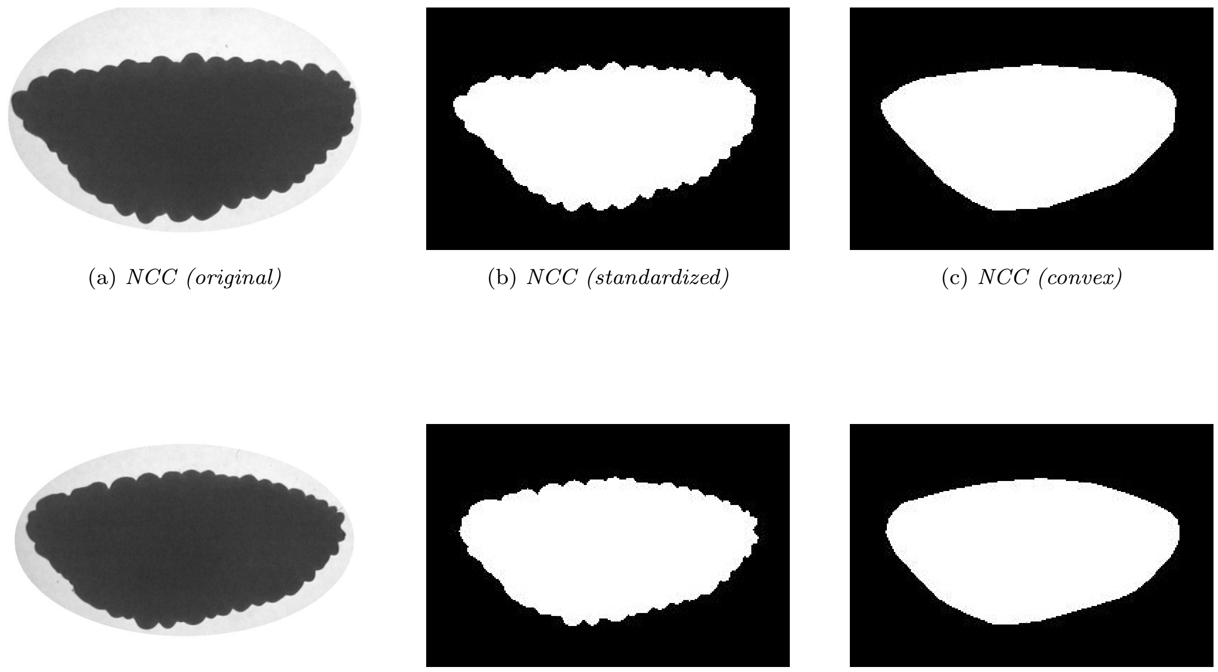
Integrated gradients is described by the following. Let $\mathbf{x} \in \mathbb{R}^n$ be an input vector, $\mathbf{x}' \in \mathbb{R}^n$ be a baseline vector and $f(\mathbf{x})$ be the output of a neural network for an input $\mathbf{x}$. The integrated gradients of $\mathbf{x}$ is defined as

$$\text{IntegratedGradients}_i(\mathbf{x}) \equiv (x_i - x_i') \cdot \int_{\alpha=0}^{1} \left. \frac{\partial f(\mathbf{y})}{\partial y_i} \right|_{\mathbf{y}=\mathbf{x}'+\alpha(\mathbf{x}-\mathbf{x}')} d\alpha \tag{12}$$

It is shown by Sundararajan et al. (2017) that Integrated Gradients does indeed satisfy the proposed axioms, and that a fundamental theoretical property of integrated gradients is the fact that the sum of all integrated gradients equals the score for the input, minus the score for the baseline, i.e.

$$\sum_{i=1}^{n} \text{IntegratedGradients}_i(\mathbf{x}) = f(\mathbf{x}) - f(\mathbf{x}')$$

Note that the integral in equation 12 is in practice computed using numerical integration. The fact that integrated gradients satisfy the completeness axiom can be used as a mean to check computations and measure the accuracy of the numerical integration.



(a) *NCC (original)* (b) *NCC (standardized)* (c) *NCC (convex)*

(d) *NEAC (original)* (e) *NEAC (standardized)* (f) *NEAC (convex)*

Figure 15: *Example images from the cod otolith dataset*

3. Examination of the datasets

In this section we will go through the two datasets subject to analysis in this thesis, and I will provide example images, plots and summary statistics that are considered relevant. I will address the set of cod otolith images first, before covering the set of Greenland halibut images.

3.1. Cod otolith data

The cod otolith data is based on 610 otolith images that are labeled by expert human readers. The original data consists of 367 Norwegian Coastal Cod (NCC) otolith images, and 243 North East Arctic Cod (NEAC) otolith images. There are two sets of images used in this thesis:

- The first set consists of binary standardized images that are reshaped such that every image has the same number of black and white pixels.
- The second set consists of convex images that are standardized in the same way, but where the ripples around the contour are removed using the *lasso* technique introduced by Harbitz and Albert (2015).

Figure 15 show a random example from each stock displaying the image types used. The processed images has a resolution of 256×256 pixels.

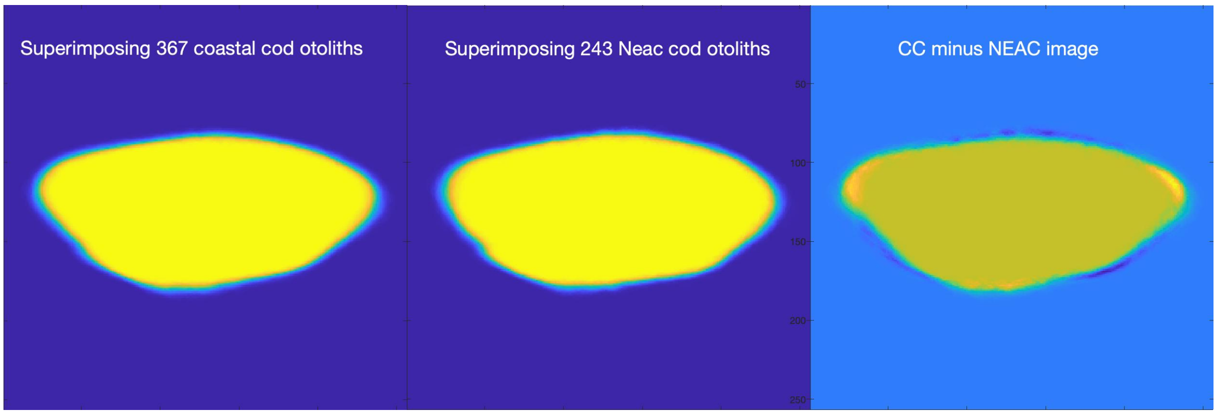


Figure 16: *Comparison of mean convex images produced by superimposing all standardized convex images for each stock (Alf Harbitz)*

Visual inspection of the images does not reveal any obvious pattern, however Stransky et al. (2008) managed to discriminate between the stocks obtaining good results using the convex images alone. Figure 16 displays mean convex images for both stocks, along with the mean difference between the standardized convex images. The images suggests a difference in shape that might be detectable using convex images alone, and a possible hypothesis is that the ripples can be interpreted as noise.

3.2. Greenland halibut otolith data

The Greenland halibut image dataset consists of 3540 images of Greenland halibut *(Reinhardtius hippoglossoides)* otoliths. The images are standardized with regard to height, and they are provided in RGB format with a resolution of 600×600 pixels. Each image is associated with a corresponding age label in the range 1 to 26, where the age is estimated by expert human readers. The samples are also associated with additional variables that consists of estimated sex and measured length. 1465 (41 %) of the data are males and 2075 (59 %) are females. Figure 17 displays a selection of of Greenland halibut otoliths ordered by age. The most evident growth pattern is the development of fingers and holes in the otolith as the fish grows older. The annual rings are not visible in all images, however we can observe a pattern in that the core size relative to the size of the otolith seems to decrease with age, and that the otolith seems to grow in an upwards direction. This is to be expected since the otolith increases in size as the fish grows older, while the core remains the same.

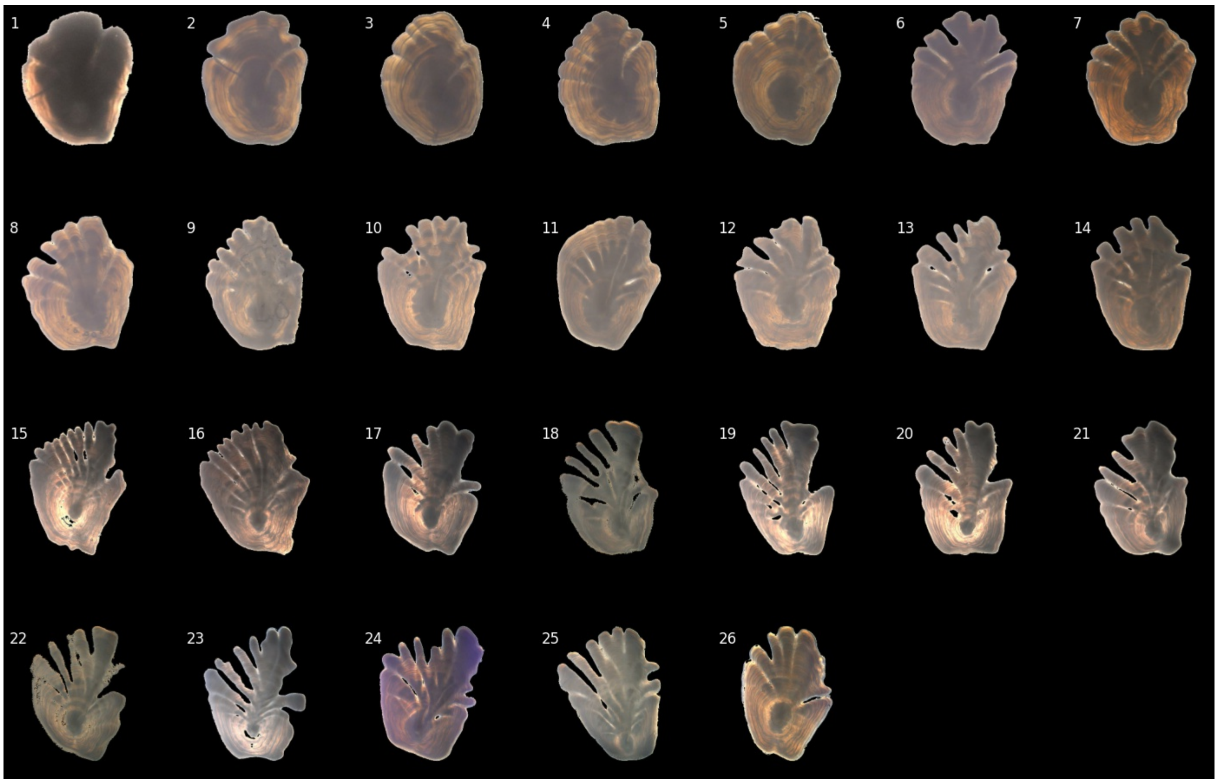


Figure 17: *A selection of Greenland halibut otoliths ordered by age*

Figure 18 shows a histogram of the read age along with age-wise image count for each sex. The age distributions for the two sexes is known to differ, which is supported by the figure. The two distributions has a rounded mean of 10 and 12 years respectively, a median of 10 and 13 years, and a standard deviation of 3.3 and 4.4. A Kolmogorov-Smirnov test (appendix B.2) with the equality of distributions as the null hypothesis, rejects the null hypothesis with a p-value of less then $2.2 \cdot 10^{-16}$.

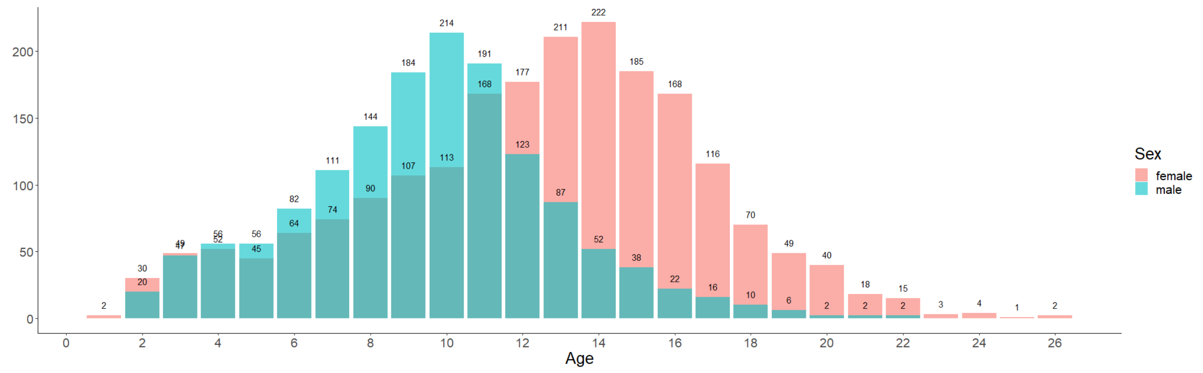


Figure 18: *Age distribution of the data for each sex*

The distribution of length for the two sexes is shown in figure 19. As with age, the distribution for the females is skewed to the right compared to the male distribution.

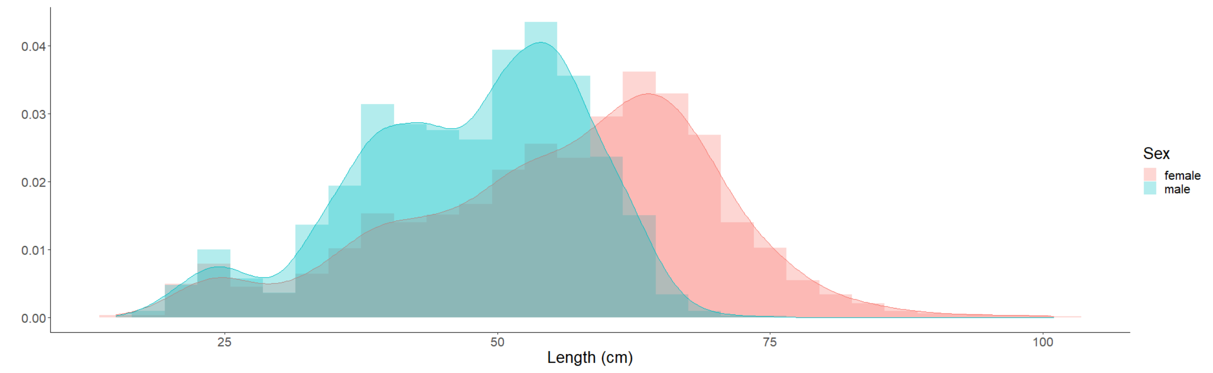


Figure 19: *Length distribution of the data for each sex*

3.2.0.1. The relationship between length and age

It is natural to investigate the correlation between length and age for fish as a preliminary analysis. Figure 20, 21 and 22 shows age plotted against length for both sexes and they suggest a relationship that appears to be close to linear.

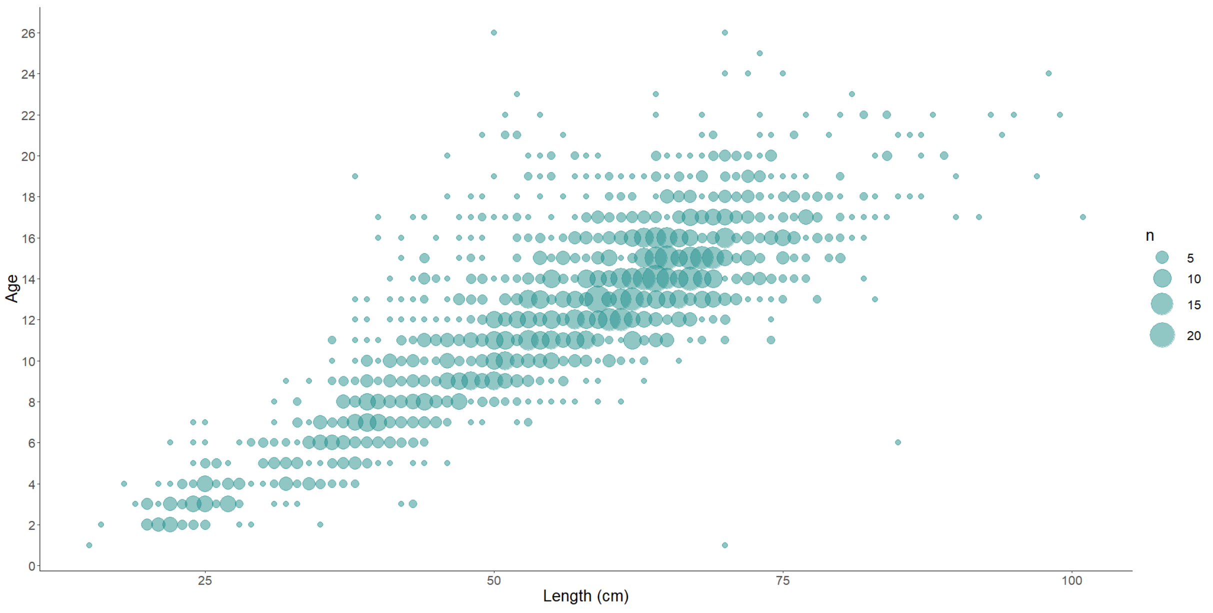


Figure 20: *Age vs length for females*

We will briefly discuss two regression models for modeling age by length.

Linear model

Let a_i and l_i be the age and length of sample i, and let z_i be a categorical variable where

$$z_i = \begin{cases} 1, \text{ if sample } i \text{ is a male} \\ 0, \text{ otherwise} \end{cases}$$

The linear regression model estimating age by length and sex is defined by

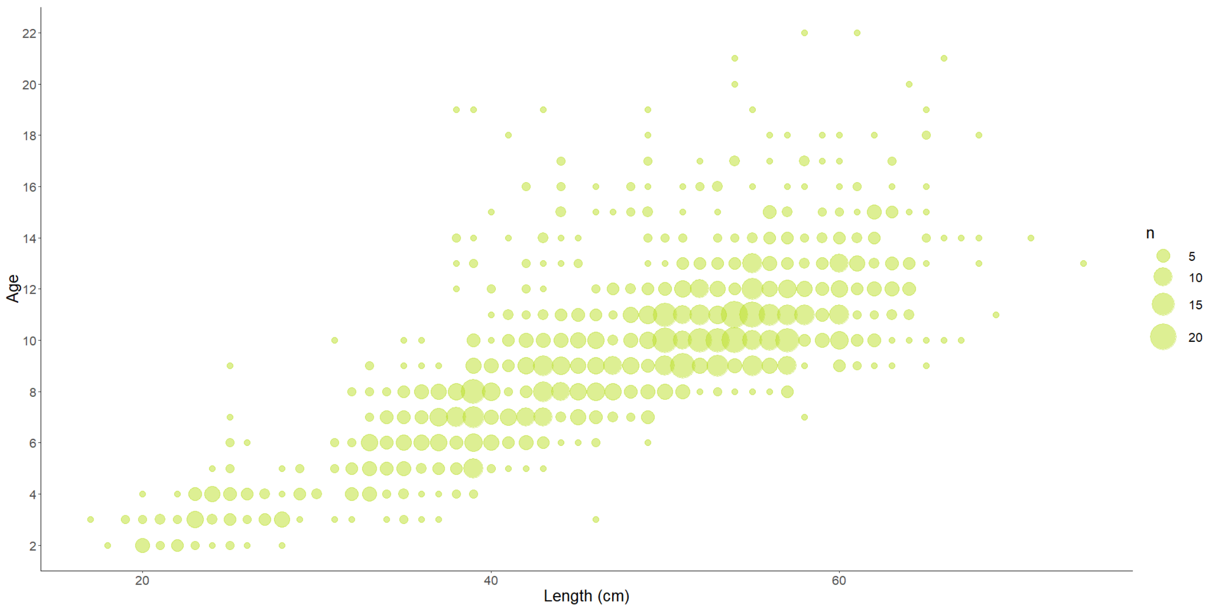


Figure 21: *Age vs length for males*

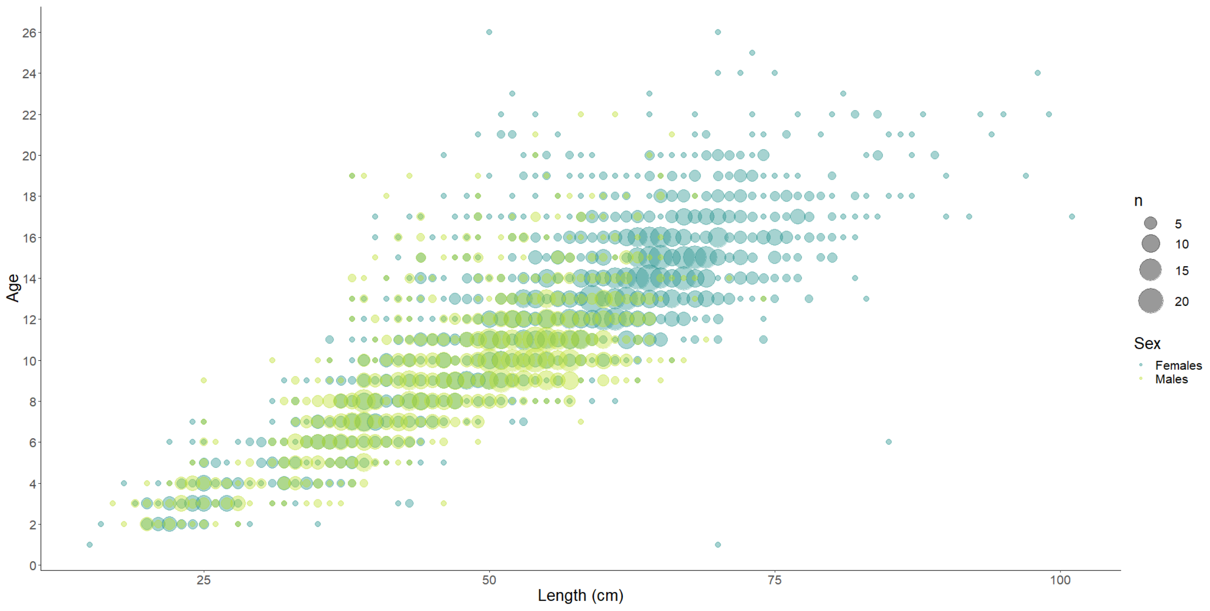


Figure 22: *Age vs length for both sexes*

$$E[A_i] = \beta_0 + \beta_1 l_i + (\beta_2 + \beta_3 l_i) z_i$$

where a test for the significance of the parameters was performed under the assumption of normally distributed error terms. β_0, β_1 and β_3 where all clearly significant, while β_2 was not, indicating equal intercepts for the two sexes. A proposed linear model is thus

$$E[A_i] = \beta_0 + \beta_1 l_i + \beta_2 l_i z_i \tag{13}$$

with a resulting MSE of 5.60 on 3537 degrees of freedom.

Non-linear model

An alternative to the linear model is to fit the well known von Bertalanffy growth function (VBGF) to the data. The VBGF is frequently used to model length as a function of age for fish, and relates mean length with age using the equation

$$l_i = L_\infty \left(1 - e^{-k(a_i - t_0)}\right) \tag{14}$$

where L_∞, k and t_0 are model parameters. Solving (14) for a_i results in an equation of the form

$$a_i = \beta_0 + \beta_1 \log\left(\frac{\beta_2 - l_i}{\beta_2}\right)$$

and if we again assume equal intercepts for both sexes, the proposed model is a nonlinear regression model with equation

$$E[A_i] = \beta_0 + \beta_1 \log\left(\frac{\gamma_1 - l_i}{\gamma_1}\right) + \beta_2 \log\left(\frac{\gamma_2 - l_i}{\gamma_2}\right) z_i$$

The parameters for this model is found numerically by minimizing the MSE between the read age and the age predicted by length. The MSE for this model is 5.61 on 3534 degrees of freedom, thus we see no need for not using the simpler linear model in this case.

4. Methods applied to otolith data

This section will address methods applied on the datasets. This includes discussions regarding the model formulations and the choice of loss functions, as well as chosen validation procedures and a discussion regarding pixel relevance attributions.

4.1. The statistical models and the choice of loss functions

4.1.1. Classification of cod otolith images

The cod otolith dataset described in section 3.1 consists of otolith images from two stocks; Norwegian Coastal Cod (NCC) and Northeast Arctic Cod (NEAC). Our aim is to discriminate between NCC and NEAC, which is a binary classification problem with the following statistical formulation.

Let $\{(X_i, y_i), i = 1, \ldots, n\}$ denote the set of image/label pairs, $y_i \in [0, 1]\ \forall i$, and let $f\left(X_i; \boldsymbol{\theta}\right)$ denote the output of a CNN with parameters $\boldsymbol{\theta}$. Assume that $y_i \sim \text{Bernoulli}(p_i)$, where p_i is a function of the associated image such that $p_i = f\left(X_i; \boldsymbol{\theta}\right)$. I.e., we assume that

$$y_i \sim \text{Bernoulli}\left(f\left(X_i; \boldsymbol{\theta}\right)\right)$$

where the predicted probability that sample i belongs to a NEAC, is given by $\hat{p}_i = f(X; \hat{\boldsymbol{\theta}}_{MLE})$. I.e., $\hat{p}_i$ is the maximum likelihood estimator (2.1.3.5) of p_i, where $\hat{\boldsymbol{\theta}}_{MLE}$ is the maximum likelihood estimator for $\boldsymbol{\theta}$, obtained by minimizing the binary cross-entropy loss function of $\hat{p}_i$ with regard to y_i (see appendix A.3.2 for details). Thus, we optimize the parameters with regard to the cross-entropoy loss

$$\text{CE}_{\text{binary}} = -\sum_{i=1}^{n} (1 - y_i) \log\left(1 - \hat{p}_i\right) + y_i \log \hat{p}_i$$

and we label sample i as a NEAC if $\hat{p}_i \geq 0.5$.

4.1.2. Regression on Greenland halibut otolith images

The Greenland halibut data described in section 3.2 includes the otolith images in addition to the variables age, sex and length. This provides some flexibility regarding the combination of explanatory variables. I will provide a brief discussion regarding regression before presenting a selection of proposed models which are subject to analysis in this thesis.

4.1.2.1. A note regarding classification versus regression

There are two ways to go about age estimation for images. One option, used by Moen et al. (2018), is to view the task as a regression problem. In this situation we are predicting a continuous quantity, namely the age, by a function usually optimized with regard to the Mean Squared Error between the read age and the prediction. The benefit of this approach is that we are able to utilize an assumed inherent relationship between the input images, namely that the closeness of two otolith images should depend upon the age difference between the two fish. Another option is to view the problem as a pure classification task as done by Ordoñez et al. (2020). In this situation we are outputting an estimated probability for each age. The benefit of a classification approach is that it is easy to convert class scores into probabilistic quantities, and as such we can gain insight into the certainty of the predictions. The obvious drawback is that we are completely disregarding the close relationship that might exist between otolith images of fish that are close in age. A problem regarding the discrimination between adolescents and adults is a problem where classification is justified. This thesis, however, is concerned with fitting a regression model using age as a continuous response variable.

4.1.2.2. Which variables to include and how to combine them

The age estimation of Greenland halibut is a regression problem where age is the response variable predicted by the otolith image. In addition, sex and length are included as categorical and continuous variables. I propose four models subject to further analysis. In these discussions the dataset is denoted by $\{(\mathcal{X}_i, y_i, l_i, z_i), i = 1, \dots, n\}$, where $\mathcal{X}_i$, y_i and l_i denotes the image, age and length of sample i respectively. z_i is a categorical variable where

$$z_i = \begin{cases} 1, \text{ if sample } i \text{ is a male} \\ 0, \text{ otherwise} \end{cases}$$

The weights of the neural network is denoted by $\boldsymbol{\theta}$.

4.1.2.3. A simple model

The first model is a simple CNN where we disregard sex and length. Figure 23 shows a diagram of the model which is expressed by

$$y_i = f(\mathcal{X}_i; \boldsymbol{\theta}) + \epsilon_i$$

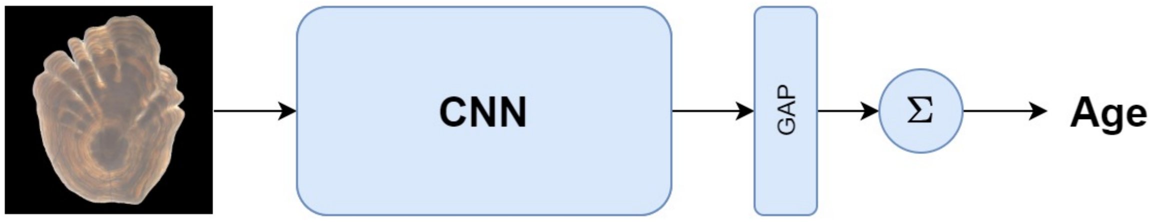


Figure 23: *A simple model that predicts age based on image inputs only*

The neural network consists of a CNN architecture followed by a GAP layer which is followed by a single output neuron with a ReLU activation function (A.2.4) mapping the output to a positive real number.

4.1.2.4. A linear model

In section 3.2.0.1 it was shown that a linear model is sufficient in modeling the relationship between age and length for this dataset. The proposed regression model using length as a predictor is the linear model

$$y_i = \beta_0 + \beta_1 l_i + \beta_2 l_i z_i + \epsilon_i \tag{15}$$

where the design matrix is given by

$$X = \begin{bmatrix} 1 & l_1 & l_1 \cdot z_1 \\ 1 & l_2 & l_2 \cdot z_2 \\ \vdots & \vdots & \vdots \\ 1 & l_n & l_n \cdot z_n \end{bmatrix}$$

For linear models, it is shown in appendix B.1 that the least squares age estimates are obtained by

$$\hat{\mathbf{y}} = X \left(X^T X\right)^{-1} X^T \mathbf{y}$$

4.1.2.5. A conditional model

Another proposed model is a conditional model using a CNN where the output is conditioned on the sex of the input. The model equation is given by

$$y_i = f(\mathcal{X}_i, z_i; \boldsymbol{\theta}) + \epsilon_i$$

and figure 24 shows a diagram of the model which consists of a CNN followed by a GAP layer followed by two output neurons that output the conditional predicted age. The final output is the dot product between the conditional outputs and the one-hot encoded categorical vector representing males by $[1 \;\; 0]^T$, and females by $[0 \;\; 1]^T$.

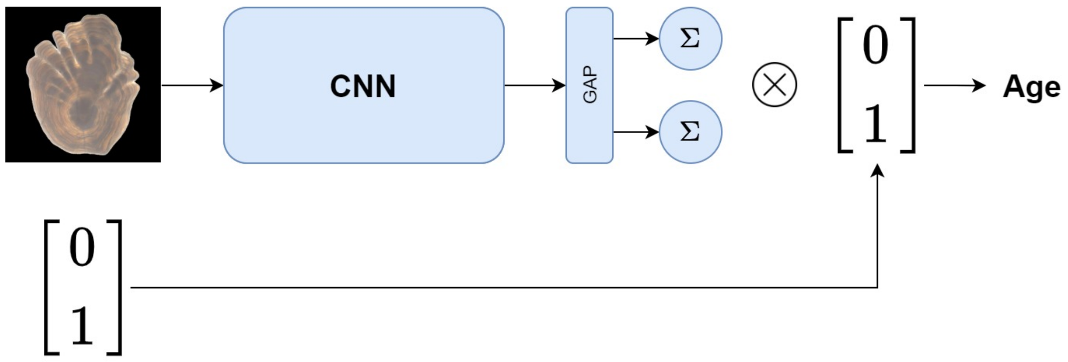

CNN
GAP
Σ
Σ
⊗
$\begin{bmatrix}0\\1\end{bmatrix}$
Age
$\begin{bmatrix}0\\1\end{bmatrix}$

Figure 24: *A model that conditions the age estimation on the sex of the input*

4.1.2.6. A combined model

The final proposed model is a model that combines the deep learning predictions from the conditional model in section 4.1.2.5, with the age predictions provided by the linear model in section 4.1.2.4. The regression model is given by

$$y_i = \beta_0 + \beta_1 f(\mathcal{X}_i, z_i; \hat{\boldsymbol{\theta}}) + \beta_2 l_i + \beta_3 l_i z_i + \epsilon_i$$

where the β parameters determines the optimal weighting of the deep learning predictions and the linear predictions.

If we let $\hat{y}_i$ denote the age predicted by the deep learning model in section 4.1.2.5, then the combined model has the design matrix

$$X = \begin{bmatrix} 1 & \hat{y}_1 & l_1 & l_1 \cdot z_1 \\ 1 & \hat{y}_2 & l_2 & l_2 \cdot z_2 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & \hat{y}_n & l_n & l_n \cdot z_n \end{bmatrix}$$

with the least squares age estimated obtained by

$$\hat{\mathbf{y}} = X \left(X^T X\right)^{-1} X^T \mathbf{y}$$

Figure 25 displays the combined model, which predicts the age by a weighted sum of the deep learning predictions in section 4.1.2.5, and the linear predictions in section 4.1.2.4.

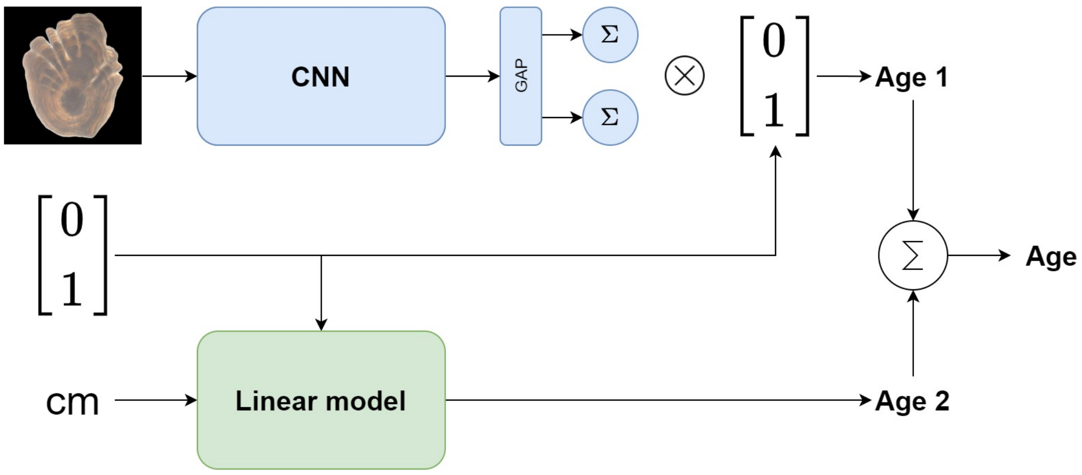

CNN
GAP
Σ
Σ
⊗
$\begin{bmatrix}0\\1\end{bmatrix}$
Age 1
$\begin{bmatrix}0\\1\end{bmatrix}$
Σ
Age
cm
Linear model
Age 2

Figure 25: *A model that combines image, sex and length in the prediction*

4.1.2.7. Minimizing the loss

All deep learning parameters in this thesis are optimized with regard to the MSE loss function described in section 2.1.3.4. It can be shown (see appendix A.3.1) that by assuming that $\epsilon_i \sim N(0, \sigma)$ for all i, the maximum likelihood estimator for y_i is obtained by finding the parameters $\boldsymbol{\theta}$ that minimizes the MSE loss function. Thus, the mean squared error is a natural loss function under the assumption of constant variance, however other loss functions can also be considered (A.3.1).

4.2. Choice of architecture for the deep learning models

Both the classification of cod otoliths and the age estimation of Greenland halibuts are under the supervised regime. Regardless of the details, each input has an associated label consisting of one or more real numbers, and the output of any model is thus restricted to that domain. This section will be concerned with the process of defining the CNN architecture used in this thesis.

4.2.1. Regarding optimizer functions

Section 2.2.6 provides a brief review of different variations in the gradient descent optimization algorithm. The last mentioned variation was the Adam optimizer (A.4.3), which provides a further improvement that is shown to outperform Adagrad, AdaDelta and RMSprop (Kingma & Ba, 2017). The Adam optimizer is considered an evolution of the former adaptive gradients method which themselves provided improvements to the basic gradient descent update rule. Ruder (2017) gives a thorough walk through addressing different optimizer functions, where Adam is suggested as the best overall choice. The Adam optimizer was used by Moen et al. (2018) and by Ordoñez et al. (2020), and I will not consider any other optimizer function in this thesis.

4.2.2. Designing your own network

Designing a CNN from scratch is a time consuming process. The number of possible specifications is seemingly endless, and trying them all is not a viable option. There exists however some heuristic principles that can serve as guidelines for the model building process, and figure 26 shows the fundamental design of a CNN; a feature extraction network followed by a classification network.

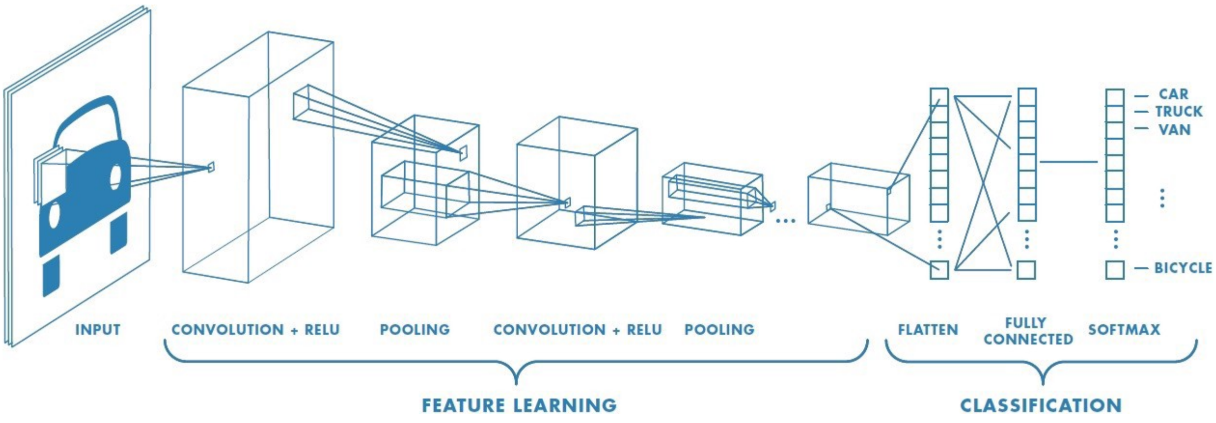


Figure 26: *A basic CNN (Medium, 2018)*

The following is a list of considerations that have to be taking into account when constructing a CNN from scratch:

- The feature extraction part of a CNN consists of a series of convolutional blocks. A convolutional block consists of one or more convolutional layers, followed by a downsampling layer. Both the number of convolutional blocks and the number of convolutional layers before downsampling can be considered hyperparameters. Note however that the input size determines the dimension of the hidden feature maps, and as a result of downsampling, the possible number of convolutional blocks is restricted by the size of the input.

- The list of hyperparameters includes the dimensions of the kernels, however this list can be reduced by using principles applied in networks such as Inception (Szegedy et al., 2014) and Xception (Chollet, 2017). Here the filter size is considered a constant with shape 3×3. These networks also applies batch normalization before every activation, and uses only a single fully connected layer at the output.

- The hidden activation functions must be chosen, and appendix A.2 provides a list of activation functions relevant to neural networks. ReLUs (A.2.4) are considered the state of the art, and are used for instance by Chollet (2017) and Szegedy et al. (2014). An argument for ReLUs is provided by vanishing gradients (A.5) for deep networks, which present a problem to which ReLUs provides a solution.

- Regularization methods such as weight decay (2.4.3) and dropout (2.3.2) should also be applied. This will not put restrictions on our model, since a regularization rate of zero is equivalent to a model without regularization. Srivastava et al. (2014) proposed dropout for fully connected layers only.

- At the end of the final convolution operation, the output feature maps needs to be reduced to a vector. This can be obtained either by a direct flattening of the convolutional output, or by applying a GAP layer (2.3.3).

The proposed default configurations are summarized by the following list:

- Set kernel dimensions to 3×3 for all kernels
- Apply batch normalization before all activations
- Apply a single fully connected layer after convolutions
- Use ReLU activations for all hidden activations
- Apply gradient descent using the Adam optimizer

The proposed set of remaining hyperparameters are summarized in the following list:

- Number of convolutional blocks
- Number of convolutional layer before downsampling
- The choice between regular convolutions and separable convolutions
- The choice between flattening and global average pooling
- The learning rate
- Weight decay rate and dropout rate

Note that the output activation and the loss function choice are problem specific.

4.2.3. Applying transfer learning

Transfer learning provides a framework for model fitting in situations where we assume that a model trained in one setting can be exploited to improve generalization in another setting (Goodfellow et al., 2016). For image classification tasks, this translates to assuming that a deep learning model trained on a set of images also will extract relevant features from a set of completely different images. A model trained on ImageNet (Deng et al., 2009) with more than 20,000 classes for instance, will presumably generalize easily to other images from classes not previously seen. In this scenario, we assume that the trained network will extract nearly all relevant image features, and that only the weighting of those features needs to adjusted.

All deep learning models in this thesis are fitted using the principles of transfer learning. In the paper by Moen et al. (2018), the regression was done using the Inception v3 architecture (Szegedy et al., 2015). In this thesis we use the Xception architecture described in section C.2. This is an evolution of the Inception models, and is shown by Chollet (2017) to slightly outperform Inception v3 on several datasets including ImageNet. The remaining hyperparameters for the thesis models are covered in section 5.

4.3. Validation of model performance

The results in this thesis are obtained by subsequent testing using $k{*}l$-fold cross-validation with early stopping (2.5.3). The fitting and validation procedure can be summarized by the following steps:

1. Split the data into training, validation and test sets.
2. Fit the model on the training set and use the performance on the validation set as a stoppage criterion.
3. Evaluate the fitted model on the test set.
4. Repeat the procedure for all k possible validation sets and the corresponding l possible test sets.

Regarding the training/testing/validation split

When splitting the data into training, validation and testing it is important to consider the size and the distribution of the sets. Ideally, each set should be distributed equally to the true underlying distribution of the data. The size of the test and validation sets should therefore ideally be large, which contradicts the need for a large training set. Thus, we have to make a trade-off between generalization capacity and test score precision. The details regarding the sample sizes for the two experiments are provided in section 5.

4.4. Techniques for visualizing pixel relevance

Section 2.6 addresses a selection of techniques aiming at attributing features with a score that reflects their importance on the output. Of the methods discussed there, gradient saliency maps (2.6.1), guided backpropagation (2.6.2) and integrated gradients (2.6.4) are used in this thesis, along with two closely related proposed alternatives - baseline gradients and integrated guided gradients - which will be introduced here. In addition, a theoretical example comparing different techniques is provided.

4.4.1. Baseline gradients: a gradient approach using baseline inputs

The gradient based method in section 2.6.1 is based on an argument involving Taylor's theorem. Another application of Taylor's theorem can be applied as follows. Let $f(\cdot)$ be the function defined by our neural network and let $\mathcal{X}$ and $\mathcal{X}_0$ be an input image and a baseline image respectively with $\mathcal{H} = \mathcal{X} - \mathcal{X}_0$. Using a first order Taylor approximation (2.1.3.3), we can write $f(\mathbf{x})$ as

$$\begin{aligned} f(\mathbf{x}-\mathbf{h}) &= f(\mathbf{x}) + \left(\nabla_{\mathbf{x}} f\right)^T \mathbf{h} \\ f(\mathbf{x}_0) &= f(\mathbf{x}) + \left(\nabla_{\mathbf{x}} f\right)^T (\mathbf{x}-\mathbf{x}_0) \\ &\Rightarrow \\ f(\mathbf{x}) - f(\mathbf{x}_0) &= \left(\nabla_{\mathbf{x}} f\right)^T (\mathbf{x}-\mathbf{x}_0) \end{aligned}$$

resulting in the relevance attribution rule

$$R_i = \frac{\partial f(\mathbf{x})}{\partial x_i} \cdot (x_i - x_{0,i})$$

This is nearly identical to the gradient saliency maps described in section 2.6.1. The important difference is that we are estimating the pixel-wise contribution to the output difference, where the difference is measured with regard to the baseline. In situations where an all-black baseline image is appropriate, this simplifies to

$$f(\mathbf{x}) - f(\mathbf{x}_0) = \left(\nabla_{\mathbf{x}} f\right)^T \mathbf{x}$$

thus, we can guarantee that only nonzero features are given nonzero scores.

4.4.2. A simple comparison of feature attribution methods

Consider a function defined by

$$f(x_1, x_2) = \sigma(x_1) + x_2$$

where σ denotes the logistic sigmoid function described in section A.2 having the property

$$\frac{\partial \sigma(x)}{\partial x} = \sigma(x)(1-\sigma(x))$$

The feature relevance attribution provided by the gradient of f (2.6.1) is thus given by

$$\text{Gradients}(\mathbf{x}) = \begin{bmatrix} \sigma(x_1)\,(1-\sigma(x_1)) \\ 1 \end{bmatrix}$$

and the baseline gradients are given by

$$\text{BaselineGradients}(\mathbf{x}) = \begin{bmatrix} \sigma(x_1)\,(1-\sigma(x_1)) \cdot (x_1 - x_1') \\ (x_2 - x_2') \end{bmatrix}$$

The integrated gradient for the first feature is given by

$$\begin{aligned} \text{IntegratedGradients}(x_1) &= (x_1 - x_1') \cdot \int_0^1 \sigma(x_1' + \alpha(x_1 - x_1'))(1 - \sigma(x_1' + \alpha(x_1 - x_1')))d\alpha \\ &= (x_1 - x_1') \cdot \frac{1}{x_1 - x_1'} \cdot \sigma(x_1' + \alpha(x_1 - x_1'))\Big|_0^1 \\ &= \sigma(x_1) - \sigma(x_1') \end{aligned}$$

and the integrated gradient for second feature is given by

$$\begin{aligned} \text{IntegratedGradients}(x_2) &= (x_2 - x_2') \cdot \int_0^1 d\alpha \\ &= x_2 - x_2' \end{aligned}$$

To summarize we have the following relevance attributions for the three methods

$$\begin{aligned} \text{Gradients}(\mathbf{x}) &= \begin{bmatrix} \sigma(x_1)\,(1-\sigma(x_1)) \\ 1 \end{bmatrix} \\ \text{BaselineGradients}(\mathbf{x}) &= \begin{bmatrix} \sigma(x_1)\,(1-\sigma(x_1)) \cdot (x_1 - x_1') \\ (x_2 - x_2') \end{bmatrix} \\ \text{IntegratedGradients}(\mathbf{x}) &= \begin{bmatrix} \sigma(x_1) - \sigma(x_1') \\ (x_2 - x_2') \end{bmatrix} \end{aligned}$$

Now consider an input where x_1 is large and x_2 is zero, i.e. $x_1 \to \infty$. The first term will converge to 1, while the second term will be 0. The feature relevance attributions however, will differ for the three methods. Comparing to a baseline input of $(0, 0)$ and using the fact that

$$\lim_{x \to \infty} \sigma(x)(1 - \sigma(x)) = 0$$

and

$$\lim_{x \to \infty} \sigma(x)\,(1 - \sigma(x)) \cdot x = 0$$

the feature relevance attributions are

$$\begin{aligned}\text{Gradients}(\mathbf{x}) &\rightarrow \begin{bmatrix}0\\1\end{bmatrix}\\ \text{BaselineGradients}(\mathbf{x}) &\rightarrow \begin{bmatrix}0\\0\end{bmatrix}\\ \text{IntegratedGradients}(\mathbf{x}) &\rightarrow \begin{bmatrix}1\\0\end{bmatrix}\end{aligned}$$

Thus, we see that the gradient based methods fails to capture what the integrated gradients does; namely that only the first feature contributes to the function output. This example suggests an inherent weakness in gradient saliency maps, and it is argued by Sundararajan et al. (2017) that integrated gradients does possess better theoretical properties.

4.4.3. Integrated guided gradients

The last example did not include guided backpropagation (2.6.2), however since guided backpropagation is a modification of gradient saliency maps, similar problems as those demonstrated for gradient based methods will be present in the guided backpropagation case as well. A proposed alternative to guided backpropagation is therefore provided by combining integrated gradients with guided backpropagation. If we let $R(x_i)$ denote the relevance attribution for feature i, obtained by guided backpropagation, then the integrated guided gradient of feature i is given by

$$\text{IntegratedGuidedGradients}_i(\mathbf{x}) = (x_i - x_i') \cdot \int_0^1 R(x_i' + \alpha(x_i - x_i'))d\alpha$$

This method attempts to combine the sparse attributions given by guided backpropagation, with the good theoretical properties of integrated gradients.

4.4.4. Choosing the right baseline

Choosing the right baseline image is of paramount importance, as the baseline image should be uninformative. Natural proposals include all-black images, all-white images, noise images etc. In a classification situation this means that the baseline image should yield an equal score for all classes, which is not easy to achieve in practice. A common problem is that a seemingly uninformative baseline results in high scores for some classes, and low scores for others. One reason for biased baseline predictions might be due to unbalanced sets, resulting in a baseline image which are predicted to the majority class. It is suggested by Taly (2018) to add an additional class consisting of baseline images alone to prevent unbalanced baseline predictions to occur. By using baseline images as positives for an additional class, we ensure that the score is close to zero for all other classes when a baseline image is the input. Note that the problem of biased baseline predictions is not specific for classification networks, as we in regression situations face a similar issue. Here we want our baseline image to yield an uninformative output, but we often end up with a baseline output equal to the mean of the data. Again, this can be counteracted by including baseline images with label values that are not present in the dataset. A suggestion if the original dataset consists of positive real numbers, is to add an additional baseline class with labels equal to zero. This is the approach used in this thesis.

5. Experimental procedure and results

This section will cover the results for the otolith experiments. This includes relevant test statistics that measure model performances, along with statistics used to analyze model uncertainties. I also will present pixel relevance heatmaps using the techniques discussed in section 4.4. The two experiments are addressed separately. Section 5.1 covers the results regarding the classification of cod otoliths, while section 5.2 covers results regarding the Greenland halibut regression problem.

I will apply the following definitions of accuracy and recall in the report of the results

$$\text{accuracy} = \frac{1}{n}\sum_{i=1}^{n} I\{y_i = \hat{y}_i\}$$
$$\text{recall}_{\text{class k}} = \frac{\sum_{i=1}^{n} I\{y_i = k \cap \hat{y}_i = k\}}{\sum_{i=1}^{n} I\{y_i = k\}}$$

All the density curves used in the following sections are obtained by the kernel density estimation method described in appendix B.3.

5.1. The classification of cod otoliths

5.1.1. Model and hyperparameters

Base model

The CNN trained for cod stock discrimination was fitted using transfer learning, with the Xception (appendix C.2) architecture as the base model. The base model was fitted on the ImageNet dataset (Deng et al., 2009). Global average pooling was applied after feature extraction, with a single dense output unit at the end. The output activation function was the sigmoid function (appendix A.2), which maps the output to the range 0 to 1. The Adam optimizer (A.4.3) was used for optimization, and the binary cross-entropy (A.3) was used to compute the loss.

The model was trained using a two-step procedure

1. Freeze the base model and train only the output layer with a learning rate of 0.001.

2. Unfreeze the base model and fine tune all layers with a learning rate of 0.00001.

Batch normalization layers were frozen during the whole training procedure to avoid ruining the patterns learned during pre-training.

Preprocessing

The input to the model were the standardized binary cod otolith images displayed in figure 15 with size 128×128. Since the base model requires 3 channel inputs, each image was repeated across the channels resulting in an input size of (128, 128, 3). The input size resulted in a global average pooling input of size 4×4, as opposed 10×10 feature maps in the original article, which used inputs of size 299×299. No data augmentation was applied.

Regularization

Dropout was applied between the GAP layer and the output layer. The dropout rate was set to 0.2 without any further hyperparameter tuning. The model was also fitted using a weight decay of 1e-5, which is the standard configuration for the Xception architecture.

Validation

The 610 cod otolith images were split into 5 subsets of size 122 using proportional stratified allocation based on class labels (section 2.1.3.1). Every possible validation/test combination was tried out. The procedure is equivalent to the k*l-fold cross validation from section 2.5.3 with $l = k - 1$. Thus the number of trials was 20 in total, with each individual image being tested 4 times.

A note about the score

The sigmoid output estimates the probability that a sample is a NEAC. Thus, sample i is classified as NEAC if $\hat{p} \geq \frac{1}{2}$. This is equivalent to using the allocation rule

$$X_i \in \begin{cases} \text{NEAC}, & \text{score}(X_i) \geq 0 \\ \text{NCC}, & \text{score}(X_i) < 0 \end{cases}$$

where the score is computed - using the inverse of the sigmoid function - by

$$\text{score}(X_i) = -\log\left(\frac{1 - \hat{p_i}}{\hat{p_i}}\right)$$

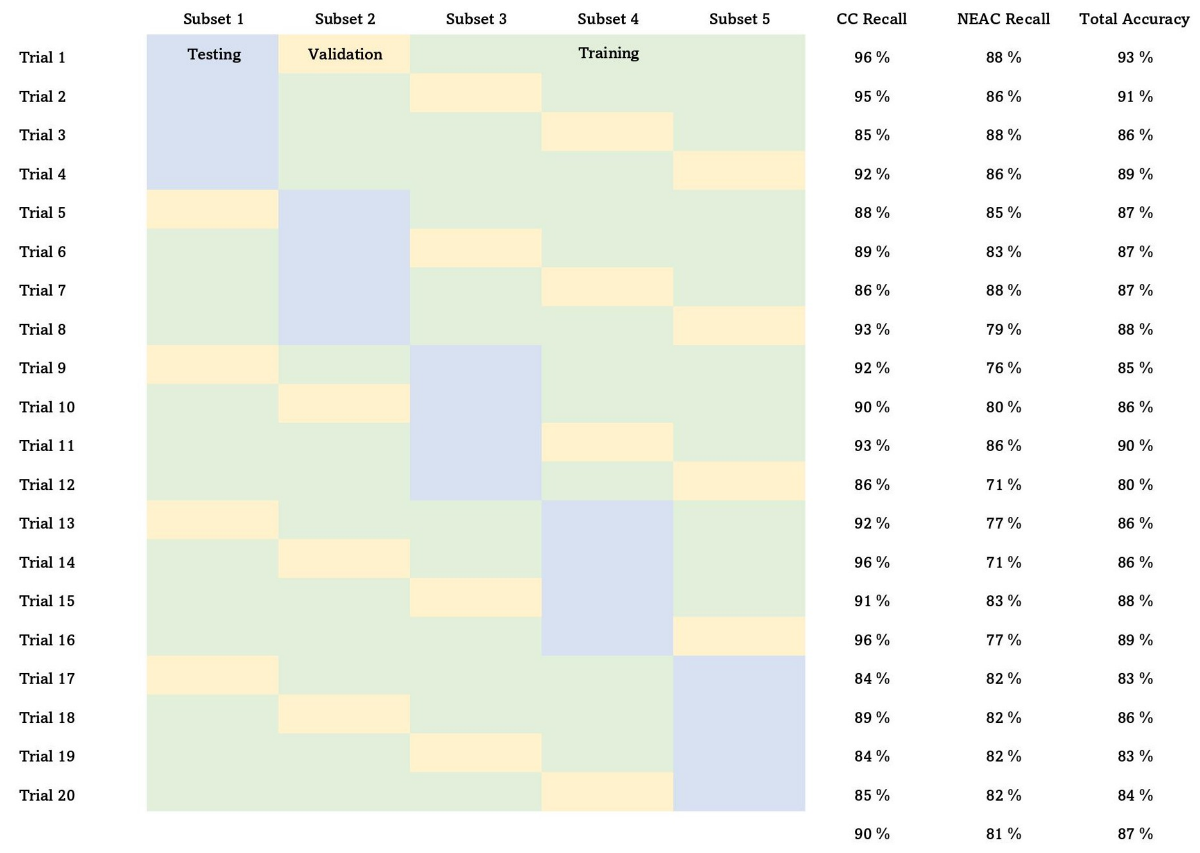
All the plots and density estimates are based on the score.

5.1.2. Model performance

Figure 27 shows the results for the cross-validation procedure. On average over the 20 trials, the measured performance was

- 90 % (84-96 %) recall for NCC
- 81 % (71-88 %) recall for NEAC
- 87 % (80-93 %) total accuracy

which displays significant variations in the test results.



	Subset 1	Subset 2	Subset 3	Subset 4	Subset 5	CC Recall	NEAC Recall	Total Accuracy
Trial 1	🟦 Testing	🟨 Validation	🟩	🟩 Training	🟩	96 %	88 %	93 %
Trial 2	🟦	🟩	🟨	🟩	🟩	95 %	86 %	91 %
Trial 3	🟦	🟩	🟩	🟨	🟩	85 %	88 %	86 %
Trial 4	🟦	🟩	🟩	🟩	🟨	92 %	86 %	89 %
Trial 5	🟨	🟦	🟩	🟩	🟩	88 %	85 %	87 %
Trial 6	🟩	🟦	🟨	🟩	🟩	89 %	83 %	87 %
Trial 7	🟩	🟦	🟩	🟨	🟩	86 %	88 %	87 %
Trial 8	🟩	🟦	🟩	🟩	🟨	93 %	79 %	88 %
Trial 9	🟨	🟩	🟦	🟩	🟩	92 %	76 %	85 %
Trial 10	🟩	🟨	🟦	🟩	🟩	90 %	80 %	86 %
Trial 11	🟩	🟩	🟦	🟨	🟩	93 %	86 %	90 %
Trial 12	🟩	🟩	🟦	🟩	🟨	86 %	71 %	80 %
Trial 13	🟨	🟩	🟩	🟦	🟩	92 %	77 %	86 %
Trial 14	🟩	🟨	🟩	🟦	🟩	96 %	71 %	86 %
Trial 15	🟩	🟩	🟨	🟦	🟩	91 %	83 %	88 %
Trial 16	🟩	🟩	🟩	🟦	🟨	96 %	77 %	89 %
Trial 17	🟨	🟩	🟩	🟩	🟦	84 %	82 %	83 %
Trial 18	🟩	🟨	🟩	🟩	🟦	89 %	82 %	86 %
Trial 19	🟩	🟩	🟨	🟩	🟦	84 %	82 %	83 %
Trial 20	🟩	🟩	🟩	🟨	🟦	85 %	82 %	84 %
						90 %	81 %	87 %

Figure 27: *Overview of trial results for the classifcation of cod otolith images*

Comparison of results

The results presented by Stransky et al. (2008) based on EFDs with 10 frequency components showed a recall of 90 % for NCC and 87 % for NEAC based on the same otolith images as in this thesis. In the work by Stransky et al. (2008), the fish length was restricted to 30-70 cm to minimize the effect of length. By restricting the EFD analysis even further, so that exactly the same length distribution as well as the same sex distribution were used for each stock (210x2 = 420 otoliths), the EFD results gave 86 % and 81 % average recall for NCC and NEAC respectively, based on 1000 simulated balanced sets by bootstrapping.

At the NORDSTAT 2021 conference, classification results obtained by Alf Harbitz (IMR), Iver Martinsen (UiT) and Filippo Maria Bianchi (UiT), that were based on a combination of classification methods, were presented. The methods used was EFDs (Kuhl & Giardina, 1982), MIRR (Harbitz, 2016) and deep learning. The results, presented in table 5 in appendix D.1, showed a significant increase in performance when combining deep learning with other methods, and the recall presented by combining EFDs, MIRR and deep learning was 93 % for NCC, and 86 % for NEAC. A cross-validation procedure identical to the procedure displayed in figure 27 was conducted for the deep learning test results, where the deep learning model was built from scratch using the principles in section 4.2.2. These results are presented in figure 51, and are very similar to the results presented in figure 27.

Ripples vs no ripples

The validation procedure presented in figure 27 was repeated using the convex images displayed in figure 15 with a total accuracy of approximately 85 %. Figure 28 displays the distribution of the paired score differences for the two stocks. If we let $y_{1,i}$ be the score of sample i obtained using standardized images with ripples, and $y_{2,i}$ be the score obtained using convex images, the mean difference $\overline{y}_1 - \overline{y}_2$ were -0.54 for NCC, 0.46 for NEAC.

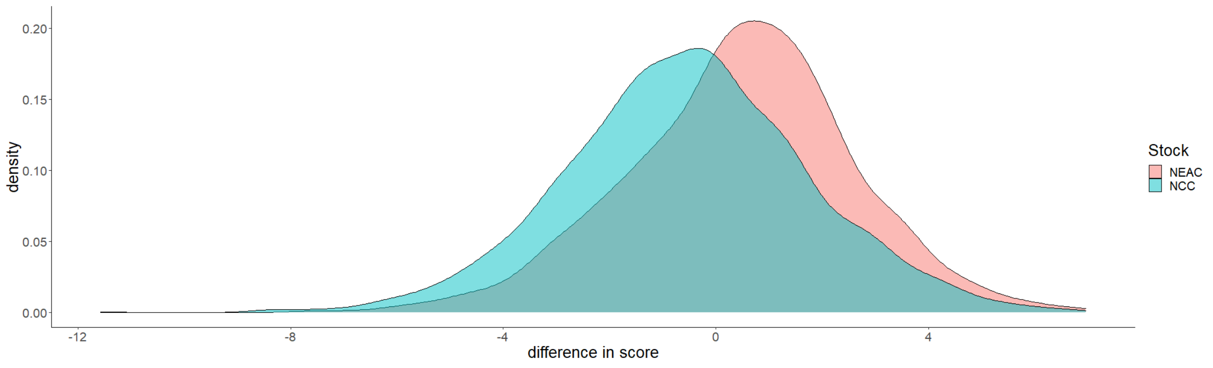


Figure 28: *Distribution of score differences for standardized vs convex images*

The significance of the mean differences can be assessed by hypothesis testing. If we assume that $\delta_i = y_{1,i} - y_{2,i} \overset{iid}{\sim} N(\mu, \sigma)$, we can test the null hypothesis

$$\mu = 0$$

against the alternative hypothesis

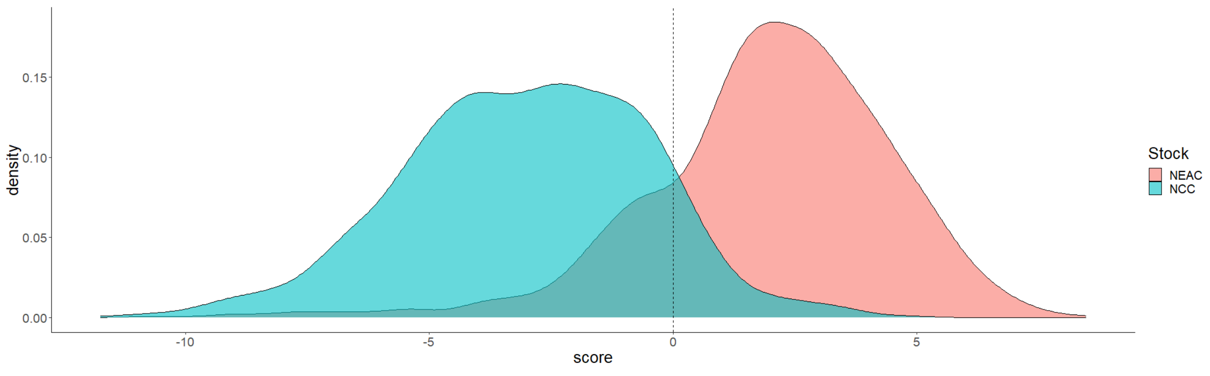
$$\mu < 0$$

to determine if a significant difference in the means are present. The null hypothesis was rejected with a p-value of $6.79 \cdot 10^{-9}$ for a difference in negative scores, while a similar test regarding the positive scores resulted in a p-value of $1.02 \cdot 10^{-7}$. This clearly indicates that there is significant information in the ripples regarding stock affiliation.

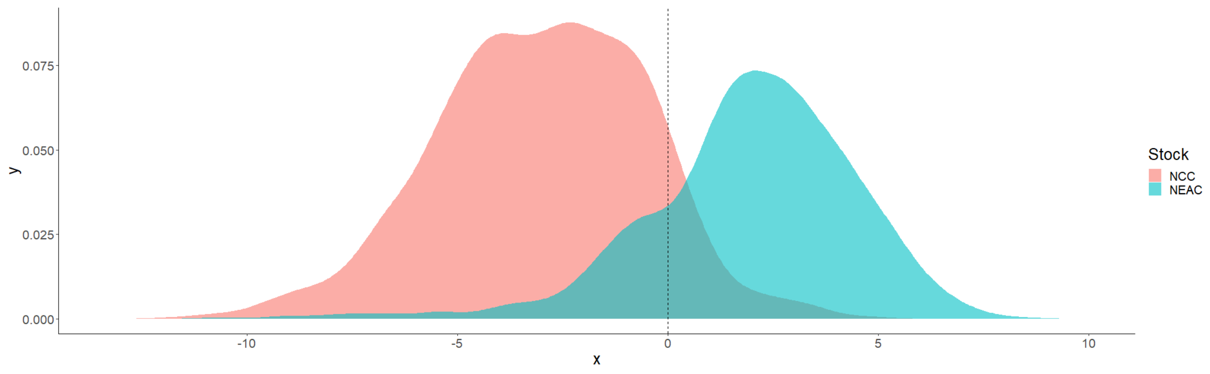
5.1.3. Model uncertainty

Model bias

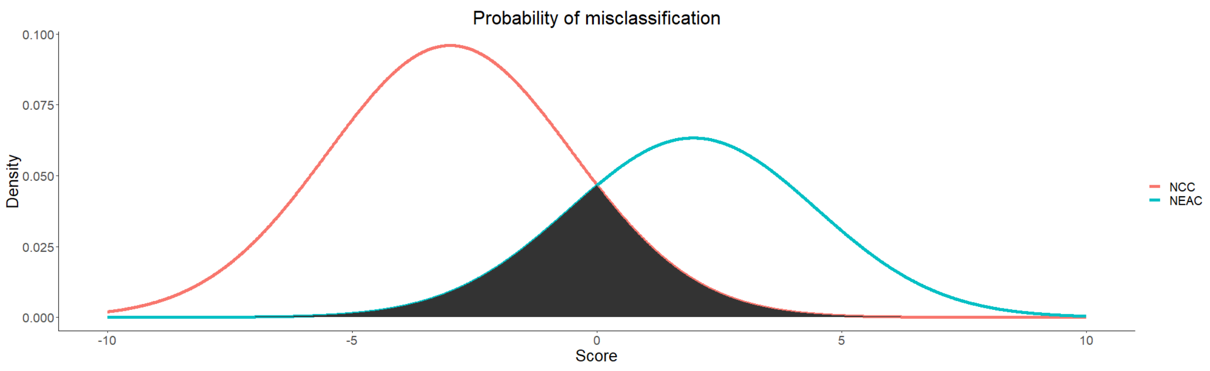
Figure 29 (a) shows the score distribution for all the cod otolith classification scores. The mean scores was -3.00 for NCC and 1.97 for NEAC. The total variance was 6.25 for NCC and 6.30 for NEAC. Thus, we note that the curves are approximately normal with unequal means, and equal variances. From the figure we see that the probability of a misclassification is largest for NEAC. Assuming normal distributions, the computed probability of misclassification is 0.115 for NCC and 0.216 for NEAC.



(a) *Density estimates scaled to 1*



(b) *Density estimates weighted according to prior ratio*



(c) *TPM under the assumption of normal distributions*

Figure 29: *Distribution of scores for cod otolith outputs*

Figure 29 (b) shows the density estimates scaled according to the prior ratio, which was approximately 0.6 for NCC and 0.4 for NEAC. Computations of the areas of the misclassified regions results in a value of 0.060 for NCC vs a value of 0.074 for NEAC.

Figure 29 (c) shows the total probability of misclassification (TPM) for the cod data under the assumption of normally distributed scores. It as apparent from the figure that a decision boundary at zero is optimal with regard to minimizing the TPM.

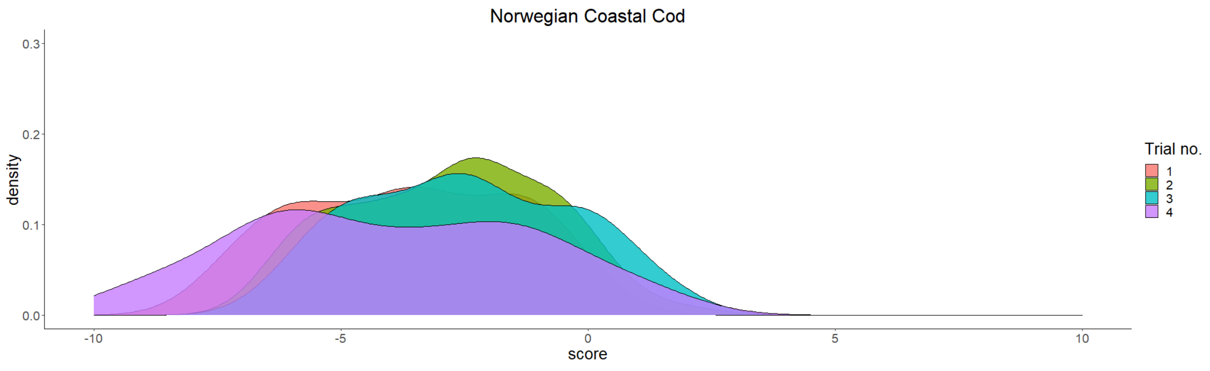
These results imply that the majority of the bias with regard to class-wise performance can be attributed to the imbalance of the data. Observations of the training procedure shows that the first step of the network is to adopt a allocation rule that classifies all samples to the majority class. This is natural, as this will indeed produce the lowest loss when the model has yet to extract any significant feature patterns from the images. An interpretation of this is that the network classifies all samples to the majority class unless presented with clear evidence of the opposite. Thus, a performance bias might serve as evidence of a model that has not yet fully converged, and in turn fails to predict the data properly.

This effect could be interesting to investigate further. One suggestion to counteract this effect is to introduce a dummy majority class, with the idea that the dummy class will absorb the initial bias. Another proposal that is frequently used, is to resample the minority class such that the network is trained with an equal amount of samples from each class.

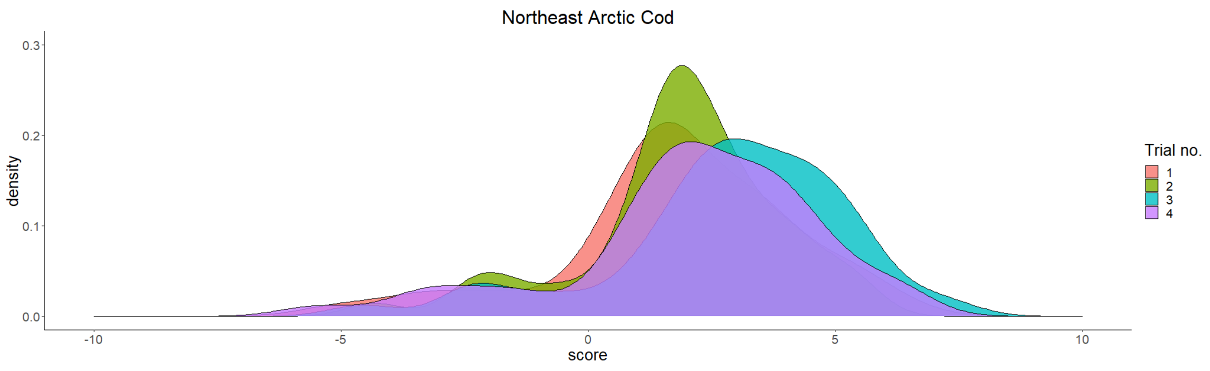
Test result variability

In figure 27 we see that the performance varies across trials. Trial 1 resulted in a recall of 96 % and 88 % for NCC and NEAC respectively, while trial 12 resulted in a recall of 86 % and 71 % for the two stocks. Thus, test results are subject to some randomness that greatly affect the performance.

Figure 30 (a) and (b) shows score distributions for repeated trials using the same test set.



(a) *Distribution of NCC scores for repeated trials*



(b) *Distribution of NEAC scores for repeated trials*

Figure 30: *Score distributions for repeated trials using the same test set*

We see from the figures that the distribution of the data is to some extent sensitive to the training and validation data selection. An interesting question is the following: How much of the total variation in the test results are attributed to the choice of a test, and how much is attributed to the choice of a validation set? We can try to answer this by looking at the two sources of variability in this trial:

1. Uncertainty as a result of the model fitting procedure itself, which includes the choice of a validation set and the procedure of parameter initialization and early stopping. In ANOVA this is often referred to as within treatment uncertainty (Walpole et al., 2007). I will refer to this as model variability.

2. Uncertainty as a result of variations between otoliths themselves. This variability is attributed to the choice of test samples. In ANOVA, this is called between treatment uncertainty. I will refer to this as sample variability.

Table 1 shows the total test variations split into model variability and sample variability using the relation

$$\sum_{i=1}^{n}\sum_{j=1^k}(\hat{y}_{ij}-\overline{y}_{..})^2 = k\sum_{i=1}^{n}(\overline{y}_{i.}-\overline{y}_{..})^2+\sum_{i=1}^{n}\sum_{j=1}^{k}(\hat{y}_{ij}-\overline{y}_{..})^2$$

with $k = 4$ trials and $n = 610$ samples.

	Total variance		Between-samples		Between-models	
	SST	s^2_{tot}	SSA	s^2	SSE	s^2_{pooled}
NCC	9171.48	6.25	7984.03	5.45	1187.45	1.08
NEAC	6125.98	6.30	5625.92	5.81	490.06	0.67

Table 1: *Sum of squares decomposition for the cod otolith scores*

We are not testing for any treatment effect here, as we expect the individual otoliths to vary in readability, but we can nevertheless observe that most of the total variability can be attributed to the variability of the test samples. This is supported by the fact that 539 of 610 otoliths was unambiguously classified.

5.1.4. Explaining decisions by heatmaps of pixel relevance

Figure 31 shows a comparison of three different heatmaps displaying pixel relevance estimations for a selection of NCC and NEAC where intensity values appearing outside the border of the guided backpropagation heatmaps are set to zero. These heatmaps display an estimated contribution to the output for each pixel, i.e. if an image has en estimated probability of 0.9 to be a NEAC image, the heatmaps display each pixel's contribution to the 0.9 output. Black images were used as baseline images (see section 4.4.4), serving as members of a third baseline class. From the images we see several things:

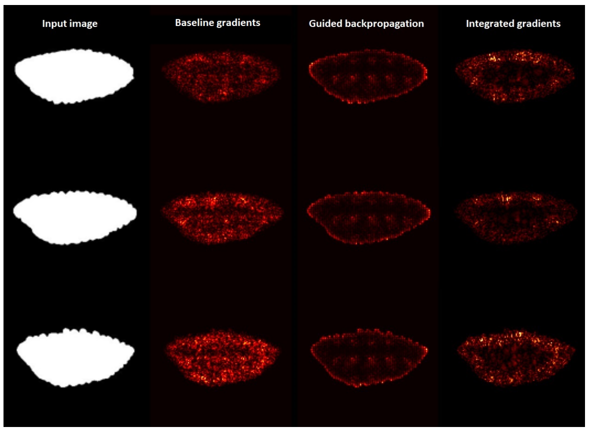
- Since the images are binary and standardized, we expect only the pixels around the contour to be highlighted. The core of the two stocks are identical and uninformative. However, because of the addition of a third baseline class, noise are added to the heatmaps and the whole otolith appears to attributing to the class score. This is to be expected, since any white pixel are clearly helping the network in discriminating between the predicted class and the baseline class.

- The gradient saliency maps and the baseline gradient method produces very noisy heatmaps compared to the heatmaps produced by guided backpropagation and integrated gradients. Using the baseline gradients, it is very hard, if not impossible, to determine which part of the otolith contour that is most influential in the class decision.

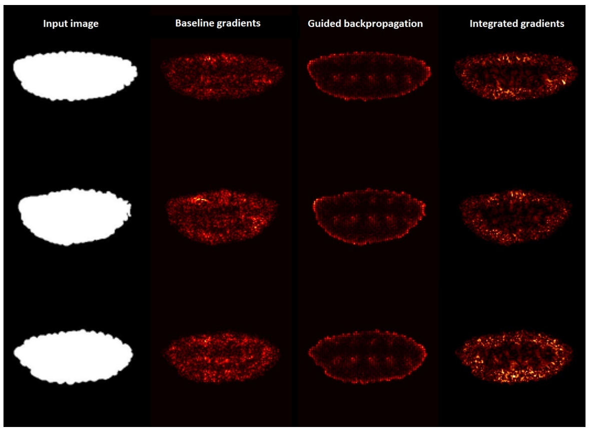
- Guided backpropagation produces the heatmaps that are most consistent with our prior belief that only the contour should be highlighted. It is difficult to see a pattern in which part of the contour that influences decisions. The images also show signs of periodic noise that should be corrected for. Image processing techniques such as Gaussian notch filters (Gonzalez & Woods, 2008) might be appropriate here.

- The integrated gradients heatmaps appear to have sparser attributions than the baseline gradient heatmaps. However, the heatmaps are still somewhat noisy, and it is difficult to draw clear conclusions on which part of the contour that is most relevant. It is possible to see that the network emphasizes small specific parts of the contour, appearing as small bright spots.

- There doesn't seem to be any pattern as to where on the contour the pixels are brightest. For the NCC, the integrated gradients maps shows some weak indications that the ridge of the contour is highlighted, however this also seems to be case for some NEAC images, for instance the second NEAC otolith image.



(a) *Heatmaps of pixel relevance for a selection of otoliths classified as NCC*



(b) *Heatmaps of pixel relevance for a selection of otoliths classified as NEAC*

Figure 31: *Comparison of different methods for computing pixel relevance heatmaps for cod otoliths*

Figure 32 and figure 33 gives a closer look into the guided backpropagation and integrated gradients heatmaps, along with heatmaps produced using the integrated guided gradients method introduced in section 4.4.3. As before, the heatmaps are uninformative with regard to class-wise patterns. Looking at the integrated gradients we see that bright spots seems to vary across the whole contour regardless of predicted class belongings. The heatmaps that appear to be most informative, are the heatmaps produced using integrated guided gradients where only pixels around the contour are highlighted.

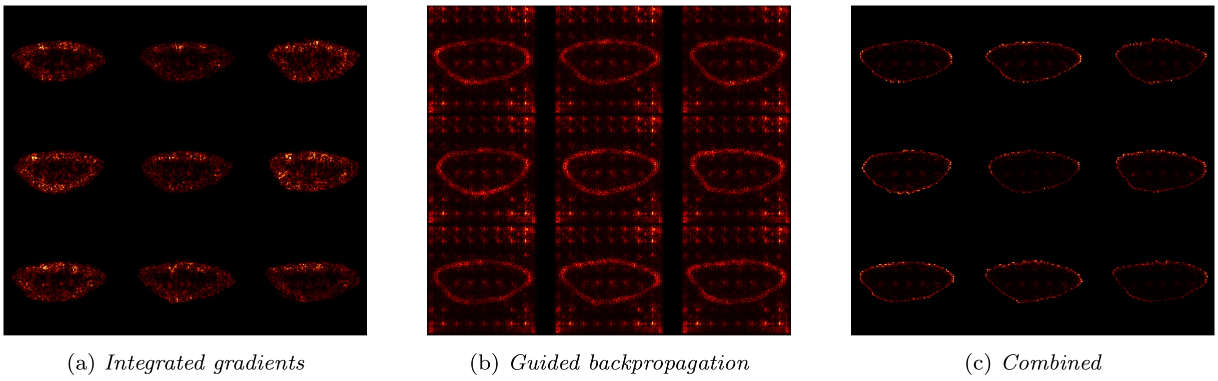

(a) *Integrated gradients* (b) *Guided backpropagation* (c) *Combined*

Figure 32: *Comparison of methods for a selection of coastal cod images*

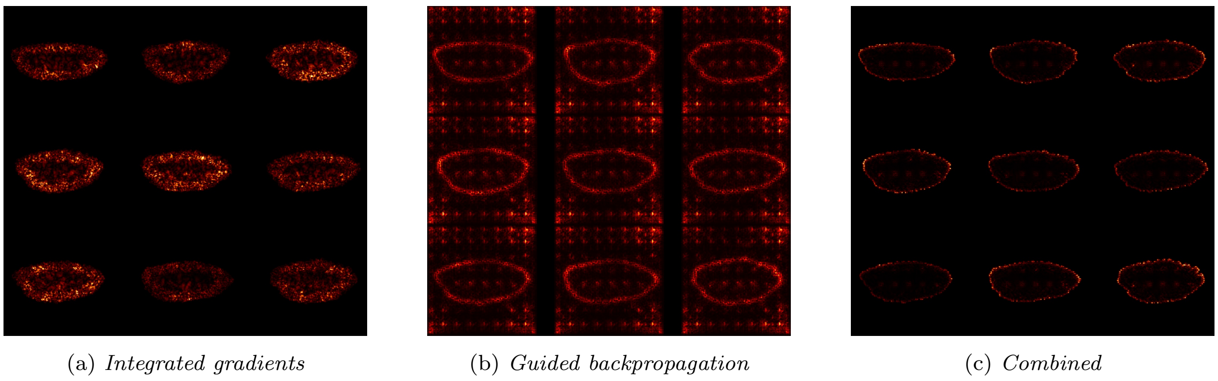

(a) *Integrated gradients* (b) *Guided backpropagation* (c) *Combined*

Figure 33: *Comparison of methods for a selection of NEAC images*

A comment regarding baseline images

The plots previously seen show the effect of using baseline images in the visualization process. In this thesis I have used black baseline images, which are clearly uninformative and neutral. The problem with black baseline images in this case is that every white pixel now serves as evidence for any other class, with the undesirable result that heatmaps becomes noisy. An all-white image would certainly be an equally well-suited baseline candidate, but we would expect an equally noisy end result only this time with highlighted pixels outside the otolith contour. A better suggestion could be to use noisy baseline images instead, e.g. by randomly sampling with replacement from $\{0, 1\}$. Another possibility is to use images with white geometric figures on a black background. Both options are neutral with regard to the classification task at hand, and both could possibly counteract the added noise on the heatmaps by forcing the network to look beyond the mere existence of black or white pixels. This effect has not been explored in this thesis, and it might be an interesting topic for further studies.

5.2. The regression on Greenland halibut otoliths

This section will address the results obtained by fitting a regression model on the Greenland halibut otolith images. I will first discuss the model selection and the choice of hyperparameters, then I will cover the model performance and an assessment of the model uncertainty. Finally, I will discuss pixel attributions to the output.

5.2.1. Model and hyperparameters

Base model

The deep learning model was fitted using transfer learning based on the Xception architecture (appendix C.2) pretrained on the ImageNet (Deng et al., 2009) data set. Only the Adam optimizer (A.4.3) was considered. GAP was applied between feature extraction and classification. The weights in the batch normalization layers were frozen during training to avoid ruining the patterns learned during pre-training, however all other weights were trainable. Since the model is predicting positive real numbers the ReLU activation function (appendix A.2.4) was used at the output. The model parameters where optimized using the MSE loss function (A.3.1).

Hyperparameters

During the process of tuning hyperparameters, 60 % of the data were used for training and 20 % for was used for both validation and testing. The sets were stratified proportionally with regard to the read age, and the sets remained constant during the hyperparameter search. The hyperparameter search can be summarized by the following steps:

1. The unconditional model outlined in section 4.1.2.3 was fitted using 128×128 images and with a learning rate of 1e-5 with a resulting test loss of 7.64.

2. The same procedure was repeated using data augmentation, resulting in a test loss of 6.84.

3. The same procedure was repeated with a reduced learning rate of 1e-4 which reduced the loss to 4.74.

4. Changing the model to the conditional model described in section 4.1.2.5 further reduced the loss to 3.82.

5. Increasing image size to 256 resulted in a loss of 3.57.

6. Finally I conducted a random search for the dropout rate and the learning rate which resulted in a learning rate of 0.00046625 and a dropout rate of 0.4.

Preprocessing

The input consisted of batches of images with shape 256 x 256 x 3, where the batch size was 32.

Data augmentation

The data augmentation applied was random translation and random rotation. Only horizontal translation with a 10 % range was used, since the input images were standardized with regard to height. The rotation was applied using a factor of 10 % in both directions, and both augmentation methods were inspected visually to ensure that the otolith remained true to the original inputs. Figure 34 shows the effect of data augmentation techniques on an input example.

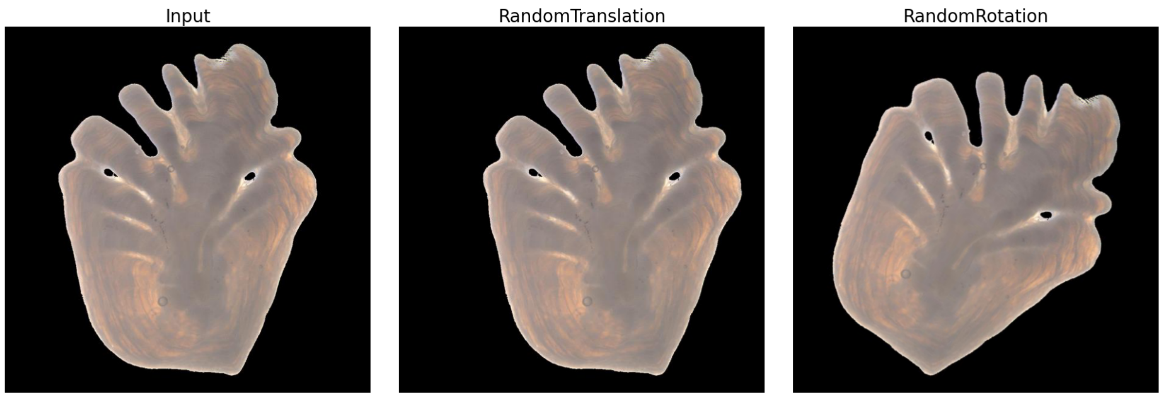


Figure 34: *Random translation and rotation applied on an otolith image*

Regularization

The model was fitted using a weight decay of 1e-5 which is the standard configuration for the Xception architecture. Dropout with a rate of 0.4 was also applied before the output.

Validation

The validation procedure was conducted using a variation of $k*l$-fold cross validation where:

- the original data set was split into 10 subsets of size 354 using proportional stratified allocation based on read age.
- each subset was chosen once for testing, while only one of the remaining 9 subsets was chosen for validation as the samples sizes were much larger than in the cod otolith data set.

In the paper by Moen et al. (2018) the test as well as the validation set was 4 % of the total sample. In this thesis we used 10 % (354 images) for both sets, as this would decrease the variance of the validation and test results, while still leaving enough images for training (2832 images).

The validation procedure included the training of three models: the linear model described in section 4.1.2.4, the conditional deep learning model described in section 4.1.2.5 and the combined model described in section 4.1.2.6. The simple deep learning model described in section 4.1.2.3 was discarded, and will not be discussed any further. The relevant models were fitted under the following regime:

- The linear model was fitted on the training and validation set combined.
- The deep learning model was fitted on the training set by applying early stopping on the validation loss, using a patience of 20 and a maximum number 100 epochs.
- The combined model was fitted as a linear combination of the linear model output, and the deep learning model output. The weights in the linear combination were optimized with regard to the training data MSE, and the validation error was used as a stopping criterion.

5.2.2. Model performance

Results

The performance were assessed by computing the coefficient of variation (CV), MSE, root MSE (RMSE) and the accuracy of the estimates with regard to the read age. The MSE and the CV is computed on the real output value, while the accuracy is computed on the rounded output. As in the paper by Moen et al. (2018), the CV is computed between two estimators - the human reader and the deep learning model - using the formula

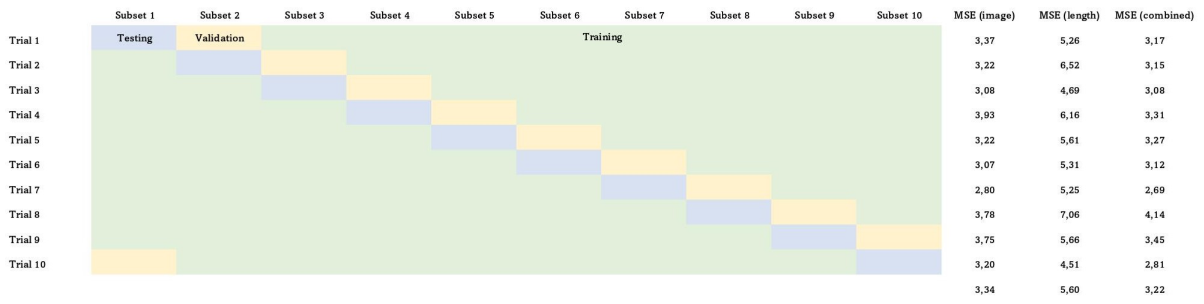
$$\mathrm{CV}_i = \frac{\sqrt{(y_i - \overline{y}_i)^2}}{\overline{y}_i}$$

where y_i is the read age, $\hat{y}_i$ is the age estimated by the model, and

$$\overline{y}_i = \frac{y_i + \hat{y}_i}{2}$$

is the mean of the read age and the predicted age. The mean CV is then given by

$$\text{mean CV} = \frac{1}{n}\sum_{i=1}^{n}\mathrm{CV}_i$$



	Subset 1	Subset 2	Subset 3	Subset 4	Subset 5	Subset 6	Subset 7	Subset 8	Subset 9	Subset 10	MSE (image)	MSE (length)	MSE (combined)
Trial 1	Testing	Validation	🟩	🟩	🟩	🟩	Training	🟩	🟩	🟩	3,37	5,26	3,17
Trial 2	🟩	🟦	🟨	🟩	🟩	🟩	🟩	🟩	🟩	🟩	3,22	6,52	3,15
Trial 3	🟩	🟩	🟦	🟨	🟩	🟩	🟩	🟩	🟩	🟩	3,08	4,69	3,08
Trial 4	🟩	🟩	🟩	🟦	🟨	🟩	🟩	🟩	🟩	🟩	3,93	6,16	3,31
Trial 5	🟩	🟩	🟩	🟩	🟦	🟨	🟩	🟩	🟩	🟩	3,22	5,61	3,27
Trial 6	🟩	🟩	🟩	🟩	🟩	🟦	🟨	🟩	🟩	🟩	3,07	5,31	3,12
Trial 7	🟩	🟩	🟩	🟩	🟩	🟩	🟦	🟨	🟩	🟩	2,80	5,25	2,69
Trial 8	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟦	🟨	🟩	3,78	7,06	4,14
Trial 9	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟦	🟨	3,75	5,66	3,45
Trial 10	🟨	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟩	🟦	3,20	4,51	2,81
											3,34	5,60	3,22

Figure 35: *Test results for the Greenland halibut validation procedure*

Figure 35 shows the results for validation procedures. We note that the MSE for the linear regression model is largest in all trials, and that the combined model has has the lowest MSE in 6 out 10 trials, while the plain deep learning model performs best in terms of loss in 3 out of 10 trials.

	MSE			RMSE			mean CV		
	Males	Females	Total	Males	Females	Total	Males	Females	Total
Deep learning	2.64	3.83	3.34	1.62	1.96	1.83	9.35 %	9.48 %	9.43 %
Length	4.86	6.13	5.60	2.20	2.48	2.37	12.79 %	12.57 %	12.66 %
Combined	2.46	3.75	3.22	1.57	1.94	1.79	9.04 %	9.26 %	9.17 %

Table 2: *Model performance measured in MSE, RMSE and CV*

Table 2 shows the MSE, RMSE and CV for all test samples combined. The combined model had a RMSE of 1.79 which is larger than the RMSE of 1.65 obtained by Moen et al. (2018). The mean CV of 9.17 % for the combined model is comparable to the CV of 8.97 % obtained by Moen et al. (2018).

	Accuracy			1-off			2-off		
	Males	Females	Total	Males	Females	Total	Males	Females	Total
Deep learning	26.42 %	23.23 %	24.55%	69.49 %	61.16 %	64.60 %	89.90 %	81.73 %	85.11 %
Length	22.18 %	18.41 %	19.97 %	59.11 %	53.16 %	55.62 %	82.46 %	75.42 %	78.33 %
Combined	26.21 %	23.76 %	24.77 %	70.78 %	61.20 %	65.17 %	91.06 %	83.33 %	86.52 %

Table 3: *Model performance measured in the percentage of predicted samples being less than 0, 1 and 2 years off the label*

Table 3 gives a summary of model performances in terms of percentage of predicted samples being less than 0, 1 and 2 years off the read age. Note that the accuracy of the deep learning model seems to be comparable to the combined model, while the MSE is noticeable higher. This could be due to the effect of rounding, however the rounded MSE is 3.44 vs 3.33 which is still a significant difference. The discrepancy can be explained by looking at figure 36. Here we see that attributions to the MSE differ only for ages in the range 11 to 15. Outside that range the MSE attributions are almost identical for the two models (2.16 for deep learning and 2.18 for the combined model). Computations of the accuracy in the 11 to 15 range, gives identical results (25.4 %) for the two models.

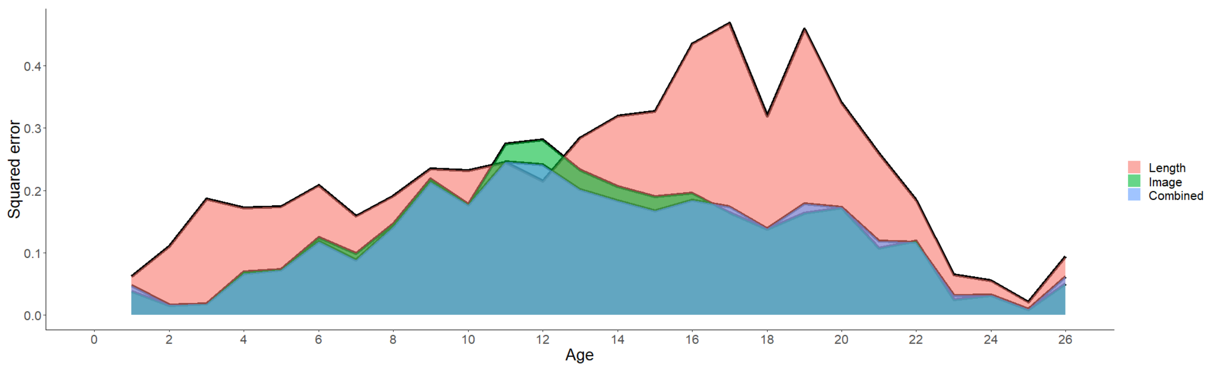


Figure 36: *Attributions to the total MSE by age*

Overall the results are better for males than for females by all performance measures. Looking at the distribution in figure 18 we see that this is natural and expected; the female age distribution has a larger standard deviation than the male distribution (4.4 years vs 3.3 years), and more samples located at the extremes than males (12 samples outside the range 2-22 vs 6 samples outside the range 2-19).

5.2.3. Model uncertainty

Bias

Figure 37 shows a comparison of the fitted values obtained by the linear model, and the fitted values obtained by the deep learning model. The figure reveals that the deep learning predictions are distributed closer to the target line.

Figure 38 shows the predicted age vs the read age for all three models, and we make the following observations:

- Both the deep learning model and the combined model seems to be unbiased from the age of 3 to 12 for males and from the age of 3 to 15 for females.
- The length model seems to be unbiased from the age of 8 to 11 for males and from the age of 9 to 15 for females.

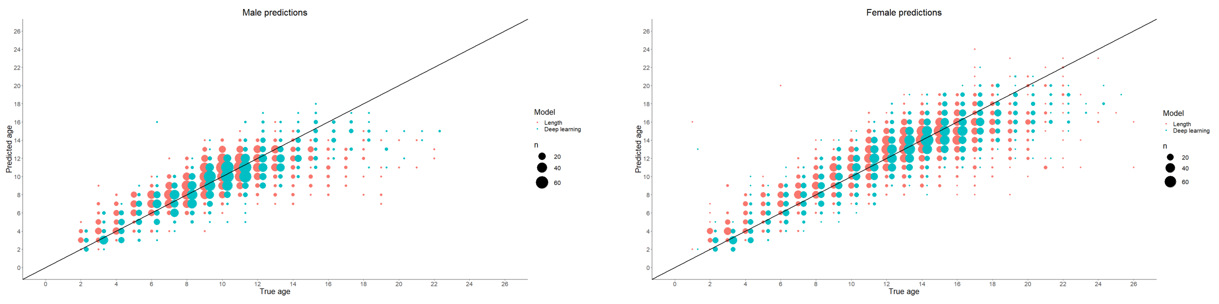


Figure 37: *A comparison of the linear model and Deep learning using predicted age plotted against read age*

	Deep learning	Length	Combined
Male	0.00088	0.17	0.019
Female	0.032	0.062	0.045

Table 4: *KL-divergence between the labeled age distribution and the distribution of the predicted age*

Recall from figure 18 that the age distribution of the data was centered around 10 and 13 years for males and females respectively, thus we expect that a model which is optimized with regard to the MSE will be biased towards mean of the data. This behavior is expected and can be observed in the figures where the bias shifts from positive to negative as the age increases. The age estimations are clearly biased at the extremes, but we note that the data set included only 2 images of 1-year old's, and only 10 images of otoliths older than 22 years. The mean bias across all samples were approximately 0 for all three models.

Model fit

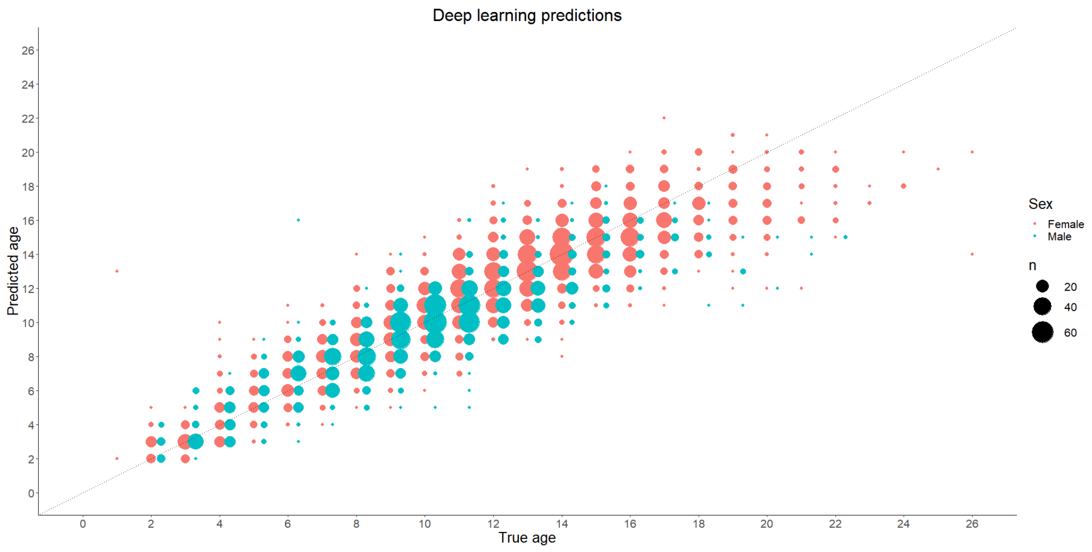
Figure 39 shows the differences in distributions between the estimated density curves for the read age and the density curves for the model predicted age where the solid line displays the label distribution. The plots seems to indicate that age predictions by the deep learning model alone has the best fit in terms of age distribution. A measure of the discrepancy between the curve can be obtained by computing the KL-divergence - defined in section 2.1.3.8 - of the predicted distribution with regard to the labeled age distribution. Table 4 displays the results, which were computed by numerical integration using the Simpson's rule (Sauer, 2011).

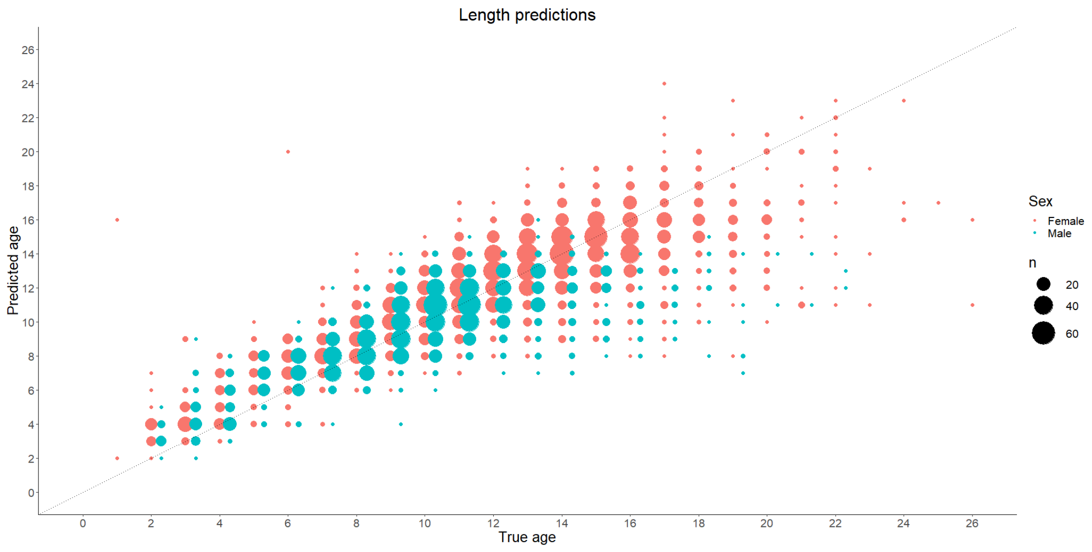
From the table we see that the deep learning model actually fits the target distribution better than the combined model. A possible explanation can be provided by figure 37, where we see that the linear model has a larger bias towards the mean for all ages. The linear model seems to direct predictions close to the mean, which in turn reduces the MSE because of the large number of samples in that range. However, the model overestimates the number of samples in that range, resulting an a density estimate that fits poorly to the true age distribution.

Model assumption

Figure 40, 41 and 42 shows residual plots and age-wise standard errors of the fitted values. There are a couple of things to note about the plots:

- There are no signs of any significant heteroscedasticity. The standard errors does not seem to vary significantly with age, except at the extremes where the number of data are limited.
- The age predicted by deep learning seems to have a linear relationship with the true age, whereas the waveform of the length residual means seems to indicate that a non linear model might be more appropriate than a linear model.
- The deep learning residuals are symmetric, the length residuals are not. Looking at predicted ages in the range 8 to 14, the length model is clearly biased and has a tendency to underestimate older fish. This tendency is not that obvious for the





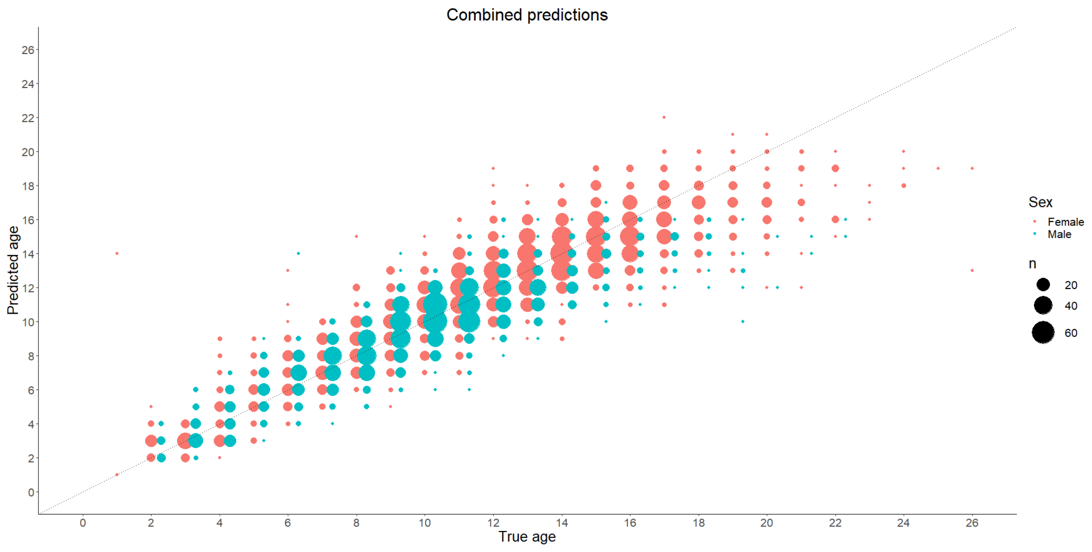


Figure 38: *Predicted vs true age for Greenland halibuts*

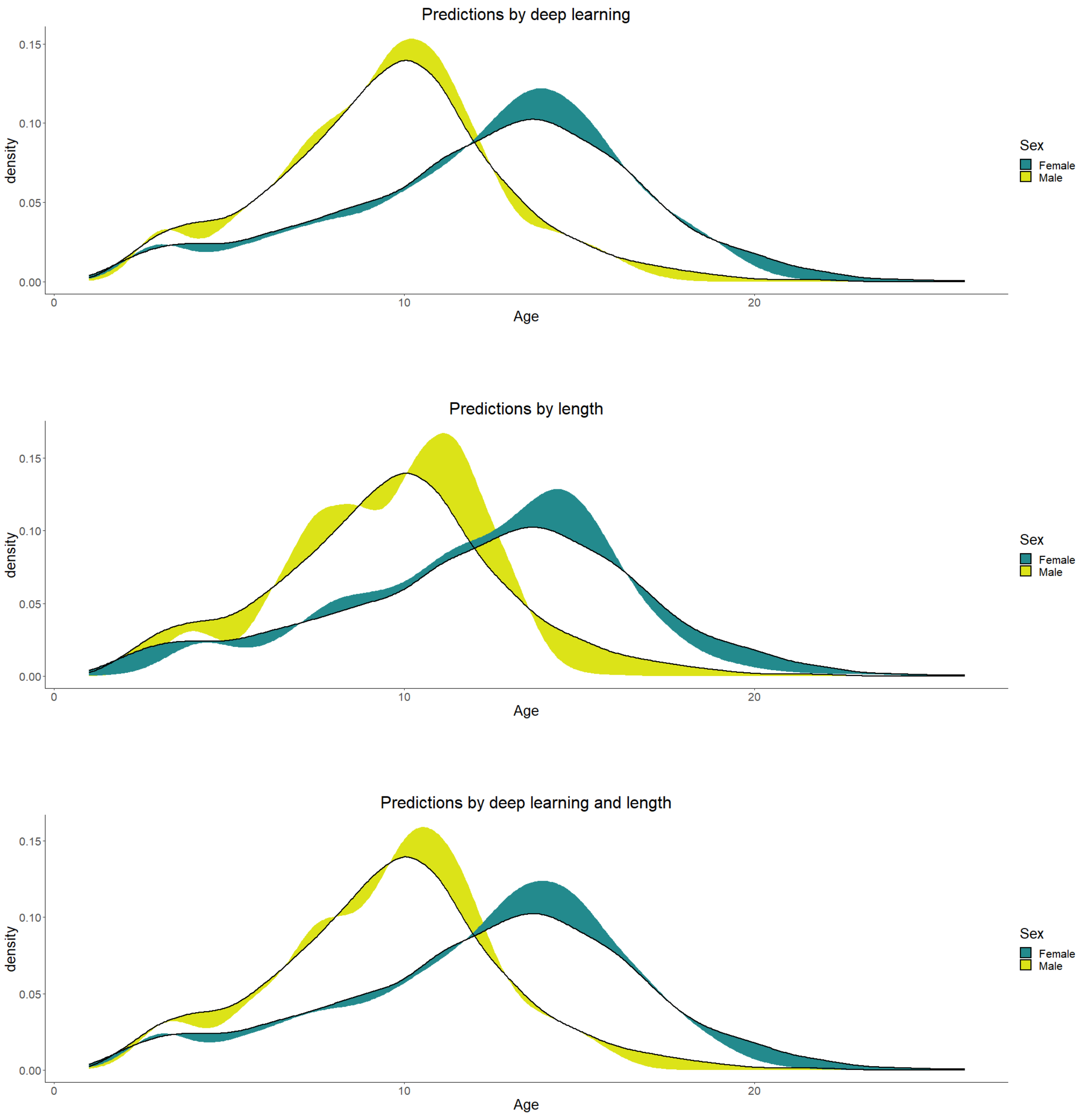


Figure 39: *Area highlighting the difference between the label age distribution and the predicted age distribution*

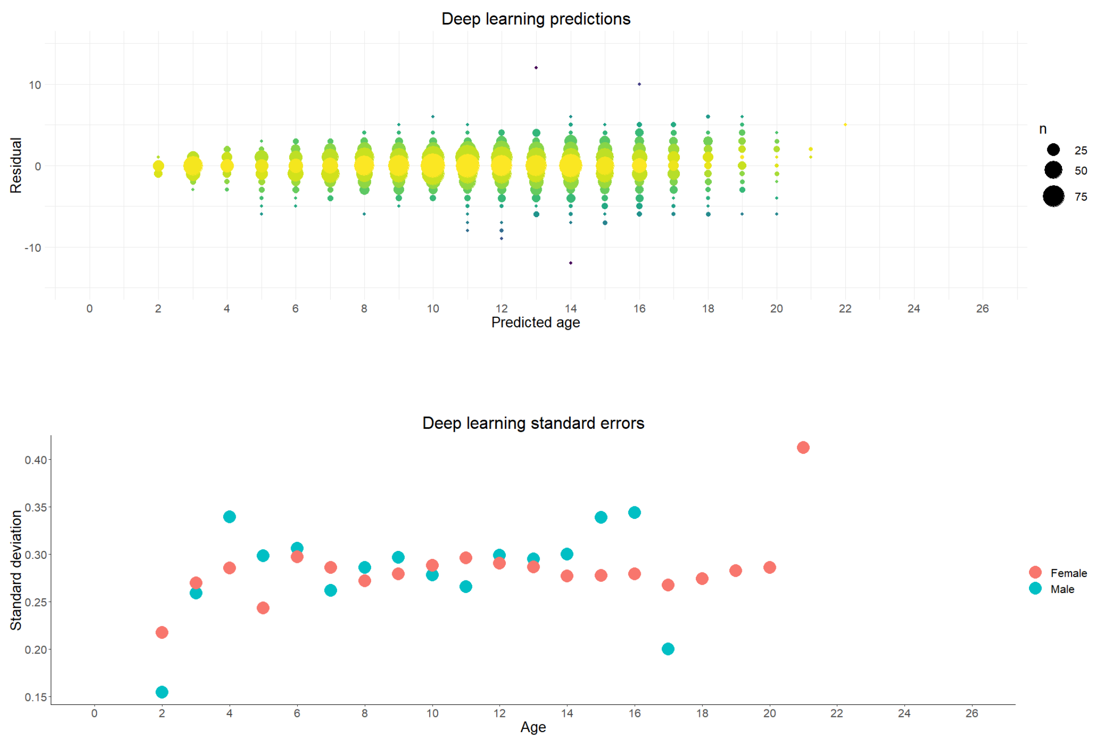


Figure 40: *Residuals and standard deviations for age predictions using deep learning*

deep learning model, which seems to be roughly symmetric around 0 for all predicted ages. The figures indicates that the deep learning model manages to estimate the age equally well for all ages, whereas the length model underestimates the age when the fish stops growing.

5.2.4. Explaining decisions by heatmaps of pixel relevance

Figure 43 shows a comparison of five different methods used to estimate the pixel relevance for a selection of four different input images. We notice that:

- The heatmaps produced by baseline gradients are very similar to the heatmaps produced by gradients, the difference being that different weights are given to the pixels, and as in the case of the cod otolith heatmaps presented in figure 32 and 33, the gradient saliency maps are noisy and difficult to interpret.
- The guided backpropagation method differ from the other methods in that it emphasizes the contour of the otolith, a feature that is especially noticeable in the first two images. This could possibly be an artifact resulting from the usage of black baseline images representing age 0, since the most noticeable difference between an otolith of age 1 and an otolith of age 0 (the baseline) is the mere existence of the otolith. We also note the presence of periodic noise in the first two images.
- Integrated gradients seems to produce heatmaps with sparser bright spots than the heatmaps produced by computing gradients. Ignoring the contour, the integrated gradients seems to highlight the same areas as guided backpropagation.

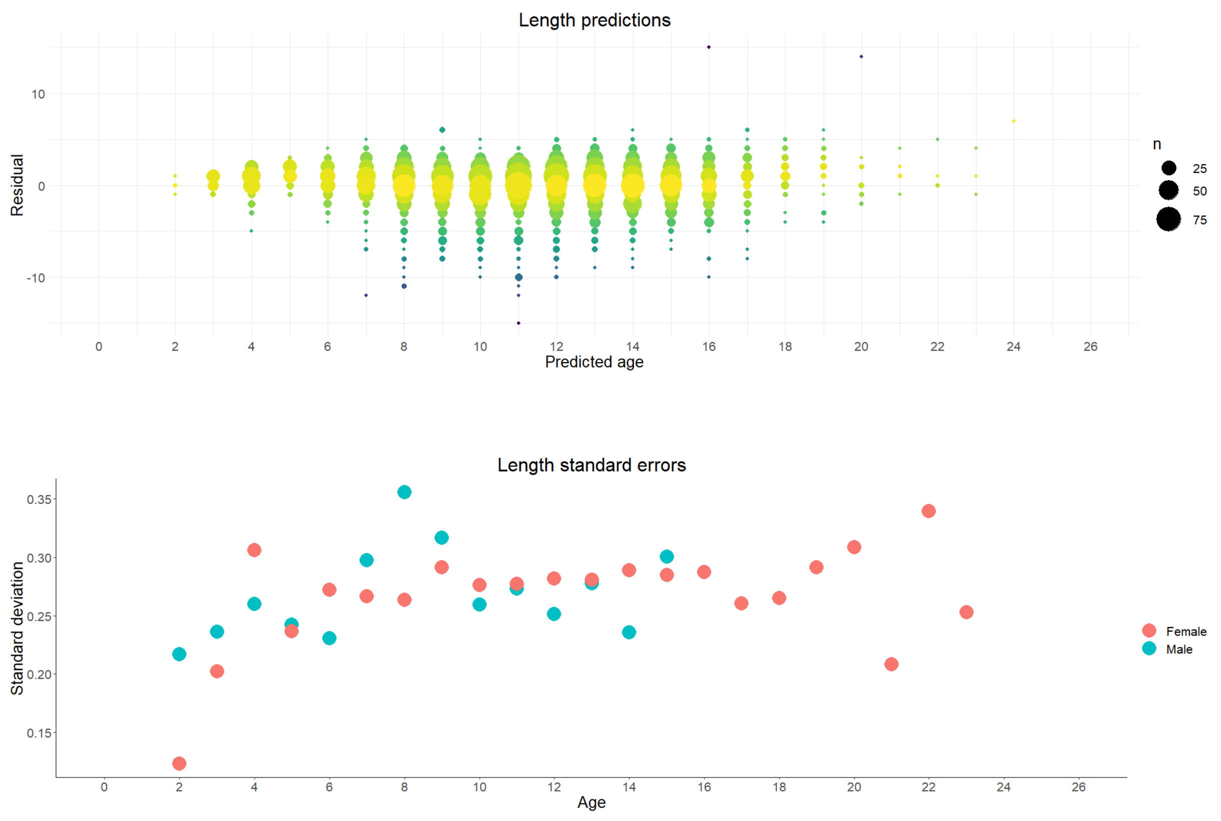


Figure 41: *Residuals and standard deviations for age predictions using length*

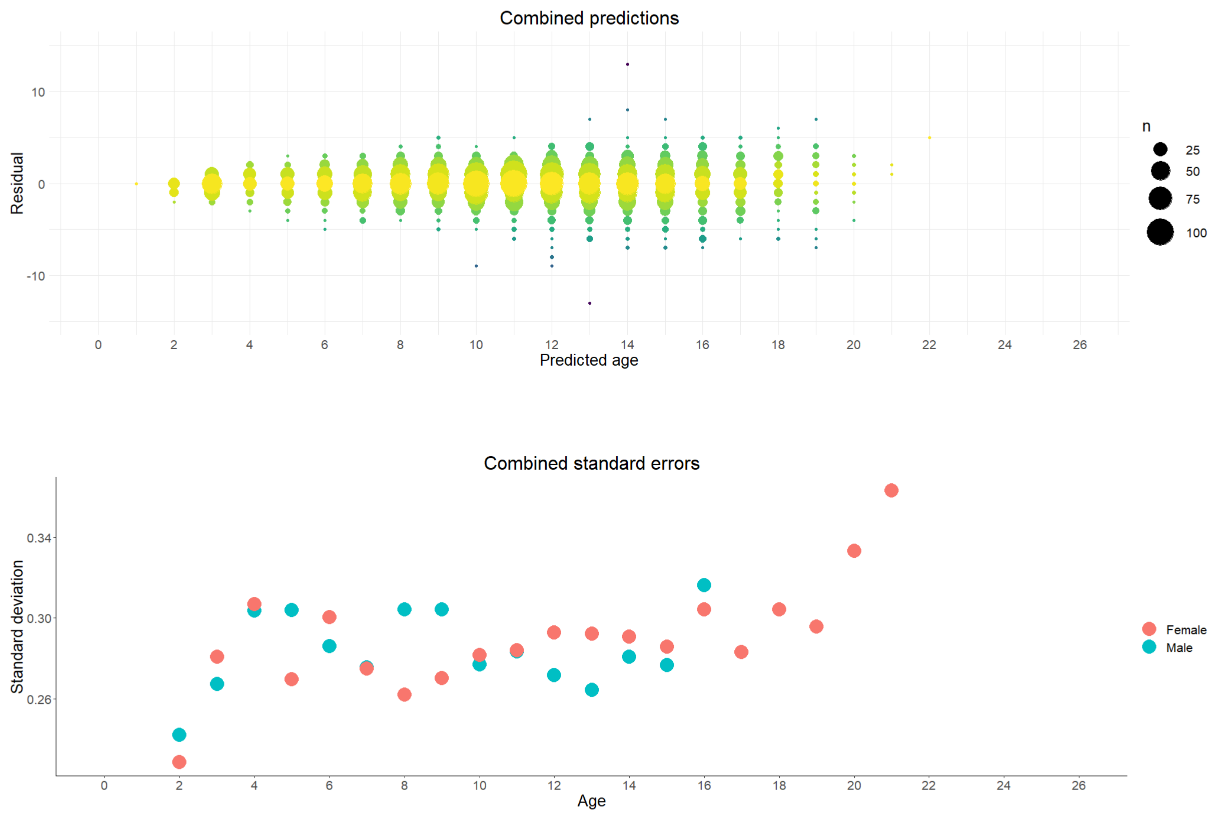


Figure 42: *Residuals for age predictions*

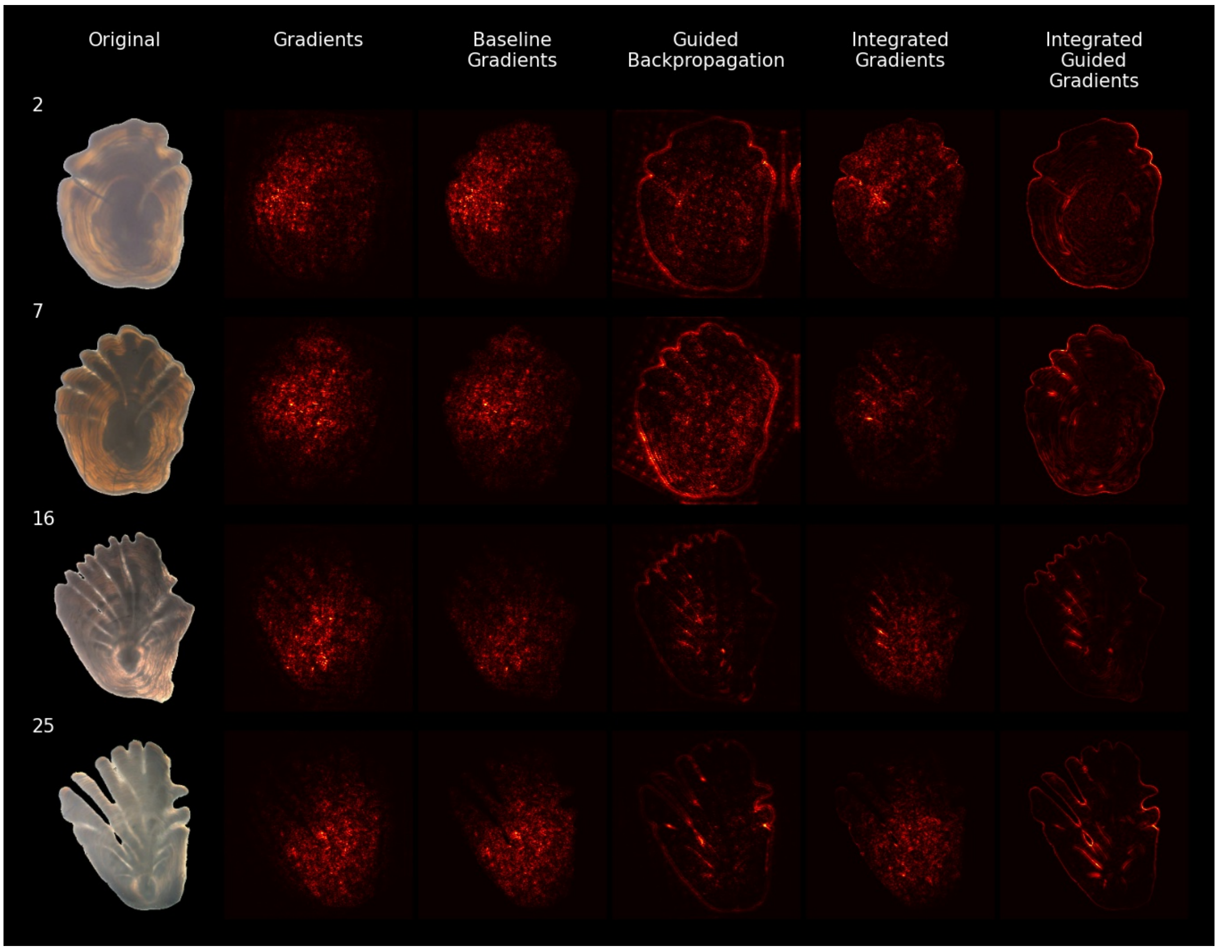


Figure 43: *Comparison of visualization methods applied on a selection of 4 Greenland halibut images. The numbers to the left are the labeled age.*

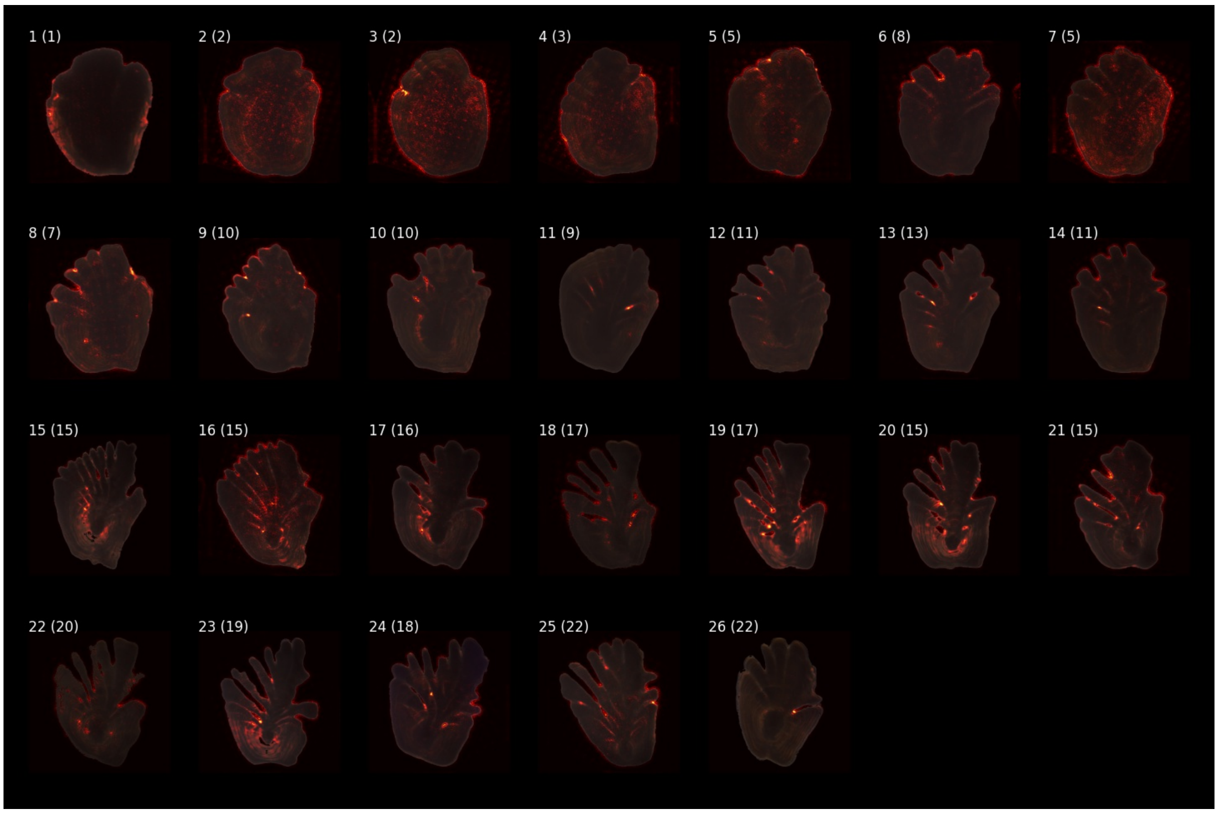


Figure 44: *Pixel relevance heatmaps produced by guided backpropagation applied on a selection of Greenland halibut otolith images. Each image is labeled with read age (predicted age) in the top-left corner.*

- The integrated guided gradients heatmaps appears very similar to the heatmaps produced by guided backpropagation. The difference between the two methods is noticeable in the first two images where the integrated guided gradients produced smoother heatmaps without any significant noise.

Figure 44, 45 and 46 displays heatmaps produced by guided backpropagation, integrated gradients and integrated guided gradients applied on the otolith selection presented in figure 17. There are three things to note about the growth patterns of the presented otoliths:

- The annual zones are considered the most important part of the otolith in age determination as they are counted by the readers to determine the age. However, since the CNN is trained on images with reduced resolution it is extremely difficult to accurately determine the number of annual zones, especially as the fish gets older. This is apparent in figure 17 which also shows that the prominence of zones also varies across samples. Based on the heatmaps, the network doesn't seem to emphasize the number of annual zones at all, except for the second image where there appears to be a single inner zone in addition to the contour, which is indicative of an age of two.

- The growth pattern indicates that the fingers develops with age, and both the length and number of fingers seems to be indicative of age. Juveniles (age 1-4) has no fingers at all, while adolescent (age 5-9) and young adults (age 10 - 13) seems to have an early development. The fingers seems to be most prominent in adult fish

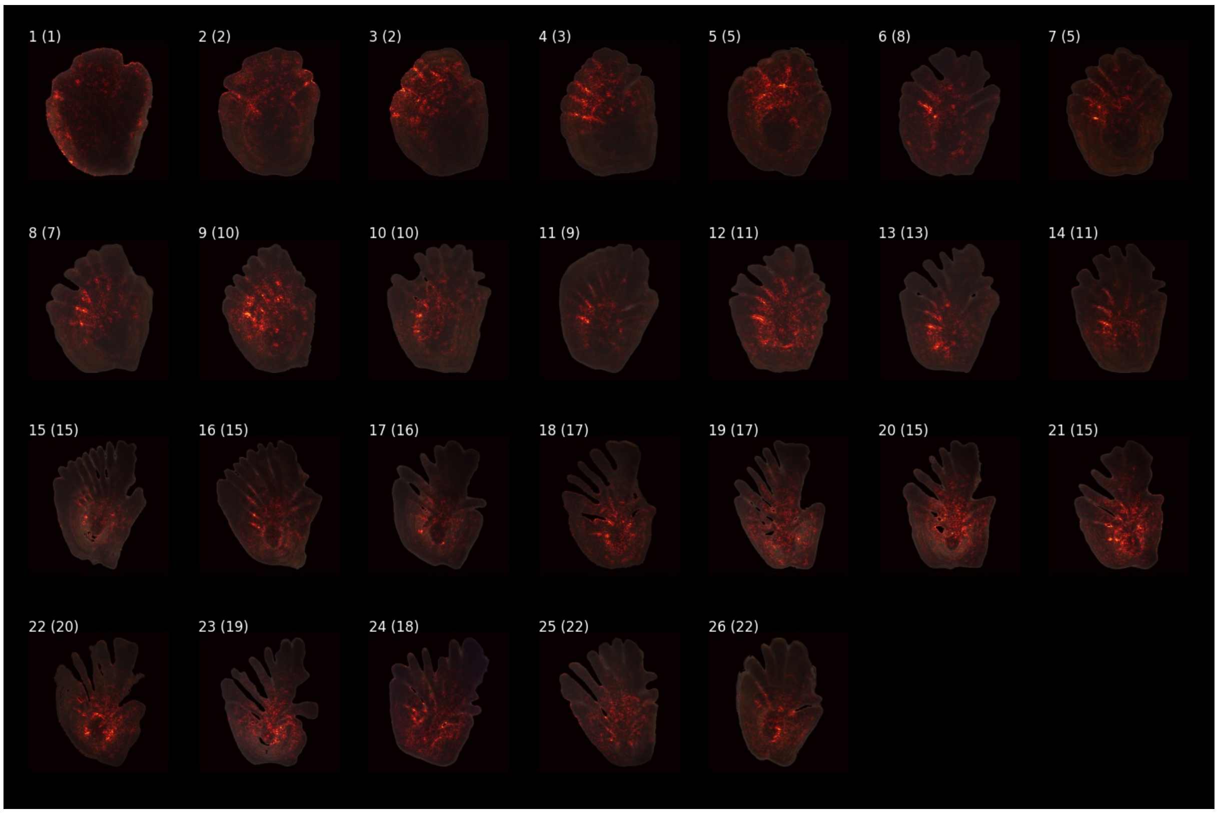


Figure 45: *Pixel relevance heatmaps produced by integrated gradients applied on a selection of Greenland halibut otolith images. Each image is labeled with read age (predicted age) in the top-left corner.*

(age 14-26). These characteristics seems to be of importance when predicting the age in adults and young adults.

- Holes in the otolith seems to be a strong indicator of an adult fish. This is reflected in the heatmaps produced by guided backpropagation and integrated guided gradients. The heatmaps produced by integrated gradients does not seem to highlight the same areas.

It is difficult to draw clear conclusions from the heatmaps in figure 44, 45 and 46. Figure 44 and 46 shows the most apparent pattern where we see that the heatmaps changes focus from the otolith contour at the early ages, to the core of the contour at the older ages which can be due to the development of fingers as the fish gets older. The heatmaps produced by integrated gradients are less consistent, however the stripe-like pattern apparent in several of the heatmaps suggests that the fjords between the otolith fingers are significant in determining the age.

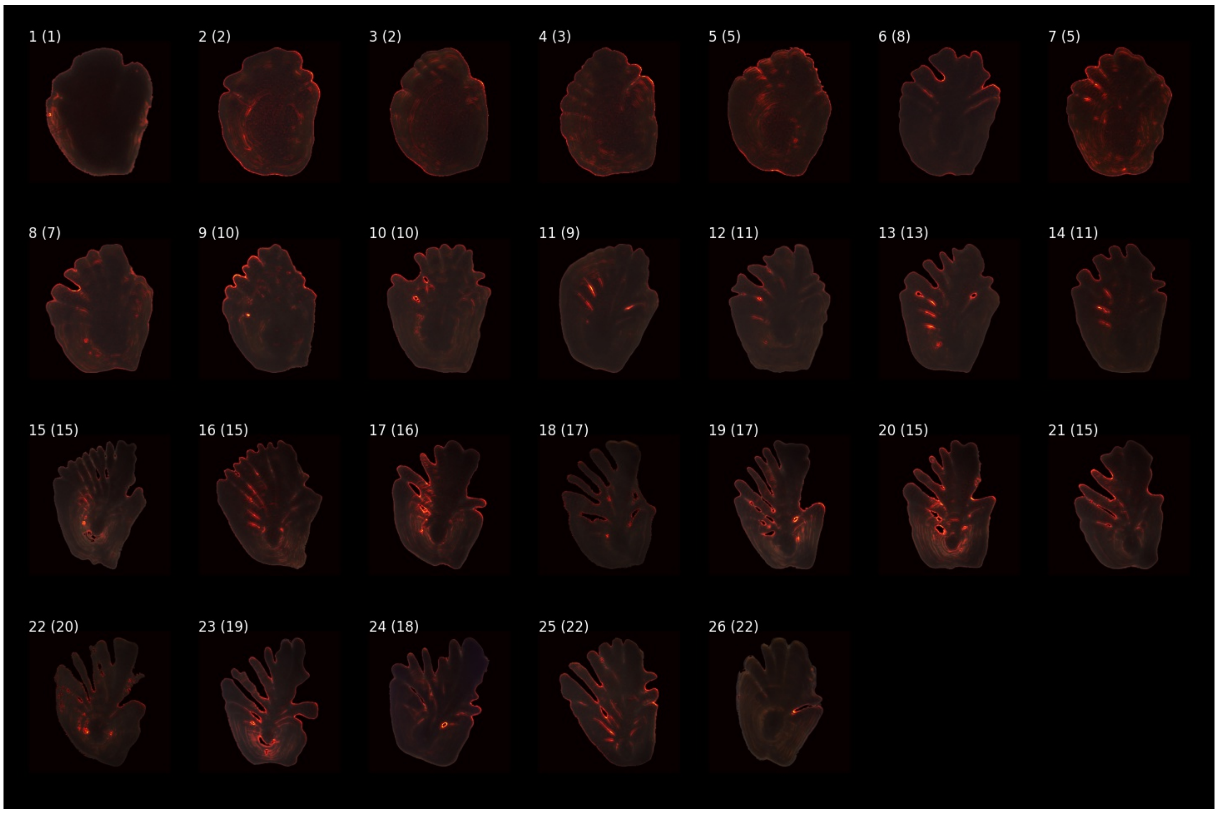


Figure 46: *Pixel relevance heatmaps produced by integrated guided gradients applied on a selection of Greenland halibut otolith images. Each image is labeled with read age (predicted age) in the top-left corner.*

6. Discussion and conclusions

6.1. Conclusions

The conclusions for this thesis can be summarized in the following five main points:

- Deep learning models were able to obtain good results in the discrimination of cod otoliths, with significantly better results obtained using images with ripples than convex images without ripples. A model constructed from scratch obtained results similar to an advanced architecture with pre-trained weights, and significantly better classification results were obtained when deep learning, EFDs and MIRR were combined.

- Determination of the age of Greenland halibut, framed as a regression task, showed that deep learning outperformed length based age predictions in terms of precision and accuracy compared to the read ages. Adding length as an explanatory variable did not increase model performance significantly. Adding length led to a slight decrease in loss, but it also led to a worse fit for the overall age distribution.

- Both the classification of cod, and the age prediction of Greenland halibut, showed that test results are greatly dependent on the test set choice, and that variations in test results are more sensitive to differences between test sets, than differences between validation sets and parameter initializations.

- A deep learning classifier exhibits a bias towards allocating input to the majority class, and the classifier adopts a classification rule that is equivalent to the minimization of the total probability of misclassification for normal distributions.

- Heatmaps produced by feature relevance attribution methods were difficult to interpret. While the heatmaps confirmed the validity of the deep learning models by giving attributions to areas perceived as important, it was difficult to discover strong patterns in the heatmaps. There were variations between the interpretability techniques, which showed that gradient saliency maps were noisy and difficult to interpret, while integrated gradients and guided backpropagation produced heatmaps that were interpretable to a greater extent. We have also seen the effect of additional baseline classes, and that baseline images should be chosen with care.

6.2. Discussion

Otoliths play an important role in fish stock discrimination and age determination. They are interpreted by human experts, who for example visually inspect the otolith to determine stock affiliation for cod, and age for Greenland halibut, which are the two examples studied in this thesis. Otolith analyses are conducted extensively across the globe, and the process is time consuming both in terms of the analysis itself, and in terms of the time used to educate the readers. A proposed improvement was to apply convolutional neural networks - a class of deep learning models that has provided excellent results on image classification problems during the last decade - for image classification and regression. Automatic analysis of otolith images is a consistent process in the sense that a trained deep learning model would produce the same output for the same input, which is a major advantage in comparison to estimates provided by human readers.

The results obtained in this thesis show that deep learning models indeed has great potential, and the deep learning model fitted to the cod otolith data contributed to a significant improvement to the results already obtained by linear discriminant analysis, while the age distribution of the Greenland halibut predictions had an excellent fit to the provided labels. The deep learning framework provides great flexibility, as models can be trained on image inputs of different input sizes without the need for any preprocessing of the inputs. The vast amount of architectures, regularization techniques and hyperparameters makes it reasonable to believe that even further improvements in model performances are possible, however there exists challenges and limitations that we will discuss.

The results obtained in this work show that a neural network might introduce a bias towards allocating inputs to the majority class, which might negatively affect the analysis. In particular, the cod otolith model performed significantly better on the NCC data than on the NEAC data which could pose a problem, as NEAC normally outnumbers the endangered NCC. This could result in a large number of false positive NCC, which in turn would result in an overestimation of the NCC stock size. It would be interesting to investigate this effect further with a dedicated study of techniques aimed at handling class imbalance. For example, the introduction of a third dummy class with a larger number of samples could counteract this effect, as the dummy class presumably would absorb that bias. The resampling of the minority class, or the modification of the loss function by weighting misclassification differently for each class, are other possibilities that would direct the model training towards the desired false positive rate.

The Greenland halibut model was trained using the mean squared error. MSE assumes homoscedasticity, an assumption that approximately holds on the dataset investigated. Other measures that capture non-constant variance better could be considered, such as the mean squared relative error. However, the evaluated coefficient of variation (CV) - a relative measure of error - showed that there were no big differences between the genders compared to previously published results.

The cross-validation procedures conducted in this thesis displayed the importance of using robust techniques to validate model performances. The test set sizes were adequately large, however there is always a trade-off between training set sizes and test set sizes. A reduction of the test set would increase the training set which in turn could improve model performance. To test for this effect a proper cross-validation procedure must be conducted.

A key element in the evaluation of a convolutional neural network is the problem of finding sensible explanations behind the model's decision. This is important for assessing the model's validity, but also for revealing the patterns in the images on which the model decisions are based. We considered several methods that aim to quantify the pixel's relevance on the output, and we provided heatmaps produced by gradient saliency maps, baseline gradients, guided backpropagation, integrated gradients and integrated guided gradients. While these tools can be useful in understanding the decision process of a deep learning model, interpretation of the heatmaps are subjective and we advise to use caution when studying the output.

Both the theoretical properties and the visual appearance of the heatmaps suggested that gradient saliency maps and baseline gradients were suboptimal for the task, as they produced noisy heatmaps which were difficult to analyze. The visually most appealing heatmaps were produced using integrated guided gradients, a combination of integrated gradients and guided backpropagation that has not been properly explored. However, all of the heatmaps produced were difficult to interpret - there were few patterns that provided clear and obvious explanations behind the model output - and post-processing of heatmaps could be a necessary aid in the search for patterns in the input. The introduction of baseline images in the feature relevance analysis is an interesting topic, and we discovered that the introduction of an additional baseline class added noise to the heatmaps. This is a topic for further studies, as the choice of baseline images is an important factor when constructing pixel relevance heatmaps.

The complexity of the model is a possible cause for the inconsistency of the heatmaps. An interpretation is that since different images exhibits different characteristics, the network is forced to base its predictions on whichever features the network is able to extract. An image with a few distinct characteristics might result in a heatmap with sparse bright spots that indicates a clear decision basis, while an image with several weak characteristics might result in a noisy heatmap as a result of a network that is forced to make its decision by weighting all the features that together made up the decision basis.

A topic that has not been addressed in this thesis is model training using semi-supervised learning. In this thesis we used labeled data, however it would be possible to acquire samples that were yet to be labeled by humans. In this case, one could use semi-supervised techniques to exploit the unlabeled data for gaining a better understanding of the underlying structure of the dataset, which in turn could improve classification and regression results. This type of analysis is left as a direction for future work.

A. Appendix A: Deep learning details

A.1. Gradient computations for multilayer perceptrons (MLPs)

This section covers the specific gradient computations for MLPs, where the gradient of the loss with regard to the weights and bias terms are used in conjunction with gradient descent. The final gradients of the weights and biases are obtained by backpropagation, and will rely on intermediate gradients apparent from the definitions below.

Definitions

Let the activation function and the cost function be denoted by f and J respectively, and let $a_j^{(l)}(i)$ denote the output from the jth unit in the lth layer for sample i, such that

$$a_j^{(l)}(i) = f(z_j^{(l)}(i)) = f\left(\left(\mathbf{a}^{(l-1)}(i)\right)^T \cdot \mathbf{w}_j^{(l)} + b_j^{(l)}\right)$$

where $\mathbf{a}^{(l-1)}(i)$ is a vector of outputs from layer *l-1*, $\mathbf{w}_j^{(l)}$ and $b_j^{(l)}$ are the weights and bias for the jth unit in the lth layer. Furthermore let n_l be the number of hidden units in layer l. We define the weight matrix and the bias vector for layer l by

$$W^{(l)} = \begin{bmatrix} \mathbf{w}_1^{(l)} & \cdots & \mathbf{w}_{n_l}^{(l)} \end{bmatrix} \tag{16}$$

and

$$\mathbf{b}^{(l)} = \begin{bmatrix} b_1^{(l)} & \cdots & b_{n_l}^{(l)} \end{bmatrix}^T$$

Similarly we define the activation matrix by

$$A^{(l)} = \begin{bmatrix} \mathbf{a}^{(l)}(i) & \cdots & \mathbf{a}^{(l)}(n) \end{bmatrix} \tag{17}$$

and we define $\delta_j^{(l)}(i) = \frac{\partial J}{\partial z_j^{(l)}(i)}$ with the corresponding δ matrix

$$D^{(l)} = \begin{bmatrix} \boldsymbol{\delta}^{(l)}(i) & \cdots & \boldsymbol{\delta}^{(l)}(n) \end{bmatrix} \tag{18}$$

The activations

To find the gradients of the loss with respect to the weights and the bias terms we need to obtain an expression for the gradient of the loss with respect to the activations. By first applying the chain rule we can compute this by

$$\begin{aligned}
\nabla_{\mathbf{a}^{(l)}(i)} J &= \left(\sum_{k=1}^{n_{l+1}} \frac{\partial z_k^{(l+1)}(i)}{\partial a_k^{(l)}(i)} \frac{\partial J}{\partial z_k^{(l+1)}(i)} \right)_{j=1}^{n_l} \\
&= \left(\left(\frac{\partial \mathbf{z}^{(l+1)}(i)}{\partial a_j^{(l)}(i)} \right)^T \nabla_{\mathbf{z}^{(l+1)}(i)} J \right)_{j=1}^{n_l} \\
&= \left(\left(\mathbf{w}_j^{(l+1)} \right)^T \boldsymbol{\delta}^{(l+1)(i)} \right)_{j=1}^{n_l} \\
&= \left(W^{(l+1)} \right)^T \boldsymbol{\delta}^{(l+1)(i)}
\end{aligned}$$

By using the definition in (16) and (18) we can generalize this to matrix form by using the definition

$$\nabla_{A^{(l)}} J = \left(W^{(l+1)} \right)^T D^{(l+1)} \tag{19}$$

The δ-term

We also need to obtain an expression for the δ terms. This is obtained by

$$\begin{aligned}
\boldsymbol{\delta}^{(l+1)}(i) &= \nabla_{\mathbf{z}(l)(i)} J \\
&= \left(\frac{\partial a_j^{(l)}(i)}{\partial z_j^{(l)}(i)} \right)_{j=1}^{n_l} \odot \nabla_{\mathbf{a}(l)(i)} J \\
&= f' \left(\mathbf{z}^{(l)}(i) \right) \odot \nabla_{\mathbf{a}(l)(i)} J
\end{aligned}$$

which we can generalize by using the result from (19) to

$$\begin{aligned}
D^{(l)} &= f'(Z^{(l)}) \odot \nabla_{A^{(l)}} J \\
&= f'(Z^{(l)}) \odot (W^{(l+1)})^T D^{(l+1)}
\end{aligned} \tag{20}$$

The weights

Finally we can obtain an expression for the gradient with regard to the weights and bias terms. By again applying the chain rule, we can compute the partial derivative with respect to the kth weight of the jth unit by

$$\begin{aligned}\frac{\partial J}{\partial w_{jk}^{(l)}} &= \sum_{i=1}^{n} \frac{\partial z_j^{(l)}(i)}{\partial w_{jk}^{(l)}} \frac{\partial J}{\partial z_j^{(l)}(i)} \\ &= \sum_{i=1}^{n} a_k^{(l-1)}(i) \delta_j^{(l)}(i)\end{aligned}$$

Using the definitions from equation 17 and 18 we can generalize this result the gradient matrix by the definition

$$\nabla_{W^{(l)}} J = D^{(l)} \left(A^{(l-1)} \right)^T \tag{21}$$

The bias terms

The partial derivative of the loss with respect to the bias is computed using the chain rule by

$$\begin{aligned}\frac{\partial J}{\partial b_k^{(l)}} &= \sum_{i=1}^{n} \frac{\partial z_k^{(l)}(i)}{\partial b_k^{(l)}} \frac{\partial J}{\partial z_k^{(l)}(i)} \\ &= \sum_{i=1}^{n} \delta_k^{(l)}(i)\end{aligned}$$

Thus, the gradient with respect to the bias can be written as

$$\nabla_{\mathbf{b}^{(l)}} J = \sum_{i=1}^{n} \boldsymbol{\delta}^{(l)}(i)$$

A.2. Activation functions

Figure 47 shows a selection of relevant activation functions along with their derivatives. The functions themselves are addressed in the subsections below.

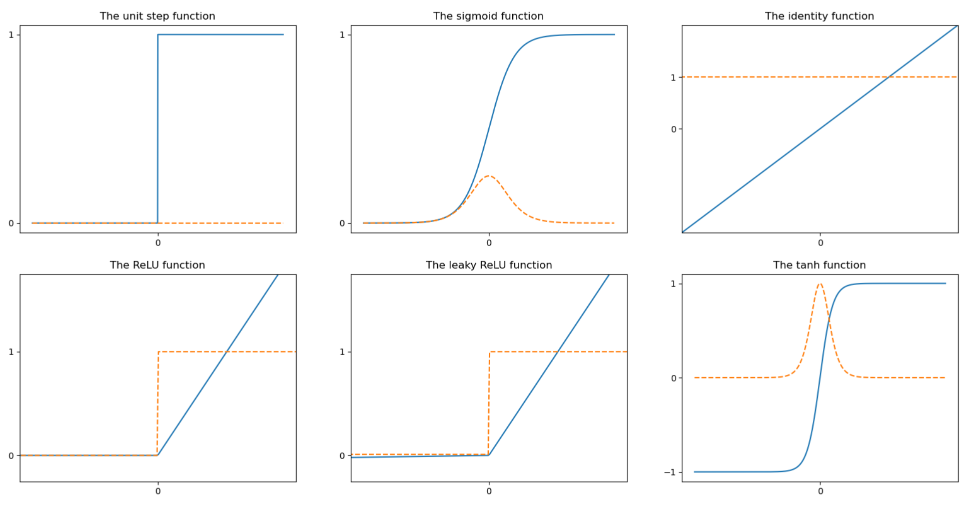


Figure 47: *Activation functions used in neural networks plotted with derivatives*

A.2.1. The unit step function

For completeness we include the unit step function which is defined by

$$f(x) = \begin{cases} 1, & x \leq 0 \\ 0, & x > 0 \end{cases}$$

This function is in compliance with the idea behind the original perceptron, which emulated a neuron with an activation that was either "on" or "off". Note that the unit step function is non-differentiable at zero and has a zero derivative everywhere else, which makes it impossible to use in a backpropagation scheme.

A.2.2. The logistic sigmoid function

The sigmoid function is defined by

$$f(x) = \frac{1}{1 + e^{-x}}$$

and it is continuous and differentiable for all $x \in \mathbb{R}$. Since the sigmoid saturates to 0 or 1, it has the ability to serve as a continuous replacement for the unit step function. It is straightforward to show that its derivative is expressed by

$$f'(x) = f(x) \cdot (1 - f(x))$$

Note from figure 47 that the derivative of the sigmoid function is always small, that is $0 < f'(x) \leq \frac{1}{4}$. This causes some problems, as networks that uses sigmoid activations are prone to vanishing gradient problems (see section A.5). Because of this, the function is generally not used in deep architectures or in Recurrent Neural Networks (RNNs). The sigmoid function is commonly used at the output for binary classification problems (see section 2.2.4).

A.2.3. The identity function

The identity function is included here for completeness. The function and its derivative are respectively defined by

$$f(x) = x$$

and

$$f'(x) = 1$$

A.2.4. The Rectified Linear Unit (ReLU)

The Rectified Linear Unit (ReLU) activation function is defined by

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

with a derivative given by

$$f'(x) = \begin{cases} 1, & x > 0 \\ 0, & x < 0 \end{cases}$$

The function returns a zero activation for half of its domain, and the unit derivative for the other half of its domain, making the function especially suitable for deep networks which would otherwise suffer from exploding or vanishing gradients (appendix A.5). The ReLU function is not differentiable at zero, but the derivative is arbitrarily defined to be either 0 or 1 when implemented.

A.2.5. The Leaky ReLU

The Leaky ReLU activation function is a modification of the ReLU activation function and is defined by

$$f(x) = \begin{cases} x, & x > 0 \\ 0.01x, & x < 0 \end{cases}$$

The derivative is

$$f'(x) = \begin{cases} 1, & x > 0 \\ 0.01, & x < 0 \end{cases}$$

which ensures a nonzero gradient for all inputs. This can be beneficial in cases where the backpropagation of zero activations and gradients would lead to weights getting stuck and failing to update.

A.2.6. The hyperbolic tangent (tanh) function

The tanh function is defined as

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

and it can be shown that the derivative is given by

$$f'(x) = 1 - f(x)^2$$

The function saturates to -1 or 1, and has a derivative ≤ 1 for all $x \in \mathbb{R}$ (see figure 47). Similarly to the case of the sigmoid function (appendix A.2.2), network using tanh activations are prone to vanishing gradient problems (appendix A.5) because of its small gradients. tanh activation functions are commonly used in Long Short-Term Memory networks (LSTMs).

A.2.7. The Softmax function

The softmax activation function can be considered a generalization of the sigmoid function (A.2.2), and is most often used at the output of a k-class classification network. The function is defined by

$$f(\mathbf{x})_i = \frac{e^{x_i}}{\sum_{j=1}^{k} e^{x_j}}$$

where the subscript i denotes the output from the output neuron corresponding to class i. Note that the function saturates to 0 or 1, and that

$$\sum_{i=1}^{k} f(\mathbf{x})_i = 1$$

ensuring that the function is suitable for probabilistic outputs.

The softmax derivative is given by

$$\frac{\partial f(\mathbf{x})_i}{\partial x_j} = f(\mathbf{x})_i \left(I\{i = j\} - f(\mathbf{x})_j\right)$$

A.3. Loss functions

This section covers the derivations of the mean squared error loss, and the cross-entropy loss using maximum likelihood.

A.3.1. Mean Squared Error

The Mean Squared Error (MSE) is defined by

$$\mathrm{MSE}\left\{\hat{\theta}\right\} = E\left\{\left(\hat{\theta} - \theta\right)^2\right\}$$

and it is well known that the MSE can be expressed by

$$\mathrm{MSE}\left\{\hat{\theta}\right\} = \mathrm{Var}\left\{\hat{\theta}\right\} + \mathrm{Bias}\left\{\hat{\theta}\right\}^2$$

which in turn leads to the conclusion that the estimator with smallest MSE is the estimator with the smallest variance among all unbiased estimators.

While the MSE requires the knowledge of the distribution of the estimator, the mean squared error loss function is concerned with computing the observed loss between a sample and a set of corresponding estimates. The MSE loss function is defined by

$$\text{MSE}_{\text{loss}} = \frac{1}{n} \sum_{i=1}^{n} (\hat{y}_i - y_i)^2$$

where the loss is computed for a set of predictions $\{\hat{y}_i, i = 1, \ldots, n\}$ with regard to a set of target values $\{y_i, i = 1, \ldots, n\}$. It can be shown that under mild conditions (Theodoridis & Koutroumbas, 2009), the minimization of the MSE loss function by gradient descent with decreasing learning rate (section 2.2.6) converges to the minimum of the expectation.

The following sections will establish the role of the MSE loss function in the context of regression.

A.3.1.1. Relation to regression under the assumption of constant variance

Let $Y_i \sim N(\mu_i, \sigma), i = 1, \ldots, n$ be a random sample, and assume that μ_i can be modeled by a function $f(\cdot)$, such that $\mu_i = f(\mathbf{x}_i; \boldsymbol{\theta})$. I.e., we have a regression model expressed by

$$y_i = f(\mathbf{x}_i; \boldsymbol{\theta}) + \epsilon_i$$

where the error terms are independent and identically distributed (iid) normal random variables.

The likelihood function of $\boldsymbol{\theta}$ (see section 2.1.3.5) is given by

$$\begin{aligned} \mathcal{L}(\boldsymbol{\theta}; \mathbf{y}, X) &= \prod_{i=1}^{n} f(y_i; \theta, X, \sigma) \\ &= \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{1}{2}\frac{(f(\mathbf{x}_i, \boldsymbol{\theta}) - y_i)^2}{\sigma^2}} \end{aligned}$$

with the corresponding log-likelihood given by

$$l(\boldsymbol{\theta}; \mathbf{y}, X) = -n \cdot \log(\sqrt{2\pi}\sigma) - \frac{1}{2\sigma^2} \sum_{i=1}^{n} (f(\mathbf{x}_i, \boldsymbol{\theta}) - y_i)^2$$

We find the maximum likelihood estimator for $\boldsymbol{\theta}$ by maximizing the log-likelihood, which is done by

$$\begin{aligned}
\hat{\boldsymbol{\theta}}_{MLE} &= \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \; -n \cdot \log(\sqrt{2\pi}\sigma) - \frac{1}{2\sigma^2}\sum_{i=1}^{n}\left(f(\mathbf{x}_i, \boldsymbol{\theta}) - y_i\right)^2 \\
&= \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \; \sum_{i=1}^{n}\left(f(\mathbf{x}_i, \boldsymbol{\theta}) - y_i\right)^2 \\
&= \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \; ||f(X, \boldsymbol{\theta}) - \mathbf{y}||_2
\end{aligned}$$

where the final equality denotes the euclidean norm between the sample and the predictions. Thus, we see that maximizing the likelihood under the normal assumption is equivalent to minimizing the mean squared error loss function defined by

$$\mathrm{MSE}_{\mathrm{loss}} = \frac{1}{n}\sum_{i=1}^{n}\left(f(\mathbf{x}_i, \boldsymbol{\theta}) - y_i\right)^2$$

or the euclidean distance between the labels $\mathbf{y}$ and the predictions $\hat{\mathbf{y}}$.

A.3.1.2. Relation to regression under the assumption of non-constant variance

Let again $Y_i \sim N(\mu_i, \sigma_i), i = 1, \ldots, n$ be a random sample, and assume that μ_i can be modeled by a function $f(\cdot)$, such that $\mu_i = f(\mathbf{x}_i; \boldsymbol{\theta})$. I.e., we have a regression model of the form

$$y_i = f(\mathbf{x}_i; \boldsymbol{\theta}) + \epsilon_i$$

where the error terms are independent normal random variables with unequal variances.

The likelihood function of $\boldsymbol{\theta}$ is given by

$$\begin{aligned}
\mathcal{L}(\boldsymbol{\theta}; \mathbf{y}, X) &= \prod_{i=1}^{n} f(y_i; \theta, \mathbf{x}_i, \sigma_i) \\
&= \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}\sigma_i} e^{-\frac{1}{2}\frac{\left(f(\mathbf{x}_i, \boldsymbol{\theta}) - y_i\right)^2}{\sigma_i^2}}
\end{aligned}$$

and the corresponding log-likelihood function of $\boldsymbol{\theta}$ is given by

$$l(\boldsymbol{\theta}; \mathbf{y}, X) = -n \cdot \log(\sqrt{2\pi}\sigma_i) - \frac{1}{2}\sum_{i=1}^{n}\left(\frac{f(\mathbf{x}_i, \boldsymbol{\theta}) - y_i}{\sigma_i}\right)^2$$

The maximum likelihood estimator for $\boldsymbol{\theta}$ is obtained by

$$\begin{aligned}
\hat{\boldsymbol{\theta}}_{MLE} &= \underset{\boldsymbol{\theta}}{\text{argmax}} \; -n \cdot \log(\sqrt{2\pi}\sigma_i) - \frac{1}{2}\sum_{i=1}^{n}\left(\frac{f(\mathbf{x}_i, \boldsymbol{\theta}) - y_i}{\sigma_i}\right)^2 \\
&= \underset{\boldsymbol{\theta}}{\text{argmin}} \; \sum_{i=1}^{n}\left(\frac{f(\mathbf{x}_i, \boldsymbol{\theta}) - y_i}{\sigma_i}\right)^2 \\
&= \underset{\boldsymbol{\theta}}{\text{argmin}} \; ||f(X, \boldsymbol{\theta}) - \mathbf{y}||_{\text{M}}
\end{aligned}$$

where the final equality denotes the Mahalanobis distance between the distribution given by $f(X, \boldsymbol{\theta})$, and the sample given by $\mathbf{y}$. If we for instance assume that $y_i > 0$ and that $\sigma_i = c \cdot y_i$ for all i, the MLE is obtained by minimizing the mean squared relative error loss function given by

$$\text{MSRE}_{\text{loss}} = \frac{1}{n}\sum_{i=1}^{n}\left(1 - \frac{f(\mathbf{x}_i, \boldsymbol{\theta})}{y_i}\right)^2$$

A.3.2. Cross Entropy

In classification problems the most commonly used loss function is the cross-entropy loss function. There are several interpretations of this, and the objective function itself can be derived as the a function to be minimized when estimating the model parameters by maximum likelihood estimation. I will derive the cross-entropy loss function by MLE using the categorical distribution defined in section 2.1.3.7.

Assume that we have a set of observations, $\mathcal{X} = \{\mathbf{x}_i\}_{i=1}^{n}$, with a corresponding vector of independent labels $\underset{(n\times 1)}{\mathbf{y}} = (y_i)$, where we assume that

$$y_i \sim \text{Cat}(k, \mathbf{p}_i)$$

with unknown probabilities

$$\underset{(k\times 1)}{\mathbf{p}_i} = (p_{ij})$$

Furthermore, assume that we can model the probabilities by some function $g(\cdot)$ such that $\mathbf{p}_i = g(\mathbf{x}_i, \boldsymbol{\theta})$. The likelihood function for $\boldsymbol{\theta}$ can now be expressed by

$$\begin{aligned}\mathcal{L}(\boldsymbol{\theta}; \boldsymbol{\mathcal{X}}, \mathbf{y}) &= \prod_{i=1}^{n} P(Y = y_i) \\ &= \prod_{i=1}^{n}\prod_{j=1}^{k} p_{ij}^{[y_j = y_i]}\end{aligned}$$

with the log-likelihood given by

$$l(\boldsymbol{\theta}; \boldsymbol{\mathcal{X}}, \mathbf{y}) = \sum_{i=1}^{n}\sum_{j=1}^{k} [y_j = y_i] \log p_j$$

Our objective is to find the parameters $\hat{\boldsymbol{\theta}}$ that maximizes the likelihood of the data. This is done by

$$\begin{aligned}\hat{\boldsymbol{\theta}} &= \underset{\boldsymbol{\theta}}{\operatorname{argmax}} \{l(\boldsymbol{\theta}; \boldsymbol{\mathcal{X}}, \mathbf{y})\} \\ &= \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \{-l(\boldsymbol{\theta}; \boldsymbol{\mathcal{X}}, \mathbf{y})\} \\ &= \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \left\{-\sum_{i=1}^{n}\sum_{j=1}^{k} [y_i = y_j] \log p_j\right\} \\ &= \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \left\{-\sum_{i=1}^{n}\sum_{j=1}^{k} [y_i = y_j] \log g(\mathbf{x}_i, \boldsymbol{\theta})_j\right\} \\ &= \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \left\{-\sum_{i=1}^{n}\sum_{j=1}^{k} P(y_j) \log g(\mathbf{x}_i, \boldsymbol{\theta})_j\right\} \\ &= \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \left\{\sum_{i=1}^{n} -E_{Y_i \sim P}\{\log Q(y_i)\}\right\} \\ &= \underset{\boldsymbol{\theta}}{\operatorname{argmin}} \left\{\sum_{i=1}^{n} H(P, Q)\right\}\end{aligned}$$

which we recognize as the minimization of the total cross-entropy (section 2.1.3.8) of $Q \sim \text{Cat}(k, p_j = g(\mathbf{x_i}, \boldsymbol{\theta})_j)$ with regard to $P \sim \text{Cat}(k, p_j = [y_i = j])$. I.e., the cross-entropy loss for a k-class classification problem is defined by

$$\text{CE}_{\text{loss}} = -\sum_{i=1}^{n}\sum_{j=1}^{k} [y_i = y_j] \log \hat{y}_{ij} \tag{22}$$

where $\hat{y}_{ij}$ is the model output j for sample i.

For a binary classification problem this simplifies to

$$\begin{aligned} \mathrm{CE}_{\text{binary}} &= -\sum_{i=1}^{n}\sum_{j=1}^{2}[y_i = y_j]\log \hat{y}_{ij} \\ &= -\sum_{i=1}^{n}[y_i = y_0]\log \hat{y}_{i0} + [y_i = y_1]\log \hat{y}_{i1} \\ &= -\sum_{i=1}^{n}(1 - y_i)\log(1 - \hat{y}_i) + y_i \log \hat{y}_i \end{aligned} \tag{23}$$

by letting

$$y_i = \begin{cases} 1, & \text{sample } i \text{ in class } 1 \\ 0, & \text{sample } i \text{ in class } 0 \end{cases}$$

and defining $\hat{y}_i$ to be the estimated probability of class 1 for sample i.

Note from equation 4, 3 and 2 in section 2.1.3.8 that the cross entropy can be written as

$$H(P, Q) = D_{KL}(P||Q) + H(P)$$

and note also that the entropy for the categorical distribution

$$H(P) = -\sum_{i=1}^{k} p_i \log p_i$$

is constant, implying that

$$\operatorname*{argmin}_{\boldsymbol{\theta}} \left\{ \sum_{i=1}^{n} H(P, Q) \right\} = \operatorname*{argmin}_{\boldsymbol{\theta}} \left\{ \sum_{i=1}^{n} D_{KL}(P||Q) \right\}$$

Thus the minimization of the cross-entropy is equivalent to the minimization of the KL-divergence, both computed for the distribution given the model, with regard to the saturated distribution - the distribution given the labels.

A.4. Variations of the gradient descent algorithm

This section covers the evolution of the gradient descent algorithm described in section 2.2.6 in very brief detail.

A.4.1. Momentum and Nestorov momentum

An improved variant of the gradient descent update equation results from the addition of a momentum term, resulting in the update rule

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^{t} - \mathbf{v}^{t}$$

where the new direction

$$\mathbf{v}^{t} = \gamma \mathbf{v}^{t-1} + \alpha \nabla_{\boldsymbol{\theta}} J\left(\boldsymbol{\theta}; \mathbf{x}, \mathbf{y}\right)\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{t}}$$

is a weighted sum of the direction of steepest descent at the last timestep - the momentum - and the direction of steepest descent at the current time step. Closely related to the momentum term is the Nestorov momentum term resulting direction update

$$\mathbf{v}^{t} = \gamma \mathbf{v}^{t-1} + \alpha \nabla_{\boldsymbol{\theta}} J\left(\boldsymbol{\theta}; \mathbf{x}, \mathbf{y}\right)\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{t}-\gamma\mathbf{v}^{t-1}}$$

where the gradients instead are computed at the estimated next position.

A.4.2. AdaGrad

AdaGrad (Duchi et al., 2011) is a proposed method with an adaptive learning rate. The update equation is written by

$$\boldsymbol{\theta}^{t+1} = \boldsymbol{\theta}^{t} - \frac{\alpha}{\sqrt{G^{t}+\epsilon}} \odot \nabla_{\boldsymbol{\theta}} J\left(\boldsymbol{\theta}; \mathbf{x}, \mathbf{y}\right)\big|_{\boldsymbol{\theta}=\boldsymbol{\theta}^{t}}$$

where G^{t} is a vector of accumulated squared gradients. While the aim of AdaGrad was to scale the learning rate for all dimensions (Gylberth, 2018), the division by the accumulated squared gradients has the unwanted effect of gradients becoming vanishingly small during training. This is addressed by methods such as AdeDelta (Zeiler, 2012) and RMSprop, which are evolution's of AdaGrad that aims to reduce the vanishing gradient effect by replacing the squared gradients with a rolling average of previous gradients.

A.4.3. The Adam optimizer

The Adam algorithm (Kingma & Ba, 2017) provides an alternative to the normal gradient descent update rule described in section 2.2.6. In Adam, the weights are updated by replacing the usual gradient with the standardized expected gradient. The Adam algorithm is described in algorithm 6, where $J(\boldsymbol{\theta})$ is the objective function to be minimized and $\beta_1, \beta_2 \in (0, 1)$ are chosen parameters along with the learning rate $\alpha \in \mathbb{R}^+$. $\mathbf{m}_t$ and $\mathbf{v}_t$ are estimates for the gradient and gradient squared respectively, both computed as moving averages over the current and previous gradients, while $\hat{\mathbf{m}}_t$ and $\hat{\mathbf{v}}_t$ are the bias-corrected first and second moment estimates. The usage of moving averages counteract the negative effect experienced in methods such as AdaGrad (Duchi et al., 2011), where gradients sometime converges to zero as the number of epochs increases.

Algorithm 6 Gradient descent using the Adam algorithm

```
Require: $\boldsymbol{\theta}_0$
  $t = 0$
  $\mathbf{m}_0 = \mathbf{0}$
  $\mathbf{v}_0 = \mathbf{0}$
  while training do
      $t = t + 1$
      $\mathbf{g}_t = \nabla_{\boldsymbol{\theta}} J(\boldsymbol{\theta})|_{\boldsymbol{\theta}=\boldsymbol{\theta}_t}$
      $\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1 - \beta_1)\mathbf{g}_t$
      $\hat{\mathbf{m}}_t = \frac{\mathbf{m}_t}{1-\beta_1^t}$
      $\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1 - \beta_2)\mathbf{g}_t \odot \mathbf{g}_t$
      $\hat{\mathbf{v}}_t = \frac{\mathbf{v}_t}{1-\beta_2^t}$
      $\boldsymbol{\theta}_t = \boldsymbol{\theta}_{t-1} - \alpha \frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{v}}_t + \epsilon}}$
  end while
```

A.5. The vanishing gradient problem

Vanishing gradients is a problem that concerns deep neural networks in general, and Recurrent Neural Networks in particular. The phenomena occurs when gradients in the early layers of a network becomes small, resulting in an optimization procedure that converges slowly, and in practice fails in optimizing the early layer parameters. A reason for vanishing gradients can be illustrated in a simplified example.

If we consider a simple network with L layers consisting of single units, and a single input sample, the partial derivative of the loss with respect to the weight in layer l can be written as

$$\frac{\partial J}{\partial w^{(l)}} = \delta^{(l)} a^{(l-1)} \tag{24}$$

where we use the definition $\delta_j^{(l)} = \frac{\partial}{\partial z^{(l)}}$. By repeated applications of the chain rule (section 2.1.3.2), we can expand the right hand side in equation 24 by

$$\begin{aligned}\frac{\partial J}{\partial w^{(l)}} &= f'(z^{(l)})w^{(l+1)}\delta^{(l+1)}a^{(l-1)} \\ &= f'(z^{(l)})w^{(l+1)}f'(z^{(l+1)})w^{(l+2)}\cdots f'(z^{(L-1)})w^{(L)}\delta^{(L)}a^{(L-1)} \\ &= \left(\prod_{i=l}^{L-1} f'(z^{(i)})\cdot w^{(i+1)}\right)\cdot \delta^{(L)}\cdot a^{(l-1)}\end{aligned}$$

where we note that since the first term consists of repeated multiplications of the activation function derivative, the magnitude of the final gradient will depend upon the chosen activation function. The vanishing gradient problem occurs when the network is deep, and the magnitude of the derivative of the chosen activation function is less than one. A similar problem - exploding gradients - can be experienced by applying activation functions with large magnitude derivatives.

The popularity of the ReLU activation function (appendix A.2.4) is largely due to the fact that its derivative is equal to one. Examples of activation functions that may induce vanishing gradients are the sigmoid function and the tanh function discussed in section A.2.

B. Appendix B: Statistical topics

B.1. Multiple linear regression and Ridge regression

Linear regression

Let $\mathbf{y}$ denote a vector of n observations, and let $\mathbf{x}_1, \ldots, \mathbf{x}_p$ denote a set of predictors. A multiple linear regression model is defined by the model

$$\mathbf{y} = X\mathbf{w} + \boldsymbol{\epsilon}$$

where we assume that $\boldsymbol{\epsilon} \sim (\mathbf{0}, \sigma I)$, and where the matrix X is defined by

$$\begin{bmatrix} 1 & x_{11} & \cdots & x_{1p} \\ 1 & x_{21} & \cdots & x_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n1} & \cdots & x_{np} \end{bmatrix}$$

If we define the error sum of squares (SSE) by

$$SSE = (\mathbf{y} - X\mathbf{w})^T (\mathbf{y} - X\mathbf{w})$$

we can obtain the ordinary least squares (OLS) estimates by finding the weights $\mathbf{w}$ that corresponds to a stationary point of SSE. These are obtained by

$$\begin{aligned} \nabla_{\mathbf{w}} (\mathbf{y} - X\mathbf{w})^T (\mathbf{y} - X\mathbf{w}) &= 0 \\ &\Rightarrow \\ -2\mathbf{y}^T X + 2X^T X\mathbf{w} &= 0 \\ &\Rightarrow \\ X^T X\mathbf{w} &= X^T\mathbf{y} \\ &\Rightarrow \\ \mathbf{w} &= (X^T X)^{-1} X^T\mathbf{y} \end{aligned}$$

Thus, the regression model weights that minimizes the error sum of squares are given by

$$\underset{\mathbf{w}}{\operatorname{argmin}}\, SSE = \left(X^T X\right)^{-1} X^T\mathbf{y}$$

and it can be shown (Johnson & Wichern, 2013) that the the OLS estimates indeed corresponds to the global minima of SSE.

If we in addition assume that $\boldsymbol{\epsilon} \sim N(\mathbf{0}, \sigma I)$, we saw in appendix A.3.1.1 that the MLE estimates of $\mathbf{w}$ are identical to the OLS estimates of $\mathbf{w}$.

Ridge regression

Ridge regression presents an alternative estimation method that is motivated by finding parameter estimates with a lower variance than the estimates obtained by ordinary least squares. The objective function in Ridge regression is expressed by

$$SSE_{Ridge} = SSE + \gamma\mathbf{w}^T\mathbf{w}$$

where the loss function is altered by the addition of the squared 2-norm of the parameters which are scaled with a constant γ. The weights are obtained, as in the case of OLS, by taking the gradient of the objective function with regard to $\mathbf{w}$, setting it equal to zero, and solving the equation for $\mathbf{w}$. The derivation is given by

$$\begin{aligned}
\nabla_{\mathbf{w}} \left(\mathbf{y} - X\mathbf{w}\right)^T \left(\mathbf{y} - X\mathbf{w}\right) + \gamma \mathbf{w}^T\mathbf{w} &= 0 \\
&\Rightarrow \\
-2\mathbf{y}^T X + 2(X^T X + \gamma I)\mathbf{w} &= 0 \\
&\Rightarrow \\
(X^T X + \gamma I)\mathbf{w} &= X^T\mathbf{y} \\
&\Rightarrow \\
\mathbf{w} &= (X^T X + \gamma I)^{-1} X^T \mathbf{y}
\end{aligned}$$

and the regression weights that minimizes SSE_{Ridge} are thus given by

$$\hat{\mathbf{w}}_{\text{Ridge}} = \left(X^T X + \gamma I\right)^{-1} X^T \mathbf{y}$$

B.2. The Kolmogorov-Smirnov test for equality of distributions

Let $\mathcal{X}$ be a random sample of size n, and let the empirical distribution function be defined by

$$F_n(x) = \frac{\text{cardinality}\{x^* \in \mathcal{X} x^* < x\}}{n}$$

The Kolmogorov-Smirnov test statistic for equality of distributions is defined by

$$D_{n,m} = \sup_x |F_{1,n}(x) - F_{2,m}(x)|$$

where $F_{1,n}(x)$ and $F_{2,m}(x)$ are the empirical distribution functions of two independent samples with respective sample sizes n and m.

The null hypothesis of equal distributions is rejected at level α if

$$D_{n,m} > \sqrt{-\log\left(\frac{\alpha}{2}\right) \cdot \frac{1 + \frac{m}{n}}{2m}}$$

B.3. Kernel density estimation

Kernel density estimation is a non-parametric method for estimating probability density functions. Given a random sample $X_1, \ldots, X_n$ from a population with pdf $f(x)$, the kernel density estimator for f is given by

$$\hat{f}(x) = \frac{1}{nh} \sum_{i=1}^{n} K\left(\frac{x - X_i}{h}\right)$$

where h is a bandwidth parameter and $K(\cdot)$ is a kernel function. A range of kernel functions exists, where the Gaussian kernel function defined by the standard normal pdf

$$\phi(x) = \frac{1}{\sqrt{2\pi}} e^{-\frac{x^2}{2}}$$

is the one applied in this thesis. The choice of bandwidth is non-trivial, and a range of bandwidth selection procedures exists. A thorough survey of methods is provided by Sheather (2004). Silverman's rule of thumb, which is defined by

$$h_{\text{SROT}} = 0.9An^{-\frac{1}{5}}$$

is used in this thesis, where A is defined by

$$A = \min\left\{\hat{\sigma}, \frac{\text{IQR}}{1.34}\right\}$$

and IQR denotes the sample interquartile range.

C. Appendix C: Deep learning applications

C.1. Residual skip connections

Residual skip connections (He et al., 2015) provides a variation to the ordinary feedforward design of neural networks, where the concept behind residual skip connections is to add a connection between layers that are otherwise separated by intermediate layers. Figure 48 displays this idea, where the input to, say layer $l + 1$ is given by the sum of the output from layer $l - 2$ and the output from layer l. I.e.,

$$\mathbf{a}^{(l+1)} = f(\mathbf{a}^{(l-2)}) + \mathbf{a}^{(l-2)}$$

The term residual refers to the fact that the function $f(\mathbf{a}^{(l-2)})$ can be considered the residual between $\mathbf{a}^{(l+1)}$ and $\mathbf{a}^{(l-2)}$, and the layers between x and the output $F(x)$ in figure 48 is said to constitute a residual block.

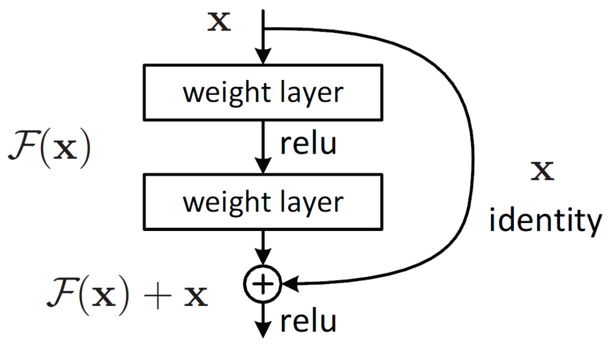


Figure 48: *A residual block example (He et al., 2015)*

Residual skip connections counteracts the vanishing gradient problem (appendix A.5) by the identity mapping. This can seen by the equation

$$\left(\frac{\partial \mathbf{a}^{(l+1)}}{\partial \mathbf{a}^{(l-2)}}\right) = \left(\frac{\partial \mathbf{a}^{(l)}}{\partial \mathbf{a}^{(l-1)}}\right)\left(\frac{\partial \mathbf{a}^{(l-1)}}{\partial \mathbf{a}^{(l-2)}}\right) + I$$

where the addition of the identity matrix ensures a nonzero gradient between the output and the input of the residual block.

C.2. The Xception architecture

The Xception architecture proposed by Chollet (2017), is an evolution of the inception models proposed by Szegedy et al. (2014) and Szegedy et al. (2015). All mentioned models are CNNs, however they differ from ordinary CNNs in how the convolutions are carried out. The hypothesis of the Xception model is that all cross-channel and spatial correlations can be decoupled, and the model relies solely on separable depthwise convolutions (section 2.3.4), which replaces normal convolutions for all layers except at the input. Figure 49 shows the architecture details and, although not illustrated in the figure, all convolutional operations are followed by a batch normalization layer (section 2.3.1). As shown in the figure, the model uses residual connections (C.1) between blocks.

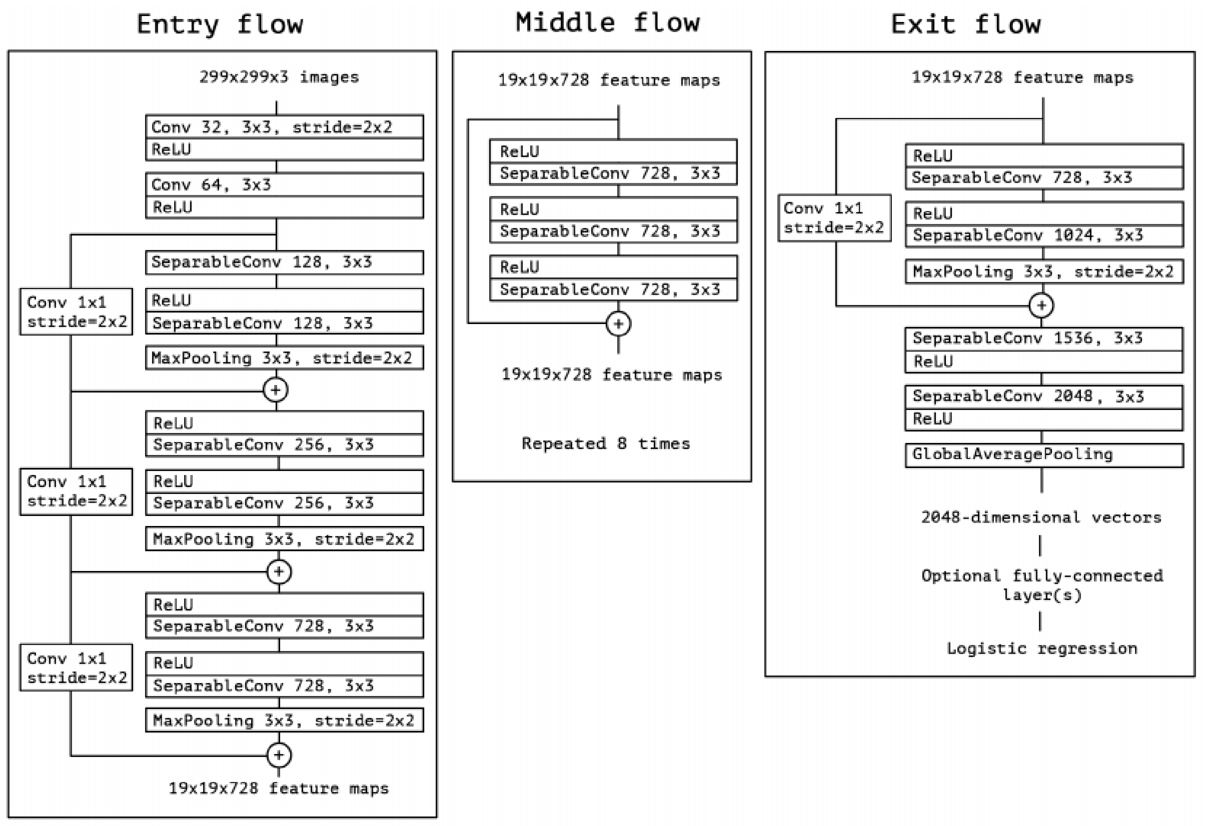


Figure 49: *Overview of the Xception architecture (Chollet, 2017)*

D. Appendix D: Related studies

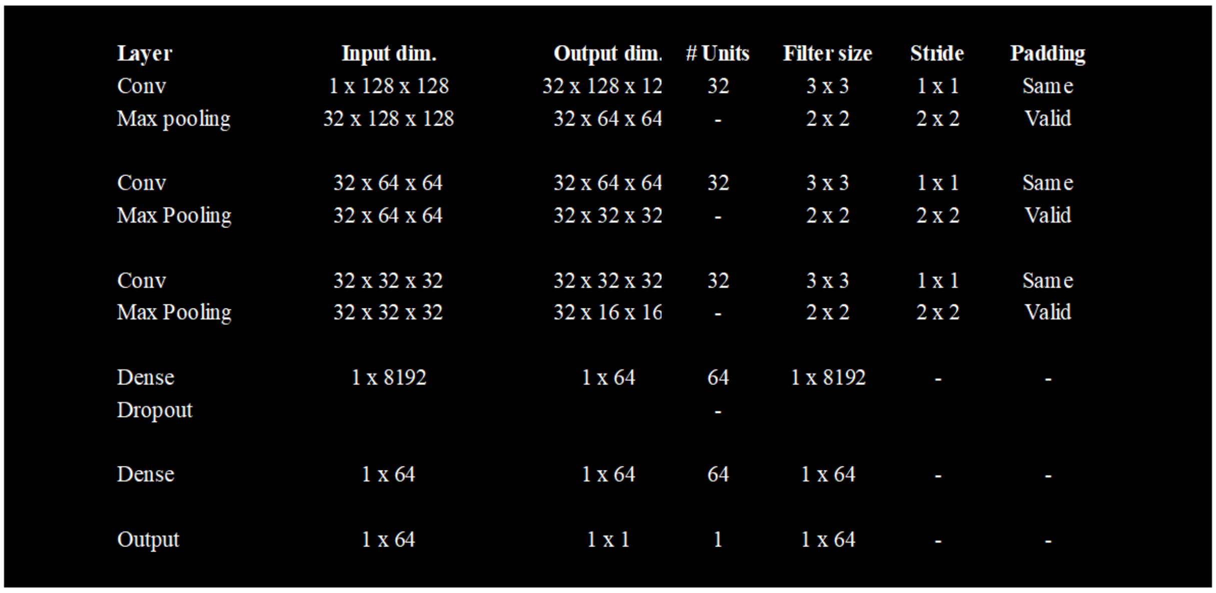
D.1. Results presented at the NORDSTAT 2021 conference

This section will briefly address the cod otolith classification results obtained by Alf Harbitz (IMR), Iver Martinsen (UiT) and Filippo Maria Bianchi (UiT), which were presented at the NORDSTAT 2021 conference. The results were obtained by combining three different classification methods applied on cod otoliths. The first method was based on EFDs (Kuhl & Giardina, 1982) with 10 frequency components, while the second method was based on the MIRR technique (Harbitz, 2016) using 5 frequency components. In both these methods the otoliths was classified using linear discriminant analysis (LDA). The third method was based on deep learning, using the CNN architecture shown in figure 50.

EFD	MIRR	DL	$\text{RECALL}_{\text{NCC}}$	σ_{NCC}	$\text{RECALL}_{\text{NEAC}}$	σ_{NCC}
x	x	x	93.28	1.02	85.88	0.72
x	x		88.40	1.29	81.30	1.05
x		x	93.16	1.04	84.17	0.70
	x	x	92.48	1.12	85.89	0.72
	x		82.47	1.34	81.33	1.05
x			85.81	1.25	80.74	1.08
		x	88.28	1.10	81.93	0.59

Table 5: *Classification accuracy using a combination of classification scores*

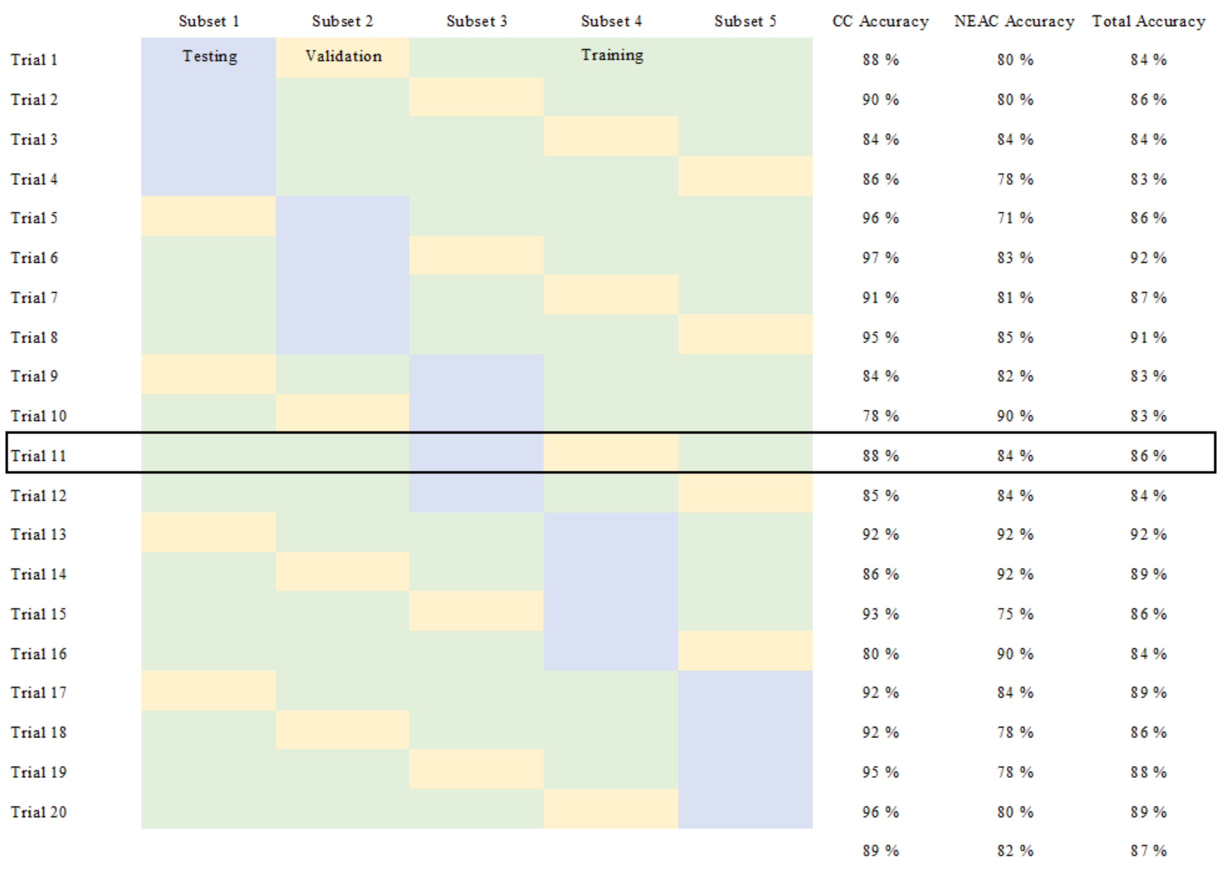
The results, presented in table 5, proved a significant increase in performance when combining methods with each other. The total test recall was 93 % for NCC, and 86 % for NEAC.



Layer	Input dim.	Output dim.	# Units	Filter size	Stride	Padding
Conv	1 x 128 x 128	32 x 128 x 12	32	3 x 3	1 x 1	Same
Max pooling	32 x 128 x 128	32 x 64 x 64	-	2 x 2	2 x 2	Valid
Conv	32 x 64 x 64	32 x 64 x 64	32	3 x 3	1 x 1	Same
Max Pooling	32 x 64 x 64	32 x 32 x 32	-	2 x 2	2 x 2	Valid
Conv	32 x 32 x 32	32 x 32 x 32	32	3 x 3	1 x 1	Same
Max Pooling	32 x 32 x 32	32 x 16 x 16	-	2 x 2	2 x 2	Valid
Dense	1 x 8192	1 x 64	64	1 x 8192	-	-
Dropout			-			
Dense	1 x 64	1 x 64	64	1 x 64	-	-
Output	1 x 64	1 x 1	1	1 x 64	-	-

Figure 50: *CNN architecture used to obtain the results presented at the NORDSTAT conference*

Figure 51 shows the recall for the two classes from the 20 different trials resulting from $k * l$-fold cross-validation procedure (section 2.5.3) with $k = 5$ and $l = 4$.



	Subset 1	Subset 2	Subset 3	Subset 4	Subset 5	CC Accuracy	NEAC Accuracy	Total Accuracy
Trial 1	Testing	Validation	🟩	Training	🟩	88 %	80 %	84 %
Trial 2	🟦	🟩	🟨	🟩	🟩	90 %	80 %	86 %
Trial 3	🟦	🟩	🟩	🟨	🟩	84 %	84 %	84 %
Trial 4	🟦	🟩	🟩	🟩	🟨	86 %	78 %	83 %
Trial 5	🟨	🟦	🟩	🟩	🟩	96 %	71 %	86 %
Trial 6	🟩	🟦	🟨	🟩	🟩	97 %	83 %	92 %
Trial 7	🟩	🟦	🟩	🟨	🟩	91 %	81 %	87 %
Trial 8	🟩	🟦	🟩	🟩	🟨	95 %	85 %	91 %
Trial 9	🟨	🟩	🟦	🟩	🟩	84 %	82 %	83 %
Trial 10	🟩	🟨	🟦	🟩	🟩	78 %	90 %	83 %
Trial 11	🟩	🟩	🟦	🟨	🟩	88 %	84 %	86 %
Trial 12	🟩	🟩	🟦	🟩	🟨	85 %	84 %	84 %
Trial 13	🟨	🟩	🟩	🟦	🟩	92 %	92 %	92 %
Trial 14	🟩	🟨	🟩	🟦	🟩	86 %	92 %	89 %
Trial 15	🟩	🟩	🟨	🟦	🟩	93 %	75 %	86 %
Trial 16	🟩	🟩	🟩	🟦	🟨	80 %	90 %	84 %
Trial 17	🟨	🟩	🟩	🟩	🟦	92 %	84 %	89 %
Trial 18	🟩	🟨	🟩	🟩	🟦	92 %	78 %	86 %
Trial 19	🟩	🟩	🟨	🟩	🟦	95 %	78 %	88 %
Trial 20	🟩	🟩	🟩	🟨	🟦	96 %	80 %	89 %
						89 %	82 %	87 %

Figure 51: *Validation results from the NORDSTAT trial procedure*

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., … Zheng, X. (2015). *TensorFlow: Large-scale machine learning on heterogeneous systems.* Retrieved from `https://www.tensorflow.org/` (Software available from tensorflow.org)

Adams, R., & Essex, C. (2013). *Calculus: A complete course.* Pearson. Retrieved from `https://books.google.no/books?id=5aaEMAEACAAJ`

Amidi, A., & Amidi, S. (2019). *Deep learning tips and tricks cheatsheet.* Retrieved from `https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-deep-learning-tips-and-tricks`

Bach, S., Binder, A., Montavon, G., Klauschen, F., Müller, K.-R., & Samek, W. (2015, 07). On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PLOS ONE*, *10*(7), 1-46. Retrieved from `https://doi.org/10.1371/journal.pone.0130140` doi: 10.1371/journal.pone.0130140

Berg, E., Sarvas, T., Harbitz, A., Fevolden, S., & Salberg, A. (2005, Mar.). Accuracy and precision in stock separation of north-east arctic and norwegian coastal cod by otoliths—comparing readings, image analyses and a genetic method. *Freshw. Res.*, *56*, 753–762.

Bergstad, O., Jørgensen, T., & Dragesund, O. (1987). Life history and ecology of the gadoid resources of the barents sea. *Fish. Res.*, *5*, 119–161.

Biran, O., & Cotton, C. V. (2017). Explanation and justification in machine learning : A survey or..

Campana, S., & Thorrold, S. (2001). Otoliths, increments and elements: keys to a comprehensive understanding of fish populations? *Can. J. Fish. Aquatic. Sci.*, *58*, 30-38.

Casella, G., & Berger, R. (2001). *Statistical inference.* Duxbury Resource Center. Textbook Binding.

Chollet, F. (2017). *Xception: Deep learning with depthwise separable convolutions.*

Chollet, F., et al. (2015). *Keras.* GitHub. Retrieved from `https://github.com/fchollet/keras`

Clairvoyant Blog. (2019). *The ascent of gradient descent.* Retrieved from `https://blog.clairvoyantsoft.com/the-ascent-of-gradient-descent-23356390836f`

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 ieee conference on computer vision and pattern recognition* (pp. 248–255).

Dhalla, A. (2021). *The rise and fall of the perceptron.* Retrieved from `https://ai.plainenglish.io/the-rise-and-fall-of-the-perceptron-c04ae53ea465`

Dobson, A. J. (2002). *An introduction to generalized linear models* (2nd ed.) [Book]. Chapman & Hall/CRC Boca Raton.

Duchi, J., Hazan, E., & Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, *12*(61), 2121-2159. Retrieved from `http://jmlr.org/papers/v12/duchi11a.html`

Gonzalez, R. C., & Woods, R. E. (2008). *Digital image processing.* Upper Saddle River, N.J.: Prentice Hall. Retrieved from `http://www.amazon.com/Digital-Image-Processing-3rd-Edition/dp/013168728X`

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning.* MIT Press. (http://www.deeplearningbook.org)

Gylberth, R. (2018). *An introduction to adagrad.* Retrieved from https://medium.com/konvergen/an-introduction-to-adagrad-f130ae871827

Harbitz, A. (2016). Parameter-sparse modification of fourier methods to analyse the shape of closed contours with application to otolith outlines. *Marine and Freshwater Research*, *67*(7), 1049-1058. Retrieved from https://www.publish.csiro.au/paper/MF15087 doi: https://doi.org/10.1071/MF15087

Harbitz, A., & Albert, O. T. (2015, 04). Pitfalls in stock discrimination by shape analysis of otolith contours. *ICES Journal of Marine Science*, *72*(7), 2090-2097. Retrieved from https://doi.org/10.1093/icesjms/fsv048 doi: 10.1093/icesjms/fsv048

Hassan, H., Negm, A., Zahran, M., & Saavedra, O. (2015, 12). Assessment of artificial neural network for bathymetry estimation using high resolution satellite imagery in shallow lakes: Case study el burullus lake. *International Water Technology Journal*, *5*.

He, K., Zhang, X., Ren, S., & Sun, J. (2015). *Deep residual learning for image recognition.*

Ioffe, S., & Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR, abs/1502.03167*. Retrieved from http://arxiv.org/abs/1502.03167

Jakobsen, T. (1987). Coastal cod in northern norway. *Fisheries Research 5*, 223–234. doi: 10.1016/0165-7836(87)90042-7

Johnson, R., & Wichern, D. (2013). *Applied multivariate statistical analysis* (5. ed.). Upper Saddle River, NJ: Prentice Hall.

Kingma, D. P., & Ba, J. (2017). *Adam: A method for stochastic optimization.*

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, & K. Q. Weinberger (Eds.), *Advances in neural information processing systems* (Vol. 25). Curran Associates, Inc. Retrieved from https://proceedings.neurips.cc/paper/2012/file/c399862d3b9d6b76c8436e924a68c45b-Paper.pdf

Kuhl, F. P., & Giardina, C. R. (1982). Elliptic fourier features of a closed contour. *Computer Graphics and Image Processing*, *18*(3), 236-258. Retrieved from https://www.sciencedirect.com/science/article/pii/0146664X8290034X doi: https://doi.org/10.1016/0146-664X(82)90034-X

Lapuschkin, S., Wäldchen, S., Binder, A., Montavon, G., Samek, W., & Müller, K.-R. (2019, Mar). Unmasking clever hans predictors and assessing what machines really learn. *Nature Communications*, *10*(1). Retrieved from http://dx.doi.org/10.1038/s41467-019-08987-4 doi: 10.1038/s41467-019-08987-4

Lin, H., Shi, Z., & Zou, Z. (2017, 05). Maritime semantic labeling of optical remote sensing images with multi-scale fully convolutional network. *Remote Sensing*, *9*, 480. doi: 10.3390/rs9050480

Medium. (2018). *A comprehensive guide to convolutional neural networks — the eli5 way.* Retrieved from https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Min Lin, Q. C., & Yan, S. (2014). *Network in network.* https://arxiv.org/abs/1312.4400.

Moen, E., Handegard, N. O., Allken, V., Albert, O. T., Harbitz, A., & Malde, K. (2018, 12). Automatic interpretation of otoliths using deep learning. *PLOS ONE*, *13*(12), 1-14. Retrieved from https://doi.org/10.1371/journal.pone.0204713 doi: 10.1371/journal.pone.0204713

Montavon, G., Binder, A., Lapuschkin, S., Samek, W., & Müller, K.-R. (2019, 09). Layer-wise relevance propagation: An overview. In (p. 193-209). doi: 10.1007/978-3-030-28954-6_10

Ordoñez, A., Eikvil, L., Salberg, A.-B., Harbitz, A., Murray, S. M., & Kampffmeyer, M. C. (2020, 06). Explaining decisions of deep neural networks used for fish age prediction. *PLOS ONE*, *15*(6), 1-19. Retrieved from https://doi.org/10.1371/journal.pone.0235013 doi: 10.1371/journal.pone.0235013

Ranjan, C. (2019). *Understanding dropout with the simplified math behind it.* Retrieved from https://towardsdatascience.com/simplified-math-behind-dropout-in-deep-learning-6d50f3f47275

Rollefsen, G. (1933). The otoliths of the cod. *Fiskeridir. Skr. Ser. Havundersøkelser 4*, 1-14.

Ruder, S. (2017). *An overview of gradient descent optimization algorithms.*

Sauer, T. (2011). *Numerical analysis* (2nd ed.). USA: Addison-Wesley Publishing Company.

Sheather, S. J. (2004). Density Estimation. *Statistical Science*, *19*(4), 588 – 597. Retrieved from https://doi.org/10.1214/088342304000000297 doi: 10.1214/088342304000000297

Shibani Santurkar, A. I., Dimitris Tsipras, & Madry, A. (2019). How does batch normalization help optimization? *stat.ML*. Retrieved from https://arxiv.org/abs/1805.11604

Simonyan, K., Vedaldi, A., & Zisserman, A. (2014). *Deep inside convolutional networks: Visualising image classification models and saliency maps.*

Springenberg, J. T., Dosovitskiy, A., Brox, T., & Riedmiller, M. (2015). *Striving for simplicity: The all convolutional net.*

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, *15*(56), 1929-1958. Retrieved from http://jmlr.org/papers/v15/srivastava14a.html

Stransky, C., Baumann, H., Fevolden, S., Harbitz, A., Høie, H., Nedreaas, K., ... Skarstein, T. (2008, 04). Separation of norwegian coastal cod and northeast arctic cod by outer otolith shape analysis. *Fisheries Research*, *90*, 26-35. doi: 10.1016/j.fishres.2007.09.009

Sundararajan, M., Taly, A., & Yan, Q. (2017). *Axiomatic attribution for deep networks.*

SuperDataScience Team. (2018). *Convolutional neural networks (cnn): Summary.* Retrieved from https://www.superdatascience.com/blogs/convolutional-neural-networks-cnn-summary

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... Rabinovich, A. (2014). *Going deeper with convolutions.*

Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2015). *Rethinking the inception architecture for computer vision.*

Taly, A. (2018). *How to use integrated gradients (ig).* Retrieved from https://github.com/ankurtaly/Integrated-Gradients/blob/master/howto.md

Theodoridis, S., & Koutroumbas, K. (2009). *Pattern Recognition, Fourth Edition.* Academic Press. Hardcover.

Walpole, R. E., Myers, R. H., Myers, S. L., & Ye, K. (2007). *Probability & statistics for engineers and scientists* (8th ed.). Upper Saddle River: Pearson Education.

Wang, C.-F. (2018). *A basic introduction to separable convolutions.* Retrieved from `https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728`

Zeiler, M. D. (2012). *Adadelta: An adaptive learning rate method.*

Zhang, A., Lipton, Z. C., Li, M., & Smola, A. J. (2021). Dive into deep learning. *arXiv preprint arXiv:2106.11342*.