Employing Deep Learning for Fish Recognition

Adrian Reithaug

Master's thesis in Software Engineering at

Department of Computing, Mathematics, and Physics,
Western Norway University of Applied Sciences

Department of Informatics,
University of Bergen

June 2018

Western Norway
University of
Applied Sciences

Abstract

Underwater imagery processing is in high demand, but the unrestricted environment makes it difficult to develop methods for analyzing it. Not only is obtaining a dataset for a single species difficult, but there are reported 33 970 fish species [1], which makes it hard to cover all bases for fish recognition.

This thesis presents 20 deep neural networks trained for fish (salmon) recognition and provides a comparison and discussion of each model and classifier combination with their respective parameter tunings. A version of SSD Inception V2 achieves 84.64% mAP on the unique fish dataset, with 3.75 FPS, and achieves state-of-the-art accuracy for salmon recognition.

Additionally, a version of Faster R-CNN ResNet 152 achieves 83.78%, at 3.76 FPS, and a version of Faster R-CNN Inception V2 achieves 83.62%, at 2.57 FPS. The thesis, in addition to proposing a solution for recognizing fish, serves as a guide for selecting a deep neural network, and finetuning its parameters.

The unique dataset in this thesis consists of 20 069 labels of salmon obtained from high-resolution videos from fish farms. Some research use an openly available, but very imbalanced dataset with 27 370 labeled images [2], and using this dataset for recognizing either salmon, cod, sea trout, or mackerel for instance, would result in failure due to the large difference in fish shape.

The thesis proposes a solution for discarding video sequences where no fish is present. The proposed solution reduces storage size, reduces time spent creating datasets from the videos, and makes it easier to analyze the data. The thesis also proposes a solution for obtaining statistics of number of fish in rivers over time, which can drastically improve the current methods used by Statistics Norway in terms of accuracy and effectiveness.

Acknowledgements

I would like to gratefully acknowledge my supervisors at The Western Norway University of Applied Sciences, Daniel Patel, Harald Soleim, and Atle Geitung, for their continuous feedback, constructive criticism, and guidance throughout the duration of this thesis.

I would also like to thank Jose Ojeda at Steinsvik AS, and Ove Daae Lampe at Christian Michelsen Research AS, for upon request, providing the videos making up the dataset.

Finally, I would like to thank Lars Erik Ørgersen at FiskeTV AS for providing some inside knowledge for ideas where such a fish recognition application might be useful.

Adrian Reithaug

Glossary

A

Activation A function of the neuron's input.

Activation Function Takes the weighted sum of all inputs from the previous layer and generates an output value to the next layer.

Algorithm A finite set of defined instructions for reaching a state or solving a problem.

Architecture The structure of multiple connected components.

B

Backpropagation Algorithm for performing gradient descent on neural networks, to gradually minimize the difference between the output and the actual output. Each neuron’s values are calculated in a forward pass, then the error rate is calculated in a backward pass.

Batch The set of examples used in one iteration (i.e. one gradient update) of network training.

Batch Size The number of examples in a batch.

Bias An extra neuron in each layer assigned a constant value to allow for more variations of weights to be learned.

Binary Classification Outputs one of two mutually exclusive classes (e.g. fish, not fish).

Big Data Volumes of data too large to process using traditional techniques.

C

Chain Rule A formula for calculating the derivatives of composite functions, where composite functions are functions composed of functions inside other function(s).

Checkpoint Data that captures the state of variables in a network at a certain time.

Class One of a set of enumerated target values for a label (e.g. fish).

Classifying The process of applying a label to an object.

Connection The link between two or more objects.

Convolution The process of extracting a set of features from local regions of an input. Can refer to either a convolutional operation or convolutional layer.

Convolutional Filter A convolutional operation. Is a matrix with (often) seeded with random numbers.

Convolutional Layer The layer in which the convolutional filter steps over the input data (a matrix), which generates a feature map.

Convolutional Neural Network A neural network that has at least one convolutional layer, that utilize prior knowledge to make better decisions by using convolution.

Convolutional Operation A two-step operation consisting of element-wise multiplication of the convolutional filter and an input matrix, as well as summation of all values in the resulting product matrix.

Cost Synonym for loss. A summation of the errors made when evaluating the network’s performance, calculating the difference between the output and the actual output.

D

Dataset A collection of examples. Refers to data related to training and testing a network.

Deep Learning A branch of machine learning where data is fed through multi-layered neural networks to make decisions about a given type of data.

Dense Layer Synonym for fully connected layer. A hidden layer in which every input neuron is connected to every output neuron by a weight.

Derivative The amount by which a function is changing at one given point.

Dropout A regularization technique which ignores/removes random neurons during training. Used to reduce overfitting by reducing the complexity of the model.

E

Epoch A full training pass over the entire dataset, such that each example has been seen once.

Example One row of a dataset, which contains features and possibly a label. Consists of the coordinates (*x*, *y*, *width*, *height*) for a label from an image.

F

False Positive An example in which the network mistakenly predicts the positive class (e.g. predicting fish when it was not a fish).

Feature An input variable used in making predictions.

Framework A library or interface which simplifies the development process by providing an environment which contains a lot of written and tested functionality.

Fully Connected Layer Synonym for dense layer. A hidden layer in which every input neuron is connected to every output neuron by a weight.

G

Generalization The network's ability to correctly predict unseen data.

Global Minimum A value at a point which is lower than any other value at any point.

Gradient The vector of partial derivatives of the network function, which points in the direction of steepest ascent.

Gradient Clipping Limiting the range of the gradient values before applying them.

Gradient Descent An optimization algorithm for minimizing a (cost/loss) function by moving in the direction of the steepest descent.

Greyscale Value A value obtained from an image where each pixel is represented by one 8-bit byte, ranging from 0 (black) to 255 (white).

Ground Truth The target variable for training and testing data; the answer to the query.

H

Hidden Layer The intermediate layer(s) in a neural network. Responsible for performing computations and transferring and transforming information from the input- to the output layer.

Hypoxia A state where there is a lack of oxygen for normal life functions.

I

Input Layer The layer that receives the input data, the first layer in a neural network.

Intersection Over Union (IoU) A classification threshold to separate the positive class from the negative class. An IoU of 0.5 means that if the confidence of the network is higher than 50%, the prediction is classified as positive, otherwise negative.

Iteration A single update of the network's weights during training.

L

Label The answer or result of an example. A label contains a class (e.g. fish).

Layer A set of neurons that process input features, and the neurons' outputs.

Learning Rate A parameter used to train a network via gradient descent. The gradient descent algorithm multiplies the learning rate by the gradient.

Linear Regression A regression model that outputs a continuous value from a linear combination of features.

Local Minimum A value at a point which is lower than its nearest adjacent points on its left and right sides.

Loss Synonym for cost. A summation of the errors made when evaluating the network's performance, calculating the difference between the output and the actual output.

M

Machine Learning A system that trains a predictive network from input data, which solve given problems by using their knowledge to make their own predictions.

Matrix A rectangular array of numbers in rows and columns, treated as a single entity.

Mean Average Precision A metric for evaluating a model's performance. The mean of the average precision scores for each query.

Mini-batch A randomly selected subset of the entire batch of examples.

Minimum Bounding Box The smallest enclosing area for a set of points (*x*, *y*, *width*, *height*).

Model A set of algorithms with purpose of predicting bounding boxes of objects in a given image.

Momentum A gradient descent algorithm in which a learning step depends on the derivatives of the step(s) preceding it, as well as the current one.

N

Network Short version for neural network.

Neural Network A series of algorithms designed to identify patterns or relationships in a dataset, inspired by the human brain. Is composed of layers consisting of neurons.

Neuron A node in a neural network. It generates a single value by applying an activation function to the weighted sum of input values.

Noise An unwanted distortion in data. Can be caused by e.g. misclassification in labeling, or a poor sensor (due to low-quality equipment).

Normalization The process of converting a range of values to a standard range, e.g. from [300, 5000] to [0, 1].

O

Open Source A program in which the source code is freely available to everyone and may be distributed and modified based on the user's requirements.

Optimizer A specific implementation of the gradient descent algorithm. Example: Adam, Momentum, RMSProp.

Output Layer The final layer in a neural network, which outputs what the neural network thinks the answer(s) is/are.

Overfitting When a network tries to fit all the data points too close to their exact value (i.e. matches the training data too closely), causing the network to fail at making correct predictions on new data.

P

Parameter An internal or external variable for a network. Internal parameters are adjusted by the network during training, while external are set before training.

Pooling A sub-sampling technique used to reduce the dimension of the layer's input, while retaining the required features for classification (i.e. reducing a matrix produced by a previous convolutional layer to a smaller matrix).

Precision Identifies the frequency with which a network was correct when predicting the class.

Prediction A network's output when provided with an input example.

Pre-trained A network that has already been trained.

R

Rectified Linear Unit (ReLU) An activation function which maps to range $[0, \infty)$.

S

Sigmoid Function An (activation) function that transform linear inputs to non-linear outputs and is used to map the output to the range [0, 1].

Softmax A function which performs multi-class classification and transforms the input values to the range [0, 1], with a sum of 1, representing a true probability distribution.

Step A forward and backward evaluation of one batch.

Stride How much the convolutional filter moves at each turn.

T

TensorBoard Displays summaries saved during the execution of one or more TensorFlow programs.

TensorFlow A machine learning platform, or framework.

Testing The phase where the network's performance is evaluated.

Testing Data A subset of the dataset that the network has never seen before and is used to evaluate the network.

Training The process of determining the ideal parameters for a network, in which the network learns by previous knowledge.

Training Data A set of examples the network uses for learning.

U

Underfitting Refers to a model incapable of modeling the training data nor generalize to new data, resulting in poor performance on the training data.

V

Vector An array of data ordered by a single index. A matrix consisting of a single column of elements.

W

Weight Strength of the connection between two neurons, where a higher weight results in a larger influence. If a weight is 0, then its corresponding feature does not contribute to the network.

List of Figures

Table of Contents

1 Introduction

Arthur Samuel coined the term “machine learning” in 1959 [3]. Over the last few decades, machine learning has slowly, but steadily matured, and many branches has developed from it, including deep learning. The early era of machine learning dealt with expensive and inefficient hardware, a lack of large datasets, and unoptimized algorithms, all of which restrained the potential of machine learning. Over the last decade, improvements in other fields have caused a reduced barrier to entry. Today there is big data, open source libraries, capable hardware, continuous software breakthroughs, frameworks, and more. Combined, these components are contributing to push machine learning to its peak of inflated expectations, where the mainstream attention to the topic is at its highest [4].

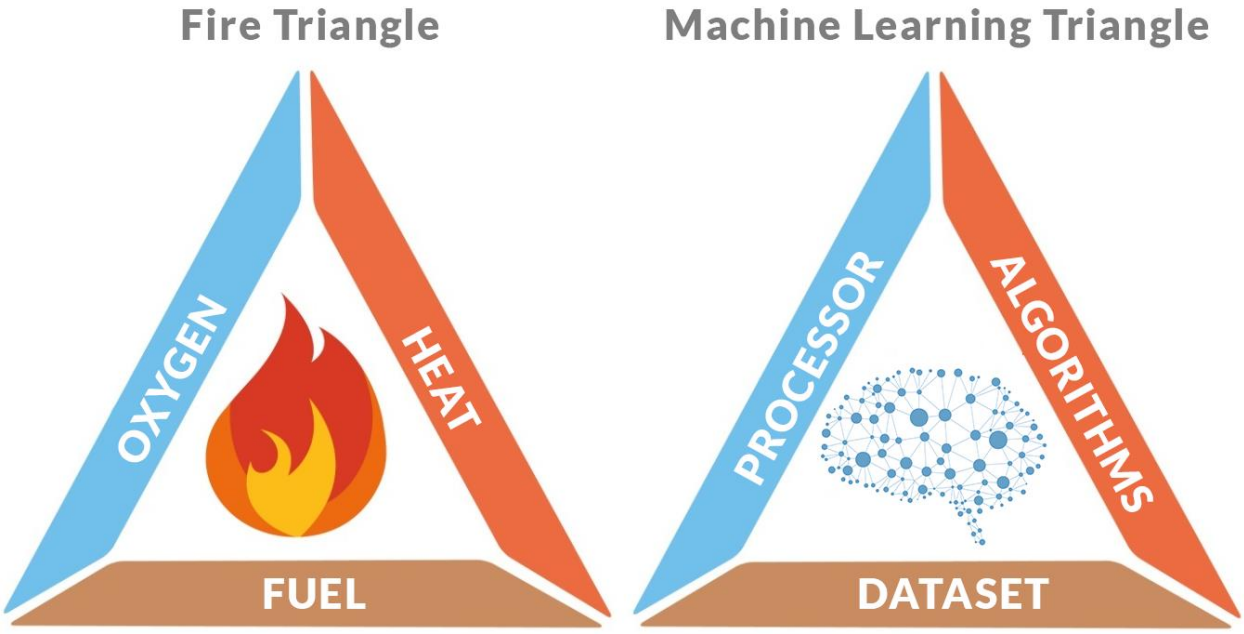

Figure 1: The ingredients for a neural network – Combined adaptations from [5, 6]

Consider the well-known fire triangle in *Fig. 1*, which illustrates the three elements a fire needs to ignite. Machine learning can be thought of in a similar way, where without an ingredient, there is no functioning neural network. The reasons as to why deep learning is now becoming popular (even though it has been around for decades) is due to (mostly) increased computational power, more efficient ways to solve problems, and increased availability of data for deep neural networks.

1.1 Motivation

Detecting fish with little to no human interference is just one of the many possibilities introduced by deep learning. Being able to find and classify any type of object paves the way for thousands of helpful applications, and there are more than enough data for it. In fact, more than 2 500 000 terabytes of data are created daily, and 90% of all existing data has been created in the last two years [7]. This abundance of data can aid solve a plethora of problems, and help humanity understand the intricacies of close to anything.

Regarding fish recognition, there are currently sensors and cameras that detect any type of movement, and while this movement could be caused by fish, it could also be caused due to any other object passing by, such as leaves or logs. This can lead to inaccurate results, either due to overestimating the number of fish, or arbitrarily removing a percentage of the registered objects to account for debris. Another issue is that fish may swim in clusters, and thus a cluster of five may only register as a single entity.

Money is an important driving factor of new technological breakthroughs. Statistics Norway, the national statistical institute of Norway, reported that in 2017, Norwegian vessels delivered 2.4 million tons of fish to a first-hand value of NOK 18.1 billion (USD 2.23 billion) [8]. For aquaculture, Statistics Norway reported a first-hand value of NOK 64 billion (USD 7.90 billion) in 2016, up from NOK 47 billion (USD 5.80 billion) in 2015 [9].

Regarding Norwegian river catches of salmon, sea trout, and migratory char, there was for 2017 reported a total of 420 258 kilograms (926 510 pounds) slaughtered fish, and 126 938 kilograms (279 850 pounds) caught and released fish [10]. The number of fish traversing the rivers are far higher, but there are no exact results for this type of information (yet). Additionally, there is a five to six step process for reporting the river catches, which is reported only once each year [10].

Other potential use-cases of deep learning is analyzing fish behavior and determining whether they are healthy. Fish behavior can change due to hypoxia, parasites, too high concentrations of dissolved oxygen levels, prey, predators, and temperature. By monitoring fish for conspicuous behavior, one can potentially reduce the number of fish deaths. For instance, **hypoxia** (i.e. a state where there is a lack of oxygen for normal life functions) is estimated to be the cause of 5.3% of all fish deaths in the United States [11].

Overall, deep neural networks have an enormous potential in underwater environments. Image classification, multiple fish detection, tracking, localization, observe conspicuous behavior, automated counting, and more, all without the need of individual, physical examination. This method of detecting and tracking can excel in smaller, controlled environments, such as in rivers and fish farms.

1.2 Goals

There are three goals for this thesis, sorted by importance:

1. Multiple objects detection and localization: Detect whether there is fish in a frame and register their position by drawing a rectangle around each fish.
2. Image classification: Tag frames in a video sequence where fish are present for more efficient surveillance of fish in rivers.
3. Use data captured from cameras to generate statistics of the amount of fish in rivers throughout the fish season.

Due to the unique dataset in this thesis only containing salmon, the first goal focuses on salmon or salmon-like species. Classifying multiple species is useful but requires a much larger dataset of each species to be operational.

Tracking is the process of following the same fish while it is in the scope of the camera. This makes the third goal of generating statistics more accurate, since fish are not counted multiple times. However, if a fish leaves the scope of the camera and at a later point returns, it will be identified as a new fish.

When a camera is recording 24 hours a day, seven days a week, it is bound to record a lot of useless footage (i.e. where no fish are present), hence removing the video sequences with no activity proves to be useful in multiple scenarios. It can be used to train the neural network further or be applied to video sequences in the past to analyze and learn from previous events.

1.3 Structure

Starting from chapter 2 Background, some words are highlighted in bold. These are key terms that can be looked up in the Glossary. Only their first instance is highlighted, but a shorter definition of the term follows its first appearance.

A brief overview of the thesis' structure:

- **Chapter 1**: A gentle introduction to what the thesis is aiming to solve, the motivation behind solving the goals, and related work.
- **Chapter 2**: Explanations of key topics in machine learning, necessary to grasp the context of upcoming chapters.
- **Chapter 3**: Discusses some present applications related to tracking and detecting fish, some of the challenges introduced by deep learning, and potential machine learning frameworks.
- **Chapter 4**: The process of gathering and labeling a dataset, setting up the environment, and creating a compatible file format for the TensorFlow framework.
- **Chapter 5**: In-depth discussion of each of the components (i.e. the methods and classifiers) making up the network.
- **Chapter 6**: The performance of the networks over time are graphed and evaluated in multiple benchmarks, such as their precision and frames per second.
- **Chapter 7**: Discussion of the thesis' results, and some insights.
- **Chapter 8**: Multiple possible extensions and improvements to the work produced in the thesis are introduced.
- **Chapter 9**: References.
- **Appendix**: Examples of calculating derivatives, forward-, and backward pass in backpropagation. Some of the networks' parameters are highlighted, and graphs of the methods are illustrated.

1.4 Related Work

For fish classification, there exists one publicly available dataset with 27 370 labels [2], and one with 3 960 labels, but requires permission [12]. The datasets related to ImageCLEF [13] are also available upon request, but permission is primarily only granted for use in the competition, and partly for research after. Thus, for now, fish datasets must be specifically gathered and labeled for use in own projects.

The dataset in [2] is, however, imbalanced, in that for every image of the least represented species, there are about 500 images of the most represented species. Also, the species in the dataset would not be applicable for use in most parts of the world since the data is gathered from the Taiwanese coral reef, with species belonging to the Indo-Pacific region.

The SeaCLEF 2017 [13] competition had a salmon identification task, but only one out of four research groups submitted their results due to the task's complexity, and their proposal achieved a 4% prevision [13]. The best official results on the SeaCLEF task overall was achieved in 2015, which achieved an 81.4% mean average precision (mAP) [14] by using a subset of the ImageCLEF dataset with 24 277 labels [13].

In 2017, two additional deep learning papers used a subset of the publicly available fish dataset with 27 370 labels, and achieved 89.0% mAP [15], and 89.95% mAP [16]. Note that the aforementioned neural networks were pre-trained on the ImageNet dataset, which contains 1.2 million images, before being trained specifically using the fish dataset mentioned above [15, 16].

By augmenting the datasets in SeaCLEF 2014 and SeaCLEF 2015, it was possible to increase their size from 20 000 to 106 956, and 29 000 to 175 200 labeled images, respectively [17]. With such a large dataset specifically tailored to a single task, a type of convolutional neural network managed to achieve 94.47% mAP, and 91.99% mAP, respectively [17].

While some of the discussed results are impressive, they are using hand-crafted features specific to the task, making them limited in generalization capability.

2 Background

Deep learning is a subset of machine learning which is a subset of artificial intelligence, as seen in *Fig. 2* which briefly explains these technologies. This background chapter discusses the bedrock of machine learning necessary to grasp the following chapters regarding deep learning.

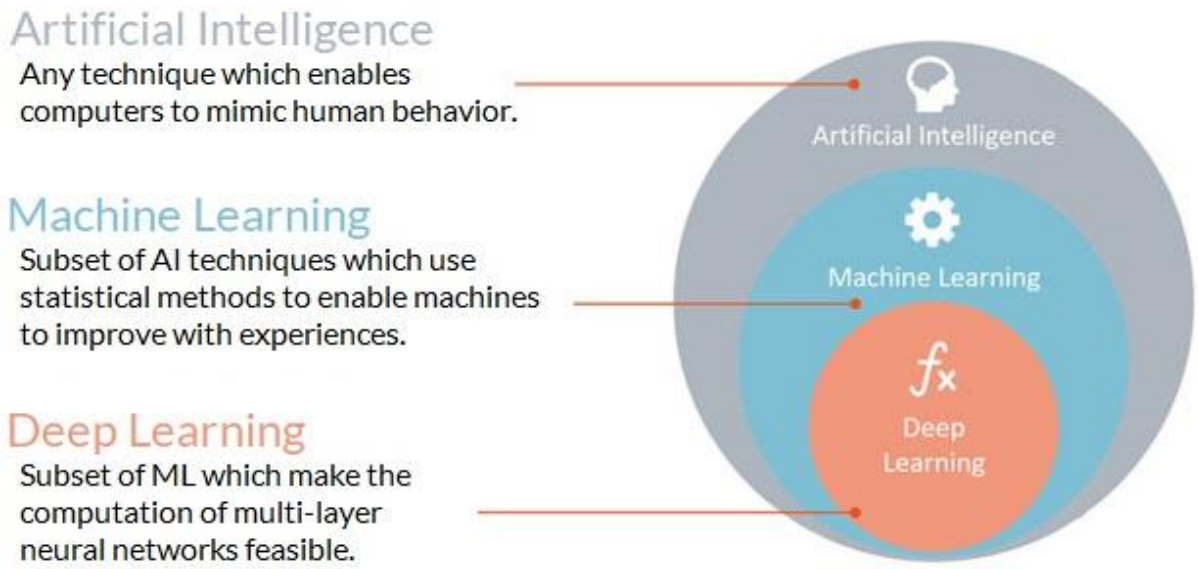

Figure 2: The relation between DL, ML, and AI – Adapted from [18]

The advantages of using deep learning compared to other techniques is that it is scalable, robust, and generalizable. It can run across multiple processors and improves with more data, it does not require predetermined features, and the same **neural network** (i.e. a series of algorithms designed to identify patterns or relationships in a dataset) can be used for other applications. On the downside, a deep neural network requires a large amount of data to operate properly (if at all), and it acts as a black box by selecting features it thinks are important without describing its decision process, making it difficult to make meaningful adjustments.

Machine learning (ML) and deep learning (DL) are similar in the way that they both are used to teach computers to be self-solving machines and are particularly useful in scenarios where humans cannot write the solution themselves, for instance in the self-driving vehicle industry. ML and DL are also dealing with the same application areas, such as image recognition and automation.

There are some radical differences, however. ML breaks down the problem into smaller pieces, solves each piece, and adds them together for a final solution, while DL solves the problem end to end. DL requires more training time and computational power due to their complex intricate neural layers. The most notable differences might be that ML handles feature extraction manually, while this is done automatically during training in DL, reducing the amount of work.

Also, ML uses precise rules to explain the decision behind choices, while these choices appear arbitrary in DL.

A strict step-by-step process works fine for problems that have a finite scope but can quickly become an entangled mess for infinite scoped problems. A finite scoped problem could be the process of registering an account on a website. An infinite scoped problem could be the process of teaching a car to drive on its own.

In a finite scoped problem, it is usually easier and quicker to explicitly tell the computer what it should do in every single scenario that might take place, as there are only a finite number of events that can occur. This, however, is not applicable for problems with an unknown or seemingly infinite number of possibilities.

2.1 Types of Learning

There are four types of learning in the machine learning field:

- Supervised: A function approximation method, where human experts feed the computer with **training data** (i.e. a set of examples the neural network uses for learning), and provide the computer with correct answers, leading the computer to eventually learn the expected patterns by itself.
- Unsupervised**:** In this case the training data is unlabeled, hence there is no human teacher. This is particularly useful when there is no structure in the data, and the expected result is unclear. This type does not output any labels for constructing model relationships, but instead attempts to detect patterns, mine for rules, or group subsets of data.
- Semi-supervised: A mixture of both supervised and unsupervised learning. Useful if the cost of **labeling** the data is high or severely restricted. Labeling refers to assigning a class to a region in the image containing an object (e.g. a fish).
- Reinforcement: An agent which continuously learns from the environment in an iterative process. There is a reward-feedback-loop for the agent to learn its behavior, and the agent is supposed to select the best option based on the current state.

Reinforcement learning received quite the media attention once AlphaGo Zero, a computer program designed to win Go games, became the strongest Go “player” in history after only three days of **training** (i.e. the process of teaching a neural network). It was designed with only the rules of Go, and only played against itself without ever watching a human play the game [19].

The **dataset** (i.e. a collection of examples containing coordinates from objects in images) used in this thesis was obtained by manually labeling fish in images, hence the appropriate choice

was to use supervised learning. This thesis has a **binary classification**, which outputs one of two mutually exclusive classes (e.g. fish, or no fish). The result of the classification is a percentage-value indicating how confident the network is that the given region of pixels is a fish. One can extend the single, salmon fish class with multiple species of fish, where each class represent a different species.

2.2 Comparing the Artificial and the Human Brain

Neural networks are inspired by the structure of the **neurons** (i.e. a mathematical function) in the human brain. A child can be told an animal's species once and will be able to name the animal's species later in any environment, with only one input sample in total. On the contrary, a deep neural network requires at least hundreds of samples in varying conditions to do the same task, where the more data it has, the more accurate it is in its predictions.

Newer research suggests that the neurons in the human brain are much more complex than just a machine with a state, as they also have an instruction set and a way to send packages of mRNA (Messenger Ribonucleic Acid) code to each other [20]. DNA (deoxyribonucleic acid) can be thought of as the blueprint, while mRNA is a template the ribosome uses to create proteins.

The sum of the signals carried by the dendrites (i.e. the segments of the neuron stimulated for the cell to activate) determine whether the neuron can fire, which sends a spike along its axon (i.e. a nerve-cell process in which impulses are carried away from the cell body). Research suggests that there are approximately 86 billion neurons in the human nervous system, which are connected with approximately 10^{14} - 10^{15} synapses (i.e. a junction between neurons, allowing for signals to be passed between one another) [21].

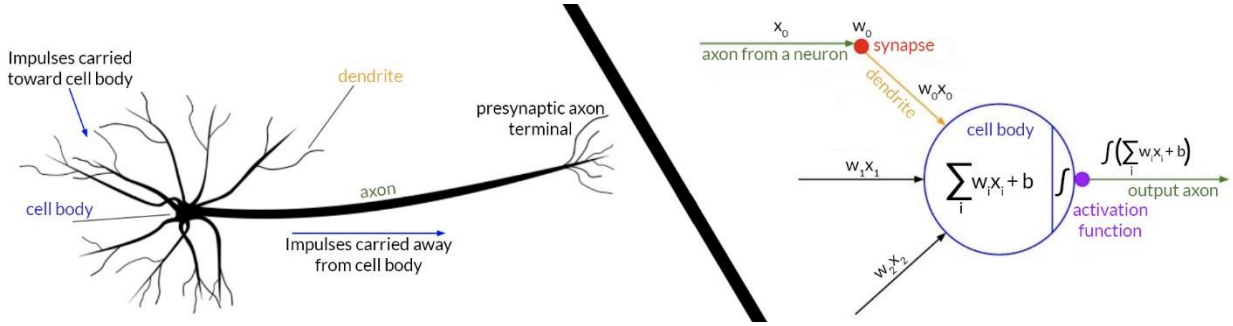

Figure 3: A biological neuron (left) and its mathematical model (right) – Adapted from [22]

The coarse cartoon drawing on the left in *Fig. 3* shows a neuron *n* (the brain's computational unit). The dendrites receive electric signals in varying strengths, which are then summed together and transferred along the axon, and eventually to the next neuron in line. The model on the right is a mathematical representation of the biological model. For reference, the mathematical model consists of weight *w*, axon *x*, bias *b*, and activation function *f*, which

decides whether the neuron's strength is large enough to fire/be active. The neuron has a connection (i.e. a link between itself and at least one additional neuron) to other neurons and learns by modifying the **weights** of the synapses, which controls how much influence each neuron has.

Oak Ridge's IBM AC922 Summit is the to-be top ranked supercomputer, and is estimated at 200 petaflops [23], while the human brain is estimated at 38 petaflops [20]. For clarity, one petaflop is one thousand trillion floating point calculations per second. The new research on the neuron's complexity might indicate that the 38 petaflops estimation is completely off, and that a closer estimate could be a ten times increase in computation, at 380 petaflops [20].

Another benchmark using the metric TEPS (Traversed Edges Per Second), has registered the highest performance by a computer to be 38 trillion TEPS [24], whereas the brain performs around 18 – 640 trillion TEPS [25]. The difference between FLOPS (Floating Operations Per Second) and TEPS is that FLOPS measures the performance, while TEPS measures the computer's ability to communicate information internally.

The brain also differs from computers in that it is more focused on reserving energy, while deep neural networks are currently closer to a brute-force method of solving problems.

While both a human and a computer can label pictures, a computer would be able to do the work non-stop without fatigue. A human expert would be more likely to accurately label the objects in the images, at least until the computer has sufficient accuracy for the labeling process. However, this can only be achieved if there are labeled pictures to begin with, hence a human is needed in the initial step nevertheless. Once the computer has reached a sufficient precision, it could save humans from spending countless hours. For instance, labeling a million pictures takes approximately 1400 hours assuming it takes five seconds to label a single picture.

2.3 The Structure of a Neural Network

Computers cannot interpret images like humans do, and to them the entire universe which they confide in consists of numbers. As such, the structural component must be designed to only deal with numbers. The goal is to have a computer look at an image and return a label of that image (i.e. what the image contains), hence the first place to start is with a pixel.

A pixel is the smallest unit of a digital image, and while a single pixel is not interesting, a region of pixels holds valuable information regarding the content of an image. For instance, a 1920x1080 image contains 2 073 600 pixels, which the computer uses as data to work with when deciding which label to set for each object in a given image.

In the scenario with the child and the computer recognizing species, the computer was unable to identify the same species in another image after only being shown a single sample. This is

due to the pixel values in the second image differing from the first. Some factors that can affect the pixel values are lightning conditions, shadows, objects obstructing parts of the object, the object's rotation, its distance to the camera, weather conditions like snow, rain, and fog, as well as the quality of the picture.

A neural network processes the pixels in an image, and outputs a value. The neural aspect asks what the values of the neurons are, while the network aspect focuses on how the neurons are connected to each other. A neuron is a node in the neural network that generates a single value by applying an **activation function** to the weighted sum of input values. There are multiple activation functions, and their purpose is to map the input value to a certain range, e.g. $[0, 1]$, to learn complicated data. The neurons reside in **layers**, which are sets of neurons that process inputs and outputs.

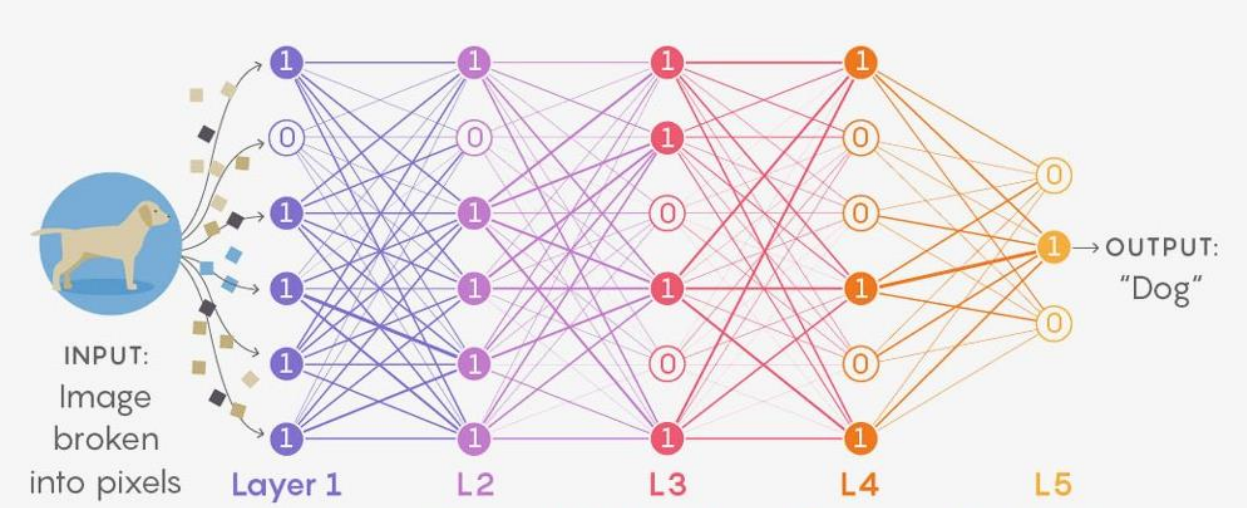

Figure 4: Overview of a deep neural network – Source [26]

Fig. 4 shows an image of a dog fed to a neural network. The initial pixel values are processed through multiple connected layers, with each layer looking for a certain pattern in the image. By combining the information gathered by the layers, the neural network can output the label for the image.

For example, consider the problem of **classifying** (i.e. the process of applying a label to an object) three animal species. One starts by collecting thousands of $56 * 56$-pixel images (or any other dimension) of the species, labels them accordingly, and feeds them to the network. In this case, the network starts with neurons corresponding to each of the $56 * 56$ pixels of the input image, resulting in 3136 neurons in total.

Fig. 4 is quite simplified, as there are only six neurons for the image (as seen in the first layer, L1), and the neurons are either marked as active (1) or inactive (0). A neuron's value could be

any real number between 0 and 1. The strength of the connection between two neurons is indicated by a weight, where a larger weight results in a larger influence.

The entirety of the 3136 neurons make up the first layer of the network. In deep learning, there can be many layers that act on the neurons, and the layers between the input layer (L1) and output layer (L5), are **hidden layers** (L2, L3, and L4 in *Fig. 4*). Hidden layers are responsible for performing computations and transferring and transforming information from the input- to the output layer.

In this example, the input layer takes many $56 * 56$-pixel images, which consists of $56 * 56 = 3136$ neurons. The neurons are sent to the hidden layers, where each hidden layer operates on the previous layer's output using various functions and operations. The final layer outputs the label for the image based on the network's confidence of either of the three classes in terms of a percentage. The final output might look like the following: [0.79, 0.04, 0.17], with a sum of 1, where 1 is 100%, for the classes [beagle, poodle, chihuahua], indicating that the image is of a beagle, with 79% precision.

2.3.1 Activation Functions

A network also carries a varying number of **parameters**, an internal configurable variable, which is estimated throughout the training process. The exact number of parameters depends on the network's size and requirements, but there are often millions of parameters. The parameters must be iteratively and gradually adjusted to be able to identify various patterns in an image.

Each connection between each layer is assigned a weight, indicating its importance, and this weight is a real number. When computing the weighted sum of the neurons in the previous layer, its real number might be exceedingly large or small. For a probability task, it might be useful to force the weighted sum in the range $[0, 1]$, instead of the existing range $(-\infty, \infty)$.

The **activations** (i.e. the strength of a neuron given by a function), in one layer determine the activations in the next layer. Neurons in earlier layers affect neurons in following layers, since a neuron's value influence other neurons' values. Biologically speaking, when a certain group of neurons in the human brain is firing, it causes other neurons to fire as well.

Some common activation functions (and their range) are as follows, and are illustrated in *Fig. 5*:

- Linear/Identity $(-\infty, \infty)$: Does not confide the output of the functions between any range, thus does not help with the complexity of the data.
- Sigmoid $[0, 1]$: Maps the input into probability spaces, making it easier to predict an event occurring. Suffers from being easily saturated due to its small range.

- TanH (Hyperbolic Tangent) $[-1, 1]$: Negative inputs are mapped strongly negative, while positive inputs are mapped closer to zero. As opposed to sigmoid, it has stronger gradients, which helps in avoiding bias.
- ReLU (Rectified Linear Unit) $[0, \infty)$: Negative inputs are mapped to zero, while inputs larger than zero, are mapped to their own value.

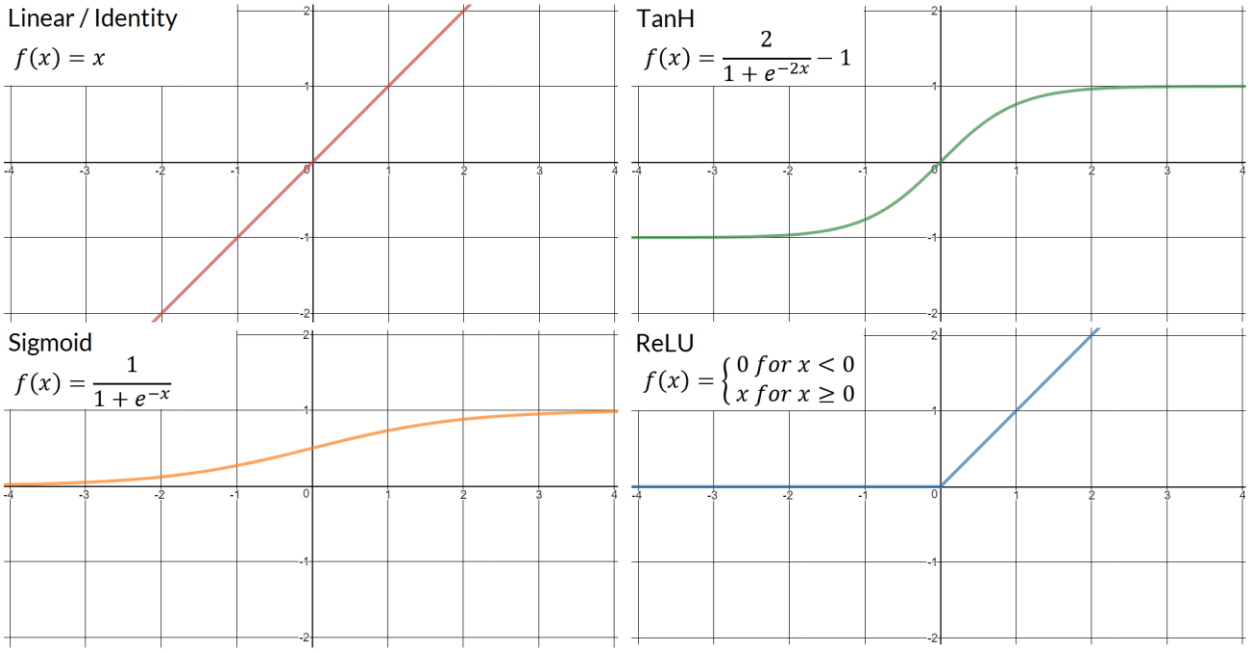

Figure 5: Common activation functions, each remapping the input values to a certain range

The essence of these functions is that they handle large numbers (either positive or negative) by remapping them according to their own ruleset. This means that the activation of the neuron is a measure of how positive the relevant weighted sum is.

Bias is used to better fit the data by shifting the curve of the activation function, resulting in a more varied calculation. A bias close to zero will have little to no impact, leading the output to be decided by the values of weights and inputs alone. The weights alter the steepness of the curve.

2.3.2 Representing the Parameters

Every neuron in the second layer is connected to every neuron from the input layer. In the case of a $56 * 56$-pixel image, there are 3136 neurons, and these have their own individual weight and bias associated with it.

To calculate the number of parameters for a neural network with 3136 inputs, two hidden layers (each with 32 neurons and 32 biases, one for each neuron), and three outputs is done as follows:

$$3136^{L1} * 32^{L2} + 32^{L2} * 32^{L3} + 32^{L3} * 3^{L4} + 32^{L2} + 32^{L3} + 3^{L4}$$

$$= 101\,539.$$

To clarify, the numbers in green are the neurons, and the numbers in blue are the biases. Their superscript $^{L1 \ldots L5}$ refers to the layer the neurons or biases reside in. Layer 1 (L1) is connected to L2, thus the neurons are multiplied with one another to get the total parameters for that connection. Each neuron in L2 also have their own bias, which is added to the total sum.

The preceding network has more than 100 000 weights and biases one can adjust to make the network behave in meaningful ways. The learning process of a neural network is essentially the process of finding the most optimal setting for all the parameters, so the computer can solve the problem at hand as precise as possible.

One can represent the connections using the following notation, where the superscript $^{(1)}$ or ^{L} indicates the layer and is not to be confused with exponents. The subscript $_0$ or $_k$ indicates the position inside the layer, and the symbol σ represents the sigmoid activation function:

$$a_0^{(1)} = \sigma\left(w_{0,0}a_0^{(0)} + w_{0,1}a_1^{(0)} + \cdots + w_{0,n}a_n^{(0)} + b_0\right).$$

This calculation can be rather cumbersome to write, thus a more compact solution is to represent all the activations in one layer as a **vector**, and all the weights as a **matrix**, where each row in the matrix corresponds to the connections between one layer and a particular neuron in the next layer, as written below [27]:

$$\sigma\left(\begin{bmatrix} w_{0,0} & w_{0,1} & \cdots & w_{0,n} \\ w_{1,0} & w_{1,1} & \cdots & w_{1,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{k,0} & w_{k,1} & \cdots & w_{k,n} \end{bmatrix}\begin{bmatrix} a_0^{(0)} \\ a_1^{(0)} \\ \vdots \\ a_n^{(0)} \end{bmatrix} + \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_n \end{bmatrix}\right) = \begin{bmatrix} ? \\ ? \\ \vdots \\ ? \end{bmatrix}, \text{ resulting in: } a^{(1)} = \sigma\left(Wa^{(0)} + b\right).$$

A neural network consists of simple building blocks, but due to the sheer number of neurons required to have a working network, one ends up with networks consisting of millions of parameters, where each parameter's value must be set to a specific value based on all the other values to have the potential of having an accurate network.

2.3.3 Selecting the Number of Neurons and Hidden Layers

There are theorems from decades ago stating that a single hidden layer is capable of universal approximations, i.e. neural networks with one hidden layer can compute any function [28]. One can combine multiple activation functions for approximation [29], which can be achieved by constructing step functions from ReLU. If a single layer is sufficient, why bother implementing multiple hidden layers?

A multi-layered network is more capable of recognizing certain aspects of input data. One could use a single layer, but it would require more neurons as it cannot rely on previous neurons to perform a sub-task of the larger task. Multiple combined activation functions make it very easy to **overfit**, and the network's ability to generalize is hamstrung. Overfitting is when a network matches the data points too closely to their exact value, causing it to fail at making correct predictions on new data.

Deeper neural networks can (due to their depth) gather more complex information from the input data, as they can use certain types of **algorithms** for each layer, slowly finding new patterns, and using the data to improve itself. An algorithm is essentially a finite set of defined instructions for reaching a state or solving a problem.

A deeper network requires more time for training, but can be far more effective, and can solve problems in which a single layered neural network cannot. That said, adding layers for the sake of having a very deep neural network is counterintuitive, as there are diminishing and potentially negative returns. As a matter of fact, a 32-layer ResNet outperformed a 1202-layer ResNet, with error of 7.51% to 7.93%, with 0.46 million parameters to 19.4 million, respectively [30]. It all comes down to a matter of design, the task at hand, and the amount of data.

One challenge with deep learning was for a long time the process of training the neural networks. Geoffrey Hinton et. al. built upon the original concept of the **backpropagation** algorithm in a paper published in 1986 [31], with the idea of repeatedly adjusting the weights of the connections in the network to minimize the difference between the output vector and the desired output vector. Backpropagation is a popular algorithm for training multi-layer neural networks.

Selecting the number of hidden layers and the neurons for each layer come down to trial and error. Choose a number of hidden layers and a number of neurons for each layer, calculate the performance, and repeat until satisfactory results are achieved.

Many of the parameters in a network are redundant, and their existence do not contribute to a better result. Pruning is an iterative process for increasing a network's speed by decreasing its size, by ranking neuron's based on their contribution, and removing redundant parameters.

On the Street View House Numbers (SVHN) dataset which consists of more than 600 000-digit images there was used a deep convolutional neural network with 1 236 250 parameters. When NoiseOut with binomial noise [32] (a pruning method) was used, the number of parameters was reduced from 1 236 250 to 194 005 (a decrease of 84.31%) while maintaining the accuracy (93.39%) [32].

The downside of pruning is that it can be challenging to implement and has a potential of reducing the overall accuracy.

2.4 How a Neural Network Learns

A neural network processes thousands of pixels and return a label based on those pixels. Learning is the process of gradually adjusting the values of the parameters (weights and biases) in the network to overall increase the accuracy of the network.

For a supervised problem, the dataset consists of manually labeled images. The process of labeling is a straightforward, but also a very time-consuming activity. This thesis labels the objects using a **minimum bounding box**, which is represented by four box coordinates (*x*, *y*, *w*, and *h*). The pixels within the bounding box is what the network will use for its training and testing. The dataset consists of the images and their corresponding minimum bounding box coordinates.

It is important to split the dataset, since it must be evaluated throughout the training process to find the optimal amount of training. Most of the data (usually 80% - 90%) is used for training, and the remainder for testing, but it depends on the size of the dataset.

When training, the network uses its existing knowledge to determine where it believes the object might be located, and it can only look at the answer once it has made a prediction. Then, it compares its own answer with the actual answer (the **ground truth**), and calculates the **loss**, which is how close its estimate was. Now, it has some additional knowledge of what it is looking for and is more likely to get it right the next time around.

Testing is similar to training, but to find the network's true performance, it must be tested on data it has never seen before, otherwise it would be memorizing, and not learning. When testing, the **testing data** (i.e. data the network has never seen before) is used, and even after the network has made its prediction, it is not permitted to look at the actual answer. This makes it possible to reuse the testing data for testing purposes, otherwise one would constantly have to collect more testing data whenever one would like to find the network's true performance.

Fig. 6 illustrates the various steps a neural network uses to learn and improve itself. These steps will be explained in the next section regarding backpropagation, but it may be useful to have an early glance at it and refer to it for clarification.

The summary of *Fig. 6* is that the weights in the network are at first randomized, and the pixel values from an image are forwarded throughout the network, which in turn produces an output (a prediction). The error of the prediction is calculated (how close the prediction was to the ground truth), and with it, the weights are updated layer by layer starting from the output layer. The process is then repeated until satisfactory results are achieved.

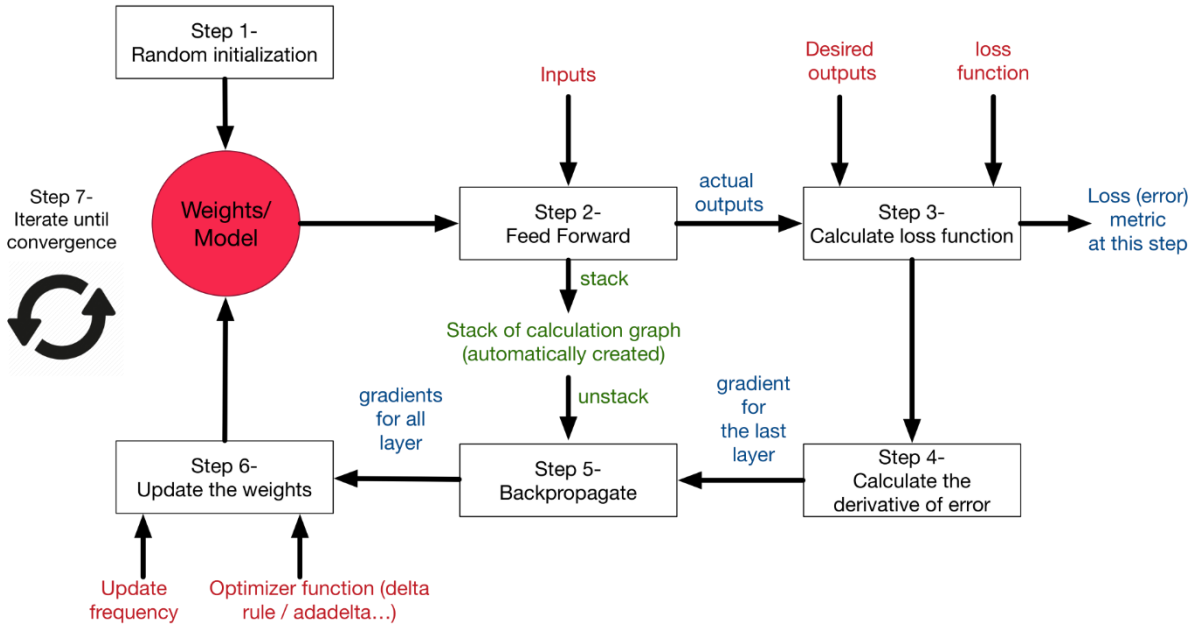

Figure 6: The learning process for neural networks – Source [33]

2.4.1 Minimizing Cost

A neural network does not always produce clear, one-sided results, in fact, those results are rare. The way to correct this, and tell the network it was wrong, is to use a **cost** (i.e. the difference between the desired output and the actual output). For the example below, the cost is calculated by adding up the squares of the differences between each of the output activations and the value that the actual result should have.

In the animal species example, there were three output neurons, each storing the network's confidence of predicting a species. Their cost could for two images be the following:

$$1.2515 \begin{cases} 0.6561 \leftarrow (0.81 - 0.00)^2 + \\ 0.5929 \leftarrow (0.77 - 0.00)^2 + \\ 0.0025 \leftarrow (0.95 - 1.00)^2 \end{cases} \quad \Big| \quad 0.0166 \begin{cases} 0.0036 \leftarrow (0.06 - 0.00)^2 + \\ 0.0009 \leftarrow (0.97 - 1.00)^2 + \\ 0.0121 \leftarrow (0.11 - 0.00)^2 \end{cases}$$

The left-hand equation has a higher cost, thus indicating that the network is not confident in its decision (but still made the right prediction), while for the right-hand equation the network classified the image correctly with high confidence. The equation is clarified below. One can take the average cost of all training data to get an indication as to how well the network performs, where a lower value is better.

For the notation of the cost $C_0(\dots) = \left(a^{(L)} - y\right)^2$ as written above, *y* is the desired output, which is "1.00" for only a single neuron, and "0.00" for the rest. Additionally,

$$a^{(L)} = \sigma\left(w^{(L)} a^{(L-1)} + b^{(L)}\right),$$

means that the action on layer *L* equals the sigmoid function (although it could be any other activation function) on the current layer's weight multiplied by the previous layer's activation, added with the current layer's bias [34].

Put in a slightly different way, the cost function uses the weights and biases as input, the parameters are the training examples, and the output is a single number, the cost. At this point the only thing the network knows is whether it is doing a decent job based on the cost its returning. This is only the first part of how the network learns, since it does not know how to improve based on this result. One must also tell it how it can change the weights and biases according to the cost to improve itself over time.

Instead of thinking of the cost function as a function with millions of inputs, think of it as a function with a single number as an input, and a single number as an output. The goal is to find an input that minimizes the value of the cost function $C(w)$, as seen in *Fig. 7*. There can be many **local minimums**, but only one **global minimum**.

In either case, one can start at any input, and figure out in which direction one should move to make the value lower. In other words, find the slope of the current function, and shift to the left if that slope is positive, and to the right if the slope is negative. Doing this repeatedly will eventually lead to a local minimum of the function. Ideally one would like to always find the global minimum, but this is a very challenging feat.

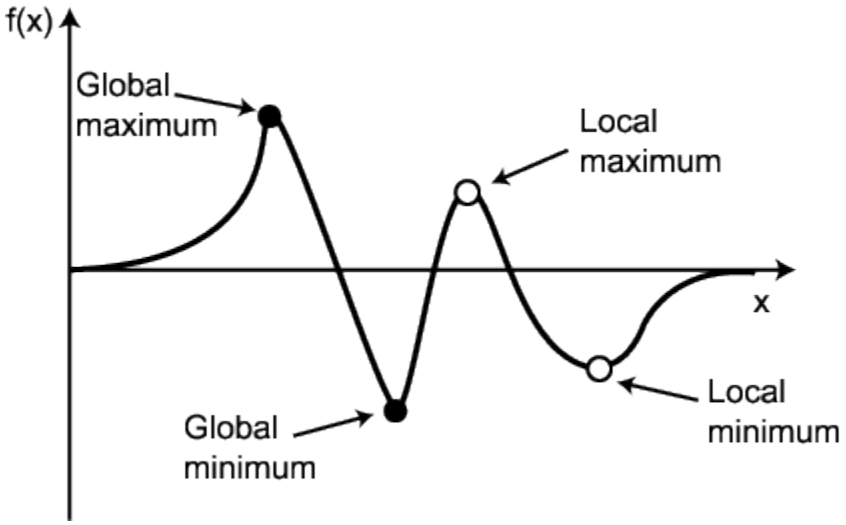

Figure 7: Gradient descent – Source [35]

For instance, if a local minimum was found, how can one then determine in which direction the global minimum is, and even so, if the global minimum was found, how would one know without trying all the local minimums first? It would be too computationally expensive, which is why the focus is to find any local minimum instead.

If the cost function had two inputs, one can look at the input space as the xy-plane and graph the cost function as a surface above it. The complexity is now increased, since there is another dimension to account for, so instead of asking for the slope of the function, one now look at in which direction $C(x, y)$ decreases most quickly. The gradient gives the direction of the steepest increase, and since the aim is the opposite, one can take the negative of the gradient to find the more optimal direction. Additionally, the length of the gradient vector indicates how steep the steepest slope is.

When a network is learning it is essentially just minimizing a cost function. Therefore, the goal is to find the negative gradient of the cost function, which in turn says how one should change the weights and biases for all connections to decrease the cost most efficiently.

Naturally, a million-dimensional input-space is difficult to wrap one's head around, but an easier way to look at it is to structure the weights and the cost of those weights in two separate vectors. Then, every cost indicates its importance, where the importance is higher the farther it is from 0. It may be more efficient to tweak a single value a lot (e.g. a weight of -14), than tweaking 10 values a tiny amount (e.g. each with a weight in the range $[-0.5, 1.5]$), but there is no definitive answer as to which of these two priorities will lead to the best result.

2.5 Backpropagation

A very loose analogy for backpropagation is the journey from being a child to becoming an adult, where all previous events and experiences affect one's decision-making for future actions. Of course, a neural network might learn too much if it is being fed the same training data repeatedly, which makes it perform worse. This is luckily not the case for humans; imagine the concept of decreasing one's reading proficiency by reading the same book over and over.

Backpropagation is a local search algorithm for computing the error contribution of each neuron once a set of examples is processed. The **gradient descent** algorithm often uses backpropagation to adjust the weights of neurons by calculating the gradient of the loss function. The gradient descent algorithm's purpose is to minimize the produced cost and is an iterative approach which takes small steps to reach the local minima of a function.

Backpropagation is also referred to as backward propagation of errors, as the error is calculated at the output and sent back through the layers.

There are three ways to alter a neuron's activation level: by adjusting the bias, the weights, or by changing the activations from previous layers. The preceding layer is a major factor when it comes to the current layer's neurons, as the connections with the brightest neurons have the most effect, due to their weights being multiplied by larger activation values.

2.5.1 Chain Rule

Forward propagation in a network can be seen as a chain of nested equations, thus one might argue that backpropagation is an application to the **chain rule** for calculating the **derivatives** of composite functions, i.e. how functions change over time. A derivative output an expression which can be used to calculate the slope at a single point on a line, and once the derivative's solved, calculate for every point on the line.

When applying the chain rule to higher-order functions like:

$$f(x) = A\Big(B\big(C(x)\big)\Big),$$

its derivative equals:

$$\frac{df}{dx} = \frac{d}{dx}A\Big(B\big(C(x)\big)\Big) = \frac{dA}{dB}\frac{dB}{dC}\frac{dC}{dX}.$$

Please refer to *Appendix A* (A.1 Calculating Derivatives) for a concrete example of how one might derivate according to the chain rule.

Derivatives are great for optimizing the network's parameters, because they can be used to figure out whether weights should increase or decrease to maximize the network's accuracy.

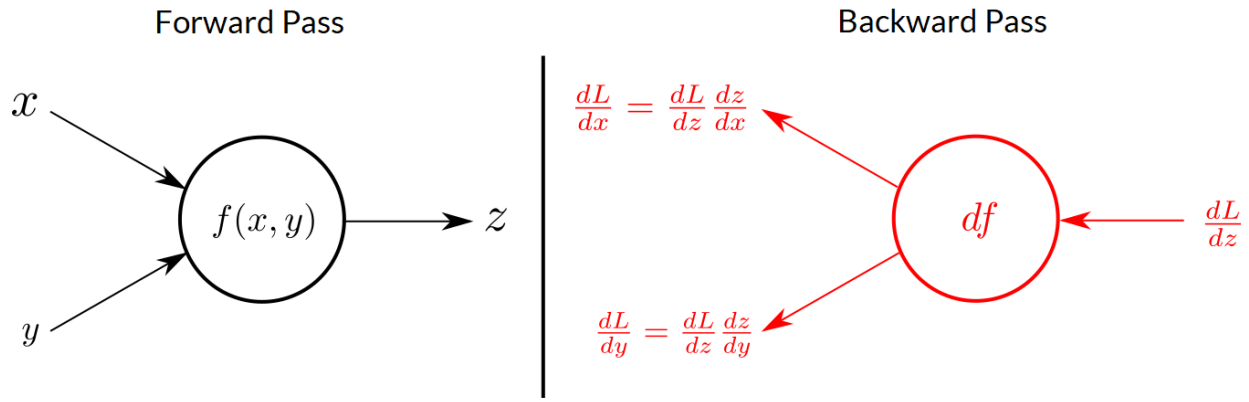

Figure 8: Forward pass and backward pass on a neuron – Adapted from [36]

The neuron $f(x, y)$ in *Fig. 8* computes a function of two inputs (*x*, *y*), and outputs a real number *z*. When doing this process backwards, the input is the gradient propagated from a deeper layer $\frac{dL}{dz}$, which says how much *L* (the loss) will change with a small change on *z*. Each layer takes the derivates of the loss *L* with respect to *z*, computes the derivatives of the loss with respect to *x* and *y*, and outputs results to the previous layer, calculating the gradient of *x*, and *y*.

Instead of dealing with abstract examples and technical notations, it might be easier to follow the process in a step-by-step example, provided in the next two sections.

2.5.2 Forward Pass

The forward pass calculates the values of the output layer based on the input values, and is a layer-by-layer approach, starting at the input layer. The produced output is what the network thinks is the label for the image.

Fig. 9 illustrates a forward pass with two hidden layers (*H1*, *H2*), a set of weights (W_{ixjy}, W_{jxky}, W_{kxly}), and operations applied to the layers (ReLU, sigmoid, softmax), where each layer has two neurons (*i*, *j*, *k*, *l*).

Please refer to *Appendix A* (A.2 Example of Forward Pass) for a detailed step-by-step example on how the values in a forward pass is calculated, based on *Figure 9*.

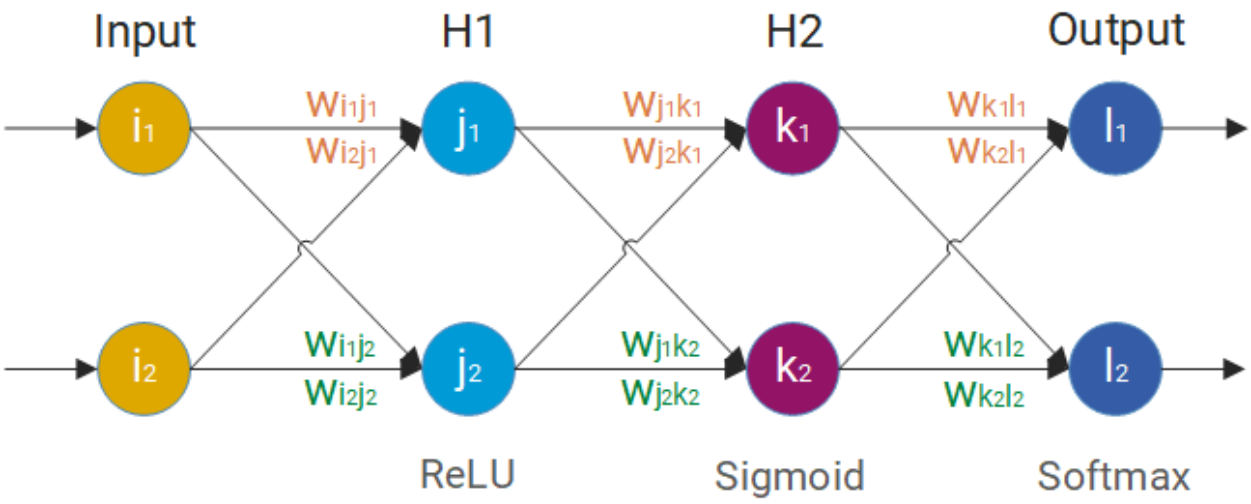

Figure 9: Notation in a neural network with two hidden layers

2.5.3 Backward Pass

The backward pass starts at the output layer and propagates toward the input layer. This is the stage where the network learns, by using the calculated cost (difference between the output and the actual output) to gradually update the network's weights.

The idea of propagating backwards is just adding together all the desired effects from the neurons, which gives a list of adjustments that should happen to the second to last layer. Once those are known, one can recursively apply the same process to the relevant weights and biases that determine those values, and continuously move backwards through the network.

Please refer to *Appendix A* (A.3 Example of Backward Bass) for a complete step-by-step on how the values from the output to the second hidden layer in a backward pass is calculated.

Adding up the influence of every single training example for every single gradient descent step uses excessive resources, hence a better method is to randomly shuffle the training data and divide it into **mini-batches**. Then, compute a step according to the mini-batch. A **batch** is the set of examples used in one iteration (i.e. one gradient update) of network training. A mini-batch is a randomly selected subset of the batch of examples.

Both are viable solutions, and each approach can be loosely tied to the following analogy. Imagine two people on top of a hill, competing to reach the bottom in the least amount of time. One person plots the entire route before descending, which leads to the shortest distance travelled. The other person starts walking immediately, while continuously adjusting the route along the way. The first option requires more computational power, while the second may have larger fluctuations for the accuracy. The result is, however, mostly the same.

Planning as you go in this case is known as stochastic gradient descent, meaning that the neural network act at a single training example for each calculation, instead of a set of examples.

2.6 Alternatives to Backpropagation

While gradient descent works well in general, it uses a lot of time to converge to the local minima of the function. Also, as it approaches the minimum, it becomes slower, as the steps are not as large. Lastly, it is ill-defined for non-differentiable functions (e.g. functions with jump discontinuity, cusps, and distinct corners) [37].

2.6.1 Decoupled Neural Interfaces using Synthetic Gradients

A constraint of backpropagation is that the weight updates require a forward propagation followed by a backward propagation. This results in the layers being locked, since a single layer must wait for the remainder of the network to execute their actions before it can be updated.

Decoupled Neural Interfaces (DNI) using Synthetic Gradients was introduced in 2016, and proposes a method where layers can communicate in a decoupled manner [38], solving the above issue.

Table 1 compares backpropagation (BP), decoupled neural interfaces (DNI), and DNI where the synthetic gradient model is also conditioned on the labels of the data (cDNI), for fully convolutional networks (FCN) and convolutional neural networks (CNN).

Layers		No BP	BP	DNI	cDNI
FCN	3	54.9	43.5	**42.5**	48.5
	4	57.2	**43.0**	45.0	45.1
	5	59.6	**41.7**	46.9	43.5
	6	61.9	**42.0**	49.7	46.8
CNN	3	28.7	**17.9**	19.5	19.0
	4	38.1	**15.7**	19.5	16.4

Table 1: CIFAR-10 (% Error) results for no BP, BP, DNI, and cDNI – Source [38]

While backpropagation has the best results overall, *Table 1* shows that DNI and cDNI can successfully update-decoupling all layers for a slight cost in accuracy. This is the first time that neural network modules have been decoupled and the update locking has been broken [38].

Fig. 10 illustrates the concept of the method. It learns a parametric model which predicts the error gradients' values based only on local information, and the predicted gradients are referred to as synthetic gradients. Layer 1 can thus be updated before the rest of the network is executed. The black lines in *Fig. 10* are the output activations, the green lines are the error gradients, and the blue lines are the synthetic gradients.

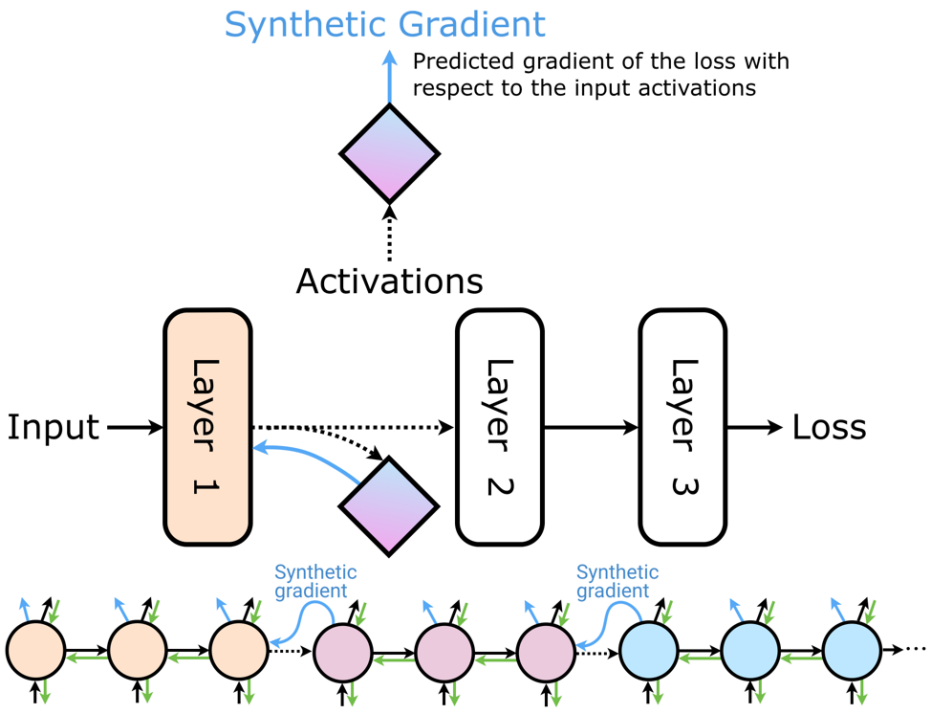

Figure 10: Illustration of Deep Neural Interfaces using Synthetic Gradient – Adapted from [39]

2.6.2 Direct Feedback Alignment

Feedback Alignment (FA) replaces backpropagation's transpose of weight matrices for updating their own weights, with fixed-random matrices. *Fig. 11* illustrates the concept of FA and Direct Feedback Alignment (DFA) compared to backpropagation (BP), with weight matrix W_i, fixed random matrix B_i, and transpose of the matrix *T*.

DFA makes it possible to use the gradient from the last layer to train all layers, instead of depending on the gradient from previous layers, thus eliminating the layer-by-layer progress.

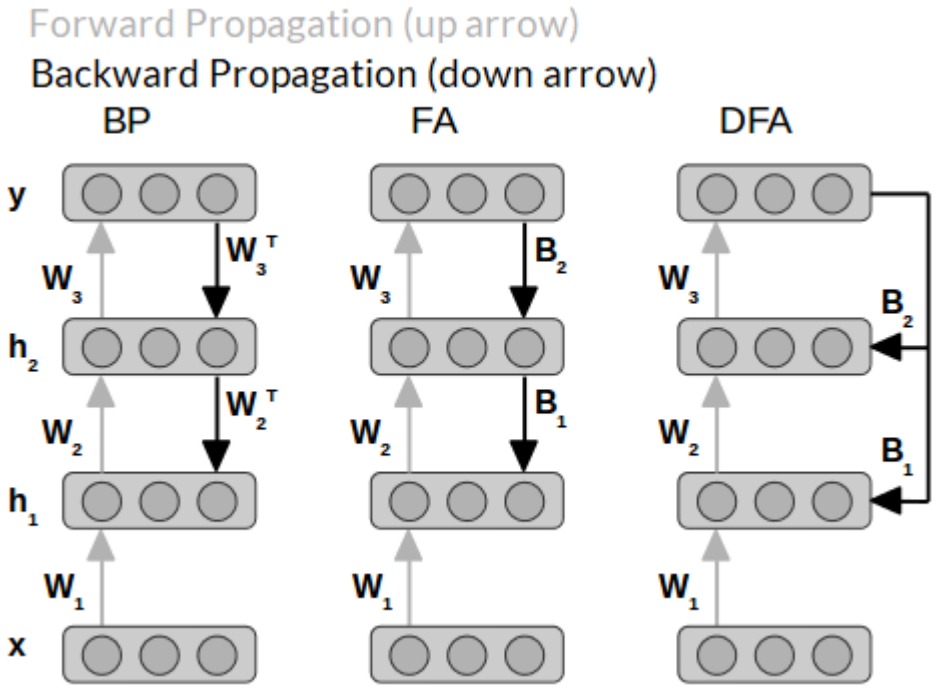

Figure 11: Backpropagation, and (Direct) Feedback Alignment – Adapted from [40]

The results from DFA in *Table 2* are in some cases better than FA on the CIFAR-10 dataset, but backpropagation outperforms both methods. DO is dropout regularization [40].

Network	BP	FA	DFA
1x1000 Tanh	**45.1** ± 0.7% (2.5%)	46.4 ± 0.4% (3.2%)	46.4 ± 0.4% (3.2%)
3x1000 Tanh	**45.1** ± 0.3% (0.2%)	47.0 ± 2.2% (0.3%)	47.4 ± 0.8% (2.3%)
3x1000 Tanh + DO	**42.2** ± 0.2% (36.7%)	46.9 ± 0.3% (48.9%)	42.9 ± 0.2% (37.6%)
CONV Tanh	**22.5** ± 0.4% (< 0.1%)	27.1 ± 0.8% (0.9%)	26.9 ± 0.5% (0.2%)

Table 2: CIFAR-10 test error for BP, FA, and DFA, with training errors in brackets – Source [40]

While there are multiple alternatives to backpropagation (in addition to the two discussed above), backpropagation is in general the better option, hence why it is still widely used.

3 Problem Analysis

3.1 Initial and Revised Objective

Initially the main objective was to count the number of fish in fish farms, hence the dataset was tailored for this specific use case. However, the third-party later stated that they already have accurate tools for estimating the number of fish in their fish farms, which involved counting the number of fish entering the farms, and the number of dead fish removed from the farms. Thus, the thesis' objective changed approximately halfway throughout the project due to the updated clarification.

After pivoting and reaching out to other potential interests, it came to light that the fish recognition technology could be more beneficial in rivers, with additional use cases for gathering statistics and discarding video sequences with no fish present.

3.2 Present Applications

There exist several methods to recognize fish. Some criteria for evaluating the methodologies are accuracy, price, ease of use, installation process, and maintainability.

3.2.1 SONAR

SONAR (sound navigation and ranging) is the term most commonly referred to for acoustic detection devices, both passive and active, and has been used to study fish populations since the 1930s [41].

When active sonars emit a pulse of sound, the sound wave spreads in the shape of a cone and bounces off objects, where the waves reflecting to the device provides information like shape, distance, and composition. Passive sonars do not emit any sound, but merely listen, and unlike active sonars, cannot measure the distance of an object on its own.

Some sonars are used in conjunction with a database, where each reflected signal can be used to query the database to identify the type of fish (i.e. if it already exists in the database), as well as their location and size, since many fish transmit species-specific sounds [41].

Methodologies such as net sounder, depth sounder, CHIRP sonar, fishfinder, and scanning/imaging sonars are all using sonar to locate submerged objects. Some differences between each are the frequencies they operate on, type of sweep, how information is displayed, effectiveness in various terrains, categorization techniques, and price.

3.2.2 Tags

Passive Integrated Transponder (PIT) tags are used for tracking individual animals throughout their life-span, and has been in use since the mid-1980s [42]. A PIT tag consists of an integrated circuit chip, capacitor, and antenna coil encased in glass. The tags are either inserted via large-gauge needles, implanted underneath the skin, or into a body cavity, depending on the size of the animal. The PIT tags are activated by a low-frequency radio signal from a scanning device, which causes the tag to reflect a unique code to the reader [42].

Acoustic tags transmit an object's identification information by acoustic pulses to multiple hydrophones (i.e. a device designed to process acoustic waves and convert them to electrical signals). Since multiple hydrophones are used to process the signals, one can calculate the real-time 3D position of multiple fish simultaneously with a sub-meter accuracy, by using the sound's time of arrival [43].

3.2.3 Appearance-Based Feature Extraction Methods

Another approach is to use an automated video processing network to identify pre-determined features. Many networks use a handful of algorithms for detecting and tracking objects. The CamShift (Continuously Adaptive Mean Shift) algorithm is a popular choice, but it has difficulties with tracking multi-hued objects, and can fail when the appearance of objects changes (e.g. due to lightning or camera movement). Several algorithms extend upon and improve CamShift in very specific ways, and while their approach may make the algorithm more robust, they may also severely impact the speed [44].

Histograms are also a core aspect of some object detection algorithms, by detecting the foreground by using background subtraction, while also incorporating edge detection. An appearance-based tracking method was used to track a school of fish simultaneously, first by creating a template of the to-be-tracked fish and then combine it with parameter estimation to find its next position [45]. The estimated positions compared to the ground truth positions were less than 4% of the body length of the school of fish [45].

3.2.4 Neural Networks

As opposed to the more conventional methods mentioned above, deep neural networks require a large amount of data, preferably tens of thousands of (in this case) images and gathering said data can be a challenging process. Large datasets can suffer from **noise**, which can drastically degrade learning performance. Noise is an unwanted distortion in data and can be caused by for instance a labeling misclassification or poor conditions (low quality sensor for image noise, or spelling mistakes for textual noise).

Nevertheless, some major accomplishments have been achieved by using neural networks for detecting fish, as seen in [15, 16, 17]. Comparing the various implementations is not a straightforward process, due to hardware differences when comparing speed, and how the datasets are structured when comparing accuracy.

3.3 Viable Options for Recognizing Fish

SONAR, appearance-based feature extractor, and neural networks are all viable solutions. Tags are not practical for this use case, as they specialize in tracking individual fish for longer periods at a time.

AF methods use templates of the object(s) to perform recognition, and since an object's appearance is different under varying conditions, one would require multiple templates to cover as many representations of the object as possible. Neural networks use large amounts of data to train itself to recognize the object(s).

Ultimately, deep learning was chosen for some specific reasons. First, the dataset had already been acquired, thus one of the largest obstacles had already been overcome. Second, a neural network's accuracy can improve when additional data is provided, which can be obtained at a later point. Third, the network's speed will increase when more capable graphics processing units (GPU) are released. Fourth, with the rapid breakthroughs seen in deep learning, one can apply the dataset to a more efficient network in the future. Fifth, companies are investing significant resources into machine learning, thus, from a purely educational purpose, getting a better understanding of ML might be a wiser investment.

3.4 Neural Network Challenges

It can be difficult to examine the decision mechanism for deep neural networks. For linear networks (i.e. a network which is specified as a continuous combination of features) one can follow the flow from start to finish and look at how each individual layer relates with preceding and succeeding layers.

In decision trees one naturally starts at the root of the tree and navigate through the nodes, where more key features are often closer to the root node. Deep neural networks on the other hand are non-linear, and have proven to work with more than 1200 layers [46] and up to 137 billion parameters [47].

When the size of the network is too large, it can become extremely difficult to make any reasonable predictions about how certain changes are going to affect the results. One solution is perhaps to reverse-engineer the entire process, by ultimately turning the network upside down and attempt to generate the wanted result from random noise with iterative adjustments

[48]. Understanding why the network behaves as it does become increasingly important, especially for critical applications, e.g. in the medical or self-driving car industry.

The amount and the quality of the data provided to the neural network is another important problem to tackle. There is no exact number for how many images a deep neural network requires to make accurate predictions for a certain task. A general rule-of-thumb, however, is to have at least a thousand unique images of each class [49], but more is better.

A large variety in images is preferred, where the angle, distance, type, and often color of the object is different. If every image in a dataset has distance *x*, and angle *y* of the object, then the neural network's understanding of the object is limited to those representations, and thus any other angle or distance than those it trained on would be unrecognizable to it.

There is also a risk in the sense that humans can create tools to mathematically manipulate images to fool a network, where the changes are undetectable by any human. This can have severe consequences, for instance in the self-driving car industry. This manipulation can be done by overlaying the original image with a pre-defined set of pixels to alter the pixel values [50].

3.4.1 Parameter and Hyperparameters

Parameter and hyperparameter are often used interchangeably, but they are slightly different. A parameter is a network's internal configuration variable and can be estimated throughout the training process. A hyperparameter is external to the network, i.e. it is defined before the training process starts. Please note that in this thesis, the term parameter will be used to cover both terms, and specific examples of parameters used in the thesis can be found in Appendix B Experiments.

Gradient descent is an optimization algorithm used to minimize the error rate by finding the optimal weights, while the **learning rate** is the variable deciding the size of the steps to find the (local) minimum. The variable usually starts with a small value, such as 0.0003, and in some cases decrease over time, logic being to first find suitable parameters quickly, and later fine-tune them. When the learning rate is too large one might risk overshooting a minimum value (the network diverges), and if it is too small, it may take too long to find a minimum value (i.e. the network converges too slowly).

The **loss function** compares the output of the network in the training phase with the ground truth, which is the actual label for the object, and is an objective measurement based on factual data. The loss function is useful for determining the performance of the network by calculating the difference between the expected output, and the retrieved output.

The mini-batch, batch, **batch size**, **iteration**, and **epoch** are all closely related, and controls the amount of data that passes through the neural network. A mini-batch is a randomly selected subset of the entire batch of examples. A batch size describes the number of examples in a batch, which is the set of examples used in one iteration (i.e. one gradient update). An iteration or a step is the number of times a batch of data has been sent forward and backward through the network. An epoch is achieved when the neural network has processed all training examples in the training set.

This can be illustrated as follows: the batch is all training examples and has a certain batch size (e.g. 100 000). The batch can be divided into mini-batches of a smaller batch size (e.g. 1000), which are passed through the network, where each pass equals one iteration. 100 iterations are thus needed to complete one epoch.

It may not be enough to run the training data through the network only once; hence multiple epochs may be necessary. A larger batch size allows for improved parallelism, which in turn increases the amount of computation per iteration. The main drawback, however, is that an increase in batch size may negatively impact the network by affecting the generalization performance [51]. Another drawback with an increased batch size is an increase in memory required to temporarily store the data for each iteration. If there is not enough memory available, the network will halt indefinitely, or simply crash.

When using a small batch size, the time it takes to train the neural network for each iteration increases since the gradient for each sample may diverge from the end-point. Determining the correct batch size for a given network is unfortunately left to trial and error but can be narrowed down by performing tests on a sample of the dataset to see when it converges fastest.

Selecting the appropriate parameters for deep neural networks are often achieved through trial and error.

3.5 Deep Learning Frameworks

There are many deep learning frameworks in active development, and in pursuit of the status as the go-to framework for deep learning. There is at the time being no framework objectively better than every other in every single aspect, hence a framework decision should be tailored to each project's individual requirements.

Some criteria when choosing a framework could be:

- Ease of installation and setup
- Quality of documentation
- Maturity of development

- Supported features
- Project criteria
- Performance
- Reliability

An important aspect is the survivability of the framework. **Open source** frameworks can be more reliable than closed projects, since anyone can (with permission) contribute to the project. One can also fork (i.e. create a copy of) the repository and continue developing it if the framework is for instance abandoned by the original creators.

A framework with a large user-base is more likely to be the leading framework over time, hence good documentation is essential! A framework where the documentation is outdated, unreliable, or missing is unlikely to attract new users.

Which hardware is required, and what performance one can expect from the framework are also important criteria. Can the framework be used with graphics cards from AMD and NVIDIA, without requiring their newest versions, and if so, what performance can one expect?

A framework's overall quality could be judged by the number of stars and forks it has on GitHub. The two metrics do not necessarily correlate with the number of users but can be useful as a proxy for popularity. *Table 3* illustrates the frameworks' "popularity" on GitHub:

Framework	Stars	Forks	Contributors
TensorFlow	100 952	63 511	1480
PyTorch	15 883	3584	644
Microsoft Cognitive Toolkit	14 517	3849	184
MXNet	14 017	5186	531
Caffe2	7982	1950	188

Table 3: Deep learning frameworks on GitHub in order of stars – Source [52]

A contributor is someone who commits code (i.e. makes a change to a file) to the default/main branch of the project.

Opinions of which framework is better are most certainly biased. They are all constantly evolving, which is why any comparison is likely to be outdated within a short timeframe.

For instance, an important distinction between PyTorch and TensorFlow was that PyTorch uses dynamic computational graphs (DCG), while TensorFlow uses static computational graphs (SCG). In short, SCG is define-and-run, meaning that the graph is defined statically before a model can run, while DCG is define-by-run, which allows for structuring nodes at runtime.

An extension to TensorFlow, called TensorFlow Fold [53] has since then been released, which provides support for DCG, making the comparisons emphasizing on this aspect outdated.

Fig. 12 compares the growth-rate in terms of stars for each framework in *Table 3*.

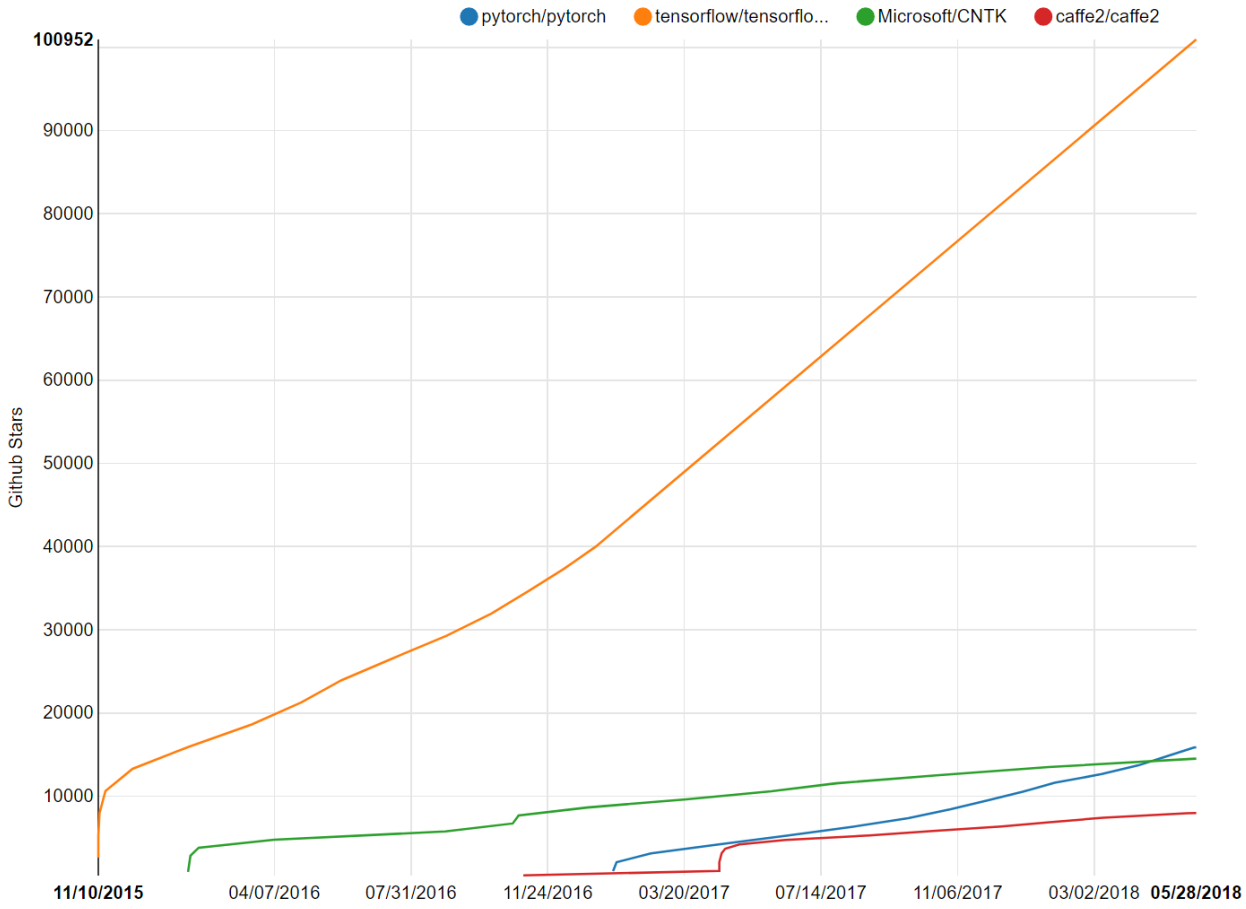

Figure 12: Stars on GitHub over time for PyTorch, TF, CNTK, Caffe2 – Source [54]

Either of the mentioned frameworks would likely suffice with the fish recognition task, but TensorFlow was primarily chosen as the framework for this thesis due to its massive growth and popularity, and TensorBoard. The more people involved with the framework, the faster new features and bug-fixes are added, as well as a more up-to-date and well documented documentation (in theory, at least).

TensorBoard is a tool which visualizes all data related to the network in the web-browser, which makes for great comparisons of various networks. This fits well with the thesis, since a large aspect of it is visualizing and comparing results.

4 Solution – Setup Phase

4.1 Datasets

Steinsvik AS, which states that they are a world-leading technology supplier to the global aquaculture industry [55], provided the training- and testing data upon request. They gathered the data using cameras in pisciculture (fish farms) located in Norway. The labels of fish were drawn by hand as part of this thesis work. Minimum bounding boxes (defined with *x*, *y*, *width*, and *height* coordinates) were drawn, and consist of the entirety of the fish (i.e. the fish has not been separated into head, body, and tail classes). The only fish species in the dataset is salmon, thus the network was not trained to detect multiple species.

The dataset consists of 13 videos recorded throughout 2015 (in January, August, and December), with a total of 32 365 frames (54.2 GB of data). Of these, only 4551 frames were used for data extraction, due to many frames having no identifiable fish, or the current frame not differing enough from its previous frames.

Most of the labeled fish are captured from its side (see *Fig. 13*, *Fig. 14*), but some are also captured from either above or below the fish as well (see *Fig. 15*, *Fig. 16*). Unfortunately, the dataset is lacking samples of fish swimming directly towards or away from the camera, thus the networks are not as efficient at recognizing fish in these situations.

The final dataset consists of 20 069 labeled fish, and *Table 4* illustrates the display resolution distribution of the images.

Display Resolution	Number of Frames	Number of Labeled Fish
2560x1920	1182	121
1920x1080	28 975	19 314
1600x1200	250	524
720x576	1958	110
Total	32 365	**20 069**

Table 4: Number of frames and labeled fish from each display resolution

Labeling the fish was not an unsubstantial part of the thesis, as it took the author close to 40 hours. Labeling was done manually using LabelImg (v1.4.3), a graphical image annotation tool.

In hindsight, the hours spent labeling could be slightly reduced if labeling had been done properly from the start, but the labeling process provided some valuable knowledge regardless. Initially, only fish clearly visible in the foreground were labeled and added to the dataset. When

this dataset was used for training and testing, it resulted in poor recognition results for fish located further in the background, as the network was trained being too specific. Therefore, the labeling process had to be repeated over all the pictures, keeping the existing labels, to also include (smaller) fish in the background as well.

Additional data was requested and provided by Steinsvik AS a couple of months after the first request. The additional videos contained 4104 frames (first request had 28 261 frames), in which 375 frames were used for data extraction (compared to first request's 4176 frames), resulting in an additional 1706 labels. The dataset is presented in *Table 4*.

The dataset was split into a training set consisting of 90% of the labels, and the remaining 10% were used for testing. The recommended distribution of each training- and testing set depends on how much data is available. If there are millions of images, then a 1% testing set may suffice in evaluating the network with enough confidence.

4.1.1 Labeling Conditions

The quality of the dataset varies, as *Fig. 13*, *Fig. 14*, *Fig. 15*, and *Fig. 16* demonstrates. The images are random samples from different sub-datasets for demonstrating not only the quality of the datasets, but also which fish were considered a good label. It would be an interesting experiment to iterate through the dataset once more and be even less critical as to which fish to label and compare the performance of this new, larger dataset, with the one currently in use.

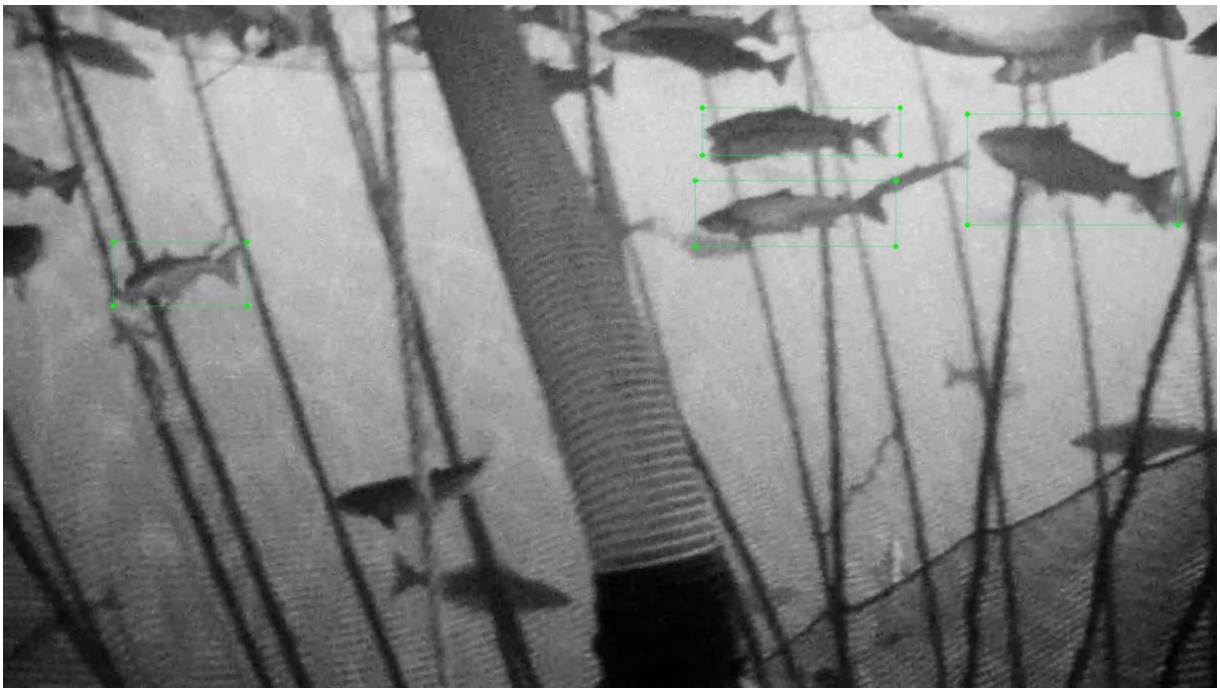

Figure 13: A 1920x1080 frame from sub-dataset 2 with four manually labeled fish

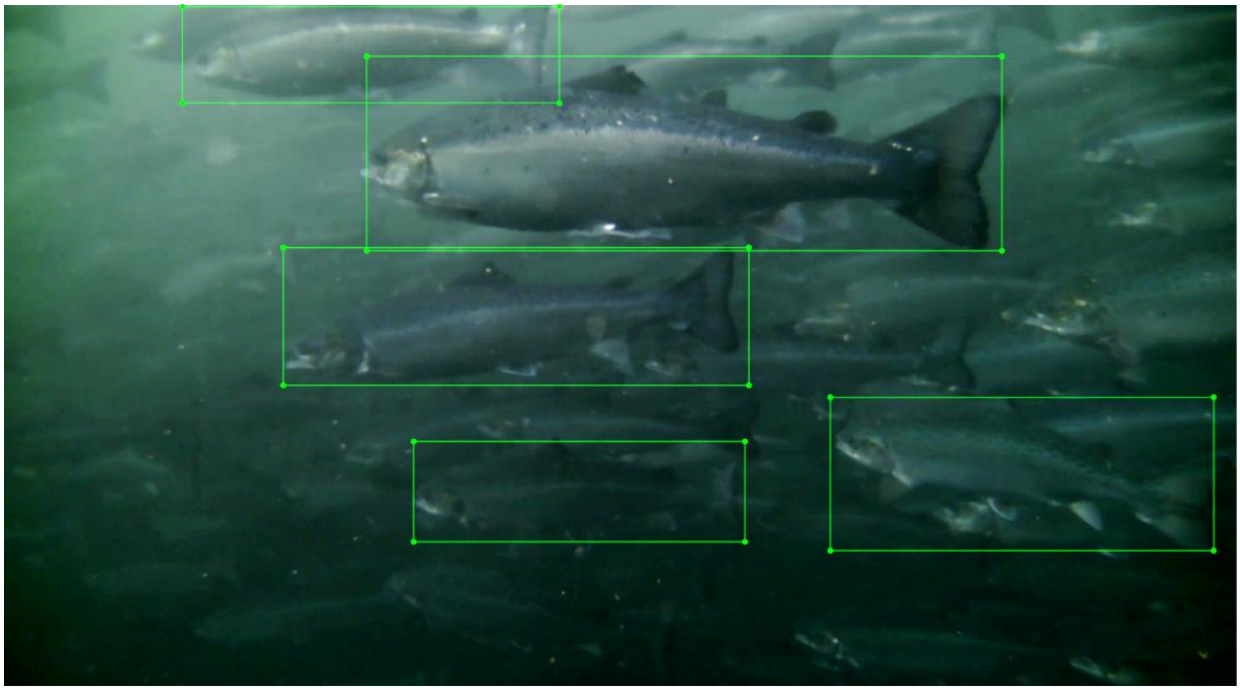

Figure 14: A 1920x1080 frame from sub-dataset 4 with five manually labeled fish

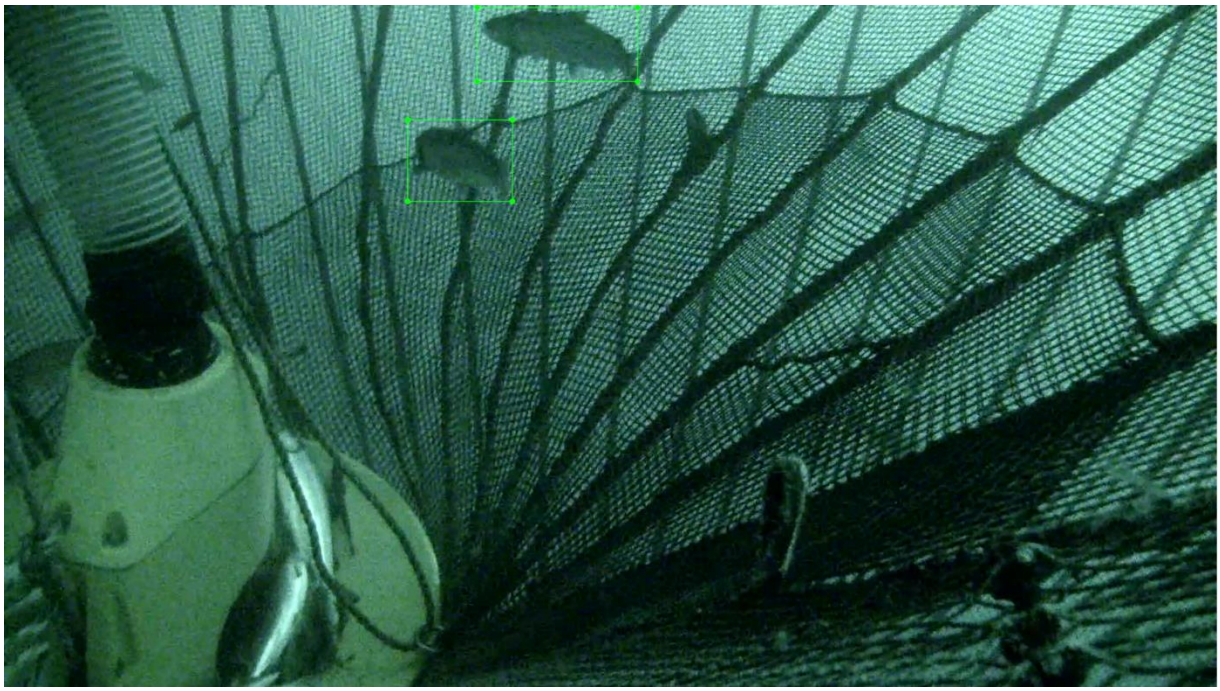

Figure 15: A 1920x1080 frame from sub-dataset 5 with two manually labeled fish

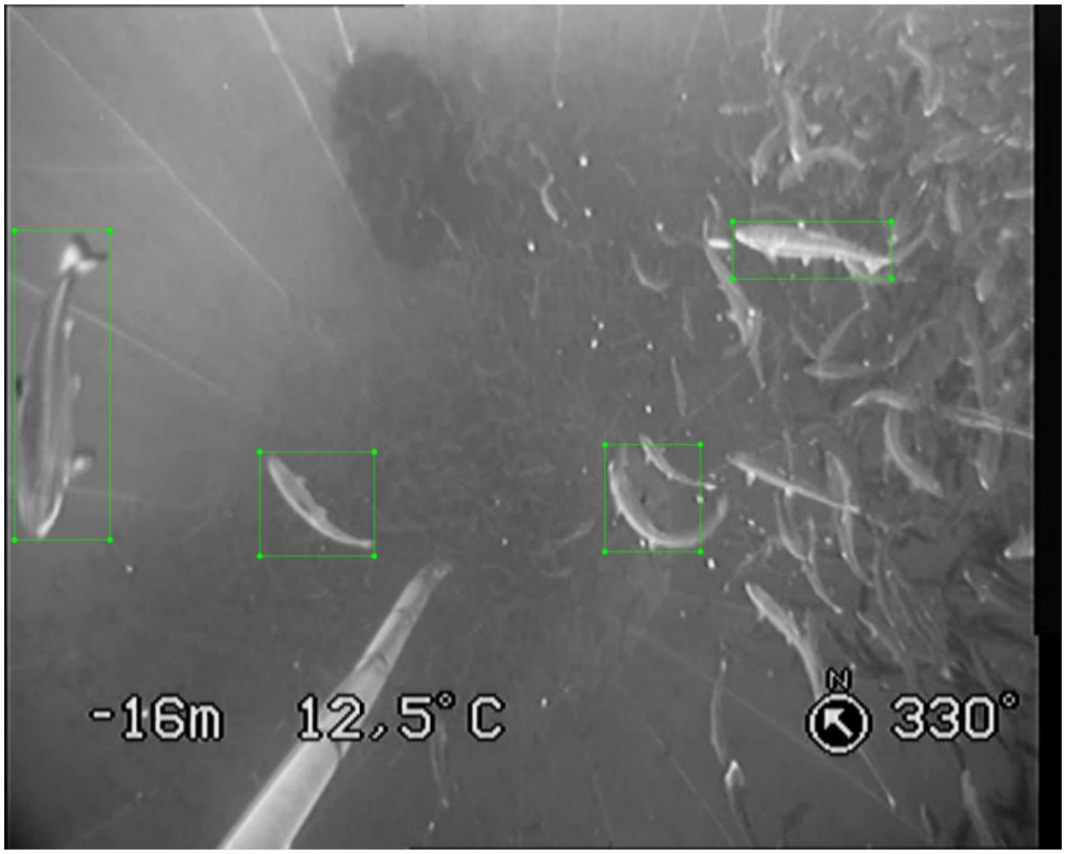

Figure 16: A 720x576 frame from sub-dataset 9 with four manually labeled fish

4.2 Extracting and Formatting Data

To retrieve all the frames in a video, FFmpeg, a multimedia framework, was used. The dataset was divided into multiple sub-datasets, where the extracted frames were separated into their respective folders, easing the process of adding or removing potential datasets in the future. Additionally, the images were renamed to comply with the following standard: class, followed by sub-dataset, followed by image id (e.g. fish_1_1). This makes converting the dataset to other formats easier when iterating through the images.

All labeling coordinates for a single image are stored in its own Excel-file with an id referencing the image, where the Excel-file was generated using the graphical image annotation tool, seen in *Fig. 17*. Each Excel-file represents all fish coordinates for one single image, thus, with 4551 images, there are 4551 Excel-files.

An example of how the data for each label is stored can be seen in *Table 5*, which contains the minimum bounding box coordinates for three of the fish in *Fig. 17*.

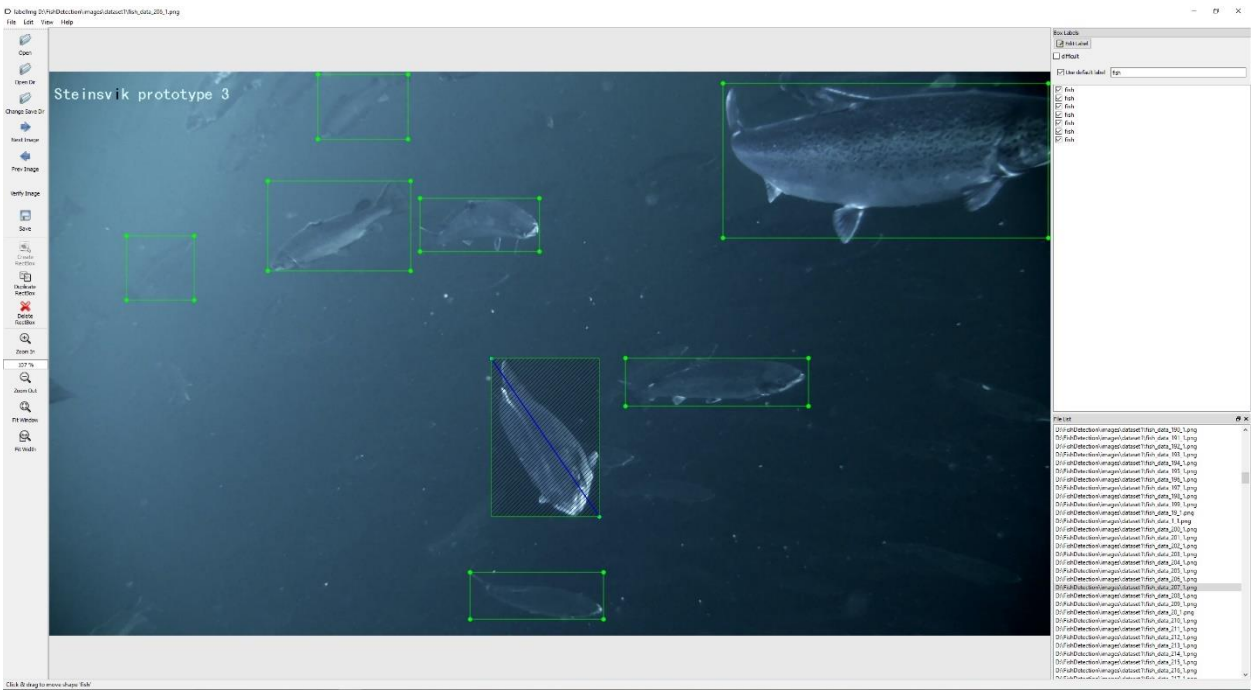

Figure 17: LabelImg v1.4.3, where tools are on the left, and classes and a list of images on the right

Table 5 contains the minimum bounding box coordinates for three of the fish in *Fig. 17*.

folder	filename	path	width	height	name	x_{min}	y_{min}	x_{max}	y_{max}
dataset1	fish_206_1.png	D:\...\fish_206_1.png	1920	1080	fish	419	209	693	381
dataset1	fish_206_1.png	D:\...\fish_206_1.png	1920	1080	fish	711	242	940	344
dataset1	fish_206_1.png	D:\...\fish_206_1.png	1920	1080	fish	1105	548	1456	640

Table 5: Structure of XML (Extensible Markup Language) data

Once the images were correctly labeled, the dataset was converted to a specific TensorFlow file format known as TFRecords, which provides an easier way of handling scalability. TFRecords makes it possible to shuffle, batch, and split (i.e. divide the data into training and testing sets) datasets with its own functions. By shuffling data, the training and testing sets are representative of the overall distribution of the data. It also helps in keeping the network's learning generalized, otherwise bias towards a certain combination of input may occur.

The 4551 Excel-files were later converted to a single CSV (comma-separated values) file to reduce the number of files and disk space (from 17.7 MB to 1.1 MB in this case).

The final step was generating the TFRecords, which are represented with the ".record" format. To convert the data to TFRecords, the properties for each annotated object must be converted from original string values into specific datatypes for TensorFlow to operate on the values. The bounding box variables (i.e. x_{min}, y_{min}, x_{max}, y_{max}) are converted to float, the *label*, *height* and

width of the image to int64, and some image specific properties such as the *filename* are converted to bytes.

For the aforementioned datasets, the size of the testing dataset, *test.record*, is 741 MB, while the size of the training dataset, *train.record* is 6.4 GB.

4.3 Environment

During the thesis, TensorFlow went through several iterations, and TensorFlow's dependencies has naturally changed over the course of its development. The first TensorFlow version used was v1.3.0, and newer versions were upgraded to when they were publicly released, or in some cases, nightly builds were used to get access to newer features and improvements earlier.

For clarification, TensorFlow v1.8.0 is the final version this thesis used. Earlier versions of TensorFlow and its earlier dependencies will not be elaborated upon, and any further reference to TensorFlow is a reference to TensorFlow v1.8.0.

TensorFlow can run either using a central processing unit (CPU), or a graphics processing unit (GPU). Using a GPU will drastically reduce the amount of time spent training and testing the network, which will be demonstrated in an upcoming chapter, 6.1.1 CPU vs GPU. Note that when installing TensorFlow, one must explicitly state whether one would like to install the CPU or GPU compatible version.

TensorFlow is dependent on multiple libraries, and *Fig. 18* illustrates the dependencies.

CUDA is NVIDIA's application programming interface (API) for communicating with the GPU. cuDNN is built using CUDA, and is a library offering building blocks for deep neural network applications. Each CUDA version is dependent on a specific version of cuDNN, independent of TensorFlow.

Each version of TensorFlow is only compatible with a specific version of CUDA and cuDNN. TensorFlow 1.8 supports CUDA versions 8.0 and 9.0, and cuDNN versions 6.0 and 7.x. To download cuDNN, one must be a member of NVIDIA's developer program, in which one must agree to their terms and conditions when creating an account.

TensorFlow uses primarily Python as an interface for interacting with it, but has some API support for Java, C, and Go, as well as a library for machine learning in the browser using JavaScript. TensorFlow's required dependencies are automatically installed when installing TensorFlow using pip, which is a tool for simplifying both installation and management of Python packages. On Windows, TensorFlow supports (Windows 7 and later) Python 3.5.x and 3.6.x.

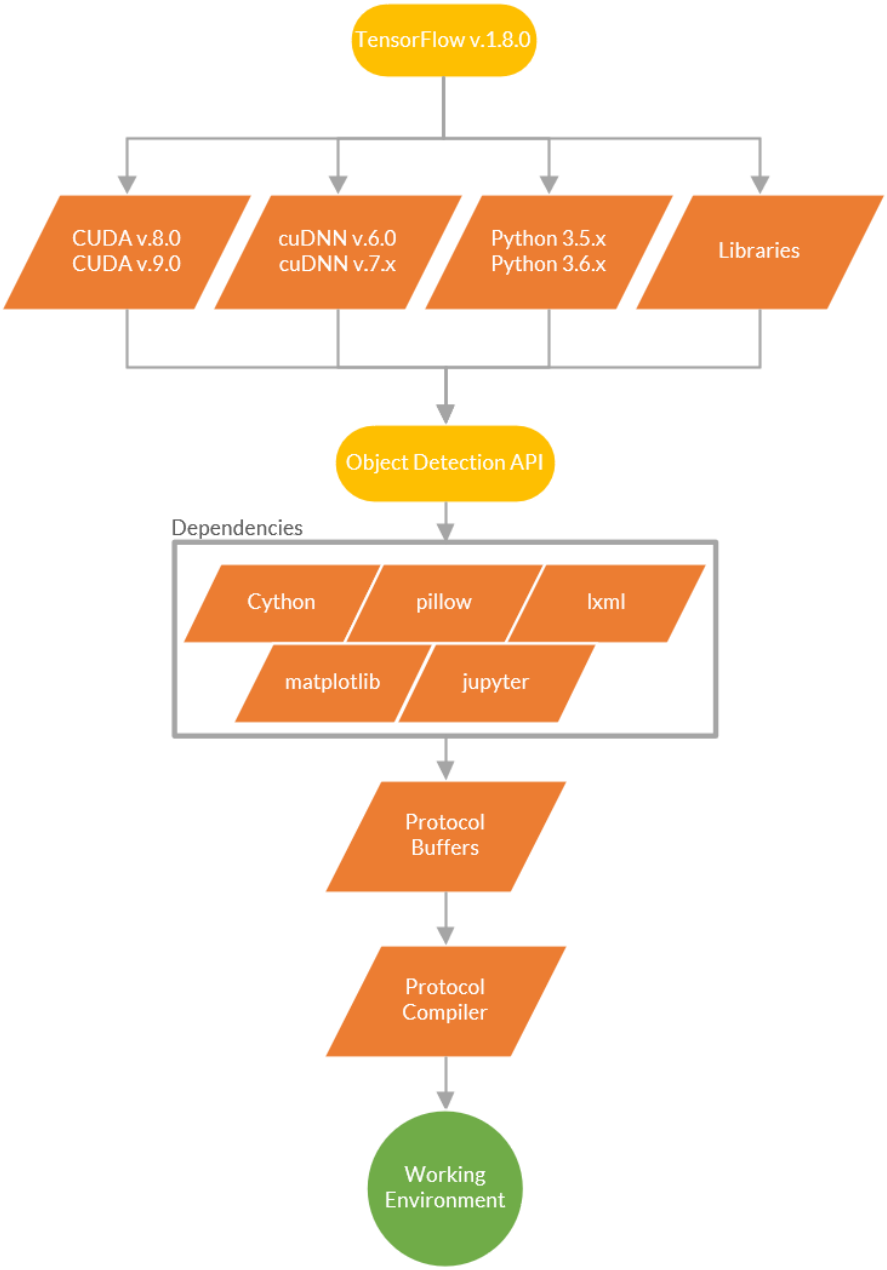

Figure 18: TensorFlow dependencies

When developing using TensorFlow's object detection API, one must also install the Python libraries Cython, Pillow, lxml, Jupyter, and Matplotlib. The object detection API uses protocol buffers (PB) to configure network parameters. PB are Google's language- and platform-neutral method for serializing structured data and requires the Protocol compiler, which compiles ".proto" files for Python.

Installing TensorFlow on Windows for the first time can be a perplex situation for the time being. There are many dependencies for each library, and functionality might break completely when updating a library to a newer version.

5 Solution – Systematic Teardown

This chapter is devoted to the many algorithms and architectures used for the fish recognition task. It is structured such that for each section, one delves deeper into how the various technological components function. There are a lot of intricacies in the component chain described in *Fig. 19*, which illustrates that a network is the combination of a model and a classifier (each of which have a certain architecture), where a classifier is the backbone of a model.

Some clarification on upcoming terms are required, to minimize the risk of confusion. For this thesis, a network refers to the system altogether, describing its model, classifier, and their underlying architecture and functionality.

A **model** is the component responsible for generating bounding boxes. A classifier aids the model and is ultimately the component responsible for providing a prediction (in combination with the model). Some models are constructed such that they can contain any type of classifier. The classifier has various operations and functions, which alters the state of the data passing through the layers.

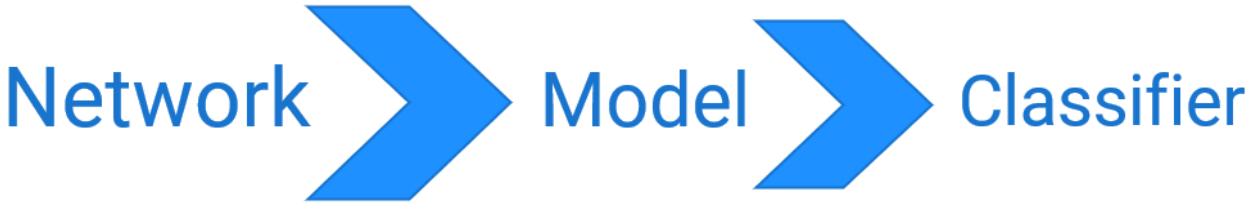

Figure 19: The component chain in underlying order

Fig. 20 illustrates some types of neural networks, but there are many more. The neurons in the figure not related to CNN will not be covered. Following is a short overview of some of the neural networks presented in *Fig. 20*:

Variational Auto Encoder (VAE): Purpose of encoding (i.e. compress) information automatically, where encoding happens before the middle layer, the middle layer is the code, and decoding occurs after the middle layer. VAEs are taught an approximated probability distribution of the input samples and attempts to rule out influence between nodes.

Restricted Boltzmann Machine (RBM): A variant of Boltzmann Machine, which instead of connecting every neuron to every neuron, only connects neurons of different types. This makes sure that input neurons are not connected to one another for instance. Useful for dimensionality reduction, filtering, classification, and regression.

Deep Belief Network (DBN): Consists of stacked architectures of RBMs or VAEs, using a stack by stack training, or greedy training, thus optimizing locally (each stack) instead of globally (the entire network). It works such that the next local network (or stack) must encode the previous network. Useful for clustering, generating, and recognizing images and video-sequences.

Recurrent Neural Network (RNN): Neurons are fed information from previous layers, as well as themselves from the previous pass. The order of the input is important, as it relies on previous information to form new decisions. Ideal for processing sequential data, such as sound and text (e.g. autocompletion).

Convolutional Neural Network (CNN): Refers to a neural network that has at least one convolutional layer, and that utilize prior knowledge to make better decisions by using convolution. Primarily used for image processing and in some cases audio. Useful for classifying data, e.g. whether the object in an image is of a volcano rabbit or a riverine rabbit.

All the mentioned networks can be trained using backpropagation. It might not always be the best local search algorithm for each specific network, but it has proven to be very efficient in general for complex tasks [38, 40].

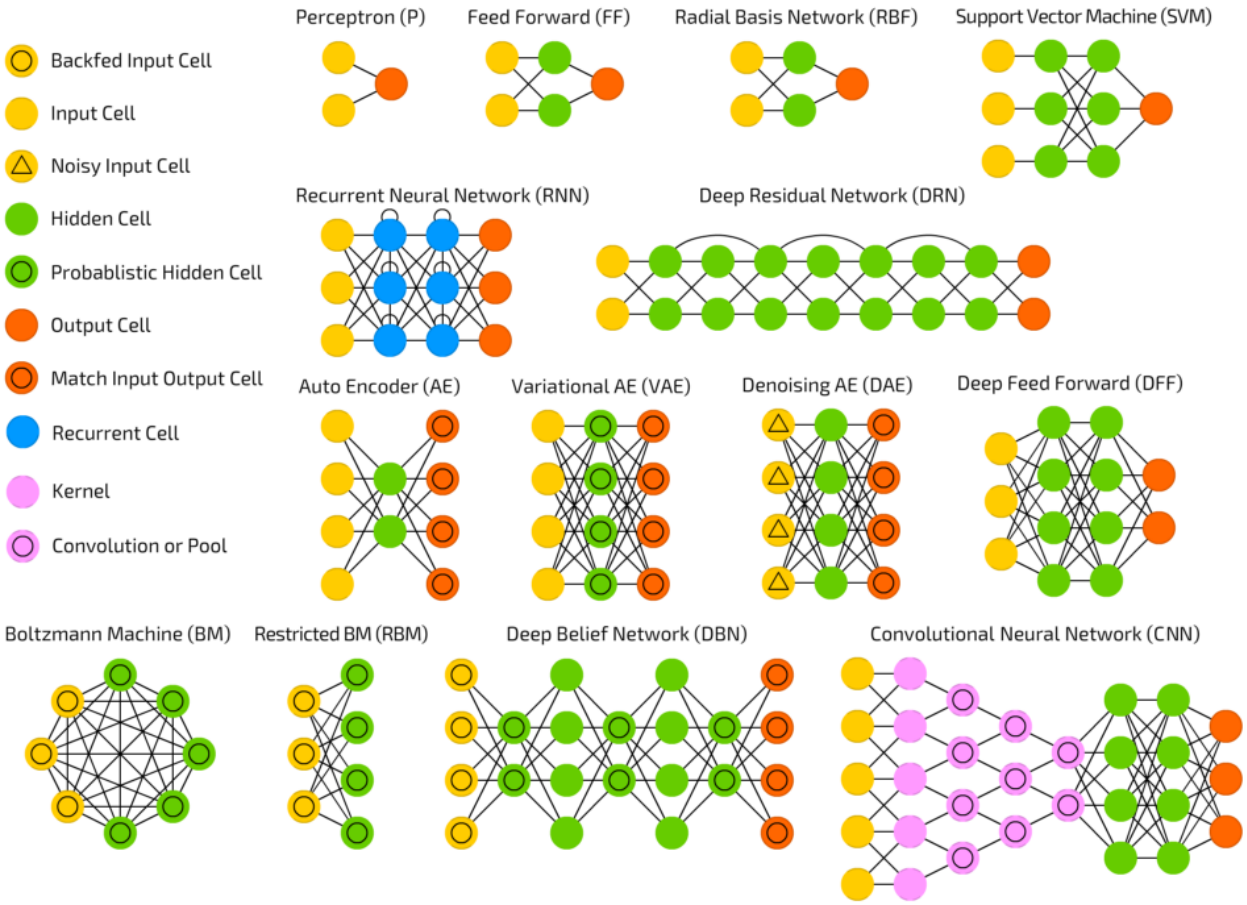

Figure 20: Chart of multiple types of neural networks – Adapted from [56]

5.1 Convolutional Neural Network

Convolutional neural networks (CNN) use convolutions to extract features from the input image and are advantageous due to their sparsity of connections and parameter sharing, essentially learning features using small squares of input data at a time.

CNN was chosen for the task since CNNs were designed with image processing in mind [57], and have proven to be successful in many image recognition applications [30, 58, 59, 60, 61], including self-driving cars [62].

The convolution operation reduces the dimension of the input at each step, which decreases the number of inputs that each output value relies on. CNNs are computationally efficient and can run on a low-end smartphone as long as it is configured appropriately.

Every CNN has a similar architecture as shown in *Fig. 21*, but there are naturally deviations from this structure too. The idea is that the input moves throughout the layers and is downsampled at each step, illustrated by the stippled lines. Once the data reaches the **fully connected** (FC) layer it is converted from a 3D volume to a 1D vector. A FC layer is a linear operation in which every input is connected to every output by a weight. The second FC layer receives the (probability) values from the neurons in the first FC layer, and the neuron in the second FC layer with the highest value (probability) determines the label for the input image.

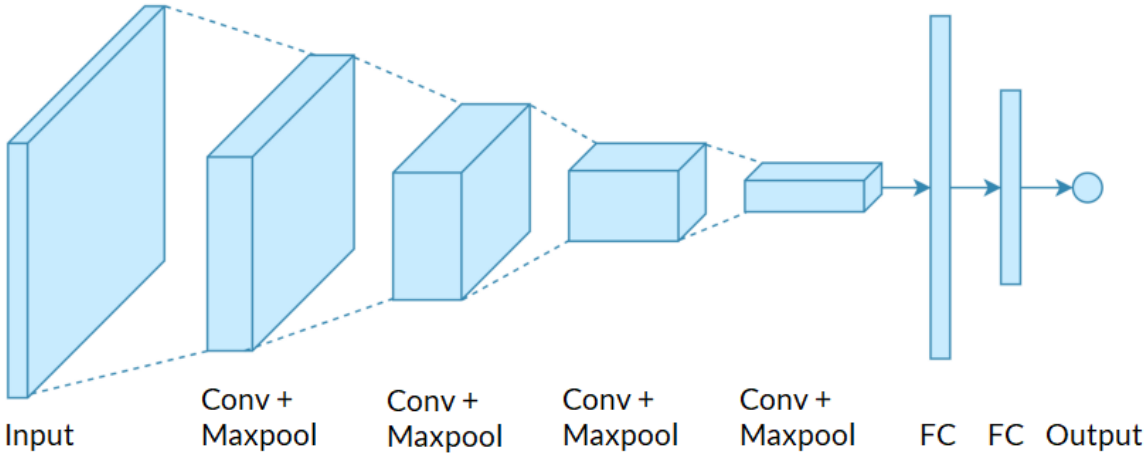

Figure 21: Convolutional neural network (CNN) architecture – Adapted from [63]

Each of these layers, operations, and the process will be explained in-depth in the upcoming sections.

5.1.1 Convolution

Convolution is an operation for extracting features from an input image, which is achieved by merging two sets of information by using element-wise matrix multiplication and computing

the sum of the results within the filter. The goal is to produce a **feature map** (i.e. the output activations of a given filter) by applying a convolution filter on top of the input data, as shown in *Fig. 22*. Note that a filter and a kernel are interchangeable terms.

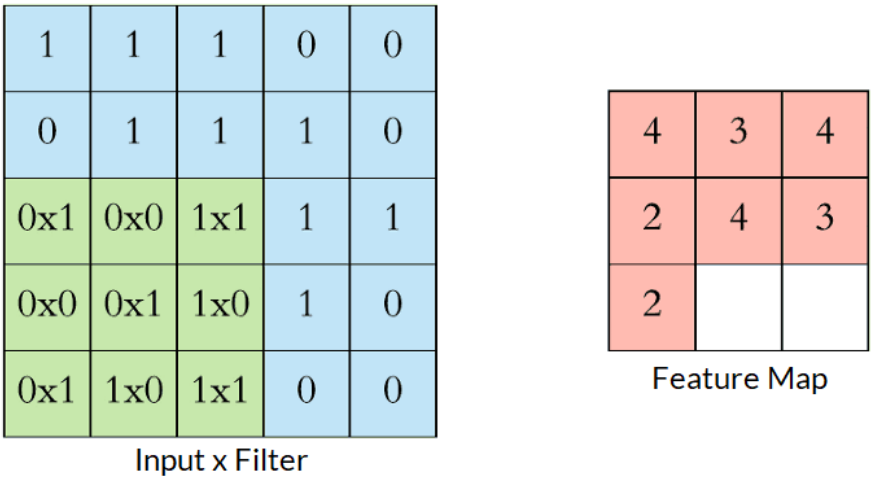

Figure 22: 2D convolution with input, filter, and feature map – Adapted from [63]

In this 2D convolution operation, the filter (green square) is applied on the input data (blue square), and the element-wise matrix multiplication is computed. The values in the blue squares are the pixel values, and in this case, they are here either white (1) or black (0) for simplification. For a greyscale image, pixel values are in the range $[0, 255]$.

The current filter position in *Fig. 22* calculates $a * b$, where a is the input value, and b is the filter value, for each square. The sum of the values produces a single value for the feature map:

$$0 * 1 + 0 * 0 + 1 * 1 + 0 * 0 + 0 * 1 + 1 * 0 + 0 * 1 + 1 * 0 + 1 * 1 = 2.$$

The sum of each square is stored in the 3x3 feature map. The filter moves one step in x-direction until it reaches the edge, in which its x-direction is set to 0, and its y-direction increases with one. For instance, in *Fig. 22*, the filter has traversed the entire input, except for the last two positions, which is why there are two empty squares in the feature map (i.e. they have not been calculated yet).

An image, however, is not two-dimensional, but a 3D matrix with dimensions of height, width, and depth (i.e. the color channels, RGB (red, blue, green)). Therefore, for a convolution to cover the entire input, it must be a 3D operation as well, and this is illustrated in *Fig. 23*.

In three dimensions, the feature map covers the entire height and width, and moves (in this case) a single step for each convolution. In the $32x32x3$ image on the left in *Fig. 23*, a $5x5x3$ filter is applied, and the result is stored as a single element ($1x1x1$) in the $32x32x1$ feature map. Note that the $5x5x3$ filter and the $32x32x3$ image have the same depth of three. The

output is the $32x32x10$ blue box as seen on the right, which means that 10 different filters are used, resulting in 10 feature maps stacked side-by-side.

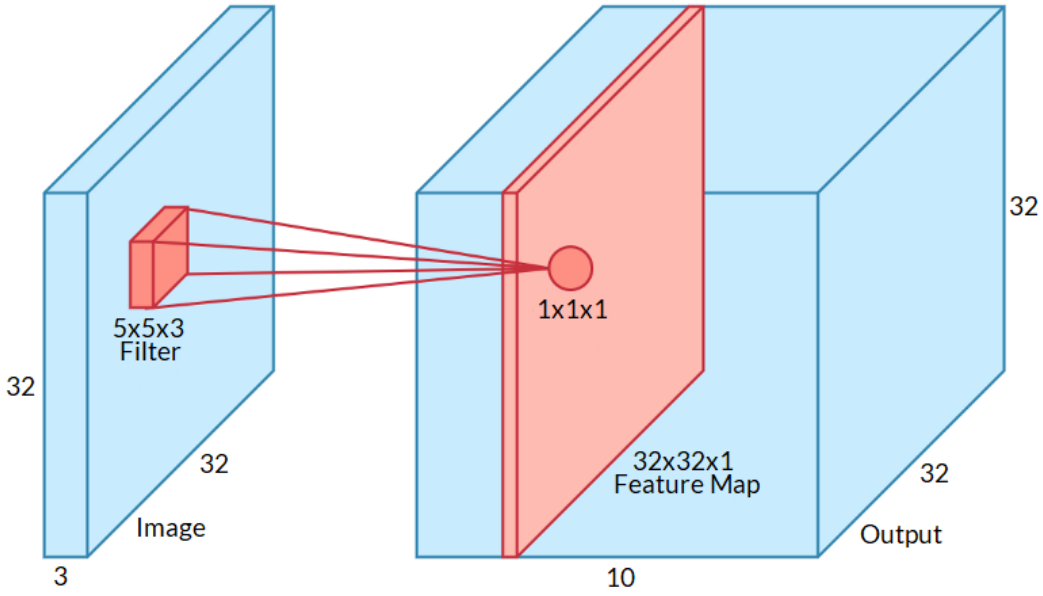

Figure 23: 3D convolution with image, filter, feature map, and output – Adapted from [63]

The output of a 3D convolution consists of multiple stacked feature maps, which are computed just as the 2D convolution in *Fig. 22* illustrates. There are in this case 10 feature maps in the depth-range $[0, 10]$ for the output. Once a feature map has been computed for position 0, the next feature map is computed for position 1, and so forth.

Since a neuron's value could be anything in the range $(-\infty, \infty)$, it is necessary to set some bounds for the neuron which tells the neuron whether it should be active. Therefore, one use activation functions, for instance a rectified linear unit (ReLU), where $f(x) = \max(0, x)$. Networks using ReLU are easier to optimize compared to networks with tanh or sigmoid units, due to how gradients can flow when the input to the ReLU function is positive. ReLU is the most widely-used activation function due to its effectiveness and simplicity [64].

Once the convolution operation has been computed, the result is passed through an activation function, and the value stored in the feature maps are thus the activation function applied to the sum, and not just the sum itself.

5.1.1.1 Parameters

A convolutional layer consists of a few key parameters, such as:

- **Filter Size**: Defines the field of view of the convolution (green area in *Fig. 24*).
- **Stride**: The filter's step size when traversing the image.

- **Padding**: Defines the border of a sample and determines whether spatial output dimensions are equal to the input or reduced.
- **Input & Output Channels**: The number of input channels determine the number of output channels for the convolutional layer.

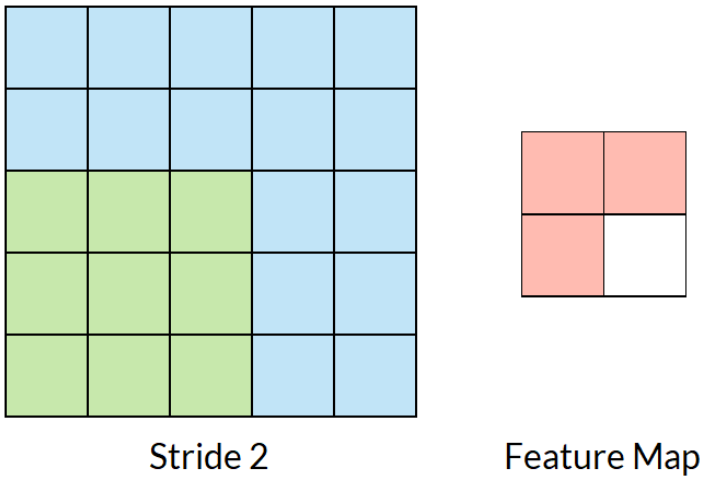

Figure 24: Filter with a stride of 2 – Adapted from [63]

If the stride is larger, as seen in *Fig. 24* which has a stride of two, the feature map becomes smaller, as the convolution filter must be contained in the input. It now jumps two pixels at a time instead of one.

One can also apply padding (grey area in *Fig. 25*) to surround the input with zeros to maintain the dimensionality of the feature map. Padding thus preserve the size of feature maps to keep them from shrinking at each layer, and this improves overall performance [57]. Otherwise, the volumes' size would decrease after each convolution, thus information at the edges would be lost too quickly.

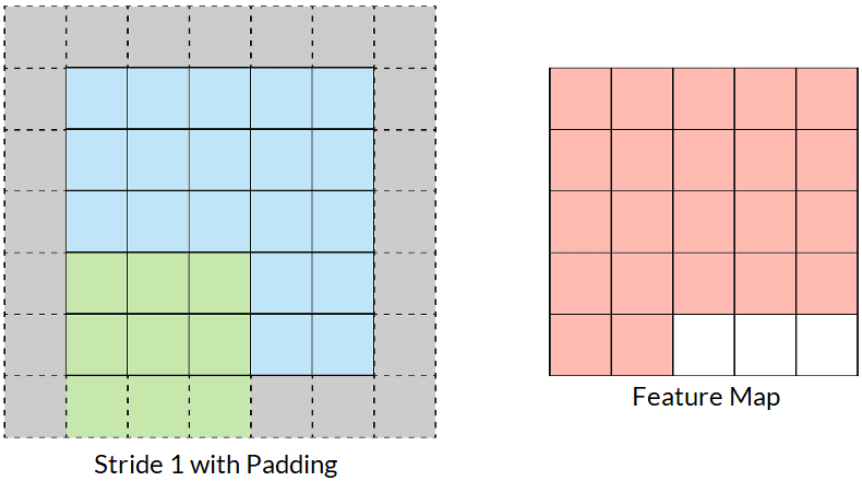

Figure 25: Padding the input to preserve the size of the feature map – Adapted from [63]

Notice in *Fig. 23*, that the height and width of the feature map is equal to the input image ($32x32$), while only the depth differs (3 for the image, 1 for the feature map), and this is due to padding. Recall that information at edges are lost without padding (the size is reduced).

There are multiple types of convolutions, but the ones used in the thesis are dilated and depthwise separable convolutions. The main differences are how they select values from the filter, or in other words, what the parameters for the filter are, and how they are defined. The following sections describe the dilated and depthwise separable convolutions.

5.1.1.2 Dilated

Dilated (sometimes referred to as atrous) convolutions have an additional parameter, called the dilation rate. It defines the spacing between the pixel values in a filter, as seen in *Fig. 26*, where the teal values are dilation rates, the red dots are the input values to a filter, and the green area is the receptive field captured by the inputs.

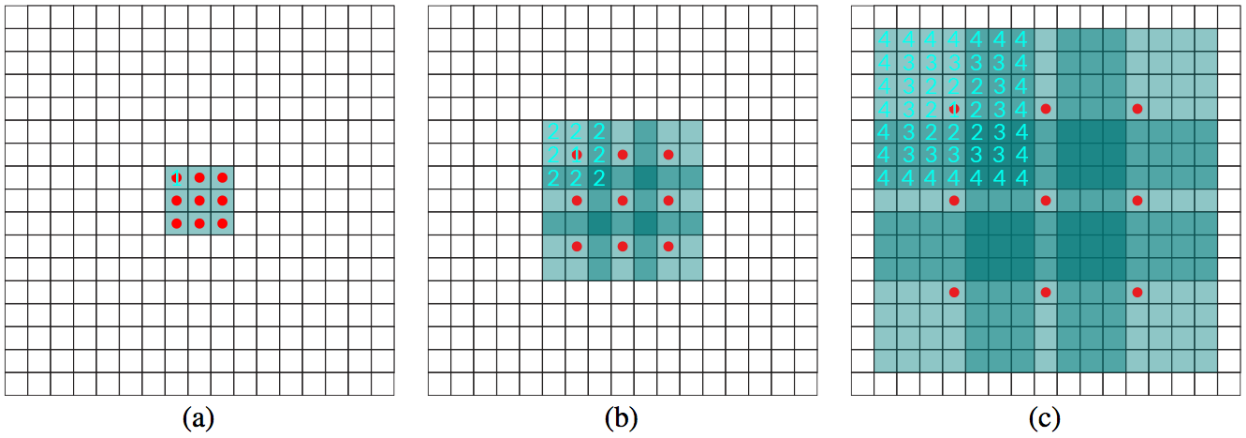

Figure 26: Increased dilation rates causes increased receptive fields – Adapted from [65]

When the number of parameters grow linearly with layers, the receptive field of units grows exponentially with layer depth. For instance, *Fig. 26 a)* has a one-dilated convolution, resulting in a $3x3$ receptive field for each element. *Fig. 26 b)* increases the dilation rate to two and *c)* to four, resulting in a receptive field of $7x7$ and $15x15$, respectively.

Dilated convolutions are practical in vision applications, as the smaller filters and reduced convolutions contribute to a lower computational cost, while delivering a wider field of units. On the other hand, the network will not be able to learn random receptive field behaviors with this type of convolution. The receptive field is the region in the input space that the network is looking at, where pixel values closer to the center contributes more to the calculation of the output feature.

5.1.1.3 Depthwise Separable

Depthwise separable convolution is a multi-convolution operation, first performing the depthwise convolution, and then the separable convolution. This process reduces the number of parameters, which lessens the risk of overfitting, and reduces the number of operations to compute, thus being cheaper and faster, with only a slight decrease in accuracy.

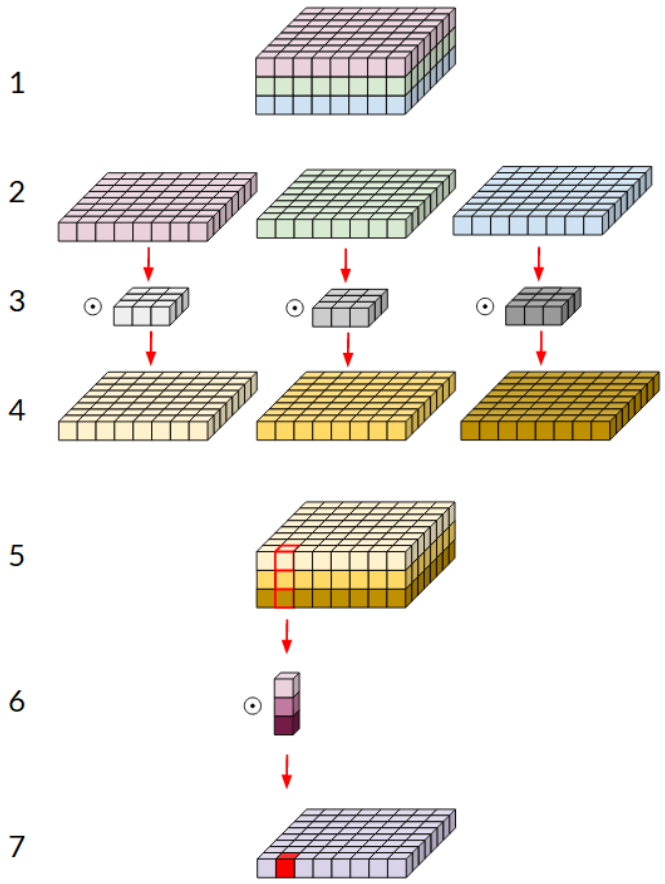

Figure 27: Depthwise (step 2 - 5) and separable (step 5 - 7) convolution – Adapted from [66]

The depthwise process is seen in *Fig. 27* and is as follows:

- The $8x8x3$ ($height\ x\ width\ x\ depth$) input (step 1) is separated into three $8x8x1$ channels (step 2), and the filter is split into three $3x3x1$ channels (step 3).
- Each channel then convolves the input channel (in step 2) with the corresponding filter (in step 3), thus producing an $8x8x1$ output tensor (i.e. a generalization of a scalar, matrix, and vector) (step 4).
- The output tensors are then stacked back together (step 5).

Following the depthwise process is the separable process, which (also seen in *Fig. 27*):

- Performs a $1x1$ convolution across all three channels (in step 5), producing a $1x1x3$ output (step 6), which eventually produces a single $8x8x1$ output channel.

5.1.2 Pooling

Pooling is performed once a convolution operation is complete and is used to reduce the dimensionality and the number of parameters, shortening the training time and combatting overfitting. In pooling layers, each feature map is downsampled independently, keeping the depth intact while only reducing the height and width.

Pooling works quite similarly to convolution, in that it slides a window over the input, and calculates the new value from the window. As in convolution, the window has a specified size and stride, but unlike convolution where one would like to preserve the feature map's size, pooling aims to reduce it.

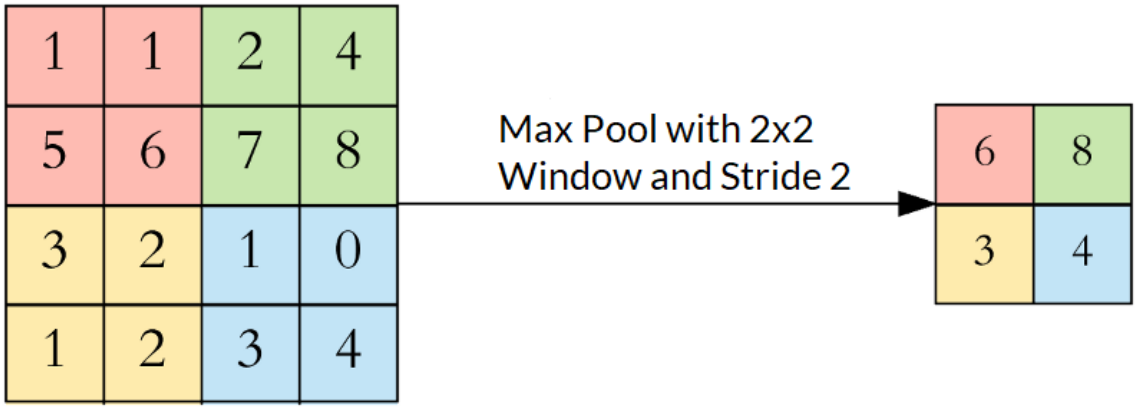

Figure 28: Applying pooling to reduce dimensionality – Adapted from [63]

In *Fig. 28*, the most common type of pooling, max pooling (i.e. taking the maximum value in each window), is applied, and in this case the $4x4$ feature map is downsampled to $2x2$.

For instance, a $32x32x10$ input will be downsampled to a $16x16x10$ feature map by using the same pooling parameters as above, which reduces the number of weights to ¼ of the input.

5.1.3 Fully Connected

Once the convolution and pooling operations are completed sufficiently, they are followed up by fully connected layers. Fully connected means that every neuron in the previous layer is connected to every neuron in the next layer, and its purpose is to classify the input image based on the high-level features from the previous layer. Recall from *Fig. 21* that the convolution and pooling layers operate with 3D volume, but a fully connected layer only accepts an input in 1D, hence the 3D volume of numbers must be flattened to a 1D vector. This is illustrated more clearly in *Fig. 29* and *Fig. 30* in 5.1.4 Visualizing a Convolutional Neural Network.

The last fully connected layer produces the values for the output layer, which consists of for instance 10 classes (one neuron for each class, where a single class represents a single digit in the range $[0, 9]$). The sum of these neurons is the value 1, where if a single neuron outputs 1, it signals that the network is 100% confident in its decision that it has chosen the correct label for the input data.

The neuron represents a class, which when activated returns the label for the image. The neuron with the highest value (confidence) is activated, as seen in *Fig. 29* and *Fig. 30* for the output layer.

5.1.4 Visualizing a Convolutional Neural Network

Fig. 29 visualizes each layer, with the name of each layer on the right-hand side. The input "5" was drawn by hand and can be seen in the lower left-hand corner in the white box.

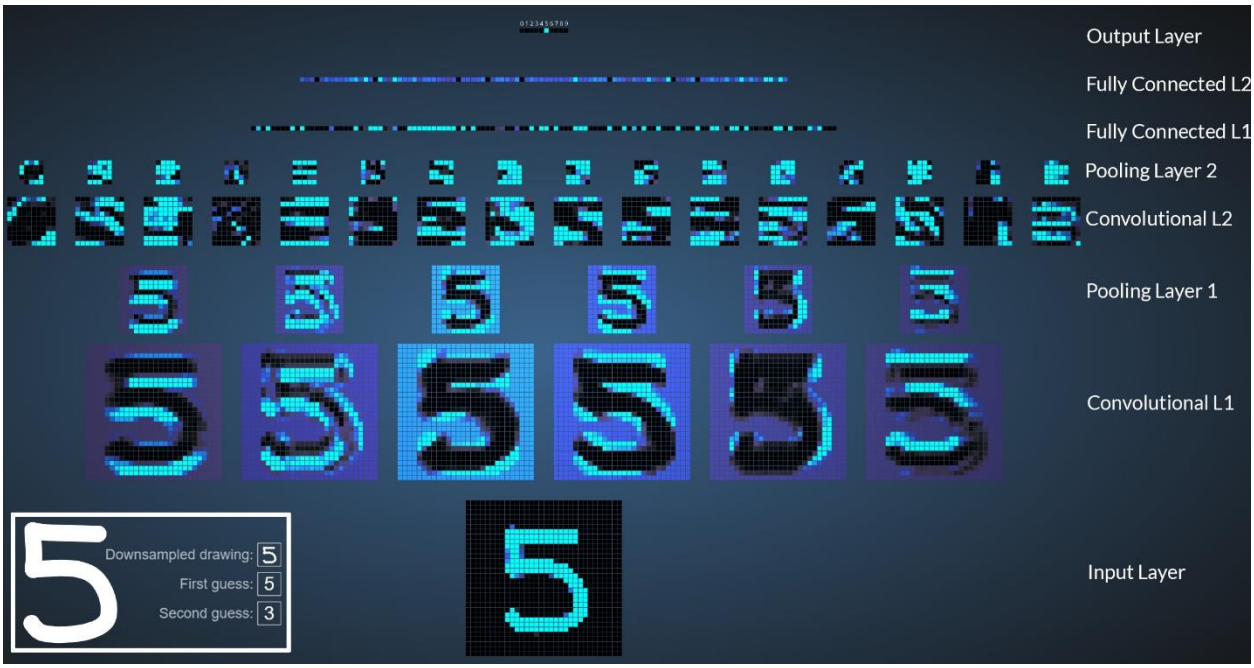

Figure 29: Visualization of a convolutional neural network (CNN) – Adapted from [67]

The convolution and pooling layers act as feature extractors from the input image, while the fully connected layers act as a classifier.

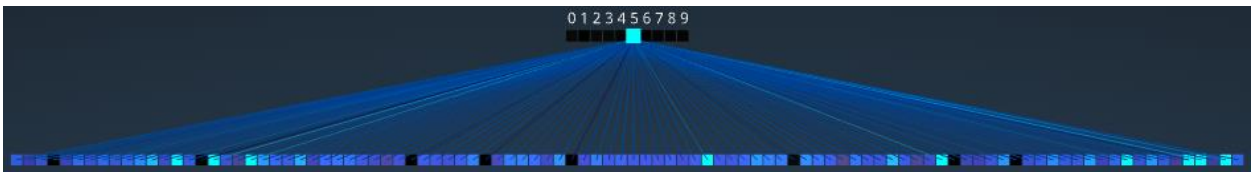

Figure 30: Correct classification from the fully connected layer – Source [67]

The network itself learns which features are useful, and which are not, by extracting the features from the image, and then use the features to classify the image.

Notice how in *Fig. 30*, each of the 10 neurons in the output layer are connected to all 100 neurons in the second fully connected layer (hence its name, fully connected). Also, in the output layer there is only a single bright neuron, “5”, which means that the network (based on its training) classified the hand-written digit correctly.

The neurons in the fully connected layer have their own weighted input and output as well, where a brighter neuron represents a higher value, and a darker neuron represents a lower value (which can be a negative value).

5.2 Model

A model is a detection method, with purpose of predicting bounding boxes of objects in a given image. A simple solution might be to just randomly add some bounding boxes to the image, and then check whether a certain object is within. However, this is not computationally efficient, and it does not perform well in terms of accuracy either.

The following sections discuss the models used for fish recognition.

5.2.1 From R-CNN to Faster R-CNN

Region-based convolutional neural network (R-CNN) built the foundation for the successors Fast R-CNN and Faster R-CNN (intuitive naming). Faster R-CNN is the model used in the results, but its predecessors play a vital role in how it functions and must be elaborated upon to grasp Faster R-CNN fully.

Please note that when the acronym FRCNN appears, it refers to Faster R-CNN, and is not to be confused with Fast R-CNN.

5.2.1.1 Region-Based Convolutional Neural Network

R-CNN executes a three-step process, as seen in *Fig. 31*:

1. Use selective search to create approximately 2000 bounding boxes or region proposals of varying sizes and positions in which objects in the input image will be scanned for.
2. The images in the region proposals are individually passed through a CNN and a support vector machine (SVM), which is a type of classifier, to determine what the object is.
3. Once the object has been classified, the bounding box is tightened by running it through a **linear regression** model, which outputs a continuous value from a linear combination of features.

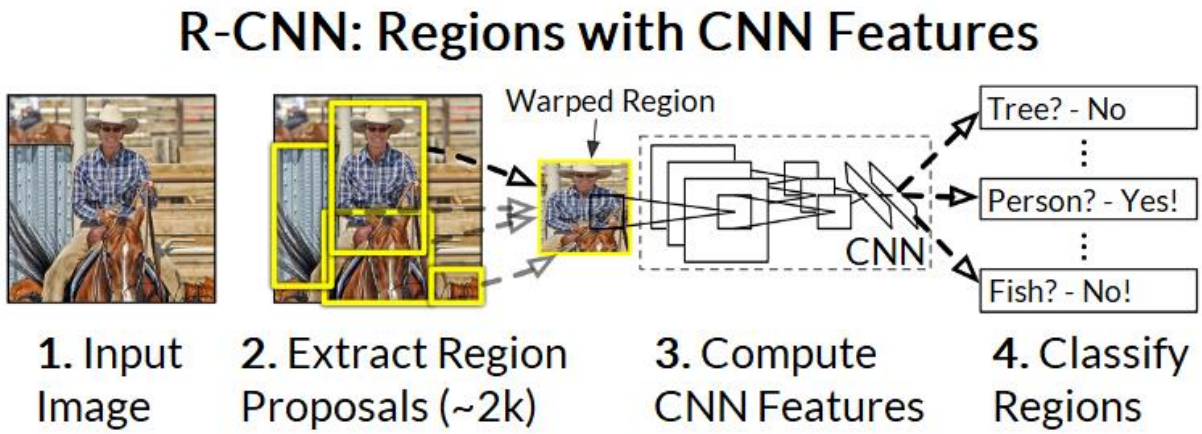

Figure 31: R-CNN step-by-step process – Adapted from [68]

While R-CNN was a tremendous breakthrough accuracy-wise, it resulted in a very slow model. First, it must pass every single region proposal (of which there are approximately 2000) through a CNN. Second, it must train three different models separately, one for generating image features (CNN), one for predicting the class (classifier), and one for tightening the bounding boxes (regression model).

Next up is Fast R-CNN, which solves these two problems quite well.

5.2.1.2 Fast Region-Based Convolutional Neural Network

The regions in R-CNN often overlapped, which caused duplicate CNN computations. Faster R-CNN handles the region of interest (RoI) issue by running the convolutional neural network once per image (instead of per region), and then uses a layer known as region of interest pooling (RoIPool).

RoIPool takes two inputs: a fixed-size feature map, and a $Nx5$ matrix representing a list of regions of interest, where *N* is the number of RoIs. *Fig. 32* shows a step-by-step process of RoIPool:

1. Region proposal is performed on the input (i.e. the activation values of a certain part in an image).
2. This creates (in this case) four pooling sections of the region proposal by dividing it in smaller sizes
3. Max pooling is then used to generate the output (i.e. select the highest value in each region).

Note that the size of the output is determined by the number of pooling sections.

The region proposal is generated from the $Nx5$ matrix, where the first column represents an image index, and the remaining four columns represent the top-left coordinates (x_{top}, y_{top}), and

the bottom-right coordinates (x_{bot}, and y_{bot}) of the region. For the region proposal in *Fig. 32*, these coordinates are (0, 3) and (6, 7).

Also, notice that the pooling sections do not need to have the same size. The values in *Fig. 32* are weights received from a previous layer and are completely arbitrary in this example. Normally, there would be multiple feature maps and multiple region proposals for each feature map.

Figure 32: RoIPool step-by-step process – Adapted from [69]

The second issue of combining the three separate models to a single network was handled by adding two parallel layers, one which replaces the SVM classifier with a **softmax** layer to produce the classification, as well as a linear regression layer to output the bounding box coordinates. Softmax is a function which performs multi-class classification and transforms the input values to the range $[0, 1]$, with a sum of 1, representing a true probability distribution.

Fig. 33 summarizes and visualizes the structure of R-CNN compared to Fast R-CNN, where the changes and new features are marked in orange.

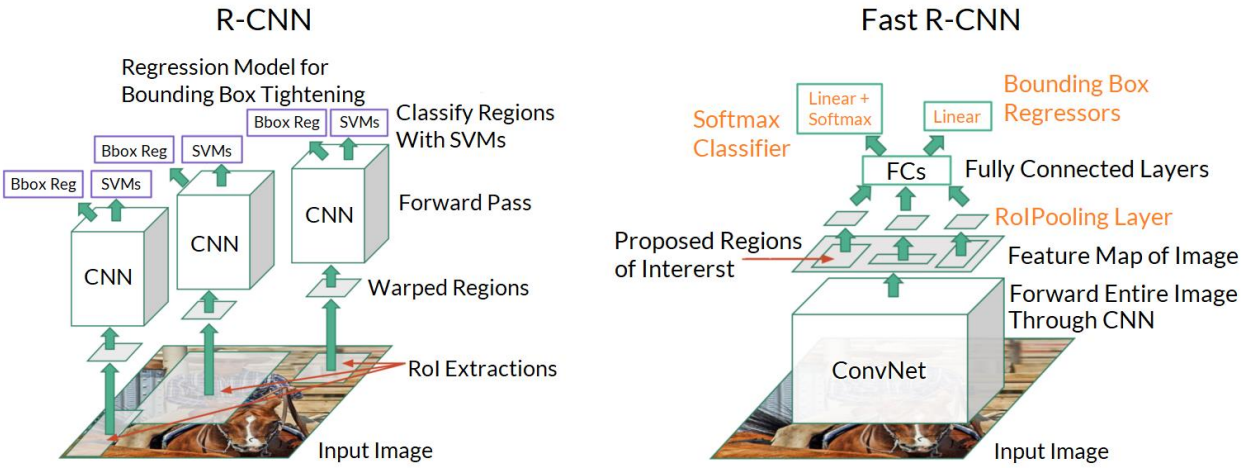

Figure 33: R-CNN compared to Fast R-CNN – Adapted from [70]

5.2.1.3 Faster Region-Based Convolutional Neural Network

The third iteration in the R-CNN series is Faster R-CNN, and this deals with yet another bottleneck, namely the region proposer.

Instead of running the selective search algorithm, the idea is to reuse the CNN results for region proposals, since they are already calculated with the forward pass of the CNN. Essentially, Faster R-CNN = Fast R-CNN + region proposal network (RPN).

RPN works as follows:

1. A sliding window moves across the feature map and reduces its dimension.
2. At each position the sliding window generates multiple region proposals based on a fixed number of anchor boxes (i.e. default bounding boxes).
3. Each of the region proposals consist of the bounding box's four coordinates, as well as an object score for that region, indicating whether it contains an object.

The region proposals are then fed to a Fast R-CNN. An overview of the architecture can be seen in *Fig. 34*.

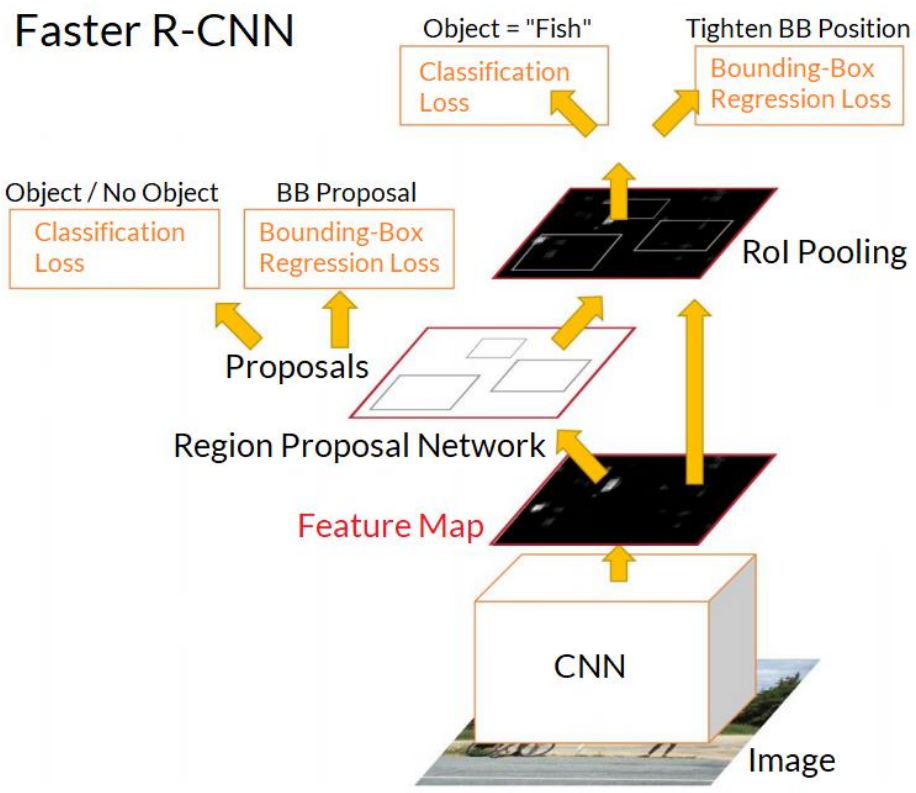

Figure 34: Overview of Faster R-CNN architecture – Adapted from [70]

While the speed and accuracy changed from the original R-CNN, the design is still the same: propose object regions and classify them.

While these improvements seem good in theory, a comparison of each model is required to check whether the improvements hold up. *Table 6* compares each model in terms of training time, testing rate, and VOC07 mAP (i.e. the network's mean average precision when trained and tested on the Pascal Visual Object Classes 2007 dataset). The VOC07 dataset consists of 24 640 labels in 9963 images.

Model	Training time (hours)	Testing rate (sec/image)	VOC07 mAP
R-CNN	84.0	47.0	66.0
Fast R-CNN	9.5	0.32	66.9
Faster R-CNN	4.4 - 17.7*	0.20	69.9

Table 6: Comparison of R-CNN models – Source [71], [72]

Interestingly, the accuracy did not increase as much as one might have expected. The testing rate, however, diminished significantly when the duplicate CNN issue from R-CNN was dealt with, as well as parallelizing the layers.

*There is no reported training time for Faster R-CNN in the official papers, and while one might estimate it by multiplying the number of iterations used in the paper, one with 80 000, and one with 320 000 with the testing rate of 200 milliseconds, it is not an official number.

Please refer to Appendix C (C.2 Faster Region-Based Convolutional Neural Network (Faster R-CNN)) for a graph illustrating the specific FRCNN model for this thesis.

5.2.2 Region-Based Fully Convolutional Network

Region-based fully convolutional networks (R-FCN) are all about maximizing shared computation across every single output. Sharing 100% of the computations, however, introduces a unique problem. Regardless of where the object (e.g. a fish) is in the image, the image should be classified as a fish (i.e. location invariance), while simultaneously drawing the bounding box at the fish location (i.e. location variance). However, all computations are shared across 100% of the network, thus a certain compromise must be made between the two invariances.

R-FCN tackles this issue by using position-sensitive score maps, where each map represents one relative position of one object class. In other words, each convolutional feature map is trained to recognize certain parts of each object. For instance, one map might activate when it sees the tail of a fish, and another for the head of a fish. If enough of the sub-region proposals indicate that they are correctly matching a part of a fish, then the region (of multiple sub-regions) is classified as a fish.

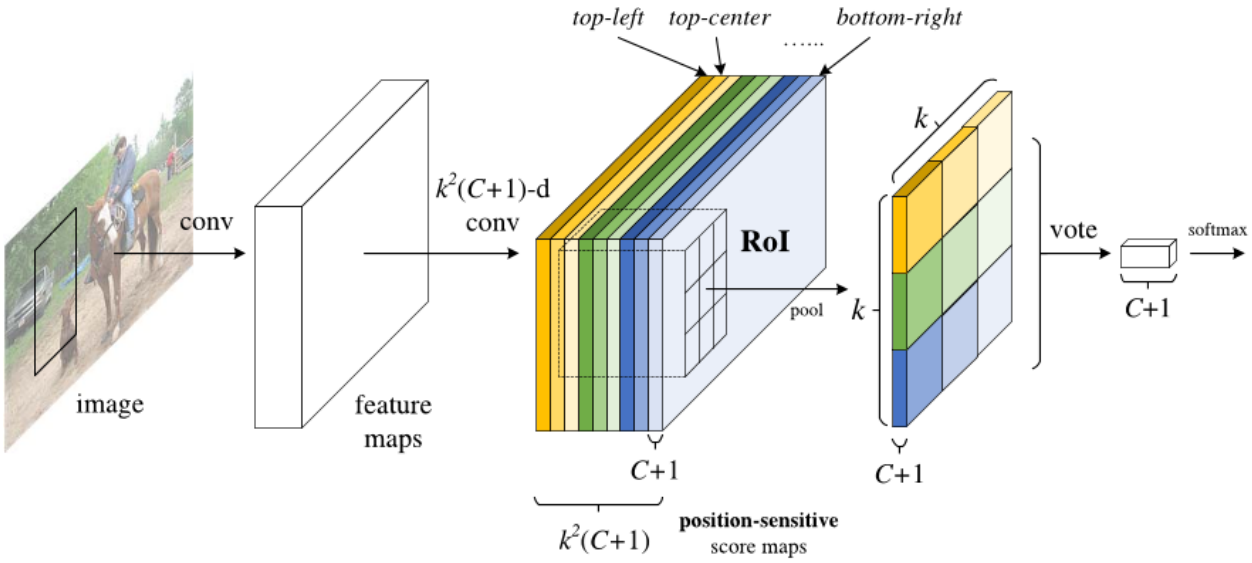

Figure 35: R-FCN architecture – Source [70, 73]

Fig. 35 illustrates the key idea of R-FCN. Here, there are $k * k = 3 * 3 = 9$ position-sensitive score maps for each class *C*, plus 1 for the background, resulting in a $k^2(C+1)$ channel output. Each score map focuses on a specific case of a class. In the figure, the first score map corresponds with the top-left region of that class. Note that *-d* is not a variable but refers to the dimensions.

Following is a step-by-step explanation of the R-FCN architecture in *Fig. 35*:

1. A convolutional neural network is run over the input image.
2. A fully convolutional layer is added to generate a score bank of the $k^2(C+1)$ position-sensitive score maps.
3. Regions of interests (RoIs) are generated from a fully convolutional region proposal network.
4. Each RoI is divided into the k^2 same sub-regions as the score maps.
5. Each sub-region is compared with the score bank to check whether the sub-region matches a corresponding position of an object by averaging the values in the RoI. This step is repeated for each class.
6. Once all the k^2 sub-regions have an object matching value for each class, the sub-regions are averaged to get a single score per class.
7. The RoI is classified with a softmax over the remaining $C+1$ dimensional vector.

With this algorithm, R-FCN can simultaneously address location invariance by having each region proposal refer to its own score bank, and location variance by proposing different region proposals, all while being fully convolutional (i.e. all computation is shared throughout the network).

Please refer to Appendix C (C.3 Region-Based Fully Convolutional Neural Network (R-FCN)) for a graph illustrating the specific R-FCN model for this thesis.

5.2.3 Single Shot MultiBox Detector

Single shot multibox detector (SSD) is rather different than the previous models. Instead of generating the regions of interest and classifying the regions separately, SSD does it simultaneously, in a "single shot".

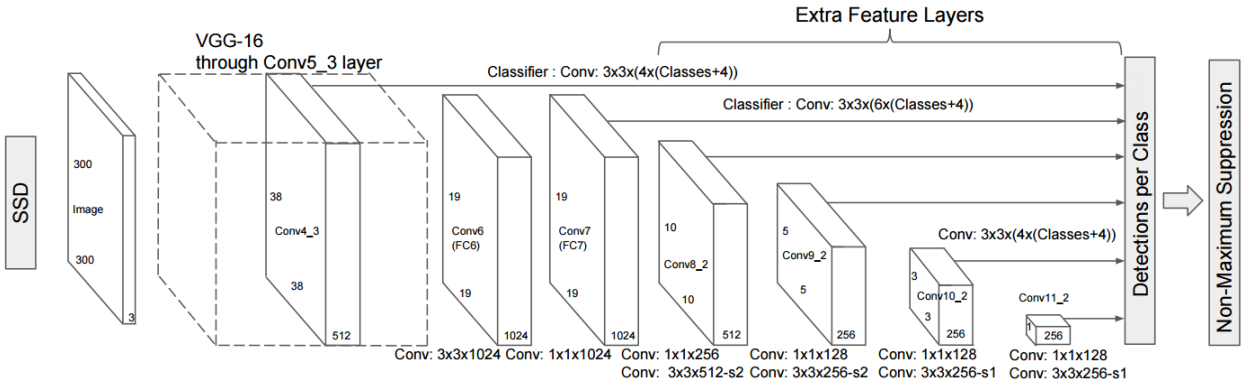

Figure 36: SSD architecture – Adapted from [74]

The step-by-step algorithm for SSD is:

1. The image passes through a series of convolutional layers, each providing a feature map in varying dimensions, as seen in *Fig. 36*.
2. A 3x3 convolutional filter is applied to each location of each feature map to evaluate a set of default bounding boxes (equivalent to Faster R-CNN's anchor boxes).
3. For each bounding box, predict its offset and class probabilities, simultaneously.
4. While training, the ground truth box is matched with the predicted boxes based on **intersection over union** (IoU), where the predictions that have an IoU with truth greater than 0.5 are labeled as positive. IoU is a classification threshold to separate the positive class (the wanted object) from the negative class (a non-relevant object).

In *Fig. 36* there is an additional layer after the objects have been detected, which is bound to raise a few questions. Recall that Faster R-CNN and R-FCN required a certain degree of confidence regarding the objects due to their region proposal network (RPN). SSD has left this filtering step out, which results in a much larger number of bounding boxes, where most of them are negative examples (IoU less than 0.5).

This is obviously not a good solution, which is why SSD takes advantage of non-maximum suppression (NMS) to group together highly-overlapping bounding boxes. The boxes are not

merged together as one, but instead, they are all but one discarded. Only the bounding box with the highest confidence is kept, i.e. the bounding box more likely to accurately contain the object, in the range $[0, 1]$, where 1 signals 100% confidence.

Additionally, SSD uses hard negative mining. The classifier requires both positive examples (i.e. of the object) and negative examples (i.e. not of the object) in form of bounding boxes. The training data contains all positive examples, while the negative examples are created arbitrarily. If a negative bounding box do not overlap with any of the positives, then it is stored as a negative example.

This could, however, potentially lead the classifier to return a lot of false positives (a fish is detected where there are none). Using a hard negative can fix this, where during training, only a subset of the negative examples with the highest confidence loss (potential false positives) are used at each iteration of training. The ratio between negatives and positives is at most 3:1.

Please refer to Appendix C (C.1 Single Shot MultiBox Detector (SSD)) for the specific network model for this thesis.

5.3 Classifier

The classifier's purpose is providing a label for the input image. To provide a label, the input image must pass through various layers with various operations, which transforms the raw data into a sequence of feature vectors, and finally, a label.

5.3.1 Residual Net

Residual Net or ResNet was created due to the observation that as the networks grow deeper, their performance degrades [30]. They did so even if the additional layers were an exact copy of the previous layers, and the final layer performed an identity mapping (a function where the input is equal to the output, i.e. $f(x) = x$). In theory, the network should at least have the same accuracy as its shallower counterpart, but this was not the case.

The reason for this behavior is that direct mappings are hard to learn, and frequently used techniques such as ReLU, **dropout**, and batch-**normalization** (to name a few) are not good enough on their own for end-to-end training. Dropout is a regularization technique which ignores/removes neurons during training to reduce overfitting by reducing the complexity of the model. Batch-normalization converts a range of values to a standard range. Therefore, instead of learning the underlying mapping from x to $H(x)$, it should learn the difference, or residual between the two. If the residual is $F(x) = H(x) - x$, then the network attempts to learn the difference, i.e. $F(x) + x$.

The reason why this solves the problem of deep network degradation, is that instead of learning an identity transformation from scratch, one can start with a reference point to learn from, which greatly simplifies the construction of identity layers.

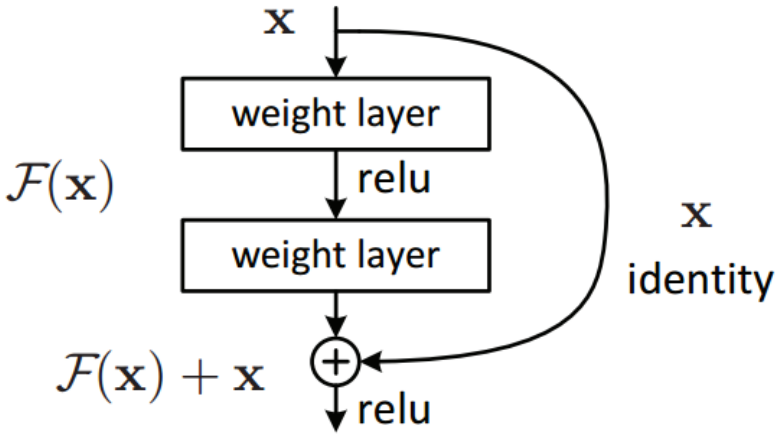

Figure 37: ResNet building block – Source [75]

Fig. 37 illustrates the residual block, in which the input is re-routed, making the next layer learn the previous layer's concepts in addition to the input of that previous layer. The addition is performed element-wise, and since the "shortcut" is adding the input block to the output block, one can for instance use padding to create matching dimensions if needed.

This routing technique eliminated the problem of vanishing gradients, in which the error function's gradient signals decreased exponentially as they backpropagated to earlier layers [76].

For the thesis, ResNet 101 and ResNet 152 were used, i.e. a 101-layer and 152-layer version of ResNet.

5.3.2 Inception

While ResNet focused on how one can go deeper, Inception (referred to as GoogLeNet, at least for the first version) focuses on how one can scale up neural networks while keeping the computational cost the same.

The building block in Inception is the Inception module as seen in *Fig. 38*, and is the result of two key insights:

1. Let the model choose which information is useful.
2. Use $1x1$ convolutions to perform dimensionality reduction.

The first insight deals with layer operations. Traditionally, each layer extracts information from the previous layer to transform the input data, but each type of layer produces different outputs, hence one cannot be sure which of these outputs are more important than the other.

This is where the wider aspect comes in play. Instead of feeding all the data layer-by-layer, one can feed the same input map in parallel, and then have the next layer decide how and which of the transformations should be used.

While computing a single $5x5$ convolution is an expensive operation, stacking multiple of these types of layers side by side achieves the opposite of the goal of not increasing the computational cost. Any constant increase in the number of filters results in a quadratic increase of computation, since by adding *N* filters, one must convolve over $N * M$ additional maps, and the module in *Fig. 38* quadrupled the number of filters. This is a major bottleneck and paves the way to the second insight of dimensionality reduction.

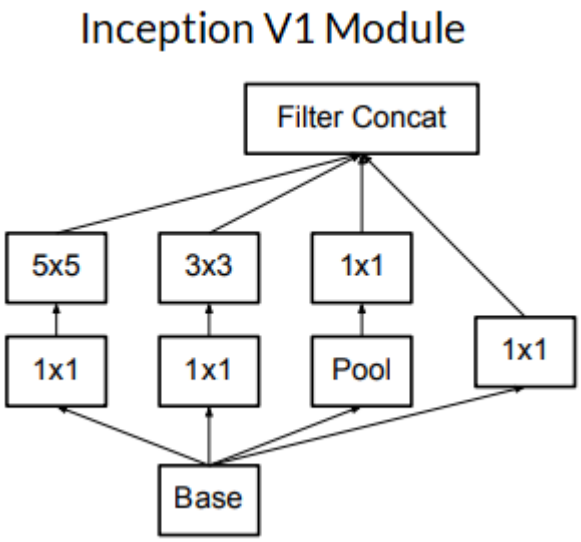

Figure 38: Inception V1 module – Source

To solve the bottleneck, $1x1$ convolutions were used to filter the depths of the output. A $1x1$ convolution looks at one value at a time, but when using for instance 20 of these filters, one can reduce an input size of for instance $64x64x100$ (with 100 feature maps) to $64x64x20$.

In the first Inception architecture, nine of these modules were used in combination with other types of single layers, such as convolution, max pool, average pool, dropout, linear, and softmax.

It is quite important to know the intricacies of the first version to grasp how the second, third, and fourth version functions, which are the versions used in the results.

5.3.2.1 Second and Third Iteration

The second iteration (Inception V2) modified the original module in *Fig. 38* with *Fig. 39* type A, namely replacing the $5x5$ convolution with two $3x3$ convolutions to promote faster learning with little to no adverse effects. *Fig. 39* type B uses a $1xn$ to $nx1$ layer structure, with purpose of dramatically decreasing the computational cost as *n* increases. Type C's purpose is to promote high dimensional representations, which are easier to process, thus resulting in faster training.

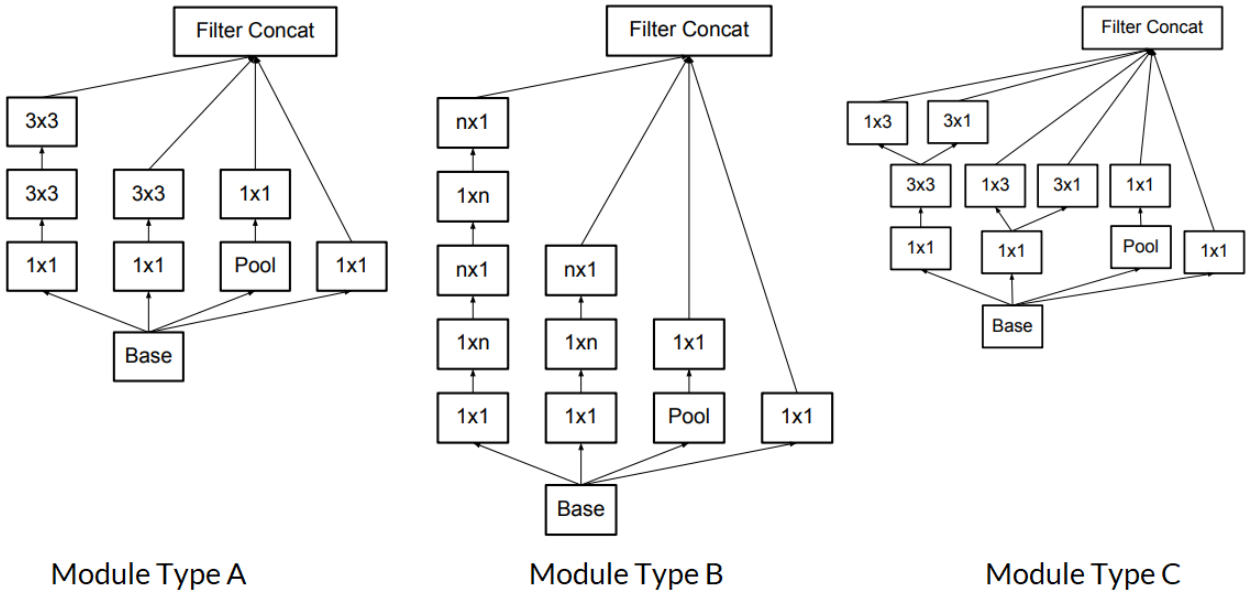

Figure 39: The three types of modules used in Inception V2 – Adapted from [77]

Inception V3 is based on the same modules as Inception V2 but has a slightly altered training algorithm. It uses:

- Root Mean Square Propagation (RMSProp): a gradient descent optimization method which divides the gradient by a running average of its recent magnitude.
- Label smoothing: to slightly decrease higher loss target values, and slightly increase values close to zero.
- Batch-normalization (BN): on the fully connected layer. BN helps with relaxing the weights to each layer to avoid shifting distributions in activations as the earlier layers' parameters are updated. Essentially making the range of the inputs more consistent for each layer.

5.3.2.2 Combining Inception and ResNet

Inception ResNet V2 is a costly hybrid version, and while its architecture is simpler than Inception V3, it has an increased size. The Inception modules in Inception ResNet V2 are also cheaper than the original Inception V1, in part due to each $1x1$ convolution (with no activation) after each Inception module, which scales up the dimensionality to match the depth of the input.

The schema can be observed in *Fig. 40*, which presents the overall architecture. Each block in the schema corresponds to a type of module, except for the softmax, dropout, average pooling, and input layers. The classifier's relating modules can be seen in *Fig. 41* and *Fig. 42*.

The architecture has changed quite drastically, but still carries over some key ideas from previous iterations, for instance the $1xn$ to $nx1$ layers in *Fig. 41* and *Fig. 42*, from the module types in *Fig. 39*.

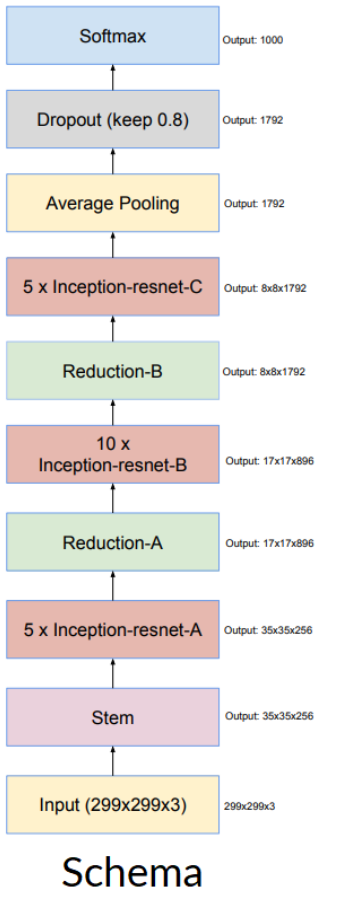

Figure 40: Inception ResNet V2 schema - Adapted from [59]

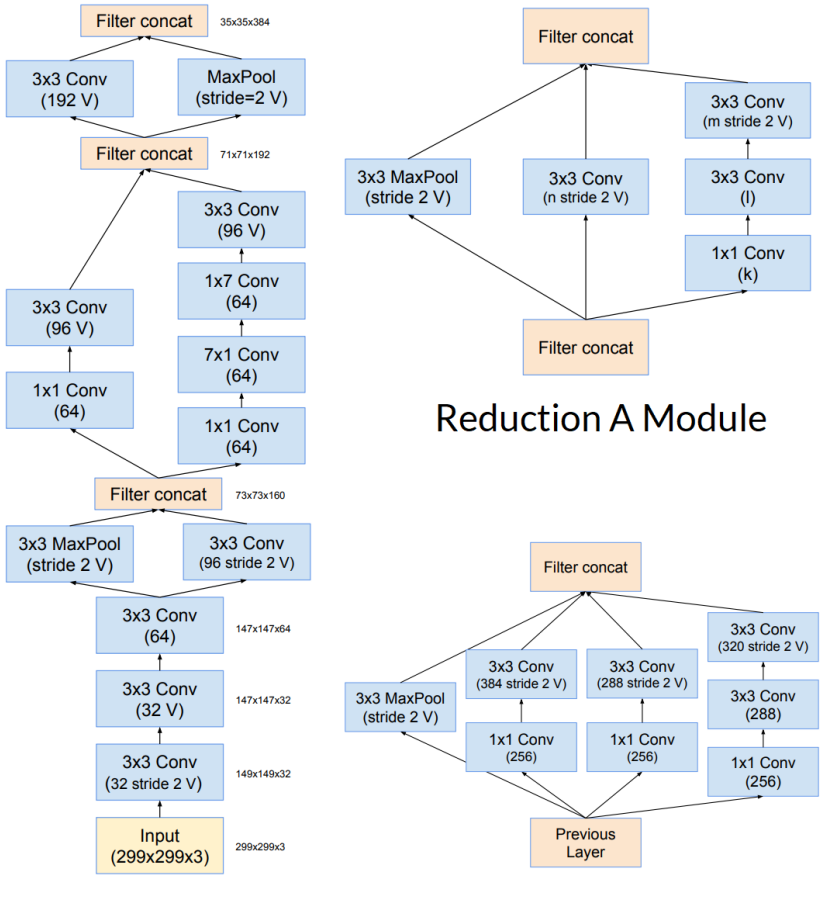

Figure 41: Inception ResNet V2 input and reduction modules – Adapted from [59]

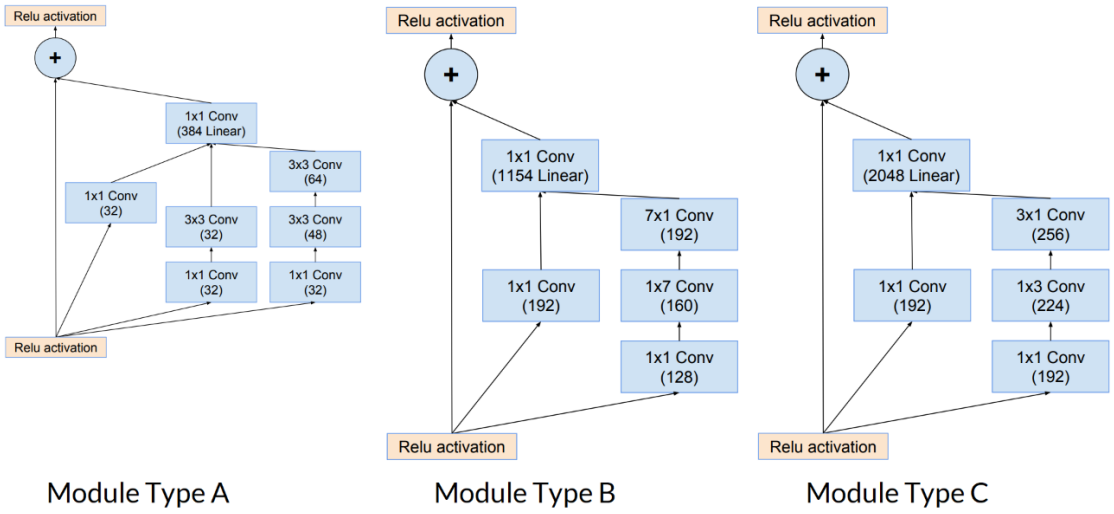

Figure 42: Inception ResNet V2 modules A, B, and C – Adapted from [59]

5.3.3 MobileNet

MobileNet was designed with speed as principle, where it should be able to run on a computationally limited platform with as small as possible loss of accuracy.

Fig. 43 illustrates convolutional filters with spatial dimension of kernel D_K, M input channels, and N output channels.

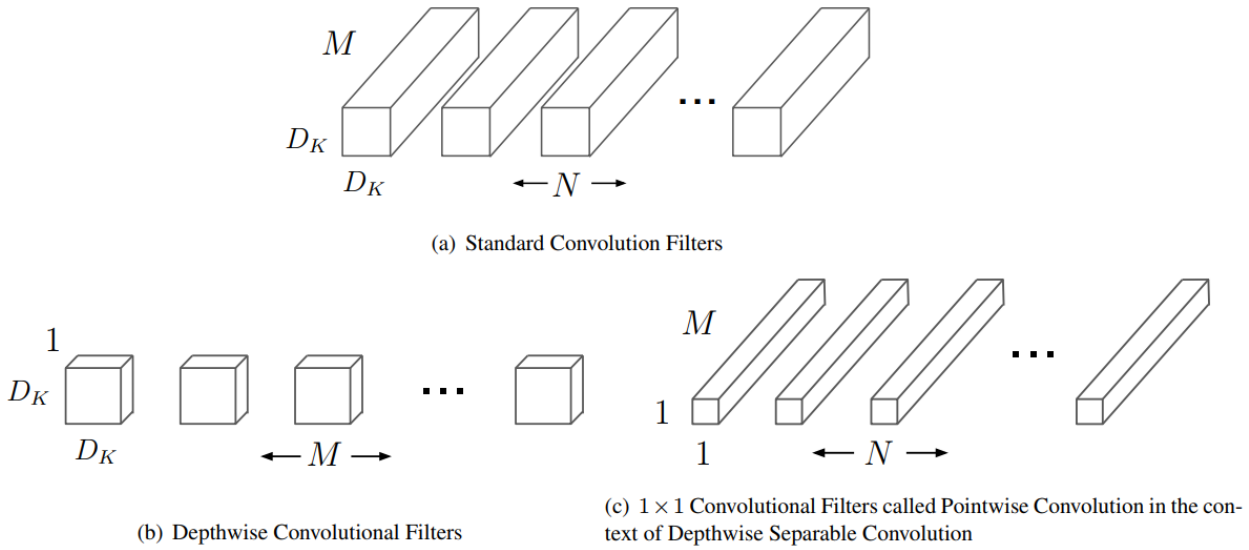

Figure 43: Standard convolution vs depthwise separable convolution – Adapted from [78]

MobileNet is based on depthwise separable convolutions, aiming to drastically reduce the number of parameters and multiply-add combined (MAC) operations. A MAC operation is the process of multiplying a set of numbers and adding them to an already existing number.

As seen in *Table 7*, both the number of MAC operations and parameters have been drastically reduced by separating the single layer into two layers, one for filtering, and one for combining.

In total, MobileNet V1 has 28 layers (counting depthwise and pointwise convolutions as separate layers), while MobileNet V2 has 32 fully convolutional layers, followed by 19 residual bottleneck layers.

For the equations in *Table 7*, the spatial width and height of a square input feature map is referred to as D_F. The filter size $D_K * D_K$ (*K* is short for kernel but is the same as a filter) and feature map size $D_F * D_F$ is used in combination with the *M* input and *N* output channels to compute cost and parameters for each type of convolution.

Layer specifications: D_K = 3, M = 512, N = 512, D_F = 14	
Standard convolution	**Depthwise separable convolution**
1. Every input channel is traversed by each output channel using a $3x3$ filter. **2.** Producing 512 feature maps $(16 * 32)$. **3.** Merge one feature map out of every input channel.	**1.** Each input channel is traversed with one $3x3$ kernel each. **2.** Producing 16 feature maps $(16 * 1)$. **3.** Traverse the feature maps with the output channel using $1x1$ convolution and merge them.
Computational cost (MAC operations): $D_K * D_K * M * N * D_F * D_F$ $= 3 * 3 * 512 * 512 * 14 * 14$ $= 462{,}422{,}016$	**Computational cost (MAC operations)**: $D_K * D_K * M * D_F * D_F + M * N * D_F * D_F$ $= 3 * 3 * 512 * 14 * 14 + 512 * 512 * 14 * 14$ $= 52{,}283{,}392$
Number of parameters: $D_K * D_K * M * N$ $= 3 * 3 * 512 * 512$ $= 2{,}359{,}296$	**Number of parameters**: $D_K * D_K * M * +M * N$ $= 3 * 3 * 512 + 512 * 512$ $= 266{,}752$

Table 7: Calculating the number of MAC operations and parameters in SC and DSC

The data in *Table 7* shows the process of each type of convolution, and the drastic reduction in MAC operations and parameters for depthwise separable convolutions, which require 8 to 9 times less computation than standard convolutions with only a slight decrease in accuracy [60].

The architecture for MobileNet V1 and MobileNet V2 is displayed in *Table 8*, and *Table 9*, respectively. MobileNet V2's inverted residual with linear bottleneck module is displayed in *Table 10*. The bottleneck layer has its name because it reduces the amount of data flowing through the network.

For *Table 8* and *Table 9*, there are *c* output channels, *n* layers, stride of size *s*, and an expansion ratio *t*. The bottleneck module in *Table 10* defines the bottleneck operator in *Table 9*, with height *h*, width *w*, *k* and *k'* channels, stride *s*, and expansion ratio *t*. The expansion ratio is the ratio between the input and the output depths, and is used to increase depth of feature maps, as opposed to a linear bottleneck which reduces it.

Input	Operator	c	n	s
224^2 x 3	conv2d 3x3	32	1	2
112^2 x 32	conv2d dw 3x3	32	1	1
112^2 x 32	conv2d 1x1	64	1	1
112^2 x 64	conv2d dw 3x3	64	1	2
56^2 x 64	conv2d 1x1	128	1	1
56^2 x 128	conv2d dw 3x3	128	1	1
56^2 x 128	conv2d 1x1	128	1	1
56^2 x 128	conv2d dw 3x3	128	1	2
28^2 x 128	conv2d 1x1	256	1	1
28^2 x 256	conv2d dw 3x3	256	1	1
28^2 x 256	conv2d 1x1	256	1	1
28^2 x 256	conv2d dw 3x3	256	1	2
14^2 x 256	conv2d 1x1	512	1	1
14^2 x 512	conv2d dw 3x3	512	5	1
14^2 x 512	conv2d 1x1	512	5	1
14^2 x 512	conv2d dw 3x3	512	1	2
7^2 x 512	conv2d 1x1	1024	1	1
7^2 x 1024	conv2d dw 3x3	1024	1	2
7^2 x 1024	conv2d 1x1	1024	1	1
7^2 x 1024	avgpool 7x7	1024	1	1
1^2 x 1024	FC	1000	1	1
1 x 1 x 1000	Softmax / Classifier	k	-	1

Table 8: MobileNet V1 architecture – Adapted from [78]

Input	Operator	t	c	n	s
224^2 x 3	conv2d 3x3	-	32	1	2
112^2 x 32	bottleneck	1	16	1	1
112^2 x 16	bottleneck	6	24	2	2
56^2 x 24	bottleneck	6	32	3	2
28^2 x 32	bottleneck	6	64	4	2
14^2 x 64	bottleneck	6	96	3	1
14^2 x 96	bottleneck	6	160	3	2
7^2 x 160	bottleneck	6	320	1	1
7^2 x 320	conv2d 1x1	-	1280	1	1
1 x 1 x 1280	avgpool 7x7	-	1280	1	-
1 x 1 x 1280	conv2d 1x1	-	k	-	

Table 9: MobileNet V2 architecture – Adapted from [60]

Input	Operator	Output
$h\ x\ w\ x\ k$	conv2d 1x1, ReLU6	$h\ x\ w\ x\ (tk)$
$h\ x\ w\ x\ tk$	conv2d dw 3x3 s=s, ReLU6	$\frac{h}{s} x \frac{w}{s} x\ (tk)$
$\frac{h}{s} x \frac{w}{s} x\ tk$	linear conv2d 1x1	$\frac{h}{s} x \frac{w}{s} x\ k'$

Table 10: MobileNet V2 bottleneck module – Adapted from [60]

The bottleneck module in *Table 10* takes as input a low-dimensional compressed representation, expands and filters it with a depthwise convolution (dw), and projects its features back to a low-dimensional representation with a linear convolution.

In MobileNet V1, all layers are followed by batch-normalization and ReLU (see *Fig. 44*), except the final fully connected layer. MobileNet V2 extends the first version by adding an additional $1x1$ convolution layer (green block).

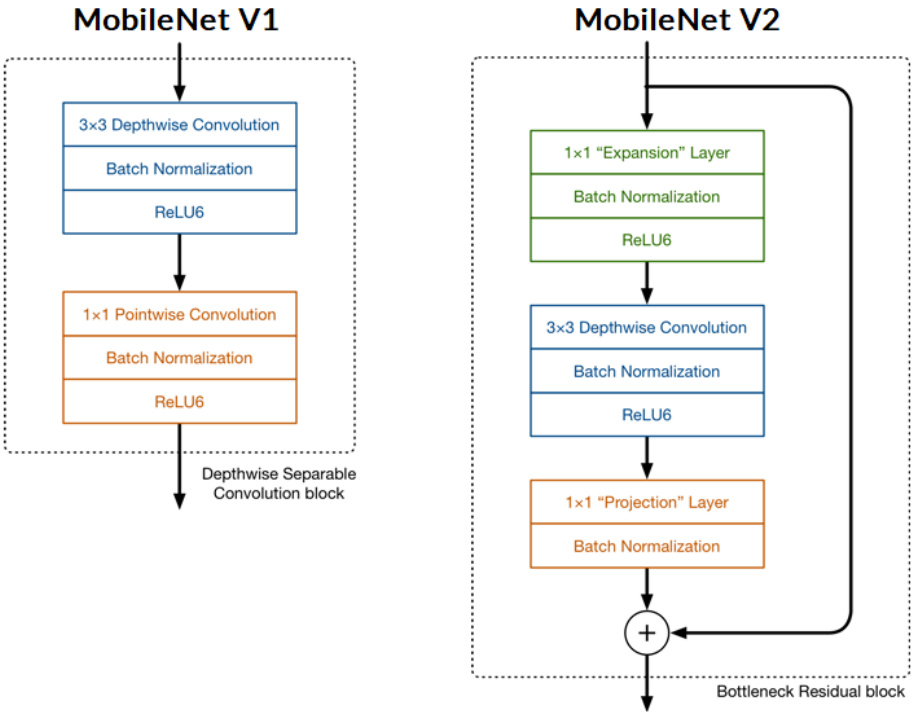

Figure 44: Comparison of the blocks in MobileNet V1 and MobileNet V2 – Adapted from [79]

5.3.4 Neural Architecture Search (NAS)

Neural Architecture Search (NAS) is an architecture in which a controller recurrent neural network (RNN) samples child networks with varying architectures, where a feedback loop is gradually increasing the classifier's accuracy over time. RNNs handle sequential information, where one task is performed for each element in a sequence, and the element's result (i.e. the output) is carried over to the next element.

Fig. 45 illustrates the NASNet-A classifier's architecture, where the input (white) is the hidden state from previous actions, and the output (pink) is the result of concatenating the branches. A single block is defined as a combination of two primitive operations (yellow) and one combination operation (green).

The cells are convolutional, where a normal cell returns a feature map of same size, while a reduction cell reduces it by a factor of two.

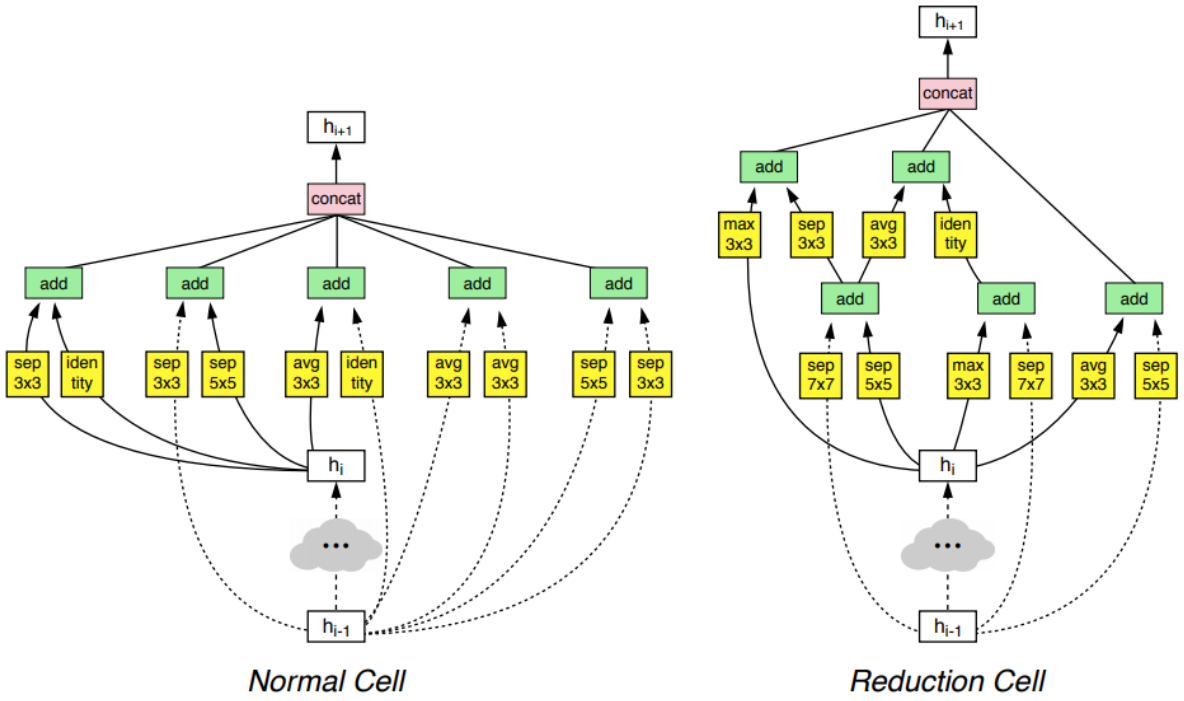

Figure 45: NASNet-A architecture with B = 5 blocks – Source [61]

Fig. 46 shows a controller model architecture recursively constructing one block of a convolutional cell. There are *B* blocks, and each block has five prediction steps made by five different softmax classifiers, depending on previous choices. Due to there being two types of cells (normal and reduction), there are in fact $2 \; x \; 5B$ predictions in total.

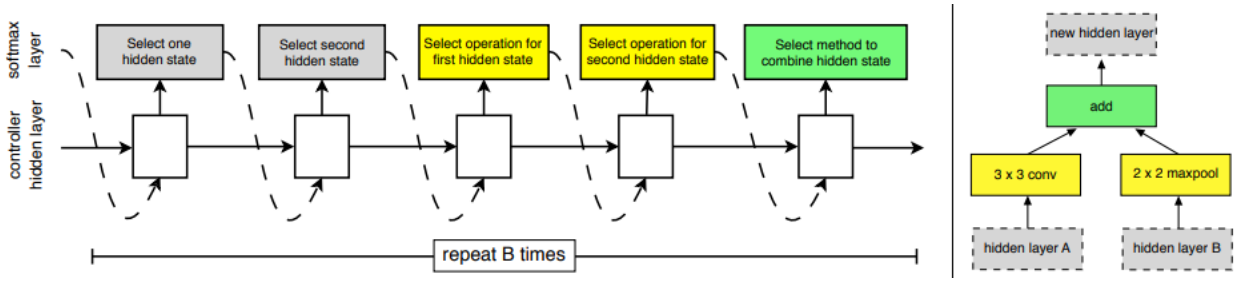

Figure 46: Controller model architecture (left) constructing one block (right) – Adapted from [61]

Once the hidden states and the operations to apply to the hidden states are selected, the outputs of the operations (yellow) must be combined by either element-wise addition or concatenation. When all the unused hidden states are concatenated together in depth, the final cell is outputted.

The controller chooses the operations to apply to the hidden states, and are either:

- identity
- 1x7 then 7x1 convolution
- 3x3 average pooling
- 5x5 max pooling
- 1x1 convolution
- 3x3 depthwise separable convolution
- 7x7 depthwise separable convolution
- 1x3 then 3x1 convolution
- 3x3 dilated convolution
- 3x3 max pooling
- 7x7 max pooling
- 3x3 convolution
- 5x5 depthwise separable convolution

5.4 Classifier Comparisons

Table 11 presents the mentioned classifiers with respect to their input image size, multiply-add combinations (MAC), number of parameters, and top 1 accuracy (%). The accuracy is based on the ImageNet 2012 dataset, which consists of 1000 classes, 1 280 000 training images, and 150 000 testing images [30].

Classifier	Image Size	MAC	Parameters	Top 1 Accuracy (%)
MobileNet V1	320 x 320	575 M	4.2 M	70.6
MobileNet V2	320 x 320	300 M	3.4 M	72.0
Inception V2	224 x 224	1.94 B	11.2 M	74.8
ResNet 101	224 x 224	7.6 B	44.6 M	76.4
ResNet 152	224 x 224	11.3 B	60.3 M	77.0
Inception V3	299 x 299	5.75 B	23.8 M	78.8
Inception ResNet V2	299 x 299	13.2 B	55.8 M	80.1
NASNet-A	331 x 331	23.8 B	88.9 M	82.7

Table 11: Comparison of architectures – Source [30], [60], [61], [80]

Top 1 accuracy means that the output with the highest confidence is equal to the actual output. For instance, imagine the output is either of the three rabbit classes: [volcano, riverine, cottontail]. If the input is of a volcano rabbit, and the network's output is: [0.48, 0.13, 0.39], then the network's output is counted as accurate.

Note that each of the metrics in *Table 11* are dependent on a lot of factors, and their value might vary slightly depending on their exact setup and dataset. Please refer to [30, 60, 61, 80] for their complete implementation details.

6 Evaluation and Results

20 networks were trained and tested in this thesis, and the results of each single network is presented. Some of the earlier networks had to be re-trained and re-tested, as the initial dataset only contained labels of fish clearly visible in the foreground. This resulted in poor precision, thus the dataset had to be expanded, where labels of fish farther in the background were included as well.

Three SSD networks were tested using a slightly smaller dataset, consisting of 18 363 labels. The remainder of the networks were trained after a request for additional data was granted by Steinsvik AS, which added another 1706 labels to the dataset, resulting in a total of 20 069 labels.

Due to memory constraints it was not possible to test and train the networks simultaneously (i.e. for networks where the batch size was required to be set to one), and it was only possible to test a single checkpoint file at a time in TensorFlow (the last checkpoint). Therefore, the source code had to be modified and extended to allow for a feature where one specified the checkpoint to evaluate from, and which evaluated all remaining checkpoints in one run. This feature has been submitted as a pull request by the author to the TensorFlow repository [81] and can be can be enabled by passing a flag at runtime.

Checkpoints are versions of the network created during training and are dependent on the code that created the network. TensorFlow saves variables in these binary checkpoint files which map variable names to tensor values. By frequently storing the checkpoint files on disk one can at a later point evaluate all of them individually to find which provides the best mean average precision (mAP). Ideally, one would like to store checkpoints frequently (for instance every ten minutes), but this requires a lot of space, while simultaneously increasing the time spent evaluating, thus one must resort to some middle-ground dependent on the project's needs.

Mean average precision (mAP) is an extension of average precision (AP) in which each class' average precision is computed, and then by taking the mean of the average class precision one ends up with the mean average precision. Note that there is only a single class used in this thesis, thus it is not necessary to calculate the mean of the average precision (mAP), just the average precision. However, for consistency, mAP will be used.

The precision is calculated by the following formula:

$$Precision = \frac{True\ Positive}{True\ Positive\ +\ False\ Positive}.$$

A detection (a positive) is either correct (true) or not (false), thus a false positive is an example in which the network mistakenly predicts the positive class (e.g. predicting fish when it was not a fish).

Intersection over union (IoU) measures the overlap between the predicted region and the ground truth region (i.e. the region with the actual answer), illustrated in *Fig. 47*:

$$IoU = \frac{Area\ of\ Overlap}{Are\ of\ Union}.$$

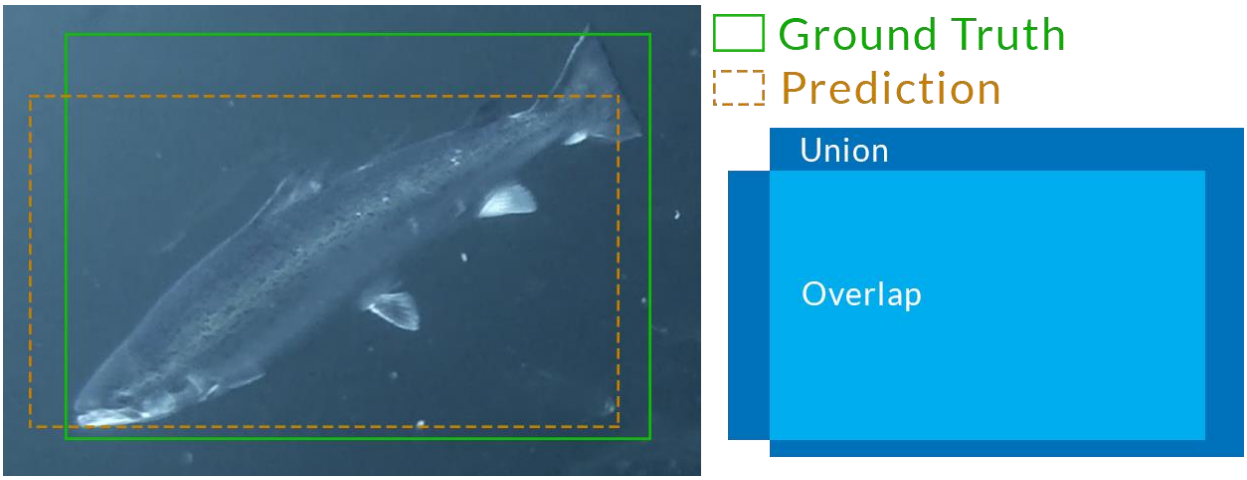

Figure 47: Intersection over Union (IoU)

The metric used in the results are mAP@0.5IoU, i.e. the detection is a true positive if the IoU is greater than 0.5, otherwise it is a false positive.

Of the 20 networks evaluated below, the total training time clocks in at 720 hours, not accounting for the earlier networks that had to be re-trained and re-tested.

Since the testing was a separate process, this too required many hours, but unfortunately, the hours spent testing were not recorded for most of the networks throughout the thesis. One example, however, is of the top-ranking network (SSD Inception V2 @1), which required 17 hours of testing for 22 hours of training. This is, however, an anomaly, since the checkpoints were stored every 10 minutes, as opposed to every hour which is the standard for 17 of the networks.

Due to the vast amount of data representing all scalars, image results, graphs, etc. in TensorBoard (a tool for visualizing the networks' data), it takes on average 4 hours and 30 minutes to generate all 20 networks' accumulated data. Both the training and testing aspect has their own set of data that must be rendered independently. When only a single network is passed to TensorBoard, however, its data is usually loaded in a matter of minutes. It is

recommended to pass only a subset of the networks to TensorBoard at once, as it tended to crash when handling large amounts of data simultaneously.

Add it all up and it is safe to say that only training, testing, and graphing the 20 networks required more than 900 hours combined using the hardware mentioned in 6.1 Hardware.

6.1 Hardware

Deep neural networks are computationally expensive to train, and thus require hardware with enough memory to store the computations in memory during the training process. More specifically, the memory in neural networks stores input data, activations, and weight parameters for each forward pass to be used when calculating the error gradients during the backward pass.

The most important components in deep learning are the graphics processing unit (GPU) and the central processing unit (CPU), as these are used to perform the necessary computations. In the following sections, when CPU or GPU is mentioned, please refer to those listed in *Table 12*.

Central Processing Unit	Intel Core i7-6700K @ 4.0 GHz
Graphics Processing Unit	NVIDIA GeForce GTX 1080 Ti 11 GB VRAM
Random Access Memory	2x16 GB G.Skill DDR4 @ 3200 MHz
Storage	Samsung 950 Pro M.2 512 GB Seagate Ironwolf 4 TB 3.5" NAS HDD

Table 12: Hardware components

6.1.1 CPU vs GPU

This section is a short discussion to emphasize that literally thousands of hours can be wasted if insufficient hardware is used.

The CPU is designed to do computation rapidly on small amounts of data, but it has problems when the amount of data is large, such as when multiplying matrices of hundreds of thousands of numbers. Since deep learning is mostly comprised of matrix operations, a better alternative would be to use GPUs, which have a large memory bandwidth and can handle a lot of parallel computations.

Table 13 showcases the difference between the GPU and CPU training time with batch size of 1 for Faster R-CNN ResNet 152 and SSD Inception V2. A step is a forward and backward evaluation of one batch.

	FRCNN ResNet 152 (step/sec)		SSD Inception V2 (step/sec)	
CPU	19.252	1x	3.708	1x
GPU	0.599	32.1x	0.742	5.0x

Table 13: GPU vs CPU training time for a single step (averaged value over 30 steps)

For a demanding network such as FRCNN ResNet 152, the CPU requires 32 hours to do what the GPU does in a single hour. Interestingly, the GPU uses slightly more time for each step in SSD Inception V2 but is still five times faster than the CPU.

To put even further perspective on the matter, if the CPU had been used to train a single FRCNN ResNet 152 network, it would have taken close to 1300 hours, whereas on the GPU it took closer to 40 hours. For reference, the total time spent training the 20 networks reported took 720 hours using the GPU. An estimate of time spent training all networks using the CPU is provided in *Table 14*, by using the data from *Table 13*.

	Actual GPU Training	Estimated CPU Training SSD	Estimated CPU Training FRCNN
Hours	720	3600	23 112

Table 14: Actual GPU training time compared to estimated CPU training time

While this is just an estimate, it still highlights the GPU's superior performance compared to the CPU. For reference, the Intel (Skylake) Core i7-6700K processor's launch date was Q3'15, while the GTX 1080 Ti's launch date was Q2'17.

6.2 Methods

The two primary evaluation methods consist of benchmarking the network's mAP and frames per second (FPS).

For all models except SSD, a batch size higher than one would eventually cause the GPU to run out of memory, and thus a reduction in batch size was necessary for the training process to run uninterrupted. Please note that a larger batch size only causes less fluctuations in the evaluation, as more samples are tested in each batch, which gives a better estimate at that point in time. Ultimately, a batch size of 1 and 64 will eventually lead to roughly the same accuracy.

The graphing was constructed and visualized using TensorBoard, which is a suite of web applications for inspecting and evaluating TensorFlow runs.

6.3 Experiments

The previously mentioned models and classifiers are combined in various ways with various parameter tunings to create a network that is capable of recognizing fish well.

> The definition and value of the specific parameters for some of the more important networks can be found in Appendix B (B Experiments, B.1 Single Shot MultiBox Detector, B.2 Faster Region-Based Convolutional Neural Network).

Please refer to Appendix B for the following discussion.

Not all networks are included in the appendix due to space and lack of noteworthy discussion. Most parameters which are equal across the similar networks are not included.

The type (@T) defines the backend for the classifier, and is specific to each classifier (i.e. SSD Inception V2 @1 is not equal to SSD MobileNet V1 @1, despite both having @1).

There are minor differences when looking at the three ResNet 152 models (@4, @6, @1) in Appendix B, *Table 26*, and one might conclude that using Adam is a better choice than Momentum when their other parameters are similar. However, ResNet 152 @9 uses the same parameters as @4, but with a lower **gradient clipping** (i.e. limits the magnitude of the gradient values before applying them) and slightly higher learning rate and yet performs poorly.

There might be a correlation between the size of the input image after resizing and the number of data augmentations applied, where a larger image with many augmentations can result in a network incapable of learning (at least for the training durations used in this thesis).

There is still no comprehensive understanding of the internal organization of deep neural networks or the optimization process, and deep neural networks are criticized for being black boxes [82]. This is, however, an accepted tradeoff (for now) due to their incredible performance on many tasks.

There is research dedicated to reverse-engineering the black box neural networks, but most of the research is either focusing on attacking the neural networks to either replicate their existing parameters, predict their model and architectural implementation, or determine whether a single data sample has been included when training the network [83].

More research focusing on the explicit study of parameters in neural networks is required to gain a more thorough understanding of how a neural network behaves. Doing isolated studies of multiple implementations where a single parameter is gradually altered over the course of multiple neural networks could provide useful information, but this can be a tedious process.

6.4 Results

Throughout this chapter there will be color coded information to ease the visual and graphical comparisons. The graphs, along with their respective tables, are structured from best (top) to worst (bottom). Keep in mind that the color coding for each section might differ.

6.4.1 Checkpoint Size Comparison

Table 15 presents how much storage space each network requires for a single checkpoint.

Network	Single Checkpoint Size
SSD MobileNet V2	57.5 MB
SSD MobileNet V1	63.2 MB
FRCNN Inception V2	98.1 MB
SSD Inception V2	150.0 MB
SSD Inception V3	384.0 MB
FRCNN ResNet 101	418.0 MB
RFCN ResNet 101	434.0 MB
FRCNN Inception ResNet V2	452.0 MB
FRCNN ResNet 152	777.0 MB
FRCNN NAS	789.0 MB

Table 15: Space required to store a single checkpoint for each type of network

In total, the 20 networks required a storage capacity of 448 GB to store all hourly checkpoints. Note that three of the SSD networks had checkpoints stored every ten minutes, but this was altered to save storage.

6.4.2 Network Accuracy Benchmarks

Table 16 provides a complete overview of the various networks' performance, where there are *H* hours and *K* steps in thousands (i.e. 20H is 20 hours, and 22K is 22 000 steps). Peak training is when the respective mAP was achieved, while total training is how long the network trained for in total. The type (notation: @T) defines the backend for the classifier, and is specific to each classifier (i.e. SSD Inception V2 @1 is not equal to SSD MobileNet V1 @1, despite both having @1).

The following sections elaborate upon these results and graphs their mAP over time.

Model	Classifier	Type	mAP@0.5IoU	Peak Training	Total Training
SSD	Inception V2	@1	0.8464	20H / 22K	22H / 25K
FRCNN	Inception V2	@2	0.8378	25H / 579K	25H / 579K
FRCNN	ResNet 152	@4	0.8362	25H / 228K	25H / 228K
FRCNN	Inception V2	@1	0.8277	21H / 467K	21H / 471K
SSD	Inception V2	@3	0.8193	70H / 68K	71H / 69K
SSD	Inception V2	@2	0.8190	17H / 23K	19H / 26K
FRCNN	ResNet 152	@6	0.7955	70H / 550K	72H / 561K
SSD	MobileNet V2	@1	0.7882	32H / 59K	33H / 61K
SSD	MobileNet V1	@1	0.7723	15H / 21K	16H / 23K
RFCN	ResNet 101	@1	0.7591	41H / 451K	42H / 458K
SSD	Inception V3	@1	0.7208	37H / 43K	39H / 45K
FRCNN	ResNet 152	@3	0.6975	36H / 279K	38H / 295K
FRCNN	ResNet 152	@1	0.6667	33H / 293K	44H / 360K
FRCNN	ResNet 152	@2	0.5044	23H / 173K	44H / 330K
FRCNN	ResNet 101	@1	0.3497	35H / 277K	36H / 282K
FRCNN	NAS	@1	0.1072	37H / 136K	37H / 136K
FRCNN	ResNet 152	@9	0.0053	3H / 23K	16H / 116K
FRCNN	Inception ResNet V2	@1	0.0050	3H / 15K	51H / 236K
FRCNN	ResNet 152	@7	0.0027	15H / 104K	27H / 200K
FRCNN	ResNet 152	@8	0.0015	4H / 33K	42H / 354K

Table 16: Complete results (mAP@0.5IoU) from all networks

6.4.2.1 SSD

Fig. 48 and *Table 17* are interdependent. The x-axis is the steps measured in thousands in range $[0, 60\ 000+]$, and the y-axis is the accuracy in range $[0, 0.800+]$. Dataset: 20 069 labels.

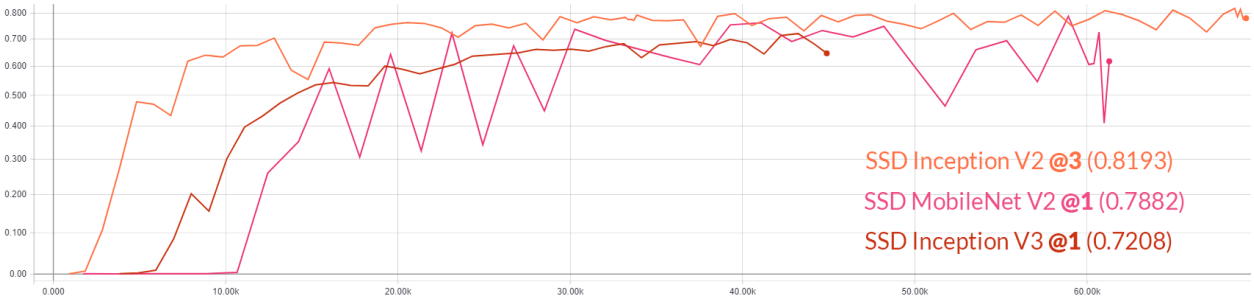

Figure 48: mAP@0.5IoU from SSD Inception V2 @3, MobileNet V2 @1, Inception V3 @1

Model	Classifier	mAP@0.5IoU	Peak Training	Total Training
SSD	Inception V2 @3	0.8193	70H / 68K	71H / 69K
SSD	MobileNet V2 @1	0.7882	32H / 59K	33H / 61K
SSD	Inception V3 @1	0.7208	37H / 43K	39H / 45K

Table 17: Results from SSD Inception V2 @3, MobileNet V2 @1, Inception V3 @1

The idea of continuously and frequently storing checkpoints are illustrated in MobileNet V2 @1. It spikes and drops repeatedly throughout its training, and one might risk not obtaining a checkpoint of when it performed well. One reason for these spikes might be that its learning rate of 0.004 is too high, so the local minima is overshot. It could also be caused due to an "unlucky" parameter space, where the updated parameters that the network thought would increase the accuracy resulted in the opposite.

Fig. 49 and *Table 18* are interdependent. The x-axis is the steps measured in thousands in range $[0, 26\ 000+]$, and the y-axis is the accuracy in range $[0, 0.800+]$. Dataset: 18 363 labels.

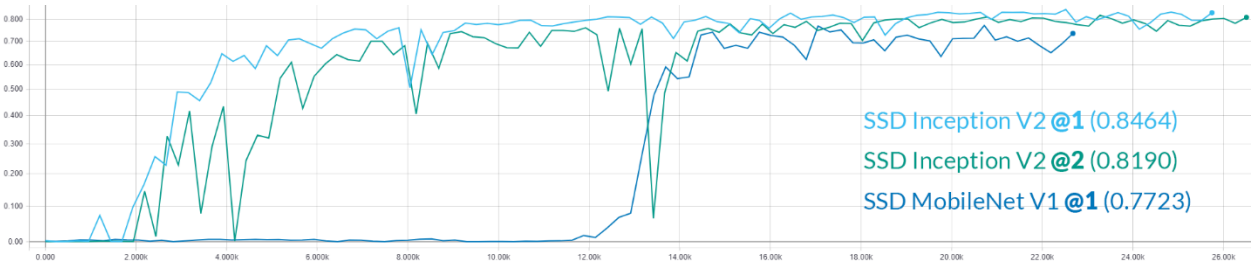

Figure 49: mAP@0.5IoU from SSD Inception V2, MobileNet V1

Model	Classifier	mAP@0.5IoU	Peak Training	Total Training
SSD	Inception V2 @1	0.8464	20H / 22K	22H / 25K
SSD	Inception V2 @2	0.8190	17H / 23K	19H / 26K
SSD	MobileNet V1 @1	0.7723	15H / 21K	16H / 23K

Table 18: Results from SSD Inception V2, MobileNet V1

MobileNet V1 @1 spends approximately 12 000 steps until its accuracy increases, and this trend is also seen in MobileNet V2 @1, hence it might relate more to an architectural design and how the weights are updated.

Interestingly, Inception V2 @1 and Inception V2 @2 has the same pattern close to 8000 steps. Inception V2 @2 drops from close to 0.73 to approximately 0.07 in less than 1000 steps, before rapidly increasing to its previous accuracy in approximately the same number of steps. This might be caused by a batch in which the network was unable to predict.

6.4.2.2 Faster R-CNN and R-FCN

Fig. 50 and *Table 19* are interdependent. The x-axis is the steps measured in thousands in range $[0, 350\ 000+]$, and the y-axis is the accuracy in range $[0, 0.800+]$. Dataset: 20 069 labels.

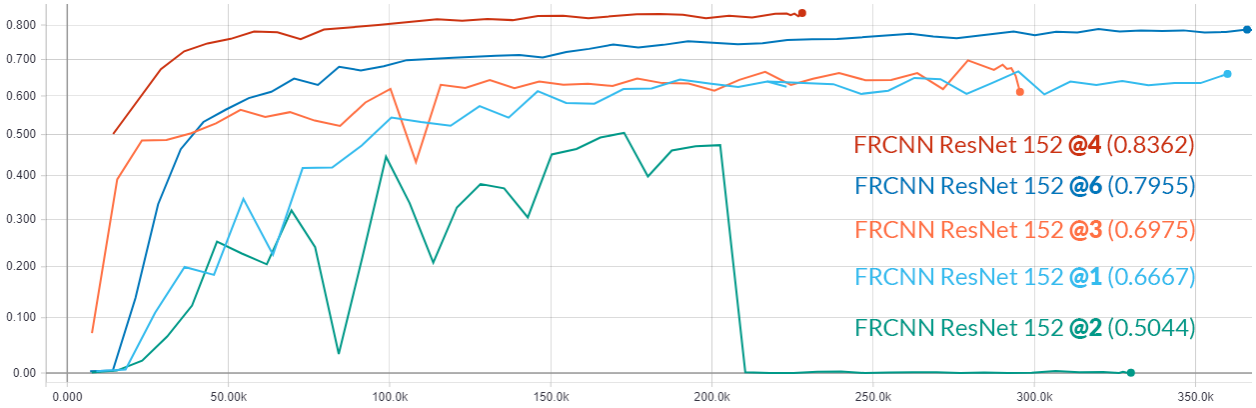

Figure 50: mAP@0.5IoU from FRCNN ResNet 152

Model	Classifier	mAP@0.5IoU	Peak Training	Total Training
FRCNN	ResNet 152 @4	0.8362	25H / 228K	25H / 228K
FRCNN	ResNet 152 @6	0.7955	70H / 550K	72H / 561K
FRCNN	ResNet 152 @3	0.6975	36H / 279K	38H / 295K
FRCNN	ResNet 152 @1	0.6667	33H / 293K	44H / 360K
FRCNN	ResNet 152 @2	0.5044	23H / 173K	44H / 330K

Table 19: Results from FRCNN ResNet 152

Each of the FRCNN networks use a batch size of 1, thus should in theory fluctuate more than the SSD networks with a larger batch size, as more samples are averaged at each step. This is, however, not the case, except for ResNet 152 @2, which after 200 000 steps drops to 0 mAP.

Fig. 51 shows how the weights have plateaued or stagnated for ResNet 152 @2 using two slightly different illustration methods. The x-axis represents the weight values, the y-axis in the left-hand figure represents the distribution of weights, and the y-axis in the right-hand figure represents steps.

Close to step 0, the weights are in the range $[-0.13, 0.13]$, with most being close to 0.014. At step 200 000, the weights are now in the range [-0.33, 0.33], thus are more spread. Incidentally, when the weights' values are reduced at the steps close to 200 000, it causes the network to be unable to accurately predict any of the batches correctly, as seen in *Fig. 50*, and the network is unable to recover.

For this case, it could be useful to lower the learning rate gradually, since the gradient descent could be overshooting in the optimal direction.

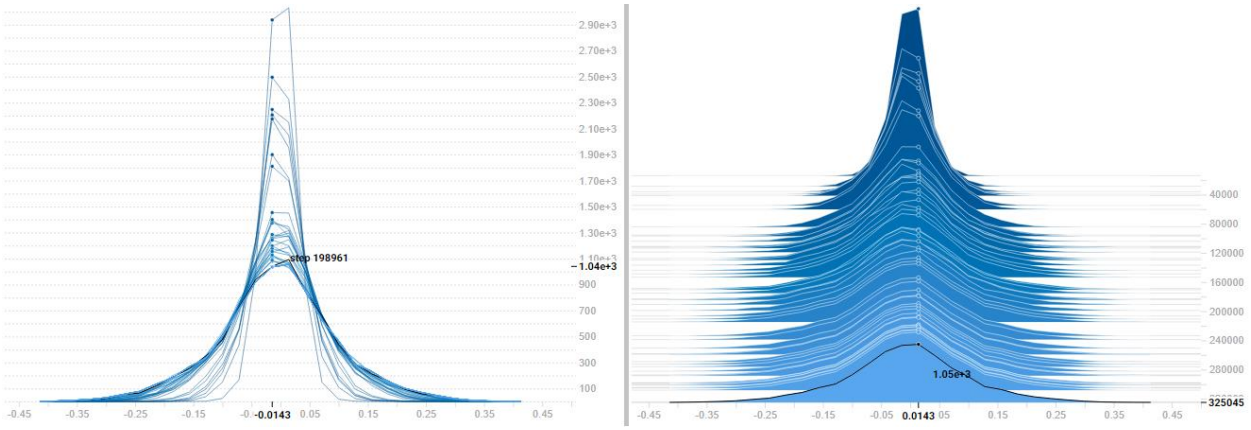

Figure 51: Change in weights over time for FRCNN ResNet 152 @2

Fig. 52 and *Table 20* are interdependent. The x-axis is the steps measured in thousands in range $[0, 450\ 000+]$, and the y-axis is the accuracy in range $[0, 0.800+]$. Dataset: 20 069 labels.

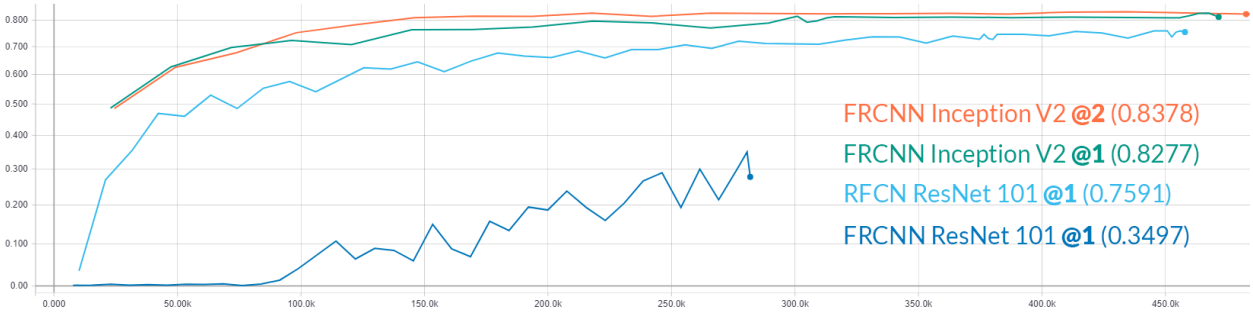

Figure 52: mAP@0.5IoU from FRCNN Inception V2, ResNet 101 and RFCN ResNet 101

Model	Classifier	mAP@0.5IoU	Peak Training	Total Training
FRCNN	Inception V2 @2	0.8378	25H / 579K	25H / 579K
FRCNN	Inception V2 @1	0.8277	21H / 467K	21H / 471K
RFCN	ResNet 101 @1	0.7591	41H / 451K	42H / 458K
FRCNN	ResNet 101 @1	0.3497	35H / 277K	36H / 282K

Table 20: Results from FRCNN Inception V2, ResNet 101 and RFCN ResNet 101

Inception V2 @1 and Inception V2 @2 are among the more stable networks over time according to their plotted lines in the graph in *Fig. 52*. FRCNN ResNet 101 @1 was training too slowly due to its learning rate of 0.0001, the second lowest among all networks, and had to be suspended to make time for other experiments.

6.4.2.3 Outliers

Fig. 53 and *Table 21* are interdependent. The x-axis is the steps measured in thousands in range $[0, 200\ 000+]$, and the y-axis is the accuracy in range $[0, 0.100+]$. Dataset: 20 069 labels.

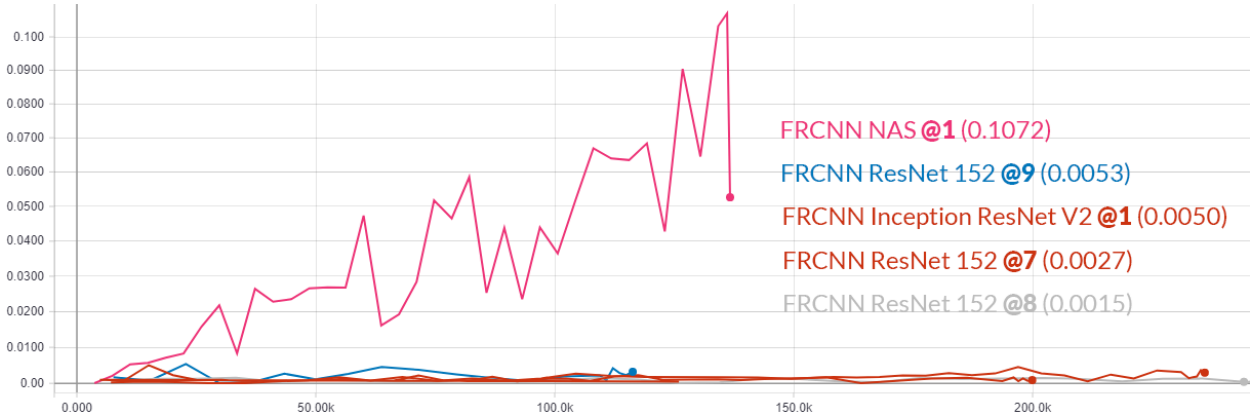

Figure 53: mAP@0.5IoU from FRCNN NAS, ResNet 152, Inception ResNet V2

Model	Classifier	mAP@0.5IoU	Peak Training	Total Training
FRCNN	NAS @1	0.1072	37H / 136K	37H / 136K
FRCNN	ResNet 152 @9	0.0053	3H / 23K	16H / 116K
FRCNN	Inception ResNet V2 @1	0.0050	3H / 15K	51H / 236K
FRCNN	ResNet 152 @7	0.0027	15H / 104K	27H / 200K
FRCNN	ResNet 152 @8	0.0015	4H / 33K	42H / 354K

Table 21 - Results from FRCNN NAS, ResNet 152, Inception ResNet V2

While the networks presented in *Fig. 53* and *Table 21* are of no practical use due to their low accuracy, they still provoke some worthy discussion. Instead of primarily focusing on the top-

performing networks and pretending failures never happen, these networks can be of great value provided the reasons for their poor performance can be highlighted.

Results not meeting anticipated goals are unfortunately often left out of research papers due to various reasons. Some might say it does not contribute practical value, while others might think that failures reflect negatively at the author(s). The reasons can be many, but they are included here for theoretical value, further understanding, and to demonstrate that not everything always goes according to plan.

Inception ResNet V2 @1 is the easiest to explain among the five. It has the lowest learning rate of 0.00006, which requires far more training than the author anticipated. It is also a very demanding classifier, with approximately 13.2 billion multiply-add combined (MAC) operations, and 55.8 million parameters (see *Table 11*). NAS @1 is an even more demanding classifier, with 23.8 billion MAC operations, and 88.9 million parameters (see *Table 11*) and might require larger amounts of training data.

ResNet 152 @7 is certainly failing due to its extreme amount of applied data augmentations. It uses random horizontal flip (50%), random 90-degree rotation (50%), random RGB to greyscale (10%), random image scaling (50%), random hue adjustment (50%), and random image cropping (50%), where the value in parentheses depict the chance of the action occurring. This amount of augmentations makes it harder for the network to learn any features, since all the data is essentially new and seemingly random.

The remaining two ResNet 152 networks are not easy to decipher, and their results might be caused by a combination of many smaller aspects. Underfitting, where there is a failure to learn the relationships in the data, or a high bias, where the network has certain assumptions causing it to ignore the training data, could be valid reasons for this behavior.

6.4.2.4 Insights

When only looking at the successful networks, it might be difficult to explain why they are performing well. However, when one can compare them to their counterparts, one gain some very valuable knowledge:

- The more data augmentations there are, the more time the network needs to achieve an acceptable accuracy, and in some cases, might not achieve a decent accuracy at all.
- The learning rate should be set based on the time constraint of the project. A learning rate that is too small will result in a network not capable of (or spends a lot of time on) learning.
- Models and classifiers should be chosen with respect to how large the dataset is, and how much the images in it varies.
- One might not always understand the behavior of a network. Deep learning is very much in a trial and error phase, where one must try to choose parameters that might work well and adjust them based on already acquired results.
- Training is very time-consuming, and some networks require large amounts of memory, which can only be obtained with expensive GPUs. Ideally, one should parallelize multiple top-of-the-line GPUs together for fast training, which leads to more time tweaking the network(s), and ultimately a better result.
- There is no one-size-fits-all when it comes to neural networks. The order in which the data is sent to the network matters, and the quality and the size of the dataset matters.
- Learn from failures, learn from success, and use previous knowledge to improve from there (just like the ideal deep neural network would).

6.4.3 Frames Per Second Benchmarks

Once the training for each network had been completed, it was necessary to compare their speed. The frames per second (FPS) benchmarks were conducted using videos with varying resolutions. Each network was tested twice for each of the four videos, and the average of the results for each respective resolution were calculated, and with that a mean average was calculated, as illustrated in *Table 22*.

MobileNet is scoring the best results, as would be expected. What is interesting, however, is that MobileNet V2 @1 is approximately as fast as MobileNet V1 @1, even though it in theory should be faster [60, 79]. A possible reason for this is that depthwise separable layers have not been fully optimized yet in TensorFlow.

One might think that a network's speed depends on the number of parameters and MAC operations (see *Table 11*). This is not always the case, for instance with Inception V3, which has

almost three times as many MAC operations, and slightly more than twice the parameters of Inception V2. While all the SSD Inception V2 networks in *Table 22* outperformed SSD Inception V3 @1 in terms of mAP, the Inception V3 network outperforms the Inception V2 networks in terms of speed.

Thus, one must consider how much mAP one is willing to sacrifice for a faster network.

Model	Classifier	720x576	1440x1080	1920x1080	2560x1920	Average FPS
SSD	MobileNet V1 @1	7.84	5.42	4.53	2.12_1	4.97
SSD	MobileNet V2 @1	7.83	4.48	4.16	2.87	4.83
SSD	Inception V3 @1	7.06	5.09	4.2	2.86	4.80
SSD	Inception V2 @3	7.18	3.89	4.07	2.78	4.48
FRCNN	Inception V2 @1	5.82	4.34	3.59	2.70	4.11
FRCNN	Inception V2 @2	5.44	3.79	3.34	2.48	3.76
SSD	Inception V2 @1	7.01_2	2.67_2	3.01_2	2.32_2	3.75
RFCN	ResNet 101 @1	4.40	3.09	2.84	2.06	3.09
FRCNN	ResNet 152 @6	3.66	2.92	2.47	2.00	2.76
FRCNN	ResNet 152 @4	3.57	2.44	2.41	1.88	2.57

Table 22: Frames per seconds from the 10 most interesting networks

Lastly, but perhaps most importantly, during the benchmarking, some peculiar results appeared. The behavior has been highlighted in subscript after each test and is discussed below.

Peculiar behaviors:

- 1: In some cases, the network would create multiple overlapping bounding boxes for a single fish swimming by, instead of one bounding box.
- 2: While the network detected fish, the bounding boxes produced were not tightly bound to either of the fish in either frame.

Exactly why SSD MobileNet V1 @1 ran into behavior 1 could be due to ill values for non-max suppression (NMS). Recall that NMS discards all bounding boxes but the one with highest confidence in an area. Instead of discarding all but one, the network kept up to three overlapping bounding boxes for each object, positioned with one in the front, middle, and back of the fish.

SSD Inception V2 @1 had too large bounding boxes for all videos. Thus, if it is necessary to detect accurately where the fish is, instead of only detecting whether there is a fish in the frame, then this network should be avoided.

6.4.4 Recording Statistics

Due to a lack of resources, it was not possible to test either network using a camera positioned in a river, therefore the results in this section had to be gathered from a local video from fish in fish farms.

The statistics are recorded and exported to Excel where they are automatically graphed as seen in *Fig. 54*, where the y-axis representing the number of fish, and the x-axis represents time. *Fig. 54* is an example of how the exported data is structured and visualized after a network has been running.

One can also implement functionality for exporting the data to a database, where a website retrieves the data, and updates the results live. This allows one to sort through the data in hours, days, weeks, months, years, etc., and can provide useful information regarding how many fish are in which rivers at certain times of the year. This data can then be compared with previous data to find whether there has been an increase or decline of fish in each river.

Figure 54: Statistics of recognized fish

Using deep learning to perform this task could drastically improve the information retrieval accuracy. Statistics Norway states the following regarding accuracy on reported river catches of fish: “Missing or delayed catch reports from fishermen and landowners can be a problem for the river catch statistics. Statistics are also lacking from some small river systems where fishing is poorly organized. Consequently, the statistics are not totally complete.” [10].

This thesis thus partially solves the issues that Statistics Norway has discussed, as the results can be instantly gathered, with no need of local fishermen or landowners to provide reports once a year. Partially, because at the time, no tracking functionality has been added, which causes duplicate results.

Also, cameras can be placed in any river as long as electricity can be provided to the equipment. Ideally, the systems in the rivers should send data to databases regularly, but if there is no internet connection, then the data must be retrieved in person.

Thus, one can get a 24/7 surveillance of rivers, resulting in more accurate statistics. This is, however, dependent on accurate systems to begin with, and for a potential implementation phase, the system should coincide with how data retrieval is currently accomplished.

6.4.5 Reduction of Footage

Whenever there is a frame with no recognized fish, the system should discard that frame. While this is a feasible solution, it can lead to what might appear as video stutter, since it is skipping individual frames instead of video sequences. Therefore, there is a three second timer between each removal window. Some basic pseudocode is provided to illustrate the core idea:

```
while video capture is running:
    if there is a fish in frame:
        if this is the first time a fish is recognized:
            set the current frame as start
            set the name of the segmented video with current date and time
        else:
            set the current frame as end
    elif at least three seconds differ between end and start:
        create a new segmented video
        set start and end to 0

concatenate all segmented videos
write a new video file with the concatenated videos
```

The results are promising when using local videos of salmon swimming in the ocean but have unfortunately not yet been tested as a standalone feature in rivers due to a lack of resources and time. In general, the higher the accuracy (mAP) of the network, the more accurate results are produced.

6.4.6 Visual Comparisons

This section illustrates that even though two networks have approximately the same accuracy, they might be better at certain scenarios. For this comparison, the networks FRCNN Inception V2 @2 (0.8378 mAP), and FRCNN ResNet 152 @4 (0.8362 mAP) is used, where two random samples from their results on the testing data is provided.

Fig. 55 (Inception V2) and *Fig. 56* (ResNet 152) illustrates that for this specific test (test 01), the difference is mostly in their general confidence. ResNet 152 detects one additional fish, while having an overall higher confidence on its detections. Inception V2 is, however, able to detect a fish (with low confidence, 51%) that ResNet 152 is not.

For the next test (test 02), shown in *Fig. 57* (Inception V2) and *Fig. 58* (ResNet 152), they detect an equal amount of fish (i.e. 11). However, they both have a unique fish that the other network is not able to detect.

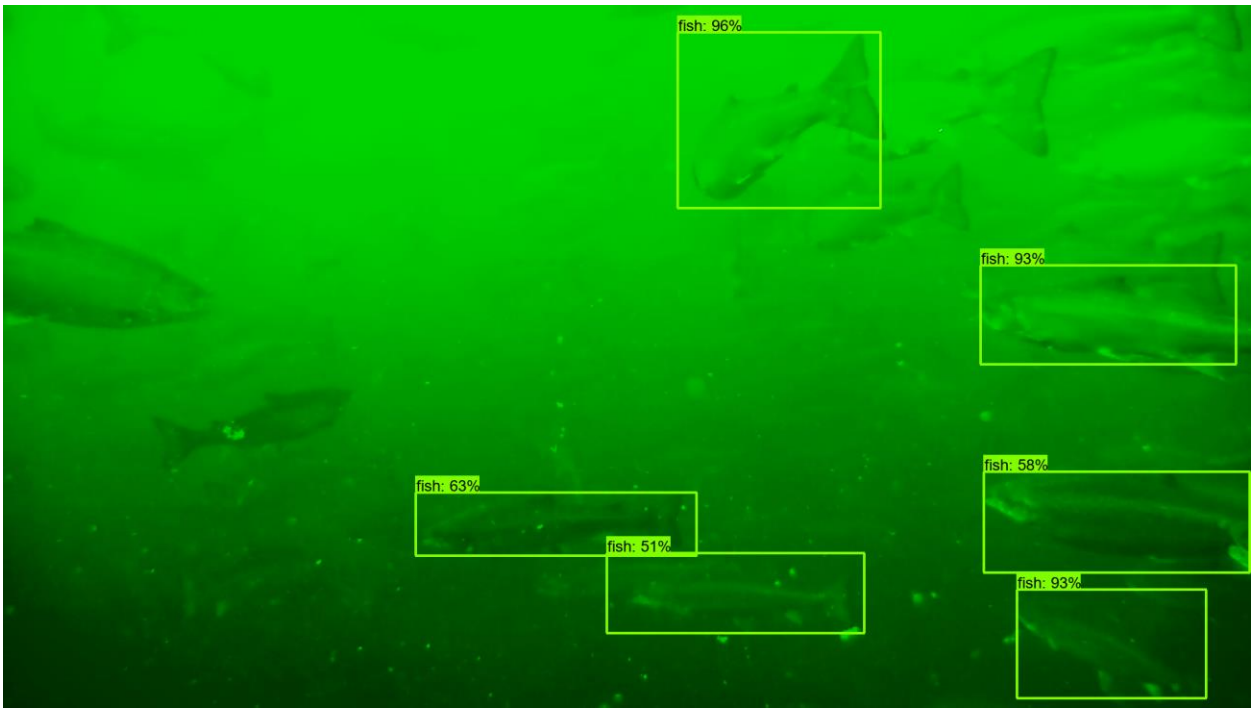

Figure 55: FRCNN Inception V2 @2, test 01

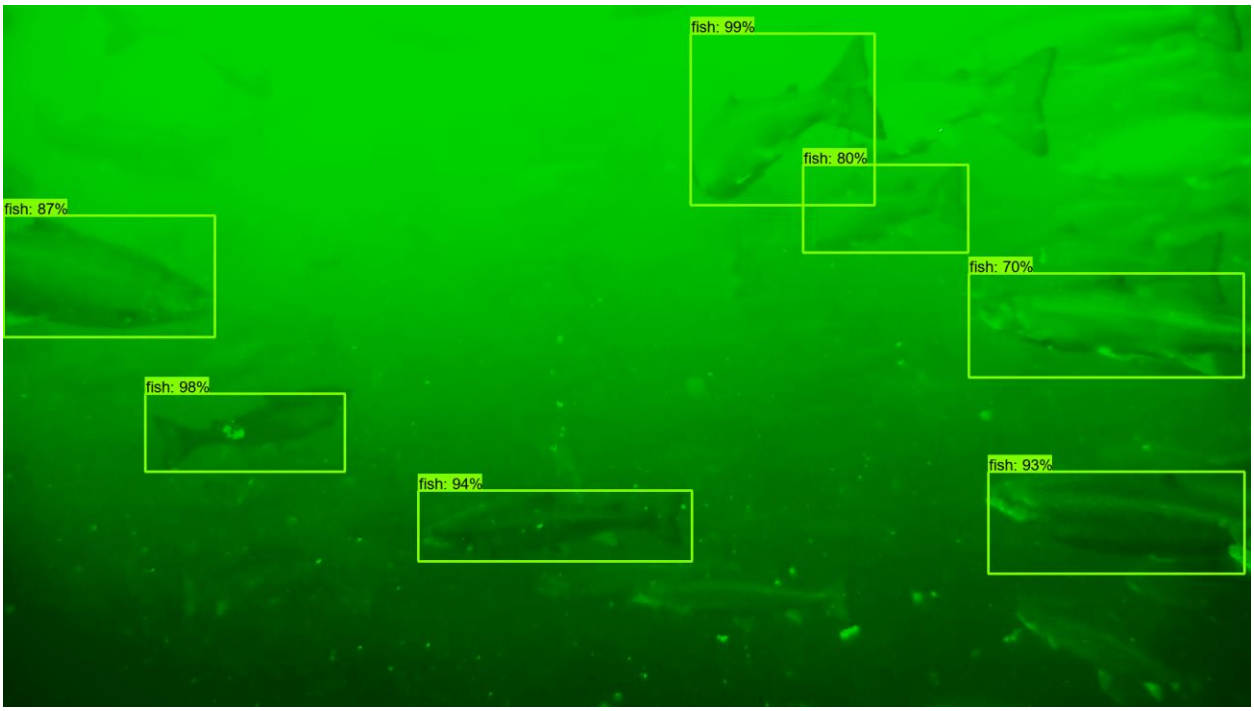

Figure 56: FRCNN ResNet 152 @4, test 01

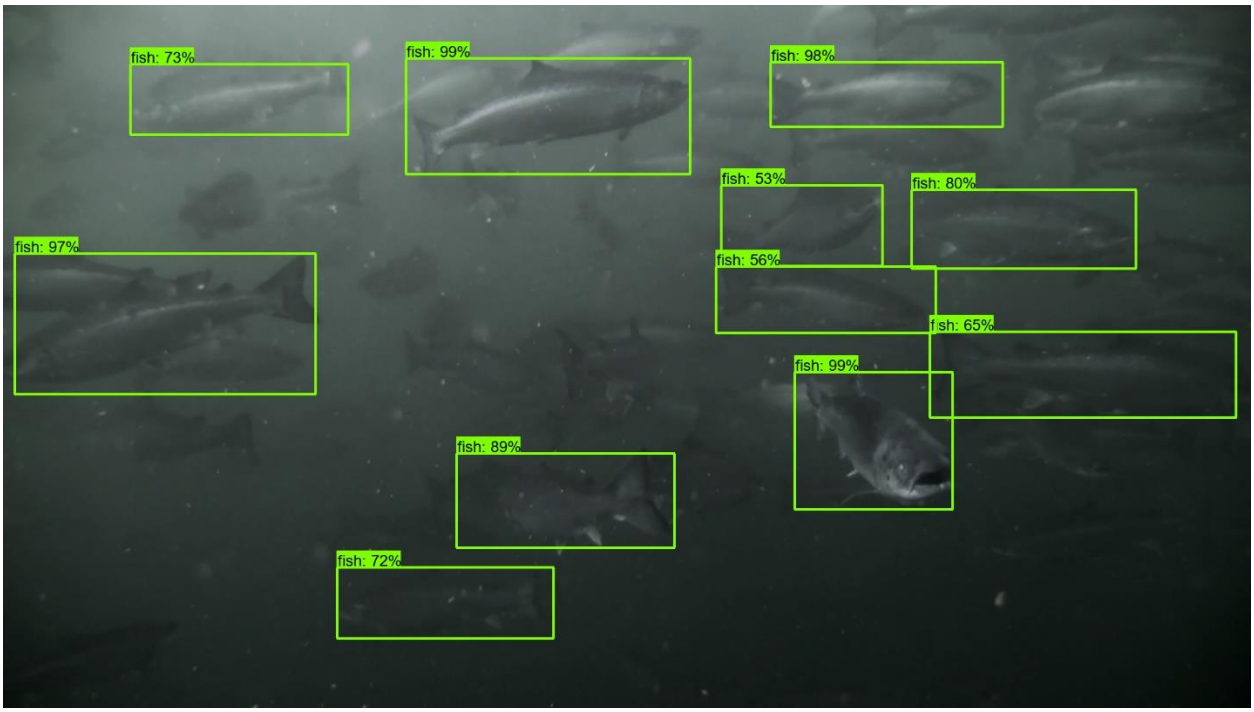

Figure 57: FRCNN Inception V2 @2, test 02

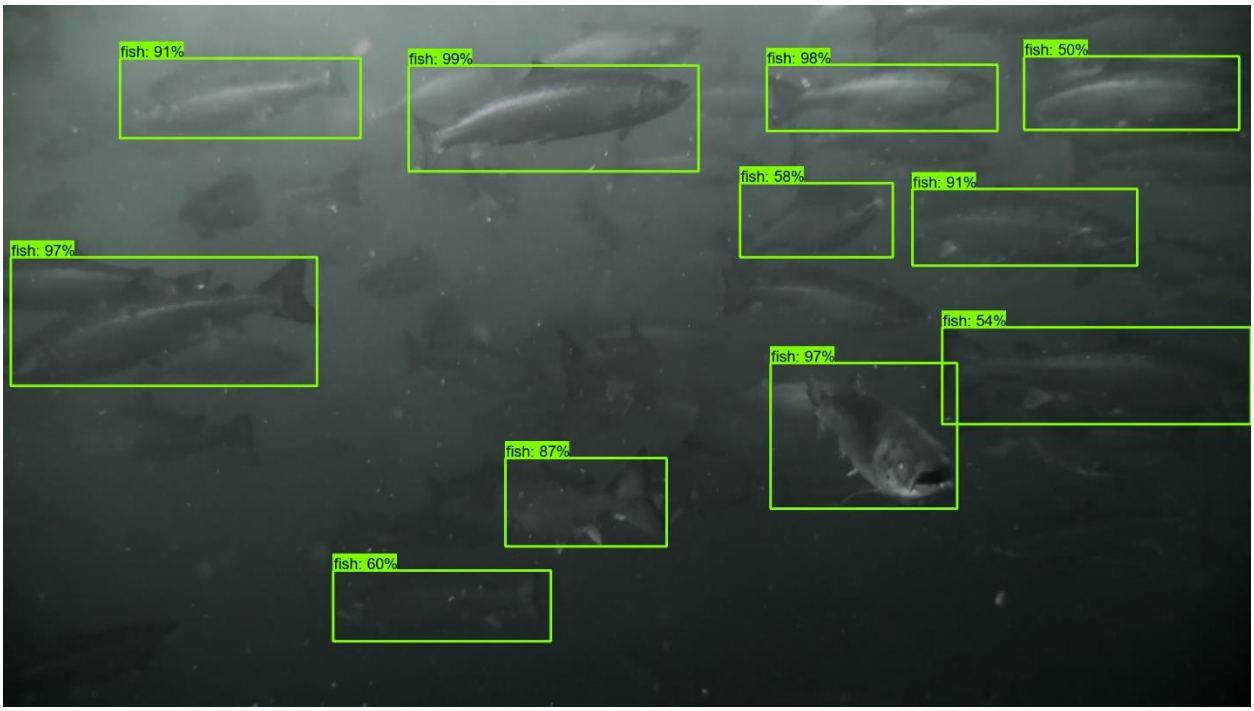

Figure 58: FRCNN ResNet 152 @4, test 02

While there are some minor differences for each network from this minor sample size of two random images from the testing data, it is in most scenarios better to use the network with the higher accuracy, and if it is similar, choose the one with a higher FPS. The FPS difference of the two networks are $3.76 - 2.57 = 1.19$, thus Inception V2 @2 would be a better choice between these two.

6.4.7 Model Graphs

Appendix C (C Model Graphs) show the structure of the SSD, Faster R-CNN, and R-FCN models, and how the modules relate to one another from a bird's-eye view. The *FeatureExtractor* module(s) in each model contains the various classifiers, but they are excluded due to not only their large overall size, but also since each module includes multiple modules, which includes more modules, etc., and depicting all these would be excessive.

7 Conclusion

Deep learning has proven to be applicable for use in pisciculture and underwater environments. This thesis presents a state-of-the-art salmon recognition deep neural network, which can achieve up to 84.64% accuracy on recognizing salmon, with an average of 3.75 FPS. Some of the other provided networks in the thesis have a higher FPS, with slightly lower accuracy.

The thesis presents a solution for removing video sequences where no recognized fish is found, which reduce storage, and more importantly, reduce manual work, while also retaining the valuable information. The video sequences can be used to obtain a larger dataset which will improve the accuracy of the networks further. The video sequences can also be used to inspect fish species, their size, and other important characteristics, which provides insight on their general health.

The thesis proposes a solution of how one can gather more accurate statistics of how many fish are in various rivers throughout the year, compared to how Statistics Norway currently do it, as well as eliminating most of their retrieval process steps, which are as follows [10]:

1. Catch reports from the individual fisherman are collected by landowners.
2. A summary catch report is sent to the county governor or entered in the register.
3. The county governor reports complete figures for each river in the salmon register.
4. Statistics Norway downloads a complete file from the salmon register from a website.
5. Data is transferred to an editing system and computerized controls are applied.
6. If necessary, the county governors are contacted to assess the data.

In addition, the thesis provides further insight on how one can initialize a deep neural network's parameters to reduce the risk of creating a network either incapable of learning or that generalizes too well on the training data.

For salmon, and salmon-like species, the results presented in the thesis achieves state-of-the-art accuracy, while also presenting the first deep neural network(s) specifically aimed at, and capable of recognizing salmon.

7.1 Multiple Fish Detection and Localization

The primary goal was to detect all fish in each frame in a video sequence, as well as draw a bounding box around each fish. This was achieved by using various combinations of models and classifiers, and finetuning the parameters in the network. Whether a network fulfills this goal based on its precision is a worthwhile discussion, and as deep neural networks improve over the coming years, the goal post is going to move farther towards 100% precision, as algorithms improve, and datasets grow larger.

One might argue that a mAP of 0.50 are considered random, thus anything over 50% precision could be considered successful. However, a network should be confident and accurate in its decisions, and networks with mAP > 0.75 fit better within those conditions, which the top ten networks presented in this thesis does.

7.2 Image Classification

The secondary goal was tagging frames that contained fish to be able to discard video sequences where no fish are present.

The goal was completed by using the networks from the first goal to register whether fish are present in a frame. This was achieved by setting certain states in video sequences, which keep track of which sequences to keep, before finally concatenating the video sequences together as a single sequence. As no presented network is 100% precise, some video sequences of where there are fish, but the network was not able to recognize, will be lost. However, all stored video sequences contain at least a single fish.

7.3 Record Statistics

The tertiary goal is partially completed in the sense that statistics are generated and exported, but fish are for now registered as a new fish in each frame.

Due to time constraints and insufficient resources, it was not possible to test the system on a camera positioned in a river. Therefore, the tests were done using local footage of fish in fish farms. Since the networks do not continuously keep track of whether a fish in a previous frame is equal to the fish in the current frame, duplicate results are recorded. It is therefore necessary to add tracking functionality to the networks to eliminate duplicate results.

7.4 Insights

Deep learning has come a long way, but the technology is very likely to improve further in a rapid pace over the coming years. Deep neural networks can be difficult to inspect since they are not currently reporting their decision process in a simple and meaningful way. Thus, improving their precision is at the time being mostly based on trial and error.

TensorBoard provides valuable insight into how the network has developed over time and makes it easier to compare multiple networks.

Labeling objects in frames can be a tedious task, but in the future, there might be software that learn the features of the currently labeled objects and suggests labels in real-time. Large datasets with precise labels are important for deep neural networks and can be hard to come

by. One is likely required to spend a lot of time (or money) to obtain and label a dataset for a new project's specific use-case.

One should also be aware that when using datasets labeled by others, they can contain wrong classifications, inaccurate classifications, lack of diversity, under- and overrepresented classes, etc. One should step through the dataset to verify that it is accurate and satisfactory for one's goals.

8 Further Work

There are many additional use cases for the deep neural networks provided in this thesis, which can be used to extend the overall system further. Some of those use cases are discussed in the coming sections.

8.1 Multi-Tracking and Monitoring for Conspicuous Behavior

By continuously tracking fish over longer periods of time in controlled environments, one can gain a better understanding of how fish behaves during certain circumstances. The deep neural networks presented can detect fish (i.e. determine whether a bounding box contains a fish), but not track (i.e. follow specific fish throughout a sequence of frames). However, it is possible to extend the application to allow for tracking capabilities, which is not only key when generating statistics, but also for monitoring fish behaviors.

The tracking functionality can for instance be added by combining a convolutional neural network (CNN) with a recurrent neural network (RNN), where the CNN embeds the fish appearance, and the RNN remembers the appearance and motion information. An example of such an application can be found in [84].

When fish encounter an area where oxygen levels are low, they will start swimming rapidly in zig-zag like patterns to escape the dangerous area, and often swim to the surface to extract oxygen from the surface layer in contact with the atmosphere. Under prolonged periods of hypoxia (i.e. low oxygen conditions), the fish tend to eat less, move less, and have weaker fright responses, which may eventually lead to death [85].

For coldwater fish, trout and salmon are very vulnerable to low dissolved oxygen (DO) levels (i.e. the level of free, non-compound oxygen in water). If exposed to areas where the DO is less than 3 mg/L for more than a few days, it will result in death. In terms of reproducing, the growth of trout and salmon eggs will be delayed at DO levels less than 11 mg/L, and decrease the survival rates and growth if below 8 mg/L. Saltwater fish have a higher tolerance for low DO concentrations, and DO levels are approximately 20% less in seawater than in freshwater [86].

On the other end of the spectrum, too high concentrations of DO can also cause death due to gas bubble disease, blocking the flow of blood through the capillary vessels. For certain species of young trout and salmon, a 100% mortality occurs in less than three days at 120% dissolved oxygen saturation [87].

Certain types of parasites will try to get to its definitive host through intermediate hosts by altering the intermediate host's behavior, and exploiting established predator seeking prey interactions. For instance, trematodes (i.e. small parasitic flatworms) have a life cycle usually

involving three hosts. In the definitive host, the parasite's goal is to reproduce sexually and release eggs, continuing the cycle [88].

As an experiment, two different killifish populations were captured and moved to outdoor pens. The killifish were separated into two groups, each with 95 parasitized fish, and 53 unparasitized fish. After 20 days, the number of remaining fish from the pen with closed surface was 91 parasitized fish and 50 unparasitized fish, while in the open pen, there were 44 parasitized fish and 49 unparasitized fish [89].

In the open pen, this is a reduction of 53.68% for parasitized fish, and 7.54% for unparasitized fish. Additionally, parasitized fish had a mean of 21 conspicuous behaviors (identified as either flashing, contorting, surfacing, shimmying, or jerking) per 30 minutes, compared to 5.3 for unparasitized fish [89]. Only parasitized fish contorted, jerked, or shimmied [89].

This indicates that host manipulation can alter fish movements, but that it may be difficult to control due to varied species of fish combined with several types of parasites. It is, however, unclear whether it is the parasite's manipulation of the host's physiology that causes the behavioral changes, or if the host itself is doing it to combat the parasite [88].

Dissolved oxygen levels, parasites, prey, and predators all affect the behavior of fish. By creating a set of rules, and continuously track fish, one might compare the fish movements to the ruleset, and determine whether something is wrong. The problem is identifying what is normal and conspicuous behavior for certain species in certain conditions.

With the ability to track copious quantities of fish over extended periods of time in both controlled and semi-controlled environments, one could train a deep neural network to identify when conspicuous behavior is apparent. This would not necessarily describe why something out of the ordinary is happening, but it would make it easier to investigate and fix potential issues in shorter time-frames.

Overall, the use of deep learning could result in healthier environments for fish in pisciculture, leading to a decrease in fish mortalities.

8.2 Classifying Multiple Fish Species or Any Object

It is possible to extend the application to recognize more than just salmon, provided there exists additional datasets for each species. Each species thus has its own class, and instead of a binary classification (either salmon or not) it is a classification based on the number of classes.

A simple example would be a dataset with salmon, cod, mackerel, and tuna. The final output would be a sum of the values up to 1 (where 1 is 100% accuracy). The classification of the mentioned species with the following results [0.02, 0.05, 0.89, 0.04] would thus output

mackerel with an 89% confidence. Extending to classify multiple species is rather straightforward, but requires, as mentioned, a large amount of data, and this data can be hard to come by.

The neural networks can also be used to recognize any type of object, not just fish. The deep neural networks presented are generalized such that they decide what to learn, instead of a programmer telling them explicitly how to learn, and as such they are generalizable for any object recognition task.

8.3 Monitoring for Growth

Another benefit of using neural networks is for monitoring fish growth. By recording the minimum bounding box for each fish at set times each week, one could identify the mean and average growth rate for various fish species.

The environment must be constructed with an obvious way of defining the growth-rate of a fish. The distance from the camera to the fish will impact the size of the minimum bounding box. This could be solved by placing a flat surface in front of the camera at a fixed distance to lower the margin of error. Another option is to have the fish swim through a tube, but the drawback is that less data may be gathered in such a strict environment.

On the topic of growth rate, the maximum body size of fish is decreasing due to the geometric limitation of the growth of gills and metabolism's response to warmer temperatures [90]. Warmer water accelerates the metabolism in fish, and causes increased activity, leading to an increase in the oxygen requirements [86].

9 References

[1] FishBase, "Fishes Used by Humans (based on FishBase 02/2018)", Feb. 2018. Available: http://www.fishbase.org/Report/FishesUsedByHumans.php. Accessed on: May 22, 2018.

[2] B. B. B. Phoenix X. Huang, Robert B. Fisher, "Fish Recognition Ground-Truth data", Sep. 23, 2013. Available: http://groups.inf.ed.ac.uk/f4k/GROUNDTRUTH/RECOG/. Accessed on: Mar. 2, 2018.

[3] A. L. Samuel, "Some Studies in Machine Learning Using the Game of Checkers" *IBM Journal of Research and Development,* vol. 3, no. 3, pp. 210-229, E-Pub. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5392560&isnumber=5392559. Accessed on: May 24, 2018.

[4] K. Panetta, "Top Trends in the Gartner Hype Cycle for Emerging Technologies, 2017", Aug. 15, 2017. Available: https://www.gartner.com/smarterwithgartner/top-trends-in-the-gartner-hype-cycle-for-emerging-technologies-2017/. Accessed on: Nov. 18, 2017

[5] H. Spruce, "The Fire Triangle", Mar. 16, 2016, [Image]. Available: https://www.highspeedtraining.co.uk/hub/fire-triangle-tetrahedron-combustion/. Accessed on: May 20, 2018.

[6] T. U. o. Munich, "DeepLearning2016", 2016, [Image]. Available: http://campar.in.tum.de/files/deeplearning2016/braincircles_small.png. Accessed on: May 20, 2018.

[7] I. M. Cloud, "10 Key Marketing Trends for 2017 and Ideas for Exceeding Customer Expectations", 2017, Available: https://public.dhe.ibm.com/common/ssi/ecm/wr/en/wrl12345usen/watson-customer-engagement-watson-marketing-wr-other-papers-and-reports-wrl12345usen-20170719.pdf. Accessed on: Nov. 18, 2017.

[8] S. Norway, "Fisheries", 2018, Available: https://www.ssb.no/en/fiskeri. Accessed on: Feb. 13, 2018.

[9] S. Norway, "Aquaculture", 2017, Available: https://www.ssb.no/en/fiskeoppdrett. Accessed on: Feb. 13, 2018.

[10] S. Norway, "River catch of salmon, sea trout and migratory char", 2018, Available: http://www.ssb.no/en/elvefiske. Accessed on: Feb. 13, 2018.

[11] R. K. K. Kade Small, Robyn J. Watts, Julia Howitt, "Hypoxia, Blackwater and Fish Kills: Experimental Lethal Oxygen Thresholds in Juvenile Predatory Lowland River Fishes" *PLoS ONE,* vol. 9, no. 4, p. e94524, E-Pub Apr. Available: https://doi.org/10.1371/journal.pone.0094524. Accessed on: Nov. 25, 2017.

[12] D. C. M. ZongYuan Ge, Prof Peter Corke, "Fish Dataset", Jun. 16, 2016. Available: https://wiki.qut.edu.au/display/cyphy/Fish+Dataset. Accessed on: May. 22, 2018.

[13] SeaCLEF2017, "SeaCLEF 2017 - Multimedia Retrieval in CLEF", May 21, 2017. Available: http://www.imageclef.org/lifeclef/2017/sea. Accessed on: May 22, 2018.

[14] M. S. Xiu Li, Hongwei Qin, Liansheng Chen, "Fast Accurate Fish Detection and Recognition of Underwater Images with Fast R-CNN ". Available: http://qinhongwei.com/academic/projects/oceans15/shang2015fast.pdf. Accessed on: May 22, 2018.

[15] A. S. Shoaib Ahmed Siddiqui, Muhammad Imran Malik, Faisal Shafait, Ajmal Mian, Mark R Shortis, Euan S Harvey, Howard Browman, "Automatic fish species classification in underwater videos: exploiting pre-trained deep neural network models to compensate for limited labelled data". Available: https://doi.org/10.1093/icesjms/fsx109. Accessed on: May 27, 2018.

[16] Y. T. Xiu Li, Tingwei Gao, "Deep but lightweight neural networks for fish detection", in *OCEANS 2017 - Aberdeen*, Aberdeen, UK, 2017, pp. 1-5: IEEE.

[17] A. J. Ahmad Salman, Faisal Shafait, Ajmal Mian, Mark Shortis, James Seager, Euan Harvey, "Fish species classification in unconstrained underwater environments based on deep learning". Available: https://doi.org/10.1002/lom3.10113. Accessed on: May 22, 2018.

[18] I. Mierswa, "What Is Artificial Intelligence, Machine Learning, And Deep Learning?", Apr. 26, 2017. Available: https://rapidminer.com/blog/artificial-intelligence-machine-learning-deep-learning/. Accessed on: May 12, 2018.

[19] J. Vincent, "DeepMind's Go-playing AI doesn't need human help to beat us anymore", Available: https://www.theverge.com/2017/10/18/16495548/deepmind-ai-go-alphago-zero-self-taught. Accessed on: Oct. 18, 2017

[20] C. E. Perez, "Surprise! Neurons are Now More Complex than We Thought!!", Feb. 10, 2018. Available: https://medium.com/intuitionmachine/neurons-are-more-complex-than-what-we-have-imagined-b3dd00a1dcd3. Accessed on: Mar. 6, 2018.

[21] F.-F. L. Andrej Karpathy, Justin Johnson, "CS231n Convolutional Neural Networks for Visual Recognition", Nov. 28, 2017. Available: http://cs231n.github.io/neural-networks-1/. Accessed on: Feb. 21, 2018.

[22] J. J. Fei-Fei Li, Serena Yeung, "Lecture 4: Backpropagation and Neural Networks", Apr. 13, 2017. Available: http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture4.pdf. Accessed on: Mar. 12, 2018.

[23] R. Harken, "Faces of Summit: Leading a Systems Expedition", May 29, 2018. Available: https://www.olcf.ornl.gov/2018/05/29/faces-of-summit-leading-a-systems-expedition/. Accessed on: May 30, 2018.

[24] Graph500, "November 2017 BFS", Nov. 2017. Available: https://graph500.org/?page_id=327. Accessed on: May 15, 2018.

[25] A. Impacts, "Brain performance in TEPS", May. 6, 2015. Available: https://aiimpacts.org/brain-performance-in-teps/. Accessed on: Mar. 6, 2018.

[26] N. Wolchover, "New Theory Cracks Open the Black Box of Deep Neural Networks", Oct. 10, 2017. Available: https://www.wired.com/story/new-theory-deep-learning/ Accessed on: Feb. 20, 2018.

[27] G. Sanderson, "But what *is* a Neural Network? | Chapter 1, deep learning," in "Deep Learning," 2017 [Online]. Available: https://www.youtube.com/watch?v=aircAruvnKk. Accessed on: Feb. 1, 2018.

[28] M. Nielsen, A visual proof that neural nets can compute any function, 2017. [Online]. Available: http://neuralnetworksanddeeplearning.com/chap4.html. Accessed on Feb. 5, 2018.

[29] C. Davis, "Is a single layered ReLu network still a universal approximator?", Feb. 25, 2017. Available: https://www.quora.com/Is-a-single-layered-ReLu-network-still-a-universal-approximator/answer/Conner-Davis-2. Accessed on: May 17, 2018.

[30] X. Z. Kaiming He, Shaoqing Ren, Jian Sun, "Deep Residual Learning for Image Recognition". Available: https://arxiv.org/pdf/1512.03385v1.pdf. Accessed on: May 17, 2018.

[31] G. E. H. David E. Rumelhart, Ronald J. Williams, "Learning representations by back-propagating errors" *Nature,* vol. 323, p. 533, E-Pub Oct. Available: http://dx.doi.org/10.1038/323533a0. Accessed on: Feb. 5, 2018.

[32] P. S. Mohammad Babaeizadeh, Roy H. Campbell, "NoiseOut: A Simple Way to Prune Neural Networks". Available: https://arxiv.org/pdf/1611.06211v1.pdf. Accessed on: May 13, 2018.

[33] A. Moawad, "Neural networks and back-propagation explained in a simple way", Feb. 1, 2018. Available: https://medium.com/datathings/neural-networks-and-backpropagation-explained-in-a-simple-way-f540a3611f5e. Accessed on: May 13, 2018.

[34] G. Sanderson, "Gradient descent, how neural networks learn | Chapter 2, deep learning," in "Deep Learning," 2017 [Online]. Available: https://www.youtube.com/watch?v=aircAruvnKk. Accessed on: Feb. 2, 2018.

[35] AIStudy, "Gradient Descent", n.d. Available: http://www.aistudy.com/math/gradient_descent.htm. Accessed on: Feb. 19, 2018.

[36] L. A. d. Santos, "Back-propagation", 2017. Available: https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/backpropagation.html. Accessed on: May 13, 2018.

[37] S. N. Varun Ranganathan, "A New Backpropagation Algorithm without Gradient Descent". Available: https://arxiv.org/pdf/1802.00027v1.pdf. Accessed on: May 27, 2018.

[38] W. M. C. Max Jaderberg, Simon Osindero, Oriol Vinyals, Alex Graves, David Silver, Koray Kavukcuoglu, "Decoupled Neural Interfaces using Synthetic Gradients". Available: https://arxiv.org/pdf/1608.05343v2.pdf. Accessed on: May 27, 2018.

[39] M. Jaderberg, "Decoupled Neural Interfaces Using Synthetic Gradients", Aug. 29, 2016. Available: https://deepmind.com/blog/decoupled-neural-networks-using-synthetic-gradients/. Accessed on: May 27, 2018.

[40] A. Nøkland, "Direct Feedback Alignment Provides Learning in Deep Neural Networks". Available: https://arxiv.org/pdf/1609.01596v5.pdf. Accessed on: May 27, 2018.

[41] A. T. John A. Fornshell, "The Development of SONAR as a Tool in Marine Biological Research in the Twentieth Century" *International Journal of Oceanography,* vol. 2013, p. 9, E-Pub Sep., Art. no. 678621. Available: http://dx.doi.org/10.1155/2013/678621. Accessed on: Nov. 28, 2017.

[42] S. N. Brian Smyth, "Passive Integrated Transponder (PIT) Tags in the Study of Animal Movement" *Nature Education Knowledge,* vol. 4, no. 3. Available: https://www.nature.com/scitable/knowledge/library/passive-integrated-transponder-pit-tags-in-the-101289287. Accessed on: Nov. 29, 2017.

[43] D. Z. Greg DeCelles, "Acoustic and Radio Telemetry" *Stock Identification Methods,* vol. 2, pp. 397–428, E-Pub Oct. Available: https://doi.org/10.1016/B978-0-12-397003-9.00017-5. Accessed on: Nov. 30, 2017.

[44] H. K. P. Hidayatullah, "CAMSHIFT improvement on multi-hue object and multi-object tracking", presented at the 3rd European Workshop on Visual Information Processing, Paris, France, July, 2011. Available: http://ieeexplore.ieee.org/document/6045533/

[45] K. H. Kei Terayama, Hitoshi Habe, Masa-aki Sakagami, "Appearance-based multiple fish tracking for collective motion analysis", in *2015 3rd IAPR Asian Conference on Pattern Recognition (ACPR)*, Kuala Lumpur, Malaysia, 2015, pp. 361-365: IEEE.

[46] Y. S. Gao Huang, Zhuang Liu, Daniel Sedra, Kilian Weinberger, "Deep Networks with Stochastic Depth". Available: https://arxiv.org/pdf/1603.09382v3.pdf. Accessed on: Nov. 18, 2017.

[47] A. M. Noam Shazeer, Krzysztof Maziarz, Andy Davis, Quoc Le, Geoffrey Hinton, Jeff Dean, "Outrageously Large Neural Networks: The Sparsely-Gated Mixture-of-Experts Layer", presented at the ICLR 2017, Palais des Congrès Neptune, Toulon, France, Jan., 2017. Available: https://arxiv.org/pdf/1701.06538v1.pdf

[48] C. O. Alexander Mordvintsev, Mike Tyka, "Inceptionism: Going Deeper into Neural Networks", Jul. 13, 2015. Available: https://research.googleblog.com/2015/06/inceptionism-going-deeper-into-neural.html. Accessed on: Nov. 19, 2017.

[49] A. L. Beam, "You can probably use deep learning even if your data isn't that big", Jun. 4, 2017. Available: https://beamandrew.github.io/deeplearning/2017/06/04/deep_learning_works.html. Accessed on: May 31, 2018.

[50] N. P. Ian Goodfellow, Sandy Huang, Yan Duan, Pieter Abbeel, Jack Clark, "Attacking Machine Learning with Adversarial Examples", Feb. 24, 2017. Available: https://blog.openai.com/adversarial-example-research/. Accessed on: Nov. 19, 2017.

[51] D. M. Nitish Shirish Keskar, Jorge Nocedal, Mikhail Smelyanskiy, Ping Tak Peter Tang, "On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima", presented at the ICLR 2017, Palais des Congrès Neptune, Toulon, France, Feb., 2017. Available: https://arxiv.org/pdf/1609.04836v2.pdf

[52] GitHub, "Deep learning", May 28, 2018. [Online]. Available: https://github.com/topics/deep-learning?o=desc&s=stars. Accessed on: May. 28.

[53] M. H. Moshe Looks, DeLesley Hutchins, Peter Norvig, "Deep Learning with Dynamic Computation Graphs". Available: https://arxiv.org/pdf/1702.02181v2.pdf. Accessed on: May 31, 2018.

[54] T. Qian, "Star History", May 28, 2018. Available: http://www.timqian.com/star-history/#pytorch/pytorch&tensorflow/tensorflow&Microsoft/CNTK&caffe2/caffe2. Accessed on: May 28, 2018.

[55] Steinsvik, "Solutions for tough environments", n.d. Available: https://steinsvik.no/en/company. Accessed on: May 29, 2018.

[56] F. V. Veen, "The Neural Network Zoo", Sep. 14, 2016. Available: https://www.asimovinstitute.org/neural-network-zoo/. Accessed on: May 20, 2018.

[57] F.-F. L. Andrej Karpathy, Justin Johnson, "Convolutional Neural Networks (CNNs / ConvNets)", May 23, 2017. Available: https://cs231n.github.io/convolutional-networks/. Accessed on: May 20, 2018.

[58] V. R. Jonathan Huang, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, Kevin Murphy, "Speed/accuracy trade-offs for modern convolutional object detectors". Available: https://arxiv.org/pdf/1611.10012v3.pdf. Accessed on: Mar. 6, 2018.

[59] S. I. Christian Szegedy, Vincent Vanhoucke, Alex Alemi, "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning". Available: https://arxiv.org/pdf/1602.07261v2.pdf. Accessed on: May 6, 2018.

[60] A. H. Mark Sandler, Menglong Zhu, Andrey Zhmoginov, Liang-Chieh Chen, "MobileNetV2: Inverted Residuals and Linear Bottlenecks". Available: https://arxiv.org/pdf/1801.04381v3.pdf. Accessed on: May 8, 2018.

[61] V. V. Barret Zoph, Jonathon Shlens, Quoc V. Le, "Learning Transferable Architectures for Scalable Image Recognition". Available: https://arxiv.org/pdf/1707.07012v4.pdf. Accessed on: May 9, 2018.

[62] D. D. T. Mariusz Bojarski, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, Karol Zieba, "End to End Learning for Self-Driving Cars". Available: https://arxiv.org/pdf/1604.07316v1.pdf. Accessed on: May 20, 2018.

[63] A. Dertat, "Applied Deep Learning - Part 4: Convolutional Neural Networks", Nov. 8, 2017. Available: https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2.

[64] B. Z. Prajit Ramachandran, Quoc V. Le, "Searching for Activation Functions". Available: https://arxiv.org/pdf/1710.05941v2.pdf. Accessed on: Apr. 25, 2018.

[65] F. Huszár, "Dilated Convolutions and Kronecker Factored Convolutions", May 12, 2016. Available: http://www.inference.vc/dilated-convolutions-and-kronecker-factorisation/. Accessed on: May 10, 2018.

[66] E. Bendersky, "Depthwise separable convolutions for machine learning", Apr. 04, 2018. Available: https://eli.thegreenplace.net/2018/depthwise-separable-convolutions-for-machine-learning. Accessed on: May 10, 2018.

[67] A. W. Harley, "An Interactive Node-Link Visualization of Convolutional Neural Networks" *ISVC,* pp. 867-877, E-Pub. Available: https://www.cs.cmu.edu/~aharley/vis/. Accessed on: Apr. 25, 2018.

[68] D. Parthasarathy, "A Brief History of CNNs in Image Segmentation: From R-CNN to Mask R-CNN", Apr. 22, 2017. Available: https://blog.athelas.com/a-brief-history-of-cnns-in-image-segmentation-from-r-cnn-to-mask-r-cnn-34ea83205de4. Accessed on: Apr. 26, 2018.

[69] T. Grel, "Region of interest pooling explained", Feb. 28, 2017. Available: https://blog.deepsense.ai/region-of-interest-pooling-explained/. Accessed on: Apr. 26, 2018.

[70] J. Xu, "Deep Learning for Object Detection: A Comprehensive Review", Sep. 11, 2017. Available: https://towardsdatascience.com/deep-learning-for-object-detection-a-comprehensive-review-73930816d8d9. Accessed on: Apr. 26, 2018.

[71] R. Girshick, "Fast R-CNN". Available: https://arxiv.org/pdf/1504.08083v2.pdf. Accessed on: May 20, 2018.

[72] K. H. Shaoqing Ren, Ross Girshick, Jian Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". Available: https://arxiv.org/pdf/1506.01497v3.pdf. Accessed on: Mar. 6, 2018.

[73] Y. L. Jifeng Dai, Kaiming He, Jian Sun, "R-FCN: Object Detection via Region-based Fully Convolutional Networks". Available: https://arxiv.org/pdf/1605.06409v2.pdf. Accessed on: May 21, 2018.

[74] D. A. Wei Liu, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, Alexander C. Berg, "SSD: Single Shot MultiBox Detector". Available: https://arxiv.org/pdf/1512.02325v5.pdf. Accessed on: May 21, 2018.

[75] L. A. d. Santos, "Residual Net", 2017. Available: https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/residual_net.html. Accessed on: May 5, 2018.

[76] J. Xu, "An Intuitive Guide to Deep Network Architectures", Aug. 14, 2017. Available: https://towardsdatascience.com/an-intuitive-guide-to-deep-network-architectures-65fdc477db41. Accessed on: May 5, 2018.

[77] V. V. Christian Szegedy, Sergey Ioffe, Jonathon Shlens, Zbigniew Wojna, "Rethinking the Inception Architecture for Computer Vision". Available: https://arxiv.org/pdf/1512.00567v3.pdf. Accessed on: May 15, 2018.

[78] M. Z. Andrew G. Howard, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, Hartwig Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications". Available: https://arxiv.org/pdf/1704.04861v1.pdf. Accessed on: May 7, 2018.

[79] M. Hollemans, "MobileNet version 2", Apr. 22, 2018. Available: http://machinethink.net/blog/mobilenet-v2/. Accessed on: May 21, 2018.

[80] J. L. Vijay Chandrasekhar, Qianli Liao, Olivier Morère, Antoine Veillard, Lingyu Duan, Tomaso Poggio, "Compression of Deep Neural Networks for Image Instance Retrieval". Available: https://arxiv.org/pdf/1701.04923v1.pdf. Accessed on: May 21, 2018.

[81] A. Reithaug, "Evaluate multiple checkpoints.", Apr. 12, 2018. Available: https://github.com/tensorflow/models/pull/2643/files. Accessed on: Apr. 12, 2018.

[82] N. T. Ravid Shwartz-Ziv, "Opening the Black Box of Deep Neural Networks via Information". Available: https://arxiv.org/pdf/1703.00810v3.pdf. Accessed on: May 22, 2018.

[83] M. A. Seong Joon Oh, Bernt Schiele, Mario Fritz, "Towards Reverse-Engineering Black-Box Neural Networks". Available: https://arxiv.org/pdf/1711.01768v3.pdf. Accessed on: May 22, 2018.

[84] A. F. Daniel Gordon, Dieter Fox, "Re3 : Real-Time Recurrent Regression Networks for Visual Tracking of Generic Objects" *IEEE Robotics and Automation Letters 2018*. Available: https://arxiv.org/pdf/1705.06368v3.pdf. Accessed on: May 29, 2018.

[85] S. G. Reebs, "Oxygen and fish behaviour". Available: http://www.howfishbehave.ca/pdf/oxygen.pdf. Accessed on: Nov. 25, 2017.

[86] I. D. O. Fondriest Environmental, "Fundamentals of Environmental Measurements" Nov. Available: http://www.fondriest.com/environmental-measurements/parameters/water-quality/dissolved-oxygen/. Accessed on: Nov. 25, 2017.

[87] United States Environmental Protection Agency. (1986). *Quality Criteria for Water*. Available: https://nepis.epa.gov/Exe/ZyPDF.cgi/00001MGA.PDF?Dockey=00001MGA.PDF. Accessed on: Nov. 25, 2017.

[88] R. Poulin, "Parasite manipulation of host personality and behavioural syndromes" *The Journal of Experimental Biology,* vol. 216, no. 1, pp. 18-26, E-Pub. Available: http://jeb.biologists.org/content/jexbio/216/1/18.full.pdf. Accessed on: Nov. 25, 2017.

[89] A. K. M. Kevin D. Lafferty, "Altered Behavior of Parasitized Killifish Increases Susceptibility to Predation by Bird Final Hosts " *Ecology,* vol. 77, no. 5, pp. 1390-1397, E-Pub July. Available: www.jstor.org/stable/2265536. Accessed on: Nov. 25, 2017.

[90] W. W. L. C. Daniel Pauly, "Sound physiological knowledge and principles in modeling shrinking of fishes under climate change" *Global Change Biology,* Aug. Available: https://doi.org/10.1111/gcb.13831. Accessed on: Nov. 27, 2017.

[91] K. Academy, "Chain rule overview", n.d. Available: https://www.khanacademy.org/math/calculus-home/taking-derivatives-calc/chain-rule-calc/a/chain-rule-overview. Accessed on: May 17, 2018.

[92] P. Jay, "Back-Propagation is very simple. Who made it Complicated ?", Apr. 20, 2017. Available: https://medium.com/@14prakash/back-propagation-is-very-simple-who-made-it-complicated-97b794c97e5c. Accessed on: May 18, 2018.

[93] S. Ruder, "An overview of gradient descent optimization algorithms". Available: https://arxiv.org/pdf/1609.04747.pdf. Accessed on: May 22, 2018.

10 Appendix

A Backpropagation

A.1 Calculating Derivatives

Assume that the functions A, B, and C are:

$$A(x) = \sin(x)\,, B(x) = e^x, C(x) = x^2 + x,$$

then their derived forms would be:

$$A'(x) = \cos(x)\,, B'(x) = e^x, C'(x) = 2x + 1.$$

Thus, according to the chain rule, the derivative of the composition is:

$$f'(x) = A'\left(B\big(C(x)\big)\right) * B'\big(C(x)\big) * C'(x),$$

which when adding the derivatives, creates the expression [91]:

$$f'(x) = \cos\left(e^{x^2+x}\right) * e^{x^2+x} * (2x + 1)$$

A.2 Example of Forward Pass

Some clarifications before moving on to the example:

1) The color in the notations represent the colors in *Fig. 9* and *Fig. 59*.
2) The weights *W* are completely random, and bias *b* is always 1.0.
3) ReLU is applied on the first layer, sigmoid on the second layer, and softmax on the third.
4) The *in* values are always followed by *out* values from either ReLU, sigmoid, or softmax.
5) The *in* and *out* values have the same color, since they interact with the same neuron.
6) e is Euler's number, and is a constant with value 2.71828.
7) The desired output from the network for this example is $[0.0 \quad 1.0]$.
8) Refer to *Fig. 59* for a full overview of the network with complete results.

The concepts on forward- and backward pass are inspired by [92].

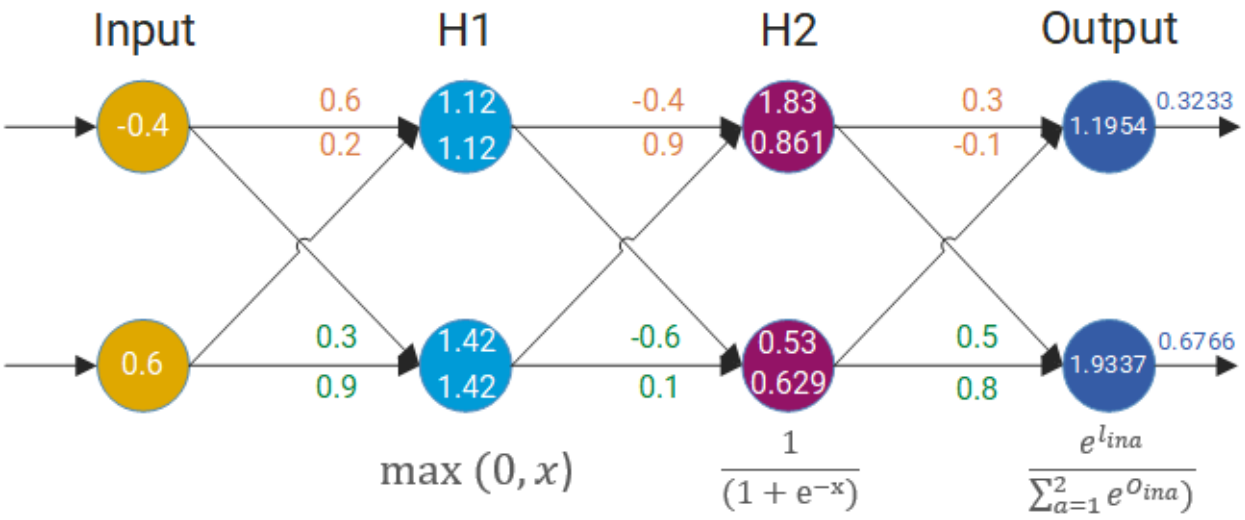

Figure 59: Results from a single forward pass in a neural network

First Layer:

- Matrix operation:

$$[i_1 \quad i_2] \times \begin{bmatrix} W_{i1j1} & W_{i1j2} \\ W_{i2j1} & W_{i2j2} \end{bmatrix} + [b_{j1} \quad b_{j2}] = [H1_{in1} \quad H1_{in2}]$$

$$[-0.4 \quad 0.6] \times \begin{bmatrix} 0.6 & 0.3 \\ 0.2 & 0.9 \end{bmatrix} + [1.0 \quad 1.0] = [1.12 \quad 1.42]$$

- Matrix multiplication (not shown for any other layers):

$$c_{11} = -0.4 * 0.6 + 0.6 * 0.2 = -0.12 + 1.0 = 1.12$$

$$c_{12} = -0.4 * 0.3 + 0.6 * 0.9 = 0.42 + 1.0 = 1.42$$

- ReLU operation ($\max(0, x)$):

$$[H1_{out1} \quad H1_{out2}] = [\max(0, H1_{in1}) \quad \max(0, H1_{in2})]$$

$$[H1_{out1} \quad H1_{out2}] = [1.12 \quad 1.42]$$

Second Layer:

- Matrix operation:

$$[H1_{out1} \quad H1_{out2}] \times \begin{bmatrix} W_{j1k1} & W_{j1k2} \\ W_{j2k1} & W_{j2k2} \end{bmatrix} + [b_{k1} \quad b_{k2}] = [H2_{in1} \quad H2_{in2}]$$

$$[1.12 \quad 1.42] \times \begin{bmatrix} -0.4 & -0.6 \\ 0.9 & 0.1 \end{bmatrix} + [1.0 \quad 1.0] = [1.83 \quad 0.53]$$

- Sigmoid operation ($\frac{1}{(1+e^{-x})}$):

$$[H2_{out1} \quad H2_{out2}] = \left[\frac{1}{(1 + e^{-H2_{in1}})} \quad \frac{1}{(1 + e^{-H2_{in2}})}\right]$$

$$[H2_{out1} \quad H2_{out2}] = [0.861 \quad 0.629]$$

Third Layer:

- Matrix operation:

$$[H2_{out1} \quad H2_{out2}] \times \begin{bmatrix} W_{k1l1} & W_{k1l2} \\ W_{k2l1} & W_{k2l2} \end{bmatrix} + [b_{l1} \quad b_{l2}] = [O_{in1} \quad O_{in2}]$$

$$[0.861 \quad 0.629] \times \begin{bmatrix} 0.3 & 0.5 \\ -0.1 & 0.8 \end{bmatrix} + [1.0 \quad 1.0] = [1.1954 \quad 1.9337]$$

- Softmax operation ($\frac{e^{l}ina}{\sum_{a=1}^{2} e^{O}ina)}$):

$$[O_{out1} \quad O_{out2}] = \left[\frac{e^{O_{in1}}}{\sum_{a=1}^{2} e^{O_{ina}})} \quad \frac{e^{O_{in2}}}{\sum_{a=1}^{2} e^{O_{ina}})}\right]$$

$$[O_{out1} \quad O_{out2}] = [0.3233 \quad 0.6766]$$

The actual output should be $[0.0 \quad 1.0]$, but the output was $[0.3233 \quad 0.6766]$.

The network's error can be calculated by using a loss function, such as cross-entropy, where *y* is the actual output (0 or 1), *n* is batch size (1), and *M* is number of classes (2):

$$-\frac{1}{n}\left(\sum_{i=1}^{M}\left(y_i * \ln(O_{outi})\right) + \left((1-y_i) * \ln(1-O_{outi})\right)\right)$$

$$-\frac{1}{1}\left((0 * \ln(0.3233)) + ((1-0) * \ln(1-0.3233)) + (1 * \ln(0.6766)) + ((1-1) * \ln(1-0.6766))\right)$$

$$= -(\ln(0.6767) + \ln(0.6767))$$

$$= -(-0.3906 - 0.3906)$$

$$= 0.7812$$

A perfect network would output an error of 0 when using cross-entropy, but this result is not bad either, and the network was able to predict the correct class with a prediction of 0.6766 (out of 1). However, it is certainly possible to improve both these scores, which is accomplished in the backwards pass.

A.3 Example of Backward Pass

Recall from *Fig. 8* that during a backward pass, one uses the derivatives of the values in the forward pass. Note: For space and relevance, only the first step (i.e. calculations from the output layer to the second hidden layer will be computed). The steps for the other layers are mostly the same. The colored values below are directly linked to the values in the forward process.

- Matrix and values of cross-entropy derivatives:

$$\begin{bmatrix} \frac{\partial E_1}{\partial O_{out1}} \\ \frac{\partial E_2}{\partial O_{out2}} \end{bmatrix} = \begin{bmatrix} -1 * \left(y_1 * \left(\frac{1}{O_{out1}} \right) + (1 - y_1) * \left(\frac{1}{1 - O_{out1}} \right) \right) \\ -1 * \left(y_2 * \left(\frac{1}{O_{out2}} \right) + (1 - y_2) * \left(\frac{1}{1 - O_{out2}} \right) \right) \end{bmatrix}$$

$$= \begin{bmatrix} -1 * \left(0 * \left(\frac{1}{0.3233} \right) + (1 - 0) * \left(\frac{1}{1 - 0.3233} \right) \right) \\ -1 * \left(1 * \left(\frac{1}{0.6766} \right) + (1 - 1) * \left(\frac{1}{1 - 0.6766} \right) \right) \end{bmatrix} = \begin{bmatrix} -1.4777 \\ -1.4777 \end{bmatrix}$$

- Matrix and values of derivative of softmax:

$$\begin{bmatrix} \frac{\partial O_{out1}}{\partial O_{in1}} \\ \frac{\partial O_{out2}}{\partial O_{in2}} \end{bmatrix} = \begin{bmatrix} \frac{e^{O_{in1}} * e^{O_{in2}}}{(e^{O_{in1}} + e^{O_{in2}})^2} \\ \frac{e^{O_{in2}} * e^{O_{in1}}}{(e^{O_{in1}} + e^{O_{in2}})^2} \end{bmatrix} = \begin{bmatrix} \frac{e^{1.1954} * e^{1.9337}}{(e^{1.1954} + e^{1.9337})^2} \\ \frac{e^{1.9337} * e^{1.1954}}{(e^{1.1954} + e^{1.9337})^2} \end{bmatrix} = \begin{bmatrix} 0.2188 \\ 0.2188 \end{bmatrix}$$

- Values of derivative of input to output layer with respect to weights:

$$\begin{bmatrix} \frac{\partial O_{in1}}{\partial W_{k1l1}} \\ \frac{\partial O_{in1}}{\partial W_{k2l1}} \end{bmatrix} = \begin{bmatrix} H2_{out1} \\ H2_{out2} \end{bmatrix} = \begin{bmatrix} 0.861 \\ 0.629 \end{bmatrix}$$

$$\begin{bmatrix} \frac{\partial O_{in2}}{\partial W_{k1l2}} \\ \frac{\partial O_{in2}}{\partial W_{k2l2}} \end{bmatrix} = \begin{bmatrix} H2_{out1} \\ H2_{out2} \end{bmatrix} = \begin{bmatrix} 0.861 \\ 0.629 \end{bmatrix}$$

- Matrix form of all derivatives in the third layer (H2):

$$\partial W_{kl} = \begin{bmatrix} \frac{\partial E_1}{\partial W_{k1l1}} & \frac{\partial E_2}{\partial W_{k1l2}} \\ \frac{\partial E_1}{\partial W_{k2l1}} & \frac{\partial E_2}{\partial W_{k2l2}} \end{bmatrix} = \begin{bmatrix} \frac{\partial E_1}{\partial O_{out1}} * \frac{\partial O_{out1}}{\partial O_{in1}} * \frac{\partial O_{in1}}{\partial O_{k1l1}} & \frac{\partial E_2}{\partial O_{out2}} * \frac{\partial O_{out2}}{\partial O_{in2}} * \frac{\partial O_{in2}}{\partial O_{k1l2}} \\ \frac{\partial E_1}{\partial O_{out1}} * \frac{\partial O_{out1}}{\partial O_{in1}} * \frac{\partial O_{in1}}{\partial O_{k2l1}} & \frac{\partial E_2}{\partial O_{out2}} * \frac{\partial O_{out2}}{\partial O_{in2}} * \frac{\partial O_{in2}}{\partial O_{k2l2}} \end{bmatrix}$$

$$\partial W_{kl} = \begin{bmatrix} \partial W_{k1l1} & \partial W_{k1l2} \\ \partial W_{k2l1} & \partial W_{k2l2} \end{bmatrix} = \begin{bmatrix} -1.4777 * 0.2188 * 0.861 & -1.4777 * 0.2188 * 0.861 \\ -1.4777 * 0.2188 * 0.629 & -1.4777 * 0.2188 * 0.629 \end{bmatrix}$$

$$= \begin{bmatrix} -0.2783 & -0.2783 \\ -0.2033 & -0.2033 \end{bmatrix}$$

- The new weights in-between the second hidden layer and output layer are thus (with a learning rate *lr* of 0.1):

$$\acute{W}_{kl} = \begin{bmatrix} W_{k1l1} - (lr * \partial W_{k1l1}) & W_{k1l2} - (lr * \partial W_{k1l2}) \\ W_{k2l1} - (lr * \partial W_{k2l1}) & W_{k2l2} - (lr * \partial W_{k2l2}) \end{bmatrix}$$

$$= \begin{bmatrix} 0.3 - (0.1 * -0.2783) & 0.5 - (0.1 * -0.2783) \\ -0.1 - (0.1 * -0.2033) & 0.8 - (0.1 * -0.2033) \end{bmatrix}$$

$$= \begin{bmatrix} 0.32783 & 0.52783 \\ -0.07967 & 0.82033 \end{bmatrix}$$

The backward pass for the other layers are similar to this, except the values and functions are different. Once all weights have been updated, one can run the forward pass again and check whether the updated weights caused the new output to be closer to the actual output. This process is done repeatedly until sufficient results are achieved. Keep in mind that this network is incredibly small, and with more layers and more neurons, the matrices grow much larger.

B Experiments

Definitions of new terms are covered in this chapter for understanding the parameter values for the presented networks. Please refer to the Glossary for any other definition.

The image resizer is responsible for altering the dimensions of the input image. The networks use either keep aspect ratio (AR) or fixed shape (FS).

Weights are initialized with a truncated normal distribution (TND) with standard deviation σ (default = 1.0). This slices off values outside the upper and lower bounds.

The following gradient descent optimization algorithms were used:

- Momentum [93], where the momentum value is specified in parentheses.
- RMSProp [93].
- Adaptive Moment Estimation (Adam) [93].

The learning rate controls how much (or how fast) the weights in the network are adjusted, where a network with a higher learning rate changes its mind more quickly. Ideally, one would like to use as low learning rate as possible, but this significantly increases the required time spent training. Most networks below use a fixed learning rate (i.e. does not change based on steps iterated).

The gradient clipping (GC) limits the magnitude of the gradient, so they do not become exponentially large (exploding gradients, opposite of vanishing gradients).

Data augmentations are specific methods which alter the input images in some way, either to generate more training data, or make the network more robust to various input sizes and shapes.

Four data augmentations are presented:

- Random Horizontal Flip (RHF): Randomly flips the image and its detections horizontally 50% of the time.
- Random Rotation 90 (RR90): Randomly rotates the image and its detections 90 degrees 50% of the time.
- SSD Random Crop (SRC): Augments the image based on [74].
- Random Crop Image (RCI): Randomly crops the image 50% of the time.

When depthwise separable convolution is used, the number in parentheses refer to the depth multiplier, which is the factor/ratio to alter the depth of the channels.

For hard negative mining, the following parameters are altered:

- Maximum negatives per positive (MAX NPP): Maximum number of negatives to retain for each positive anchor (bounding box).
- Minimum negatives per image (MIN NPI): Minimum number of negative anchors to sample for a given image.

B.1 Single Shot MultiBox Detector

B.1.1 Parameters for Inception

Type	Parameter	SSD Inception V2 @1 (0.8464)	SSD Inception V2 @3 (0.8190)	SSD Inception V3 @1 (0.7208)
Resizer	Fixed Shape	600x600	300x300	300x300
Box Predictor	Activation	ReLU	ReLU	ReLU
	Filter Size	3	1	3
	TND σ	0.03	0.1	0.03
Training	Batch Size	10	20	24
	Optimizer	Adam	Adam	RMSProp (0.9)
	Learning Rate	Manual Step 0k: 0.001	Manual Step 0k: 0.0006	Exponential Decay 0k: 0.004
	Gradient Clip	5.0	7.5	0.0
	Augmentations	RHF, SRC	RHF, SRC, RR90	RHF, SRC

Table 23: SSD Inception V2 (@1, @3), V3 (@1) specifications

B.1.2 Parameters for MobileNet

Type	Parameter	SSD MobileNet V1 @1 (0.7723)	SSD MobileNet V2 @1 (0.7882)
Resizer	Fixed Shape	300x300	300x300
Box Predictor	Activation	ReLU	ReLU
	Filter Size	1	3
Architecture	Depthwise	False	True (1.0)
Loss	Max NPP	3	3
	Min NPI	0	3
Training	Batch Size	32	24
	Optimizer	Adam	RMSProp (0.9)
	Learning Rate	Manual Step 0k: 0.001	Exponential Decay 0k: 0.004
	Gradient Clip	5.0	0.0
	Augmentations	RHF, SRC	RHF, SRC

Table 24: SSD MobileNet V1 (@1), V2 (@1) specifications

B.2 Faster Region-Based Convolutional Neural Network

B.2.1 Parameters for Inception

Type	Parameter	FRCNN Inception V2 @2 (0.8378)	FRCNN Inception V2 @1 (0.8277)	FRCNN Inception ResNet V2 @1 (0.0050)
Resizer	Aspect Ratio	AR 600 – 1024	AR 600 – 1024	AR 600 – 1024
Model	Initial Crop Size	14	14	17
	MaxPool Filter Size	2	2	1
	MaxPool Stride	2	2	1
Training	Batch Size	1	1	1
	Optimizer	Momentum (0.8)	Momentum (0.9)	Momentum (0.9)
	Learning Rate	Manual Step 0k: 0.0007	Manual Step 0k: 0.0005 90k: 0.00005 120k: 0.000008	Manual Step 0k: 0.00006
	Gradient Clip	8.0	10.0	10.0
	Augmentations	RHF	RHF	RHF

Table 25: FRCNN Inception V2 (@2, @1), Inception ResNet V2 (@1) specifications

B.2.2 Parameters for ResNet 152

Type	Parameter	FRCNN ResNet 152 @4 (0.8362)	FRCNN ResNet 152 @6 (0.7955)	FRCNN ResNet 152 @1 (0.6667)
Resizer	Aspect Ratio	AR 600 – 1024	AR 600 – 1024	AR 600 – 1024
Training	Batch Size	1	1	1
	Optimizer	Adam	Adam	Momentum (0.9)
	Learning Rate	Manual Step 0k: 0.0003	Manual Step 0k: 0.0003	Manual Step 0k: 0.0003
	Gradient Clip	10.0	10.0	10.0
	Augmentations	RHF	RHF, RR90	RHF

Table 26: FRCNN Inception ResNet 152 (@4, @6, @1) specifications

Type	Parameter	FRCNN ResNet 152 @2 (0.5044)	FRCNN ResNet 152 @9 (0.0053)	FRCNN ResNet 152 @8 (0.0015)
Resizer	Aspect Ratio	AR 600 – 1920	AR 600 – 1024	AR 600 – 900
Training	Batch Size	1	1	1
	Optimizer	Adam	Adam	Adam
	Learning Rate	Manual Step 0k: 0.0003	Manual Step 0k: 0.0009	Manual Step 0k: 0.0007
	Gradient Clip	10.0	2.0	7.0
	Augmentations	RHF, RR90	RHF	RHF, RCI, RR90

Table 27: FRCNN Inception ResNet 152 (@2, @9, @8) specifications

C Model Graphs

C.1 Single Shot MultiBox Detector (SSD)

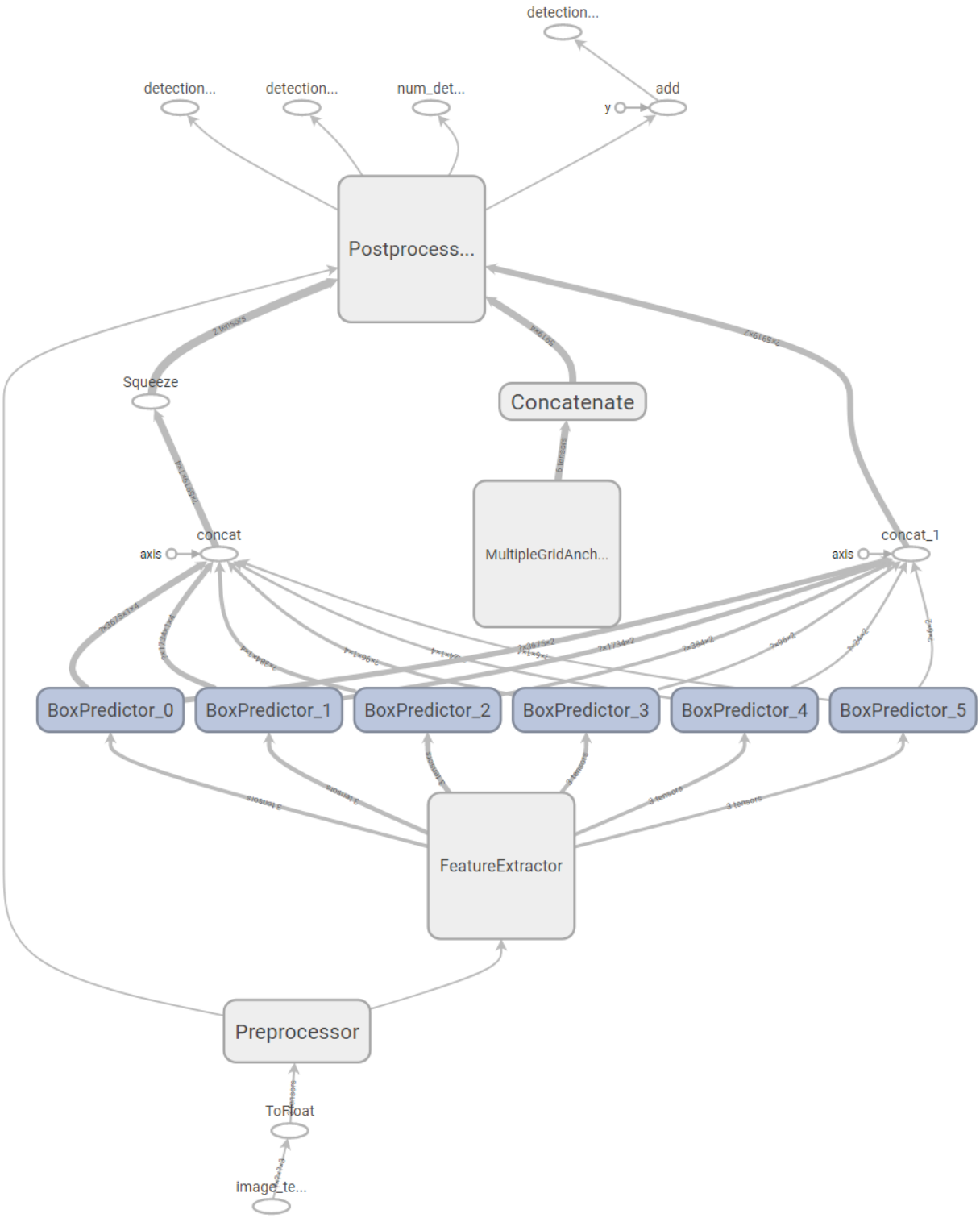

Figure 60: SSD model

C.2 Faster Region-Based Convolutional Neural Network (Faster R-CNN)

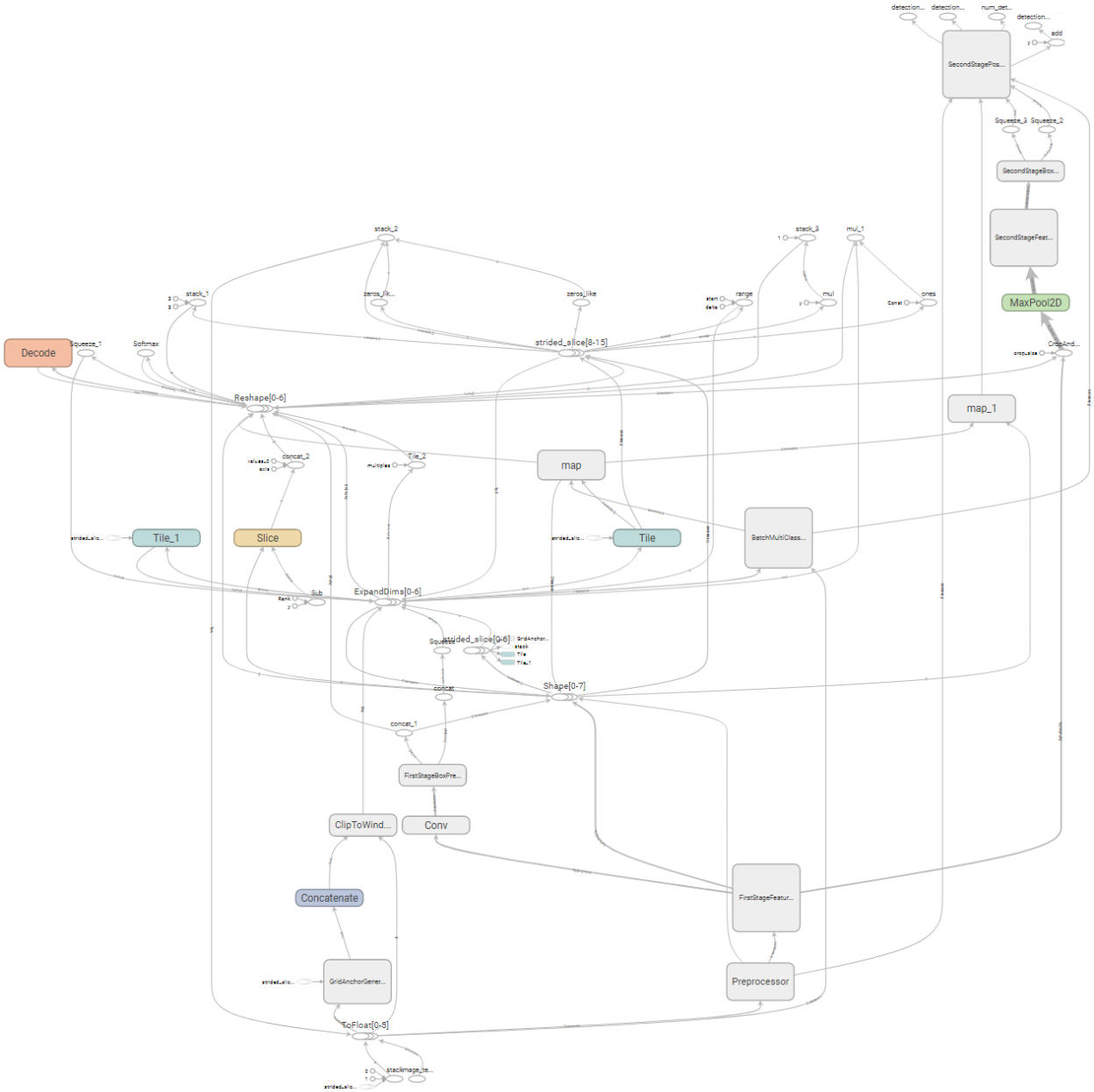

Figure 61: Faster R-CNN model

C.3 Region-Based Fully Convolutional Neural Network (R-FCN)

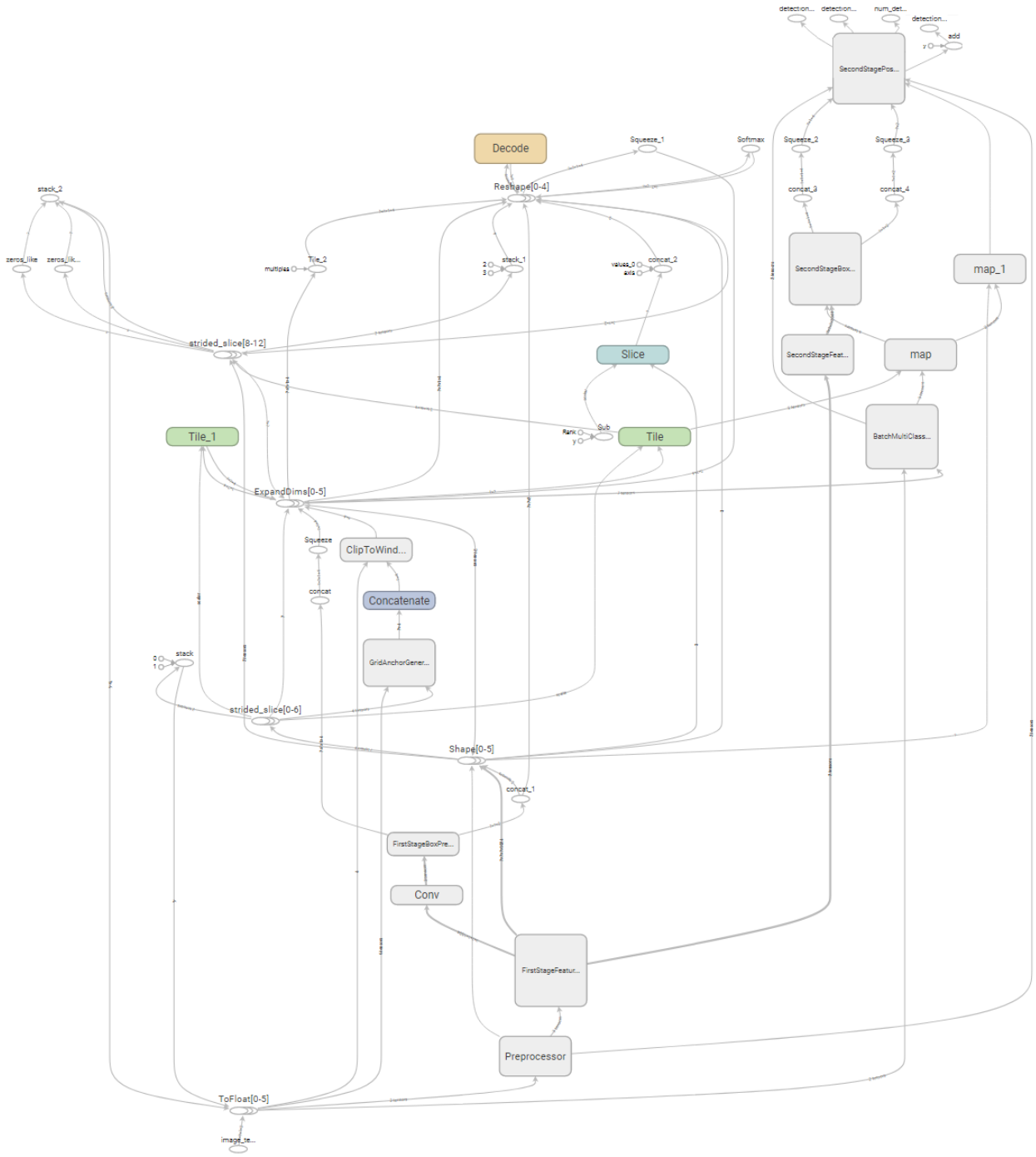

Figure 62: R-FCN model