UPPSALA
UNIVERSITET

UPTEC W 20037
Examensarbete 30 hp
Juli 2020

Automatic identification of northern pike (Exos Lucius) with convolutional neural networks

Axel Lavenius

Abstract

Automatic identification of northern pike (Exos Lucius) with convolutional neural networks

Axel Lavenius

The population of northern pike in the Baltic sea has seen a drastic decrease in numbers in the last couple of decades. The reasons for this are believed to be many, but the majority of them are most likely anthropogenic. Today, many measures are being taken to prevent further decline of pike populations, ranging from nutrient runoff control to habitat restoration. This inevitably gives rise to the problem addressed in this project, namely: how can we best monitor pike populations so that it is possible to accurately assess and verify the effects of these measures over the coming decades?

Pike is currently monitored in Sweden by employing expensive and ineffective manual methods of individual marking of pike by a handful of experts. This project provides evidence that such methods could be replaced by a Convolutional Neural Network (CNN), an automatic artificial intelligence system, which can be taught how to identify pike individuals based on their unique patterns. A neural net simulates the functions of neurons in the human brain, which allows it to perform a range of tasks, while a CNN is a neural net specialized for this type of visual recognition task. The results show that the CNN trained in this project can identify pike individuals in the provided data set with upwards of 90% accuracy, with much potential for improvement.

Handledare: Torbjörn Dunér
Ämnesgranskare: Gabriele Messori
Examinator: Alexandru Tatomir
ISSN: 1401-5765, UPTEC W 20037
Detta examensarbete är sekretessbelagt enligt avtal mellan projektägare, student och Uppsala universitet.
Tryckt av: Publicerad digitalt vid Institutionen för geovetenskaper, Uppsala universitet

Acknowledgements

I would like to thank Torbjörn Dunér who recruited me for this project, and who has been a steady and reliable support from day one. I also have to recognize the invaluable assistance from my supervisor Mattias Åkesson who quickly became my ML mentor, consistently guiding me and the project in the right direction. Furthermore, I want to thank Gabriele Messori for his work in thoroughly reviewing the report, and for all his constructive feedback and helpful advice. Last but not least, I would like to thank all the members of the Pic-a-pike crew who, apart from providing the necessary resources, data, and high doses of enthusiasm for everything pike, also made sure to support and encourage me throughout.

Populärvetenskaplig sammanfattning

Östersjön är idag hårt ansatt av människans härjningar från jordbruk, industrier, kommersiellt fiske, och många andra aktiviteter. När känsliga eller värdefulla naturområden så som Östersjön diskuteras, talas det ofta om nyckelarter som är särskilt betydelsefulla för att säkerställa balans och stabilitet inom dessa naturområdens ekosystem. Predatorer högt upp i näringskedjan brukar inräknas bland dessa nyckelarter eftersom de kraftigt kan reglera bestånden av andra arter i ekosystemet, och i sjöar och hav antas dessa roller ofta av stora rovfiskar. Torsken och dess kraftigt minskade levnadskraftiga bestånd i Östersjön har historiskt sett varit i fokus, men även gäddan är en sådan rovfisk som idag minskar i antal.

På grund av sin utbredning i svenska sjöar och kustområden utgör gäddan en mycket viktig komponent i många svenska ekosystem. Utöver sina ekologiska värden är det samtidigt ytterst tveksamt om någon fisk är mer beryktad, eller har en större plats i den svenska kulturen, än gäddan. Med sin torpedliknande form, glupska aptit och inte minst sina kannibalistiska tendenser intar den status som rovfiskarnas rovfisk varhelst den förekommer. Rimligtvis är det därför varken som nyckelart eller matfisk, utan istället som jagad trofé av sportfiskare, som gäddan framförallt uppskattas och vördas. I gäddfiskarens värld hägrar ständigt nästa vidunder i vassen, och löftet om ett häftigt hugg följt av en utdragen kamp lockar årligen hundratusentals sportfiskare – inhemska så som utländska - ut i svenska vatten. Den näring som detta ger upphov till omsätter miljardbelopp, och för den som någonsin ämnat införskaffa modern fiskeutrustning borde inte detta förefalla särskilt förvånande.

I och med upptäckten av att bestånden av gädda minskar i Östersjöområden investeras det nu mycket resurser på att bevara populationerna och skydda gäddan från vidare påverkan. Projekt inriktas nu till exempel på återställning av kustnära våtmarker vilka är uppväxtmiljöer för gäddor, så kallade ”gäddfabriker”. Parallellt med sådana åtgärder ökar samtidigt kraven på noggranna populationsmätningar så att man kan följa gäddans utveckling, och därtill styrka effektiviteten hos de åtgärder som hittills gjorts. För att kartlägga populationernas storlek och tillstånd behövs dock betydligt mer kunskap om antalet individer, och information om till exempel ålder och storlek hos fiskarna. Denna information har hittills erhållits via provfisken,

där professionella fiskare ger sig ut och fångar och fysiskt märker så många gäddor de kan vid olika tillfällen på året. Dessa metoder är ineffektiva, kostsamma, och genererar förhållandevis lite data. Det är inom detta område som detta arbete har undersökt om inte detta föråldrade manuella system kan ersättas av automatiserande metoder där istället gäddor fotograferas och identifieras visuellt av datorer.

Forskning har visat att gäddindivider har unika mönsterteckningar som de kan identifieras via, och under de senaste åren har datorer kunnat tränas för bildigenkänning av precis sådan karakteristik. Inte minst finger- och ansiktes igenkänning har visat sig utgöra relativt enkla uppgifter för datorer, som dessutom effektivt kunnat implementeras i var människas vardag via mobiltelefonen. Hemligheten bakom denna utveckling – som också ligger bakom framgångar så som Teslas självkörande bilar – heter Artificiell Intelligens (AI), och särskilt den del inom AI som kallas maskininlärning och neurala nätverk. Maskininlärning (ML) innefattar principen att låta system tillämpa självinlärande algoritmer för att tränas på att utföra uppgifter utifrån mycket stora datamängder, och neurala nätverk är ML-system med den specifika twisten att de också efterliknar neuronerna i en biologisk hjärna. För detta projekt visade det sig just att ett neuralt nätverk, och då ytterligare mer specifikt att ett "Convolutional Neural Network" (CNN), kunde implementeras för att känna igen bilder tagna på gäddor av samma individ. Förenklat gjordes detta möjligt genom att ett tusental bilder på många olika individer gäddor först sorterades, och därefter tilläts nätverket träna på att själv sortera in bilderna på samma sätt. På så vis har nätverket hela tiden ett facit att förhålla sig till, där det belönas för korrekta identifieringar, och bestraffas för inkorrekta sådana, till dess att det helt enkelt gör statistiskt bättre och bättre försök att sortera in bilderna.

När nätverket till sist testades på helt nya bilder som den inte tränat på, gavs lovande resultat. Nätverket lyckades sortera 9 av 10 nya bilder korrekt, och då ska det här ändå betraktas som ett ganska simpelt CNN. Utöver att detta CNN har mycket stor utvecklingspotential, skulle det också vara möjligt att tillämpa andra nydanande metoder för bildigenkänning av djurindivider med CNN. När detta görs parallellt med en fortsatt och utvidgad insamling av gäddbilder, så finns det mycket god potential för att till slut helt och hållet fasa ut rådande metoder för populationsmätning av gädda, till förmån för ett automatiserat system.

Contents

Acronyms and keywords

Semantic segmentation An image processing method where for example an object of interest is cut out and separated from the raw image

Ground truth The "key" presented to the segmentation network which represents the desired segmentation output

IoU Intersection over Union: A metric with which to determine how well an image was segmented based on its overlap with the ground truth

NN Neural network: a machine learning system that imitates biological neurons to find patterns within large sets of data

Weights Values that neurons learn and are assigned during training, which determine how input is processed and ultimately how the network performs

FC-layer Fully Connected layer: contains the weights of the classic neural network, where each weight in a layer is connected to all weights in the previous and next layer

Loss function A function that calculates how well the network predicts solutions based on the assigned values of weights. A low loss infers better predictions

Gradient descent The standardized concept behind self-learning algorithms. The loss function is a function of weights, and gradient descent occurs when it is minimized by changing the weights in the direction of the loss function's negative gradient

Optimizer Algorithms that alter the way the solution approaches its minimum values during gradient descent **accuracy** A metric which assigns a percentage to how many of the networks predictions were correct

Image classifier A NN which is able to "look" at images and determine their content in the form of pre-defined sets of classes (for example objects, or different human individuals).

CNN Convolutional Neural Network: A type of NN specialized in multi-dimensional input, such as images, which makes it suitable for image classification tasks

Convolutional filter The main constituent of CNN:s. Is essentially a matrix of weights which can be superimposed on image input to extract spatial features

CONV-layer Convolutional layer: same principle as with the FC-layer, only consisting of convolutional filters instead

Network structure Refers to the properties of a NN with respect to how many layers it is composed of, how they are structured, and what types of parameters are utilized in its setup

VGG Visual Geometry Group: Refers to the specific network structure developed by Oxfords Visual Geometry Group, which is characterized by its simplicity and straight forward design.

1 Introduction

1.1 Decline in Baltic Sea pike

Anthropogenic activities in and around the Baltic Sea, such as over fishing and eutrophication, are rapidly changing the Baltic Sea ecosystem (Larsson, Tibblin, and Koch-Schmidt 2015). One imminent problem is the declining populations of predatory fish, since predatory fish constitute important parts of complex nutrient web systems. When nutrient web systems are significantly disrupted, such as with the dramatically reduced populations of Cod over the past 50 years, trophic cascading effects can be observed on the whole ecosystem. Currently, Cod is not the only concern regarding fish populations in the Baltic Sea, as studies also show that populations of northern Pike (*Exos Lucius*) are in decline (Berggren 2019).

The northern Pike holds great values in many Swedish freshwater and coastal systems in terms of both ecological and economical functions, as it is popular for both commercial and recreational fishing. With the current decline of pike in the Baltic Sea, actions directed at preserving pike populations - such as wetland restoration - are key in preventing further loss of these values (Bryhn et al. 2019). Simultaneously, it is equally important to employ a functioning and effective system to monitor pike populations, so that the effects of preservation measures can be assessed and verified over time. However, methods typically used for systematically quantifying populations of various species of fish do not work well for pike. Because of its living habits, it rarely gets caught in fishing nets, and neither fishing with explosives nor electric fishing work on fully grown specimen. Instead, pike is usually caught by rod-fishing (angling), where the pike is physically marked before being released. This is a system which relies heavily on recapturing the same marked specimen at later occasions. Additionally, it consists of heavy manual work which occupies hundreds of people every season, while still not generating particularly large sets of data.

1.2 Problem formulation

Instead of relying on a system in which pike is identified through manually marking specimen through organized angling, it is thought that this could be done through identification of the unique patterns of individual pike. Already in 1982 it was shown in a study that individuals of pike could be identified based on their patterns (Fickling 1982), but the manual identification methods would be highly impractical for several reasons. A recent study, utilizing a partially automatic software (still requires continual human input), further proves the potential of pike-pattern analysis for individual pike recognition (Kristensen et al. 2020). Today, there is however no reason to believe that pattern recognition can not be performed on a fully automatic scale.

Methods of automatic pattern recognition are currently used for other species of animals, where much progress has been made only in the past few years via advances in machine learning algorithms. The success of many of these projects altogether proves that machine learning, and in particular convolutional neural networks (a type of Artificial Neural Network), can be implemented to automatically classify individuals of animal species with excellent precision, even for problems that seem very difficult. Examples of such projects are entries to Kaggle competitions which tasked its competitors to classify individuals of whales with limited quality and quantity of data.[1] [2]. For other projects, more general algorithms have been implemented which are able to identify individuals within populations of different species.[3] [4] The previous success of convolutional neural networks in classifying individuals of animal species speaks in strong favor of the possibility for applying such methods to classification of pike individuals.

1.3 Project goal

The goal of this project is to develop an automatic system which, via a convolutional neural network, can take images of pike and with a high confidence classify them as specific individuals within a data set based on their unique patterns. An automatic system could replace the currently expensive and ineffective methods of pike monitoring used in coastal areas of Sweden, thereby greatly benefiting preservation efforts.

[1]https://www.kaggle.com/c/whale-categorization-playground

[2]https://www.kaggle.com/c/noaa-right-whale-recognition

[3]https://www.wildbook.org/doku.php

[4]https://www.groundai.com/project/similarity-learning-networks-for-animal-individual-re-identification-beyond-the-capabilities-of-a-human-observer9272/2

2 Theory

2.1 Pike patterns

The pike pattern is characterized by specks of lighter colors (typically yellow), which are often round or oval shaped, on a background of primarily green and brown (see Fig 1). The previous works on the subject, primarily Fickling (1982) and Kristensen et al. (2020), indicate that there are two important general properties of pike patterns to consider for an identification task. Firstly, it has been shown that the pattern significantly differs in shapes and structures between a pike-individual's right and left sides. Secondly, it is important to consider that a pikes pattern may change over its lifespan. Research has shown that the pattern remains stable over at least 1.5 years, but probably longer than that (Kristensen et al. 2020). Since some pike live for 20-30 years according to Bryhn et al. (2019), there is however no evidence to support that the pattern is stable over the pike's whole lifespan.

For this project, no particular part of the pike is specifically targeted for analysis, but the fish body (abdomen and back areas) is overall prioritized as it provides the most pattern information. The fins do however provide some pattern information, and since shapes of fins and mouth, and eye position, can be characteristic features, it seems that the best approach is to include as much of the pike as possible.

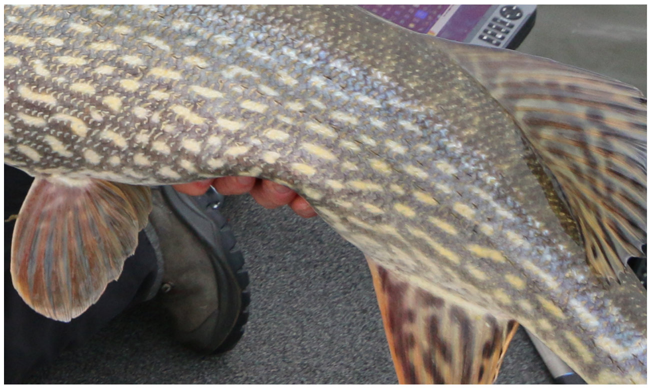


Figure 1: The pike-pattern on the left side posterior abdomen and back of a quite large pike specimen. Note also the patterns on the dorsal and anal fins

2.2 Artificial neural networks

Artificial neural networks (ANNs), often referred to as just neural networks (NNs), are computational systems which originated from attempts to replicate the functions of the nervous system, hence the term "neural" (Rosenblatt 1958). Much like the nervous system with its clusters of connected neurons, NNs consist of sets of connected nodes, of which some receive information on one end (i.e. input), transform it via some computations, and pass it on via other sets of nodes which repeat the process. When the input has been passed through all the nodes of the network and finally arrives at the end of the network, a decision is made (i.e. output) based on the now transformed input. This decision can be anything from taking an action, to calculating a maths problem or detecting the contents of an image - much like the range of tasks governed by the human nervous system. Therefore, the function of a NN is in principle similar to the function of a nervous system, and the nodes of NNs are commonly referred to as neurons. However, biological neurons, and the nervous system as a whole, are in the end far more complex than any artificial networks and neurons currently developed. Therefore this analogy is generally only used in the highest level descriptions of NNs. (Schmidhuber 2015)

2.2.1 General structure

A standard NN has the neurons divided into a layered structure, with an input layer and an output layer (visible layers), and so called "hidden layers" in between, see Figure 2. For the standard NN, each neuron in a layer connects to all the neurons in the preceding and subsequent layer, referred to as "Fully Connected" (FC), or Dense, layers.

A neuron receives an input - either from data in the input layer, or from a previous neuron in a hidden layer - multiplies it with a real value called a "weight" assigned to the neuron and adds a "bias term". Summed up for all neurons, this can be formulated as:

$$f(X, W) = \sum_{i}^{n} x_i \cdot w_i + b \tag{1}$$

where X contains all input values, and W all weights w_i and biases b for the n neurons.

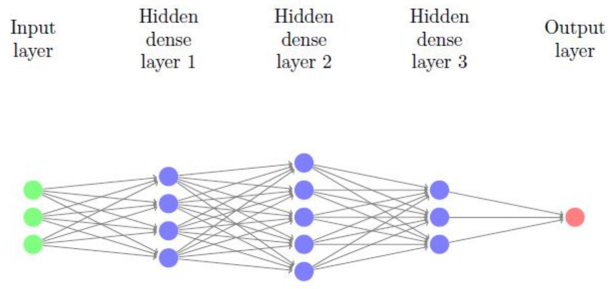


Figure 2: A classic fully connected neural network with an input layer, three hidden layers, and an output layer

Thereafter, the output of each neuron is transformed via an activation function before it is passed on as part of the system's "forward pass". The transformed output can either be passed on directly to the output layer as the final output of the model, or further on to the neurons of a subsequent layer, which repeat the same computational process.

2.2.2 Gradient descent and back propagation

The artificial intelligence, i.e. learning, of NN:s occurs through algorithms aimed at automatically improving the prediction performance of NN models during run time. One way, which has become standardized for modern machine learning aimed at, for example, classification problems, is achieved through the concepts of ”gradient descent” and ”back propagation” (Lecun et al. 1998). A function is set to measure prediction quality in the final layer, referred to as a ”cost”, or ”loss” function. The loss function assigns high values to poor predictions and vice versa, thus introducing a minimization problem with respect to ”learning” the optimal set of weights in the final hidden layer which reduces the value of the loss function. Minimization is achieved through having the values of the weights adjusted in the negative direction of the gradient, i.e. for the solution to move in the direction in which it decreases the fastest - thus gradient ”descent”. Since the values of the weights in each layer are a function of the weights in the previous layer, the calculation is performed through applying the chain rule repeatedly for each layer while going backwards in the network, which is referred to as ”back propagation”.

2.3 Convolutional neural networks

For image classification problems, the most common approach today is implementing a type of ANN called convolutional neural network (CNN) (Goodfellow, Bengio, and Courville 2016). The Keras API from the open source platform Tensorflow was used in programming a CNN in Python language for this project.

The CNN takes data with one or more spatial dimension as input, such as images (two dimensions), and learns to detect spatial patterns which in the end are combined to predict some type of identity trait of an input image. For image classification problems, the model is set to learn the connections between identity traits of images and a set of classes (objects, people, etc.), transforming raw pixel data into class scores in the output layer.

2.3.1 Convolution filters and operations

A problem with images as input in a NN, is that the input is very large, even for a low resolution image (for example: a grey scale low-res image of 100x100 pixels = 10000 input neurons). Since every connection from the neurons of one layer to those of the next is associated with a unique weight, having FC layers quickly become computationally expensive for image input. Therefore, a mayor advantage of CNN:s as opposed to the classical types of NN:s, is that they introduce convolutional (CONV) layers which are not fully connected.

The layers of a CNN (CONV layers) consist of weights which make up learnable filters. The filters are three dimensional, with a set width and height generally far smaller than the input image, as well as a depth corresponding to the number of channels (three for RGB, with 3 color channels) in the input image. Filters slide across the width and height of an image at a certain step length, referred to as stride, and convolve with the input in each position (see Figure 3). This outputs a feature map which is a representation of the patterns that the filter has extracted. For a RGB image, the filter will convolve each color channel, before adding them together as one feature map, three channels deep.

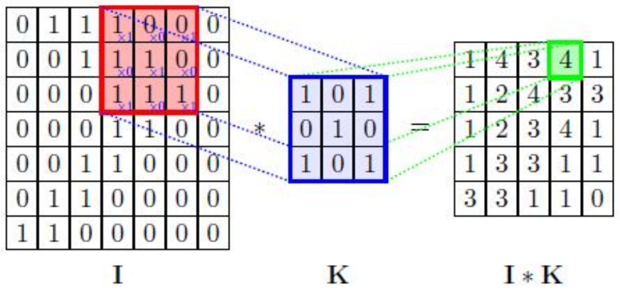


Figure 3: The convolution operation

Each filter looks at, and learns from, specific parts of the input, called "the receptive field", but is applied over the whole input space (Lecun et al. 1998). This is intuitive in the sense that the property of a filter in detecting certain patterns, such as edges, corners, or blobs of color, is universally applicable to an image. This has the implication that weights are shared within CONV layers, referred to as parameter sharing, and the computational cost significantly reduced compared to that of a FC layer. The number of filters with the same receptive field is referred to as the filter depth, or depth column. This is separate from the **network depth** which increases with an increasing number of CONV layers. With successive convolutions in multiple CONV layers, i.e. a deeper network, the feature maps become increasingly complex, detecting higher dimensional patterns. During the last decade, with the introduction of many deep networks, this has proven to be one of the key properties allowing for CNNs to excel at classification tasks.

2.3.2 Activation functions

Due to the potentially complex and heavy computation required for the multiple chain rule computations, it is paramount for the activation function to have a well defined and simple derivative. The ReLu function, short for Rectified Linear unit, defined as:

$$f(x) = max(0, x) \tag{2}$$

is one such function commonly used today in CNN:s, and is exclusively used in this project for hidden layers (Krizhevsky, Sutskever, and Hinton 2012).

For the output layer, a classifier CNN typically uses the Softmax function, defined as:

$$S(x)_i = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{3}$$

Softmax normalizes the input from the last set of neurons into values between 0-1, all values adding up to 1 in the output layer. This can be interpreted as a probability distribution for an input to belong to a range of labels (classes), which combines well with the cross entropy loss function (see section 2.4.4.1 and equation 5).

2.3.3 Padding and pooling

In order for the data passing through a CNN to remain "intact", as well as compact without losing too much information through sets of convolutions, it is common to utilize padding and pooling.

Zero padding ”pads” the input with zeroes on the borders, allowing for the filter to run along the whole width and depth of the input. In this way, the output dimensions can be controlled, and potentially (as is the case in this project) rendered to have the same dimensions as the input; called SAME padding.

Pooling is often applied to the output before being passed on to subsequent CONV layers, and consists of operations to down sample the feature maps by extracting more compact representations of the vital activations. In this process, a majority of the activations are discarded, which drastically reduces computational costs, and also mitigates over fitting. The most common version of pooling, which is consistently used in this project, is Max pooling, see Figure 4.

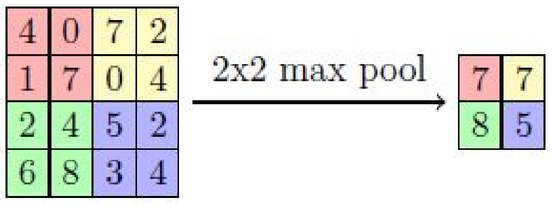


Figure 4: The max pooling operation

2.4 Training and optimization

With the highly accessible and comprehensive packages and libraries for ML applications comprised within Tensorflow and Keras, the network structure could be formalized rather quickly, while far more time instead was devoted to training, evaluation and optimization.

2.4.1 Data division

When training and evaluating a NN, the standardized way of utilizing data, is to divide it into three parts: one larger part for training (training data), and two smaller parts for evaluation and testing (evaluation- and test data). The training data for a CNN comprises all the images which the network will be allowed to see when training, and consequently learning the features of.

The evaluation data is often picked out randomly as a chunk of the training data prior to training, which (for a case of rather homogeneous data, as is the case for this project) means that its images generally hold similar traits and features as the training data, although the network will never see those exact images. This has the consequence that the network's performance on the evaluation data is a good indicator of when/if the network is over fitting, i.e. learning the features of the training data too well, generalizing poorly to other data. The last piece of data, the test data, preferably holds images which constitute diverse features true to the real nature of the problem, which the network has not previously been trained on, or optimized to perform on. Therefore it is of vital importance that the test data is left altogether untouched until the final moments in optimizing a model. Model performance on the test data forms the ultimate test of the network's ability to generalize to the real-life problems.

2.4.2 Data pre-processing

Inputting raw images into a CNN is problematic, both computationally and performance wise. Regardless of task, it is more or less always advised/necessary to perform two pre-processing techniques:

- **Down sampling:** A CNN requires for all images to have the same input dimensions. Most images taken today with mobile phone cameras or other, are of very high resolution (often the scale of several mega, 10^6, pixels) which are input dimensions way too computationally costly for this project to deal with. It also likely that much of the information in such a high resolution image can be represented just as well by a lower resolution image. This is why it has been common in similar projects to reduce the input image dimensions to somewhere around 100-300x100-300 pixels. For this project, images were down sampled and fixed within that interval.

- **Normalization:** Large input values, such as the range of pixel intensities between 0-255, can disrupt a network and slow down its learning since weights are generally assigned much smaller values. This can be fixed by normalizing the input. One common way of normalizing input, which is used throughout on all input in this project, is by dividing each pixel value by 255, thereby having the input range from 0-1.

- **Segmentation:** For a classification problem where the object of interest is located at a particular place in the input image, it might prove beneficial, or even necessary, to isolate that part of the image before analysis. For the identification task, a segmentation CNN was set up to perform binary semantic segmentation, i.e. label each pixel as either belonging to a foreground (pike) or a background (everything else in the image) class. The labeled data is provided to such a network as "ground truth images", which are manually segmented images which show the network the desired output for a given image. The output from the segmentation CNN could then be used to crop out the pike in the down sampled image, and place it on a black background.

2.4.3 Mini-batch gradient descent

When training a model on a data set, it can be very impractical and computationally expensive to perform the gradient descent with back propagation after a full iteration (referred to as epoch) through all of the input. Instead, it is common practice to divide the data into subsets of equal size, called batches, and have the network update its weights after each epoch over a batch. This is called Mini-batch gradient descent (MBGD), where the extreme case of having a batch size equal to the size of the input data is called batch gradient descent (BGD), and having it set to iterating over only one image at a time is called Stochastic Gradient Descent (SGD). In this project, MBGD is exclusively utilized.

2.4.4 Setup for gradient descent

The objective of minimizing the loss function with gradient descent requires the selection of a loss function to be minimized, and an optimizing function which calculates the updated weights for each step in the negative gradient.

2.4.4.1 Cross entropy loss function

Many different loss functions have been developed for implementation in NN:s with different purposes. For classification tasks, cross entropy loss is most commonly used. Cross entropy is a measurement of the difference in two distributions of information (bits), which in the case of a NN can be viewed as the additional number of bits which would be needed for our input image to fully represent a certain class. This concept can be utilized as a loss function by one-hot encoding the input. One hot encoding means that a prediction can only belong to one class, and is implemented by assigning binary labels (example: input cat, classes: [dog, human, cat], label: [0, 0, 1]). Entropy decreases with more certain probability distributions, which implies that cross entropy approaches zero as the prediction becomes increasingly accurate. Therefore, one-hot encoding transforms cross entropy into a minimization problem, which a CNN solves by becoming increasingly proficient in correctly classifying input images (Murphy 2013).

For this project, both binary and categorical cross entropy are implemented. Binary cross entropy is used for the segmentation task with only two classes, and is calculated by:

$$L_b = -log(p) + (1 - y) \cdot log(1 - p) \tag{4}$$

Categorical cross entropy is used for pike recognition where the number of classes is larger than two, and is calculated by:

$$L_{ca} = -\sum_{c}^{M} y_{o,c} \cdot log(y_{o,c}) \tag{5}$$

where M is the number of classes, y is the binary indicator, and p is the probability that the observation o belongs to the class c.

2.4.4.2 Optimizer and learning rate

As with the loss function, there are also a dozen different gradient based optimizer algorithms to choose between when setting up gradient descent for a NN. The optimizer algorithm governs the manner in which the weights are updated between epochs, and different algorithms are specialized in addressing different problems associated with gradient descent. The algorithms are briefly described conceptually below, leaving the technical details out since optimizers are not extensively explored for this project.

The analogy of viewing the optimizer algorithm as a ball rolling down a hillside is often used, where the goal is of reaching the bottom i.e. a minimum solution to the loss function. In the ideal case, the deepest part of the topography is targeted and reached (global minimum), but in most practical cases the objective is to roll down into a satisfyingly deep enough pit ("good enough" local minimum solution). This is in large due to the ball being hindered by for example flat areas (saddle points) and more or less shallow pits (non-satisfying local minima solutions) during its rolling downhill. Therefore it is important to alter the rolling properties of the ball so that it can roll past such areas, or avoid them altogether. The main property which governs how the algorithm behaves around such areas, is the learning rate which tells the algorithm how large steps to take in the direction of the negative gradient. Algorithms such as Nesterov gradient descent, Adagrad, RMSprop and Adam are aimed at adapting the learning rate to its environment, allowing it to move faster in "steep" areas, and more cautiously (although without getting stuck) in less steep areas. Adam (Adaptive Moment Estimation) is one of the most popular and widely used algorithms in recent projects, and is also implemented in this project (Ruder 2016).

2.4.5 Optimizing model performance

When training a CNN and aiming to optimize its model performance, it is common to perform cross validation to compare model performance with different sets of hyper parameters. Hyper parameters are settings which govern some of the training aspects of the network, such as learning rate, kernel (filter) size, batch size, etc. For a project such as this one with very limited computational power, cross validation is simply too computationally expensive and time consuming. Most hyper parameters were therefore set at constant values for the training session of a network once it seemed to produce acceptable results. Since the issues with training models for this project could almost entirely be traced down to over fitting, optimization relied more heavily on regularization combined with using Keras checkpoints (see 2.4.5.2 and 2.4.5.3) than hyper parameter fitting.

2.4.5.1 Evaluation metrics

Monitoring the value of the loss function on validation data gives an understanding of how to progress with training a model. Once this function is minimized, and training has stopped, it is however necessary to express the model performance on validation- and test data in terms of an intuitive metric with real world applicability. For this, the metric "accuracy" was used for the identifier CNN, and intersection over union (IoU) as well as pixel accuracy for the segmenting CNN. The "accuracy" simply returns the percentage of correctly classified images after an epoch (i.e. 90% would infer 9/10 correctly classified images), while "IoU" infers a percentage as to how well a predicted segmentation overlaps with the ground truth image. The IoU is the overlap between the segmented object in the prediction and the ground truth divided by their union, seen in Figure 5, and is calculated by:

$$IoU = \frac{TruePositives}{TruePositives + FalseNegatives + FalsePositives} \tag{6}$$

Since the IoU is calculated for each object separately, it can be expressed as class-wise IoU, or for example as a mean of all classes (mean IoU).

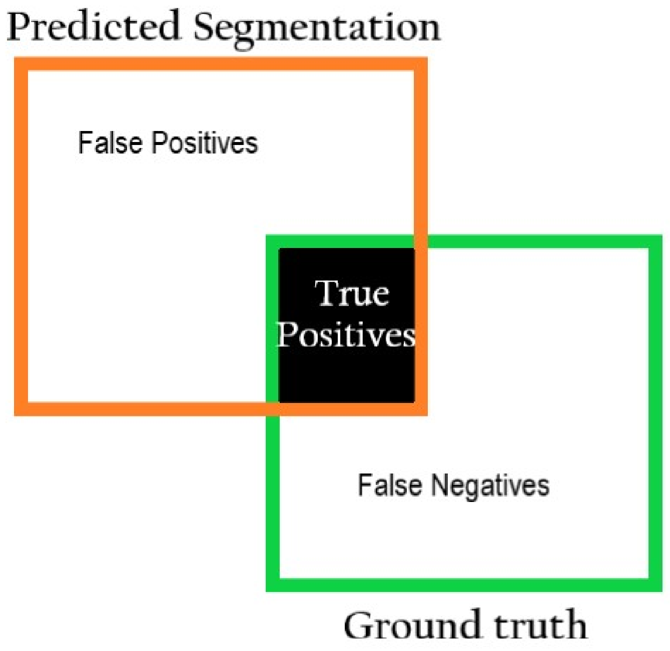


Figure 5: A figure illustrating the Intersection over Union (IoU) metric used for validating segmentation. The black square consists of the overlapping pixels between prediction and ground truth with respect to the segmented object. A better overlap means a higher IoU index

Lastly, pixel accuracy gives the percentage of pixels correctly classified in an image, i.e. true positives/total number of pixels. For tasks with class imbalance where the background is the dominating class, pixel accuracy poorly represents how well the target has been segmented - it then only gives an indication of whether the background is mostly in place. Therefore, IoU is generally preferred over pixel accuracy for this type of task, although it can still be useful for validation and optimization purposes.

2.4.5.2 Regularization techniques

Regularization techniques can be used to great effect when a NN is over fitting on the training data set. For the pike identifier, this was certainly the case as the lack of comprehensive and diverse raw data was a highly limiting factor. This was combated through implementation of two regularization techniques:

- **Data augmentation:** Instead of increasing the pool of training images, which would be difficult and time consuming for this project, an alternative is to augment images in different ways as they are generated as input for the CNN. With on-the-fly data augmentation, the images in each new batch introduced to the CNN are randomly augmented during training, as opposed to in-place data augmentation where a new set of augmented images is statically added to the training data before training. This has the main advantage that each batch is ”new” to the network, which allows it to learn new features, even if the training data originally represented only a very limited set of features.

  Augmentation techniques applied during training of the pike identifier were vertical and horizontal shifts, rotations, zooming, vertical flips, and brightness alterations. These are all augmentations which represent the diversity which the network would likely have to deal with in a real world situation.

- **Dropout:** To mitigate over fitting, another approach is to randomly cancel out (drop) a fraction of a layer's (either hidden or visible) neurons during each epoch. The idea behind this is to prevent the network from relying too heavily on a small set of neurons while others are left redundant, and instead find alternative ways of solving the problem with other paths of activations. In this way, the model can become better at generalizing (Srivastava et al. 2014).

2.4.5.3 Early stopping and save best only

The Keras ML library provides different checkpoint functionalities during training, for example providing the possibility to save the weights after the best performing model epoch instead of only saving the weights from the last epoch. It is also possible to automatically stop a training session before it has completed all its epochs if the monitored validation loss-value does not improve. Both of these checkpoint functionalities were utilized when optimizing the pike identifying CNN.

2.5 Implemented structures

This section describes the different network structures used for the segmentation and pike identification task, as well as the concept of transfer learning.

2.5.1 Unet

For the segmentation task, the U-net (Ronneberger, Fischer, and Brox 2015) architecture was utilized. U-net is a so called ”encoder - decoder” network, which (simplified) first applies a CNN architecture with convolutions and pooling to down-sample the input and learn features (encoder), and then follows that up by up-sampling (decoding) the input to its original size with ”reversed” convolutions (transposed Conv-layers). Through the decoding layers, spatial information is regained (for example where an object was located in the input image); a property which allows for semantic segmentation.

2.5.2 VGG

The Visual Geometry Group (VGG) from Oxford University developed the VGG-block CNN structure which is used throughout this project (Simonyan and Zisserman 2014). A VGG block consists of one or more convolutional layers with small CONV-filters (3x3), ReLu activation, and lastly a max-pooling layer. Sets of VGG blocks are then stacked on top of each other, followed up by one or more FC-layers with ReLu activation, and ending in a FC output layer with soft-max activation. The VGG team's very deep networks VGG16 and VGG19 (numbers indicating the number of CONV+FC layers) have performed well (although no longer state of the art) in competitions on benchmark data sets. Even though other, deeper, networks have now performed better, the uniform and quite simplistic structure of the

VGG network makes it relatively easy to grasp and modify which was of high importance for this project where network depth was not necessarily a key factor. The segmentation model utilizes the VGG16 as the encoder, while several different VGG network structures were implemented for the identifier.

2.5.3 ResNet and InceptionResNet

This project briefly dealt with the CNN structures called ResNet (He et al. 2015) and InceptionResNet (Szegedy, Ioffe, and Vanhoucke 2016) for the identifying task. Both these networks have both performed better on benchmark data sets than the deep VGG networks. ResNet is a deep network which utilizes residual learning, while InceptionResNet is a version of ResNet which combines with Googles Inception network (also called GoogleNet). Neither network is elaborated upon further, since the project never explored them in any depth, and the test results from these networks are therefore merely intended as reference.

2.6 Transfer learning

For many classification tasks, the problem can be broken down into very similar parts, where low and mid level features are detected in shallow layers, and high level (more abstract) features are detected in the really deep layers. Especially the low and mid level features are often similar between different classification problems (most objects for example have low level features like edges and corners), and therefore a trained network might work well on a completely different task. It is however rarely the case that a network is completely transferable from one problem to another. Transfer learning is therefore the idea of merely extracting layers with weights that have (preferably) been trained on massive data sets, such as Imagenet[5], as opposed to transferring the whole network.

Transfer learning can be particularly beneficial for problems with limited data, since much deeper features can be extracted from larger data set. After transferring some, or all, layers and weights with these features, the network can however also be customized to fit the problem. It can either be allowed to continue training the imported weights on the target data set (referred

[5]http://www.image-net.org/

to as ”fine tuning”), or have its layers ”frozen”, which renders the weights static (Pan and Yang 2010). Transfer learning was applied when testing out different model structures for the identifier.

3 Method

In image 6, the workflow is described visually as a workflow with the purpose of providing an overview over the comprehensive method section.

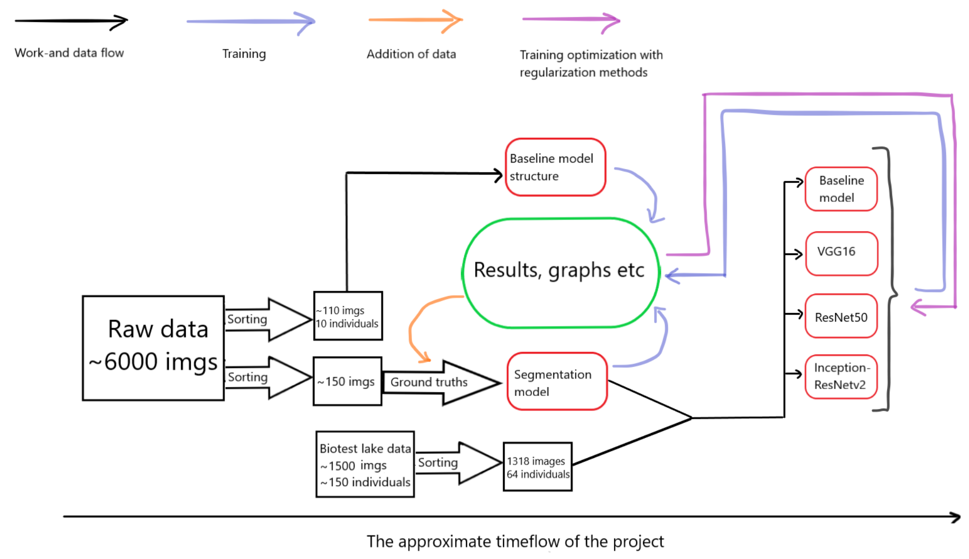


Figure 6: A rough outline of the workflow described in the method section

3.1 Data

The raw data was made available through contacts at the County Administrative Board, who provided approximately six thousand images of pike. The data did not adhere to any consistent format or labeling, and most pictures in the data set were not sorted according to individuals of pike. This generally had to be done by hand, where intended use of the data demanded different measures of sorting, labeling, and pre-processing. These processes of preparing data are further described for each part of the project below.

3.1.1 Data for segmentation

Choosing data for the segmentation model to be trained on could be done quite simply and efficiently, since it, for example, did not matter whether the same pike occurred more than once. The only significant information to look for in an image, was that it captured a shape of pike which is representative of those shapes which generally occur in images of pike. Therefore, the main focus in choosing images, was trying to cover all of the most common cases with sufficient data.

Of the total set of data, of about six thousand images, the most common image-type is one where a pike is held by one person, to a varying horizontal or vertical degree, with the background mostly consisting of the interior of a boat, water and sky. The second most common type is one where a pike has been placed on the ground, or on a homogeneous mat, and photographed almost perfectly horizontal. The first type was considered a more difficult problem for the model, since the pike did not have a consistent shape in those images, and they simultaneously contained very different types of backgrounds. To compensate for this, the model would likely need to train to a larger extent on the first type images, in order to get them right. The data set was therefore created as to consist of about two thirds of the first type image, and one third of the second type, with pike held pointing both to the left and right.

The major limiting factor to the size of the data set was the time-consuming pre-processing that had to be performed on each image in order to create its ground truth images. Due to this, only one hundred and fifty images were picked out and pre-processed for the initial segmentation model.

3.1.2 Data for pike identification

For the pike-identification model, the selection of data posed a more sensitive problem than for the segmentation model. First and foremost, a larger data set of many different sorted individuals was required for this task. Because of the assumption that the the patterns differ for each side of a pike, only images with pike exposing the same side were selected. This ruled out many images of pike exposing the ”wrong” side. Secondly, it was very important to make certain that every image of pike was labeled correctly, since the model would otherwise learn to identify images of the same pike as different, and vice versa.

3.1.2.1 Baseline model

The first step in creating a model for identification of pike, was to establish a small network with enough data to get results of any kind. For setting up this baseline model, the aim was to find ten individuals of pike with at least around ten images, of decent quality and representation of typical pike-images, per pike. At the time of setting up the baseline model, no sorted data of individual pike with ten or more images per pike was available. Instead, parts of the total data set was scanned for sets of images which showed clear signs that they had been taken of the same pike at the same time (i.e. same background, same person holding the fish, or other obvious markers). This was a way of minimizing the risk of the network being fed with incorrectly labeled images.

Ultimately, one hundred and eleven images divided over 10 pike individuals were picked out to form this data set. No pre-processing, either manual or automatic, was performed on this data, since it was in the end only intended for setting up a baseline model setup, getting it to run properly, and for receiving rough indications of its performance on a non-ideal data set.

3.1.2.2 Biotest lake data

For developing the baseline model, as well as for implementing other model structures, pre-processed data was produced by the segmentation model from the "Biotest lake" data set. This data set was not available initially, which is why the baseline model was set up on a different data set. The images in the Biotest lake data set had been photographed by pike experts at the county board during test fishing trips, and had subsequently been sorted into folders for each pike individual by the same experts. Because of this, the identity of each individual of pike could be verified. The data set consisted of 1500 high quality images of around 150 individuals, and consequently contained far better representation of the real world situation than previous data used. Note however that only a small proportion of all pikes in this set were represented by images taken with many years apart. Most pike individuals were simultaneously only represented by a handful images.

3.2 Model setup

Different types of convolutional neural networks were trained for both the task of segmenting images, and the task of identifying individuals of pike in different images.

3.2.1 Segmentation

The model used for the segmentation was a U-net model with a VGG16 as encoder network, developed for semantic segmentation. The model was imported from an open source Github repository [6] and was already fully constructed and operational with respect to model structure and hyper parameters. This approach was chosen with the purpose of not using up too much time on a secondary (although necessary) task, which – due to the quite simple nature of the problem – would probably not require the developing of a highly customized network. Therefore, the data and the way it was organized had to be adapted towards suiting the model requirements for input. The model required the ground truth models to be of the same shape, and with the same labels, as their corresponding training images, and the pixel values of type uint8, being either 0 or 1. All images also had to be formatted to (.png) files. A script was set up to perform this and organize a directory for all the images, with training images divided into a training and a test set in respective folders, along with two other folders containing the corresponding pre-processed ground truth images. Twenty images were selected for the test set, which left 110 for training. The model was then trained and evaluated on 10 epochs.

Acceptable results were produced for most images, see results in Figure 7, with around eighty percent of the white pike-pixels in the correct place. Clearly, images which were well represented in the training set with respect to light conditions, pike-position, etc, were already no match for the model to classify with almost perfect precision. Other, more deviant images to the training set, were evidently more difficult for it to process, and this indicated over fitting. A first approach to try to deal with the model over fitting, was to increase the data set with additional data, and thereafter train it more. After the first instance of added data to the training set (around 30 arbitrary images from the raw data set) and an additional 8 epochs, results were

[6]https://github.com/divamgupta/image-segmentation-keras

improved, Table 1 where particularly the class wise classification of pike improved by 4% see Figure 9a). Also, from comparing the two images in the middle top and left in Figure 8 and 9a, it is evident that the model to some extent produced better predictions. From these images, it is however also evident that there is a lot of room for improvement.

The next approach was to perform predictions on images from the raw data set which were not in the training or test set, and pick out those which it performed poorly on. The hypothesis for the poor prediction on the top left image in Figure 9a , was that the lighting of the pike was brighter than the model was used to, and therefore the color scheme of the pike did not match well with what it was trained on. Therefore, most of the images now chosen for prediction were ones with different - and particularly over exposed - light conditions, or simply images of pike with different color schemes than previously. Some of the predictions of these images are presented in Figure 9b) (note that no training on similar images preceded these predictions), and it is evident that the model performs poorly on most of them. This indicated that the hypothesis might be correct, and thereby additional images with new or difficult properties were added to generalize the model.

The model was now also set to validate during training, which could help in analysing whether the model was over fitting on this new extended data set. This meant that a certain amount of images from the training set had to be removed and sorted for validation instead. A script was created to randomly pick out 20% of the images from the training set. Consequently, the number of images for training did not increase much from the last training session, but more importantly they would represent a wider diversity and more features. In total, the model was now trained on 172 images, validated on 43, and tested on 41 with results presented in Figure 11. Comparing figure 10b) with Figure 11b), it now appears that some of the recently introduced test images are better predicted, some have gotten worse, and some are more or less identical. Meanwhile, comparing the images from the original test data set in Figure 10a) with those in Figure 11a), no improvement seems to have been made, and especially the image in the top left is clearly predicted worse. The evaluation IoU scores with regards to the test set are also virtually the same as from the previous training, which indicates that no evident progress has been made in better generalizing the model from adding the new data.

Figure 11 shows that the training and validation pixel accuracy reach very high percentages during training, where they both start leveling out at 98%, and 95% respectively after 4 epochs. This indicates that further training might only result in over fitting. Meanwhile, the loss function follows a similar trend for the training set, but not for the validation set. Here, the model loss for the validation set fluctuates highly throughout training, and ends up at a model loss after 20 epochs, several times higher than after only 3 epochs. This provides a strong indication that the model is quickly over fitted on the training data set during training. Regularization techniques could mitigate this and improve model performance.

Despite not having the best performance in metrics, or necessarily the best image predictions so far, the 3d training session model was assumed to still generalize better than the other models since it had been trained and tested on a larger and more diverse data set. Its performance could clearly be improved on metrics and some more difficult images by implementing regularization and checkpoints. However, the best predictions of those shown in Figure 11, and the 90% mean IoU of the 3d training session, indicated that it could probably already be used on a less difficult data set such as the Biotest lake data set.

A script was set up to output segmented pikes from the Biotest lake data set by reformatting the images to the same size as the segmentation model output, and having the model predict the segmentation stencils for each image. In the final step, the reformatted image and the stencil from 12 were added together to produce fully segmented images, with only the pike as foreground, and non activated (black) pixels in the background, see Figure 13. As expected, it turned out that the more homogeneous and high quality images of this data set were generally no match for the segmentation model. Even the seemingly difficult images in Figure 14, where the pike occupies a small area of the image, with a person additionally obscuring its shape, were segmented very well, see Figure 15. Going through all of the images, only a small fraction of the approximately 1500 image had to be removed due to poor segmentation predictions, and those were often of such poor quality, or irrelevant format to begin with, that it was of little importance.

3.2.2 Pike identification

3.2.2.1 Baseline model

The baseline model was constructed largely through following the instructions in a tutorial on setting up a CNN with Keras for the Cifar photo classification problem [7].

Before being able to train this model structure on the pike data, the data had to be formatted for implementation in Keras, and labeled correctly. This process is described below, and was subsequently repeated for all other data sets used in this project .

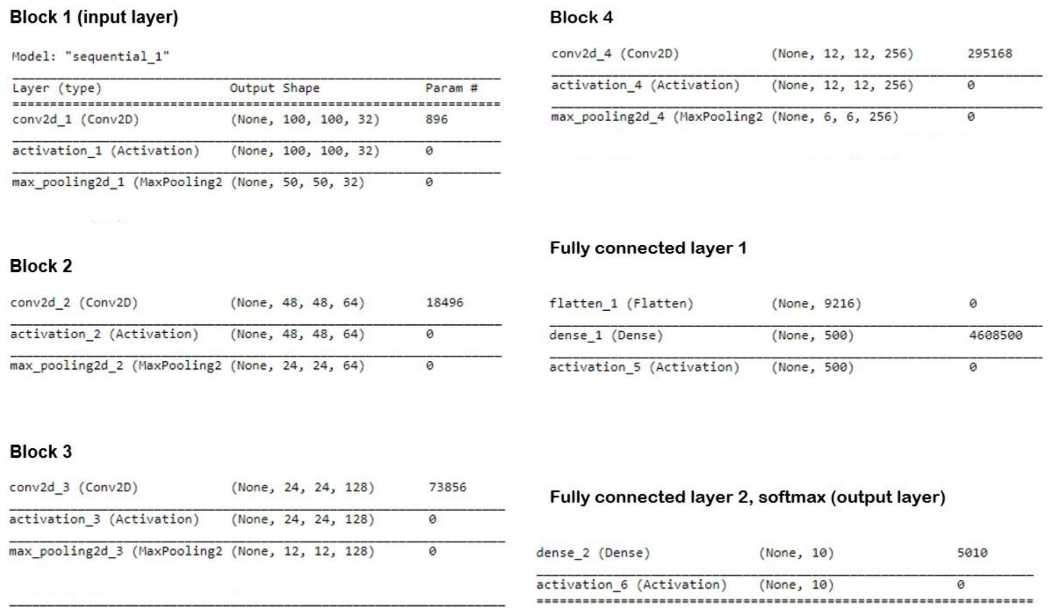
The ten individuals of pike constituted ten classes for a typical CNN model to categorize images into, so each image had to be labeled with its respective pike-id. This ID was defined through sorting each set of pike-images into a respective folder with a unique number. From this training set, two images from each pike-folder were extracted and put into another, identical, folder structure which was to make up the test data. Next, a function was set to loop through both training and test folders, loading each image, re-formatting it to a given size, and turning it into an array of all the pixel values. The images were then placed in the training or test array, containing the information for each image of that category. In the same process, the label values for each image were placed in two other arrays mirroring the training- and test image arrays, so that the position of each label corresponded correctly to its image. The label arrays were then one-hot encoded, so that each image would be represented by a label of 10 digits, where only the digit corresponding to the right id was 1, and the rest 0.

The data, formatted into arrays and labeled correctly, was ready for implementation. Setting up and customizing the model structure and its hyper parameters was therefore the next step. A Keras sequential model was constructed with a VGG-type block structure of four convolution blocks with max-pooling, and two fully connected (dense) layers. ReLu was used as activation function for the convolutional layers and the first dense layer, while Softmax was used for the dense layer before the output layer. The optimizer

[7]https://machinelearningmastery.com/how-to-develop-a-cnn-from-scratch-for-cifar-10-photo-classification

was set to a learning rate of 0.001, with the adaptive learning rate Adam. The model was compiled with categorical cross entropy as its loss function, and accuracy as the evaluation metric. The model performed at around 40% accuracy after 15 epochs on the test-set at this point. With some tweaking of the number and order of filters within the convolutional layers, and the number of fully connected units in the first dense layer, results could further be improved. Having an increasing number of filters for each added convolutional layer proved to significantly improve results, up to around 50-60% accuracy.

Since it was not important for this model to be optimized for performance on the small data set of questionable quality and representation of the problem, this model structure, presented in Figure 7, was now set to function as the baseline model. This model structure is referred to as the VGG6 network, and could later be improved on with the introduction of more, and higher quality, data. No validation data was picked out and analyzed alongside the training data-set for this baseline model, since these early results on the test data-set sufficed as indicator that the approach was valid and might work well going forward.



Block 1 (input layer)

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 100, 100, 32)      896
_________________________________________________________________
activation_1 (Activation)    (None, 100, 100, 32)      0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 50, 50, 32)        0
```

Block 2

```
conv2d_2 (Conv2D)            (None, 48, 48, 64)        18496
_________________________________________________________________
activation_2 (Activation)    (None, 48, 48, 64)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 24, 24, 64)        0
```

Block 3

```
conv2d_3 (Conv2D)            (None, 24, 24, 128)       73856
_________________________________________________________________
activation_3 (Activation)    (None, 24, 24, 128)       0
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 12, 12, 128)       0
_________________________________________________________________
```

Block 4

```
conv2d_4 (Conv2D)            (None, 12, 12, 256)       295168
_________________________________________________________________
activation_4 (Activation)    (None, 12, 12, 256)       0
_________________________________________________________________
max_pooling2d_4 (MaxPooling2 (None, 6, 6, 256)         0
```

Fully connected layer 1

```
flatten_1 (Flatten)          (None, 9216)              0
_________________________________________________________________
dense_1 (Dense)              (None, 500)               4608500
_________________________________________________________________
activation_5 (Activation)    (None, 500)               0
```

Fully connected layer 2, softmax (output layer)

```
dense_2 (Dense)              (None, 10)                5010
_________________________________________________________________
activation_6 (Activation)    (None, 10)                0
=================================================================
```

Figure 7: The baseline model structure (VGG6) following the VGG structure, starting at block one in the left column and ending in the fully connected Softmax layer in the right column. It consists of 4 blocks and 2 fully connected layers, which infers the "6" in "VGG6". Each block contains a convolutional layer followed by a ReLu activation and a max pooling operation

As a small experiment to test regularization techniques, a training session was initiated with dropout and data augmentation. A 10% dropout was applied after each convolution, and a 50% dropout after the first dense layer, and data augmentation via keras ImageDataGenerator was implemented on the training data, such as shifting, shearing, zooming and rotating. This resulted in a model that performed between 60-80% on the test data set after 15 epochs with batch-size set to 16 - a significant improvement. As before however, results fluctuated highly between runs, which indicated that the model performance was unstable and susceptible to the behaviour of the stochastic learning algorithm. Regardless, this experiment still showed that regularization techniques might be very effective for this problem going forward.

3.2.2.2 Baseline model on Biotest lake data

To create a reference for performance on the Biotest lake data set, the VGG6 model structure was trained on the larger Biotest lake data set that had been processed by the segmentation model. An issue was that more than half of the individuals of pike were only represented by one or just a few images, and this posed problems for distributing the images of each individual into training, validation and test data sets. Also, some of the images were poorly segmented by the segmentation models prediction, and had to be removed. Individuals which were left with less than 6 images were subsequently removed from the data set. This left 64 individuals of pike with a total of 1318 images amongst them. Consequently, most of the image information was intact for training, but the model would have to train on far fewer classes.

From the remaining data, a set fraction of the full set was randomly chosen by a python function for validation and testing. Approximately 30% of the full data set was set for the function to select from the full data set: 10% (138) and 20% (250) for validation and testing respectively.

Running the VGG6 model with the new bigger data set, the model performed similar to how it did on the smaller set, see Figure 17. The test accuracy topped out at around 60%, while training accuracy plateaued at almost 100%, which was a clear sign of over fitting. Running the model for far longer, while applying regularization could resolve these issues. After 1000 epochs with 20% dropout after block 1,2,3 and 4, and a 50% dropout after FC-layer 1, as well as various forms of data-augmentation, results were vastly improved, see Figure 18.

3.2.2.3 Deep networks on Biotest lake data and transfer learning

Even if the shallow VGG6 model performed well with regularization, it was also reasonable to implement a deeper network structure which could potentially be more robust as it could learn more complex features. Utilizing these models, and also testing them with transfer learning, was a feasible first step in applying a deeper model structure. Three model structures, with accessible weights generated from being trained on the Imagenet database, were selected for this: VGG16, ResNet50, and InceptionResNetv2.

For VGG16, training seemed to get stuck almost immediately (no improvement of performance, either on training or validation data) without transfer learning, while the training of ResNet50 got stuck in either case. For InceptionResNetv2, no difference could be made out between training with or without transfer learning, which implies that transfer learning is unnecessary. This is why no results are presented for these training sessions. All remaining results are presented in Figure 19, 20, 21 and 22. Sufficient training sessions of different models were now completed for the results to be compiled and compared in Table 2, and for a final assessment to be made of what the best approach seems to be.

4 Results

4.1 Unet VGG16 segmentation model

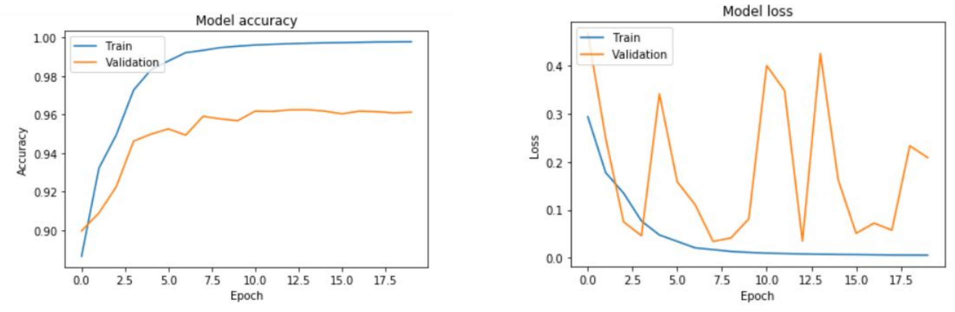
4.1.1 Results from training

The different training sessions generated three models with different weights. The IoU accuracy metrics of these three models are presented in Table 1. The amount of data and number of epochs increased for each session, with model 1 being trained on less data over fewer epochs, and model 3 being trained longest on most data. The best results on all the IoU metrics are given by model 2, with model 3 performing almost as well on the metrics. Model 3 was however assumed to generalize well to more diverse types of images, since it had been trained and tested on bigger and more diverse data sets. Therefore, only the training of model 3 is presented below.

Table 1: Results of Intersection between Union (IoU) on test data from three models trained in different sessions. b = background, f = foreground (pike)

	Model 1: 10 epochs 110 Train-imgs 20 Test-imgs	**Model 2:** 18 epochs 140 Train-imgs 20 Test-imgs	**Model 3:** 20 epochs 172 Train-imgs 41 Test-imgs
Frequency weighted IoU	94.3%	95.0%	94.7%
Mean IoU	87.7%	90.1%	89.5%
Class wise IoU (b/f)	97.0%/79.0%	97.0%/83.2%	96.9%/82.2%

As seen in Figure 8, the training pixel accuracy approaches 100% after about 4-5 epochs for model 3, while the validation pixel accuracy stagnates around 95-96% also at about 4-5 epochs. Model loss fluctuates heavily through the whole session, and seems to be minimized at either the 7th or 12th epoch.



(a) Pixel accuracy for training and validation data

(b) Cross entropy loss value for training and validation data

Figure 8: Validating model 3 over 20 epochs

In Figure 9, 10, and 11 are presented segmentation predictions generated by model 1, 2, and 3. In Figure 10 and 11, the b) figures show the respective model's predictions on some of the new test images, while the a) figures show predictions of images from the original test data set, same as in Figure 9.

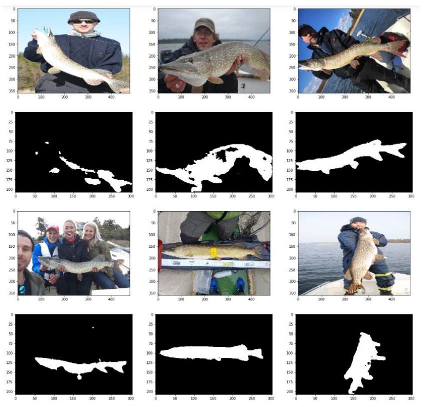
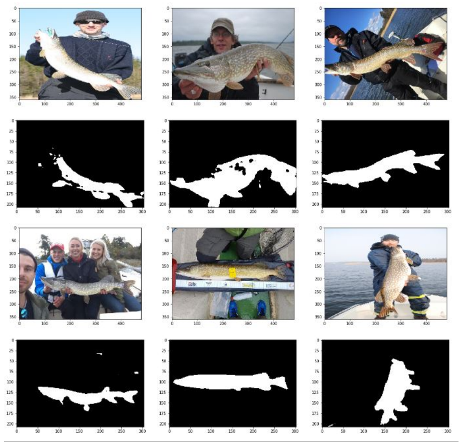
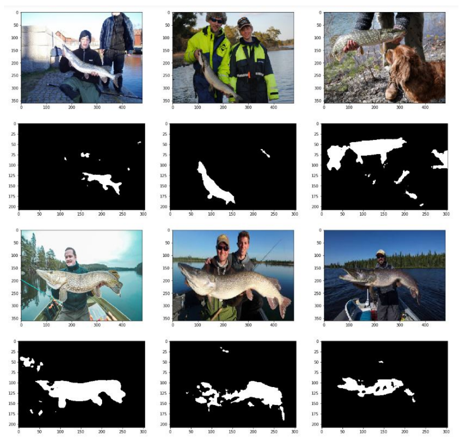


Figure 9: Segmentation predictions of images from the original test data set by model from training session 1

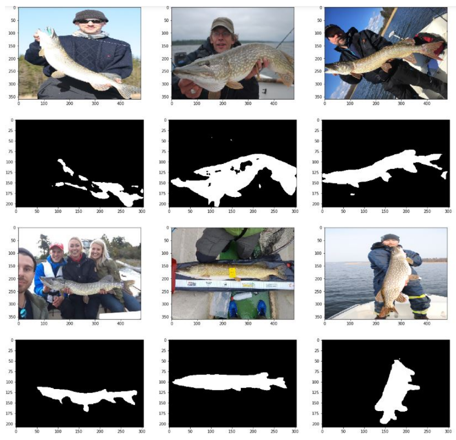


(a) Results on some of the 20 test images



(b) Some of the predictions on new images which the network was expected to perform poorly on

Figure 10: Segmentation results of test images, from training 18 epochs on a 150 image training set


(a) Results on same test images as figure 10)

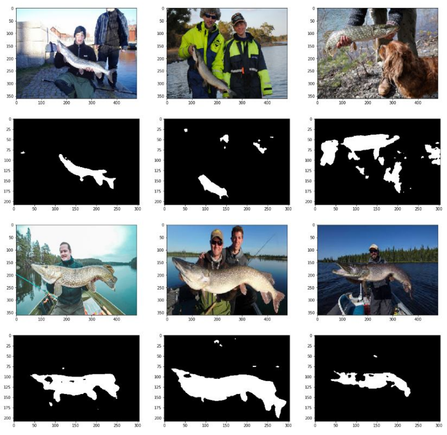
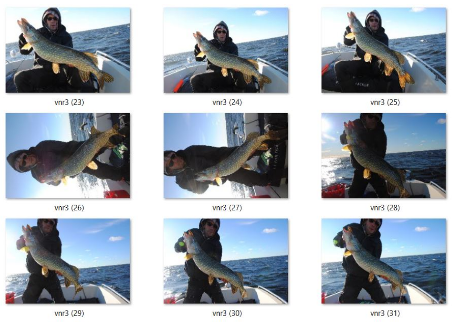

(b) Results on some of the images from the new test set of 41 images

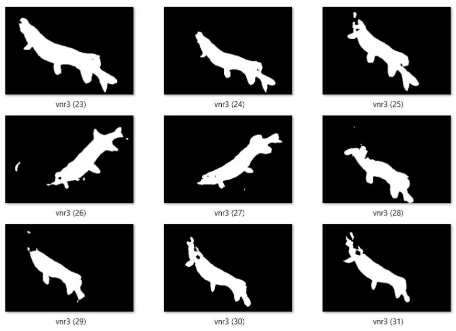
Figure 11: Results on images from training 20 epochs on a larger data set of 172 images

4.1.2 Performance on Biotest lake data set

Here are presented some results from implementing segmentation model 3 on the Biotest lake data set. In Figure 12 and 14 the a) figures show the (reformatted) raw data images, onto which the images in the b) figures ("stencils") are superimposed to create the fully segmented images in Figure 13 and 15. The raw data images in Figure 12a) represent typical images within the Biotest lake data set, while those shown in 14a) represent more outlying types of images, which seem less trivial to segment adequately.



(a) Re-formatted raw data



(b) Modeled stencils

Figure 12: Example of a set of typical pike-images in Biotest lake data set

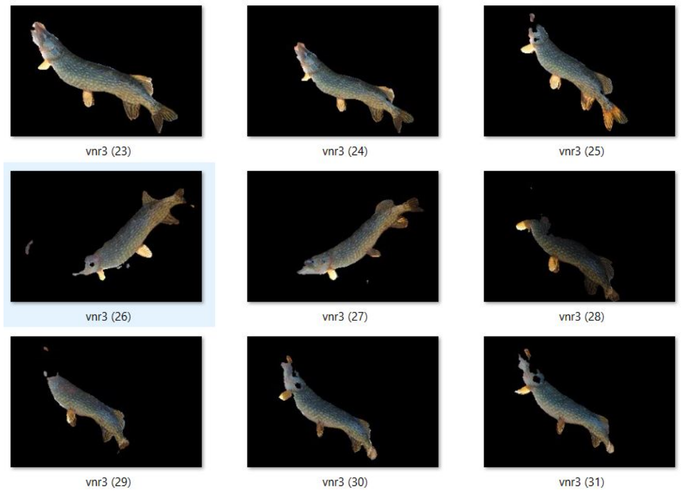
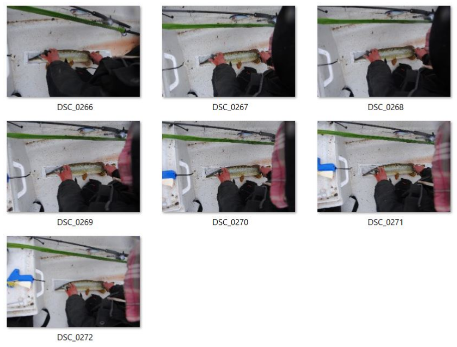
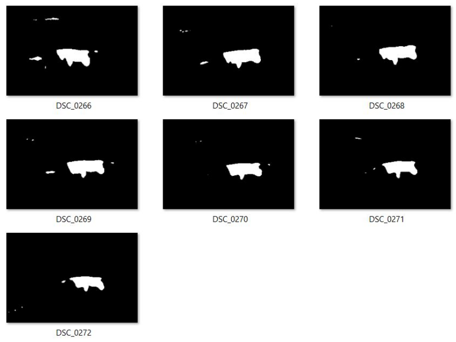


Figure 13: Superimposing stencils from Figure 12b) onto images in 12a) forms fully segmented images



(a) Seemingly ”difficult” re-formatted raw data



(b) Modeled stencils

Figure 14: Example of a set of a more ”difficult” set of pike-images to segment from the Biotest lake data set, where the pike is small and obscured

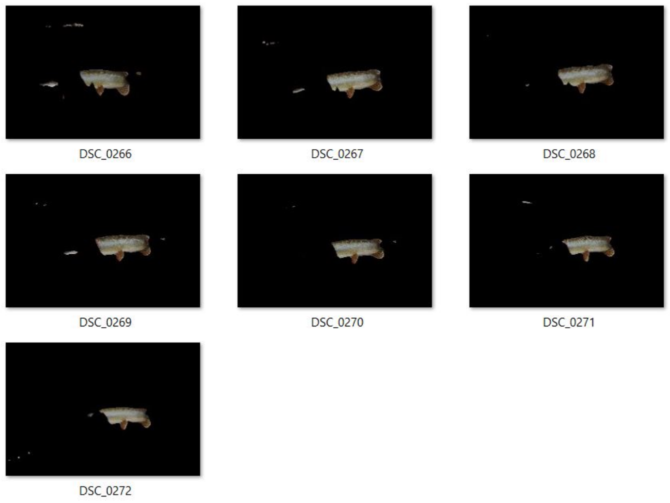


Figure 15: Superimposing stencils from Figure 14b) onto images in 14a) forms fully segmented images

4.2 Pike identification model

4.2.1 Model pipeline

All models presented in this section follow the pipeline in Figure 16, where the raw data is from the Biotest lake data set which is pre-processed by normalization, reformatting, and segmentation. The network structure presented therein represents the general format of the VGG networks, but is much different from the ResNet and InceptionResNet structures.

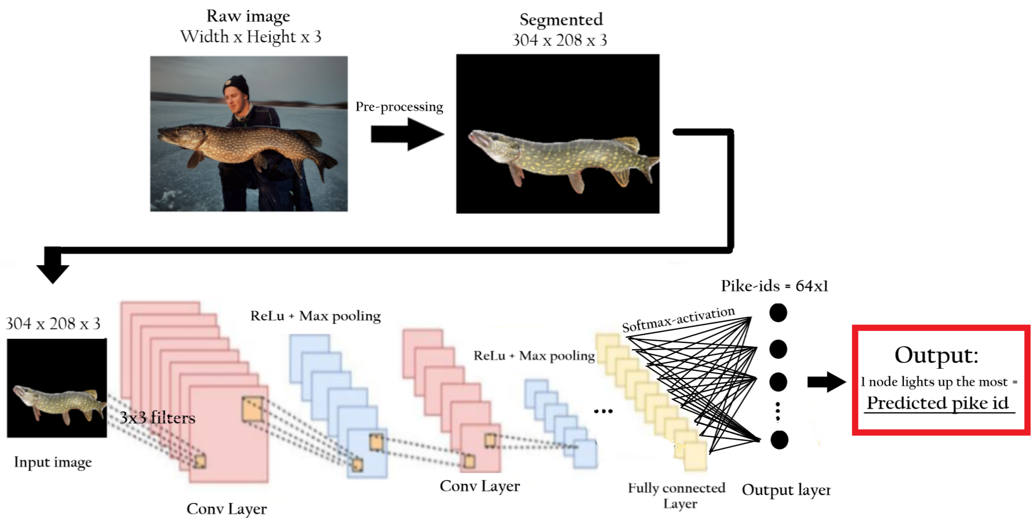
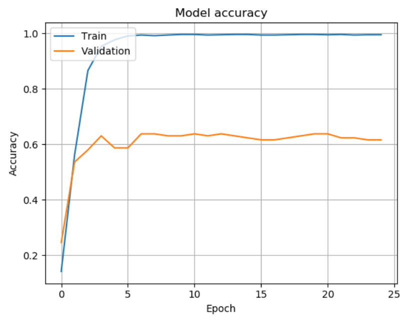


Figure 16: The general model pipeline for the VGG based models

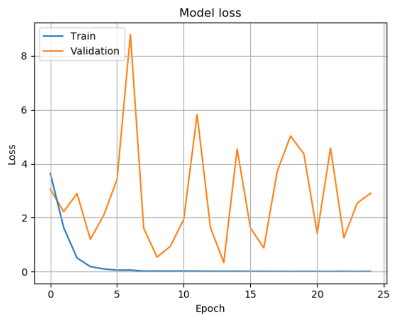
4.2.2 Baseline VGG6 model

Figure 17a) shows that the performance of the baseline VGG6 model quickly converges during training. Training accuracy reaches 100% after only a few epochs, indicating over fitting, while validation accuracy stops increasing around 60%. The cross entropy loss for validation data fluctuates heavily, potentially showing a decreasing trend over all 25 epochs. The loss is in the b) figure is minimized after 13 epochs.

The training and validation accuracy and loss when running the VGG6 model much longer (1000 epochs instead of 25) with regularization techniques are presented in Figure 18a) and b). Performance converges much slower during this training session, stagnating after approximately 200 epochs for both training and validation accuracy while still improving. Accuracy is potentially still increasing after 1000 epochs for both training and validation data, and ends up at max value after several epochs of around 90%. The loss fluctuates heavily throughout, but reaches its minimum at a loss value of 0 at the 913th epoch.

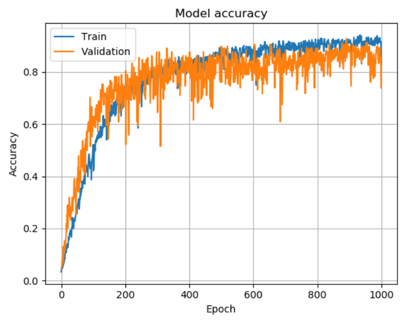


(a) Value of accuracy metric during training

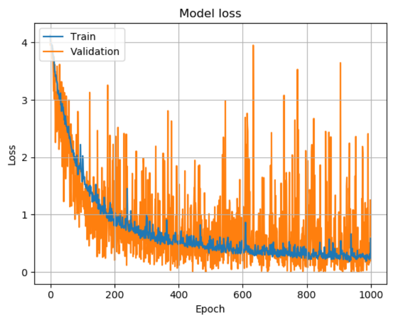


(b) Value of cross entropy loss function during training

Figure 17: Accuracy and loss results from training the baseline model 25 epochs



(a) Value of accuracy metric during training

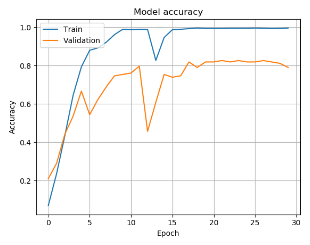


(b) Value of cross entropy loss function during training

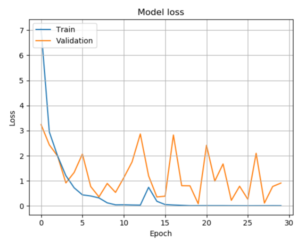
Figure 18: Accuracy and loss results from training the baseline VGG6 model 1000 epochs with regularization on segmented images

4.2.3 VGG16

When training VGG16 With transfer learning, it is shown in Figure 19a) that the training and validation accuracy of the VGG16 model converges quickly to 100% and approximately 80% after about 10 epochs. The loss value in the b) figure is minimized at the 27th epoch. Adding regularization and training over more epochs, training accuracy and validation accuracy reaches 80% after 120 epochs, and seem to still be increasing according to Figure 20. The loss value in the b) figure is minimized at the 116th epoch.

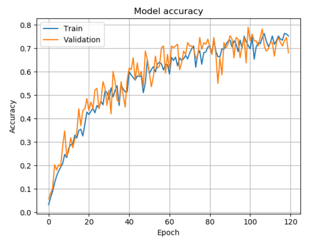


(a) Value of accuracy during training

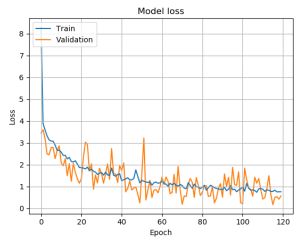


(b) Value of cross entropy loss function during training

Figure 19: Accuracy and loss results from training VGG16 over 30 epochs with transfer learning from Imagenet. All layers were frozen except the 2 dense layers at the end



(a) Value of accuracy during training

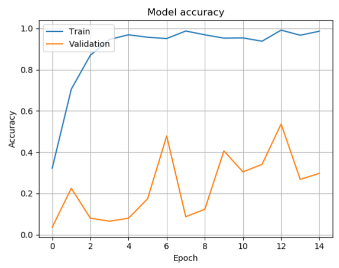


(b) Value of cross entropy loss function during training

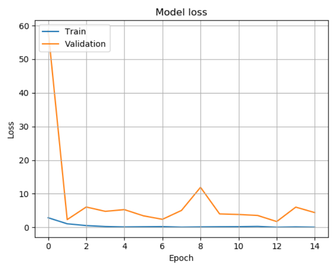
Figure 20: Accuracy and loss results from training VGG16 120 epochs with transfer learning from Imagenet. All layers were frozen except the 2 dense layers at the end, and data-augmentation and dropout were applied

4.2.4 InceptionResNetv2

Training and validation accuracy for InceptionResNetv2 showed highly unstable behavior during training, either with or without regularization, as is shown in Figure 21a) and 22a). With regularization, validation occasionally reaches an accuracy of around 90%. The cost is not as visibly shown to fluctuate between epochs, which is simply due to the graphs in Figure 21b) and 22b) being zoomed out as the loss values are initiated at very high values. In these graphs, the loss is minimized at the 13th and 39th epoch respectively.

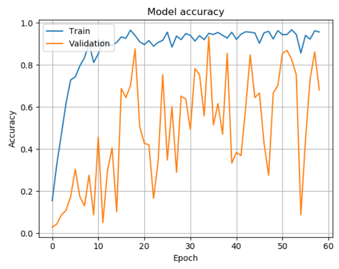


(a) Value of accuracy during training

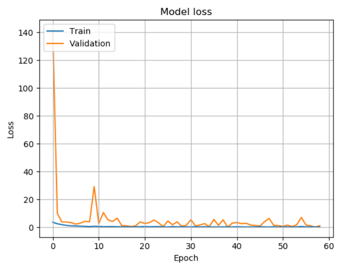


(b) Value of cross entropy loss function during training

Figure 21: Accuracy and loss results from training InceptionResNetv2 16 epochs



(a) Value of accuracy during training



(b) Value of cross entropy loss function during training

Figure 22: Accuracy and loss results from training InceptionResNetv2 60 epochs with data-augmentation

4.2.5 Model performance summary

The performance results from various training sessions, of the different CNN models for the pike identification task, are presented here. The hyper parameters during training were the same for each training session, but they were run for different amounts of epochs depending on training time. Transfer learning was employed for all models except the custom made baseline VGG6 model, but only provided presentable results for the VGG16 model.

Table 2: Test data results of the different model structures on Biotest lake data set

Model structure	Optimized Epoch	Regularization	Test accuracy	Test loss
Baseline (VGG6)	15	None	59.2%	3.283
	913	Data-augmentation and dropout	89.2%	0.635
VGG16	x	x	x	x
VGG16 TL* (Imagenet)	20	None	82.0%	0.952
	116	Data-augmentation	68.4%	1.120
ResNet50	x	x	x	x
Inception-ResNetv2	13	None	49.2%	2.200
	39	Data-augmentation	86.0%	0.821

*Transfer learning, non trainable convolutional layers, 2 trainable dense layers

5 Discussion

The performance results of the models utilizing CNNs developed in this project indicate that the task of identifying individuals of pike with a high confidence is certainly possible. It is however important to understand what these results represent at this point, and the limits to its representation. It is also important to understand in which ways the networks can be further analysed for deeper insights and improvements. Moreover, it might prove vital to view the problem of pike identification from a broader perspective, as there is much potential for discussing other approaches to the problem.

5.1 Current approach

5.1.1 Data limitations

An imperative for the success of the current approach is careful selection, sorting and pre processing of images from a diverse data set with high quality images. After all, the idea is to create feature ids for each pike individual, and this relies heavily on proper representation of the data for each pike. Therefore, the representation limit of the data inevitably constitutes the most significant limit to the representation of the results attained in this project. The Biotest lake data set is a relatively representational data set compared to other available data, but would be considered very small for many similar classification tasks. This has the effect that it is still lacking representation in some crucial ways. For the purpose of this project, one problem with the Biotest lake data set is that it does not have a prevalence of re-captures, i.e. images of the same pike from completely different periods of time. Therefore this project can produce no evidence that it is possible to classify images taken many years apart. Until there is clear evidence that pike patterns are stable, and/or identifiable by automatic systems over periods of time corresponding to the longest recorded lifespans of pike (approximately 20-30 years), there is similarly no reason to believe that the networks developed would be able to classify such images. Even though it was never specified as a goal that the network would have this capability, it can be assumed that it is an essential property in the real world application.

5.1.2 Pre-processing

In the pre-processing stage, segmentation constitutes the most comprehensive measure. Whether segmentation is an absolutely necessary pre-processing for this task is not established since the high performing models were not tested without preceding segmentation of the raw data. Only the baseline VGG6 model on the smaller initial data set was trained without first segmenting the raw data, but it was never trained again on the same data with segmentation. Therefore, the results are not decisive on the matter. It can however be assumed that segmentation reduces noise and enables the identification models to focus more directly on relevant features. Even if a deep network could perhaps look past this if properly trained on a large, diverse and high quality data set, segmentation will almost certainly reduce the complexity of the problem which simplifies training, and potentially renders a very deep model superfluous. For this project, it is unlikely that the results of any of the networks on the Biotest lake data set - and especially the quite shallow baseline VGG6 model - would have been of the same quality without segmentation, since the images of a given pike id displayed a high level of background homogeneity. Moreover, it is important to consider that the segmentation model performed well even when it was hastily set up without exploring more ways of improving it, and without optimizing it during training. In that way, it posed a small investment which at least ensured that the model did not over fit on irrelevant features in the background. If it would be required for the segmentation model to perform better, there is a variety of rather simple measures that could be taken. For example, performance would likely improve if the pike was roughly cropped before segmentation, so that there was less noise in the image. At the moment, segmented images sent as input to the identifier mostly consist of background. For pike with very complex or subtle pattern characteristics that require high resolution, this could be a problem. By first cropping the images, a larger portion of the segmented images would now instead be occupied by pike-pixels (foreground), thus preventing the background from becoming a problem. In general, more pike pixels occupying the 308x204 pixels (current format of segmented images) means more pike-pattern information per image as input to the identifier CNN, which is always desirable as we are not interested in the background.

5.1.3 Identifier model structure

When it comes to the pike-id models, the current approach is to employ the softmax classifier at the end of the network, where each node essentially learns to pick up on a specific pike-id. An advantage of this approach is that inspiration, know-how and network model structures can be derived (and in the latter case even transferred) from the vast amount of existing projects which solve similar classification problems in this way. For this project, it enabled for easy and intuitive adoption and testing of networks without them having to be extensively adapted and customized for pike classification. For some networks, attaching the 64 neuron

FC layer (pike classifier) at the end sufficed as adaptation. Focus could instead be on training the models and evaluating and optimizing them, which produced comparable and insightful performance results.

The results generated quite convincingly indicate that the pike classification task can largely be solved through adopting existing network types. As shown in Table 2, especially convincing results were those obtained from training the quite shallow baseline VGG6 model (a scaled down version of the VGG nets) with regularization. Based on the high accuracy (90%) it appears that model size and depth is not necessarily a determining factor for individual pike identification, although the InceptionResNetv2 performed almost as well without being allowed to train as extensively. The trade-off between performance and computational cost will however work in the baseline models' favor when compared to very deep models, such as InceptionResNetv2.

For a fair assessment, the obtained results do not in the end suffice as evidence for superiority of one model or the other, and it should be stated that the evaluation metrics have limited representational power. For one, they only evaluate accuracy on the data fed into the networks thus far. Despite data augmentation, certain image types which could occur in a real world situation might not be well represented. Connected to this, is the problem of the network acting as a "black box", i.e. it is not particularly clear what either of the networks look at to determine an image's class belonging. Perhaps a model that performs worse actually focuses on parts of the pike such as the pattern, which is arguably a complex and difficult feature, but one that we would desire for it to focus on. Simultaneously, a better performing model might focus on more simple features which work very well for this data set, but would not hold for the complexity of a "real world" data set. Analysing this is unfortunately not a trivial task and beyond the scope of this project. This might however change in the near future, as much current research is dedicated towards understanding the mechanisms of neural networks, where "Explainable AI" (XAI) has emerged as a field specifically intended to counter balance the "black box" problem (Gilpin et al. 2018).

5.1.4 Further analysis

The results generated thus far can be improved both when it comes to performance, and when it comes to interpreting them. In terms of performance, there are many straight forward ways of increasing the accuracy metric for the different networks since few of them got to completely finish training, i.e. stop only when they had reached their full potential. This can for example be clearly observed for the VGG16 model in Figure 20 since both training and validation curves have increasing trends by the last epoch, but might also be the case for the baseline model, see Figure 18. Experimenting with data augmentation could also be relevant in the case of extending training, since only a particular (and somewhat randomly selected) degree of data augmentation was utilized in this project. Moreover, one could also invest in tweaking hyper parameters and performing cross validation. However, extending the training sessions with more epochs, testing different forms of data augmentation, and/or performing cross validation would be a computationally expensive processes (which is why it was never done in this project), and a GPU would be preferred for such tasks.

Beyond just trying to improve the results, there are plenty of ways to analyse them further to attain deeper insights. For example, it might be worthwhile to investigate if certain pike individuals within the data set are particularly difficult for the network to classify, and therefore account for most of the error, or if the error is evenly distributed between all of the individuals. This type of analysis could be connected with attempts to visualize the model behavior via plotted feature maps and/or heat maps. Displaying what the feature maps look like, and what type of input they are activated on, gives insights into what the network "sees"[8], while heat maps highlight where the model "looks"[9]. Even if the assumption within this project is that networks will utilize the individual patterns of pike for identification, that might not necessarily be the case. The network could over fit on other trends in the data, and especially do so in undesired ways if the model is shallow and/or the data is not representative of the real world problem. Analysis of the model behavior could thereby provide evidence that the pike pattern is (or is not) the most significant trait for automatic pike identification.

Despite potential efforts aimed at developing the current approach, and further analysing the results generated from it, there are reasons for why it is probably not the most optimal approach to begin with. The main drawback is that it is impractical, and possibly difficult to implement for pike-identification in the real world situation, once the network is active. The potentially endless stream of new pike-ids would have to be learnt individually and be appended to the network retroactively once it is running, since the final FC-layer is by its nature stable and not dynamic. This is computationally expensive since the network has to be re-trained each time a new id-neuron is added to the final FC layer. It is also a process of high uncertainty, since it is not a completely trivial task for the network to determine whether a pike individual is new or already existing. For example,

[8]https://cs.nyu.edu/ fergus/papers/zeilerECCV2014.pdf

[9]https://jacobgil.github.io/deeplearning/class-activation-maps

one could introduce a threshold which the networks would have to satisfy with a certain confidence before adding a new pike, but this still renders the risk of omitting certain pike from the network, or representing a certain pike individual with more than one id-neuron. Another solution could be to only implement predictions made by the network after verifying them via a control-step. This step could consist of comparing and verifying meta-data between the new image and perhaps the top five or so pike-ids that the network predicts, such as pike location, length, or height. It could also consist of human supervision, such as having experts verify the identity of recently introduced pike which the network is less certain of.

In any case, a control step is also associated with many problems and in itself could potentially render the functionality of the network overly complicated and impractical. For one, attaching, storing and accessing many different types of meta data is not a computationally light task, and requires a large degree of manual data management. More importantly, if the manual and human-reliant aspect is something that is intended to be phased out, introducing heavy manual data management contradicts the purpose in the first place. This is especially the case if the system also relies on human supervision of the model output.

It will be advantageous (or even necessary) to include aspects of human control, like manual data management or human supervision, in the initial stage of running the system, and perhaps beyond that to some extent. In the end, after evaluation and verification, it is however more desirable to have all the key functionalities, of which automatic addition of new pike individuals is one, compacted within the automatic parts of the system pipeline. This currently seems like a difficult criterion to fulfill with the current approach.

5.2 Alternative problem approach

An alternate approach currently being considered consists of instead implementing a "Siamese network". This is a method which has been successfully employed in a few previous projects of individual animal identification with limited data (Schneider et al. 2019). The core idea is that instead of having the network learn what each individual looks like, it is instructed to learn to distinguish between two arbitrary individuals. This means that if we introduce an individual to the network it will not find its specific signature and match it with an existing signature, and this in turn means that we do not need the final FC-layer. Instead, it will run two identical parallel (Siamese) CNNs where one takes in the input image, and one takes in images from one pike-id at the time from the database. The networks will output two feature maps in parallel, and a type of loss function (discriminator) will measure the discrepancy between the outputs, thereby predicting whether the input was of the same individual or not. This is then repeated for each pike-id in the data base, where each id could be represented by one or more images. The discriminator will be motivated by gradient descent to learn how to distinguish between individuals, while the Siamese networks are frozen since they must be identical at all times.

A Siamese network would be convenient for this project, and for pike identification in general, in several ways. First of all, the identical parallel networks can be any type of CNN which successfully extracts and outputs relevant features. This seems to be achieved by the classifier CNNs used in this project so far, so after removing their final FC-layers they could just be transferred to the Siamese network. The same type of pre-processing of data would still be useful or required, and the parallel networks would be more or less operational in their current state. Where the major alterations and innovations to the system would have to be made concerns data selection and structure, along with discriminator architecture. The requirements on data would be less strict, since there is no need for an abundance of images per individual for the Siamese network. Many of the individuals omitted for this project because they only consisted of 2-4 images could be added to the Siamese network data base. In fact, even pike ids consisting of only one pike can be used, since the network learns to distinguish between pike from both correctly and incorrectly predicting true positives (matches) and true negatives (non-matches). Many of the pike ids with few pike images were (coincidentally) re-capture data, so a Siamese network would directly open up for better representation of re-captures. The currently available re-capture data might not suffice regardless, but it would be a start nonetheless.

During training, the Siamese network would have to be carefully set up so that all images are set up in pairs with the correct label, being either true positives or negatives. This involves a slightly more complicated data structuring, but is straight forward once set up. The main trade off of the Siamese network when it comes to data, is that it will need many different individuals rather than many images per individual. At the moment, it is difficult to say if the Biotest lake data set fulfills this, and perhaps it would have to be extensively expanded upon with new pike individuals from many other data sets.

Another advantage of the Siamese network is that it could generalize to identification tasks beyond just pike. Compared to properties of typical classification networks, distinguishing between individuals is a property which in theory could be implemented for other animal species in general, and fish species in particular.

In other similar projects, Siamese networks have been used in this way, where they have been successful (although to different degrees) in identifying same individuals for completely different species. At least in the long term perspective where the goal could be to expand the application of the network, this speaks in great favor of the Siamese network approach.

6 Conclusion

This project has shown that individual pike identification with high confidence can be achieved with a system that segments images before inputting them to a convolutional neural network. The accuracy reached upwards of 90% for the quite shallow VGG6 network (4 convolutional layers) on a test data set that represents many features present in the real world situation. Data augmentation and drop out helped the network generalize well, and improved performance significantly. A very deep network is therefore not necessary to solve the task with the current data.

Further analysis should be aimed at investigating how well the network generalizes to a larger (or differently augmented) and more diverse data set, as the data augmentation implemented thus far can not account for all types of diversity that might appear in the real world. Particularly identification of re-captures of pike many years apart can not be verified based on the current data set, as no sufficient amount of such data is contained therein. Going forward, other CNN systems which avoid problems inherent to the typical soft-max classifier approach should also be considered. The Siamese network is one such promising system which could be adopted for pike identification.

Bibliography

Berggren, Terese (2019). “Increased body growth rates of northern pike (Esox lucius) in the Baltic Sea – Importance of size-selective mortality and warming waters”. In:

Bryhn, Andreas et al. (2019). *Fisk- och skaldjursbestånd i hav och sötvatten 2019*. Havs och Vatten myndigheten.

Fickling, Neville J. (1982). “The Identification of Pike by Means of Characteristic Marks”. In: *Aquaculture Research* 13, pp. 79–82. URL: https://doi.org/10.1111/j.1365-2109.1982.tb00033.x.

Gilpin, Leilani H et al. (2018). “Explaining Explanations: An Approach to Evaluating Interpretability of Machine Learning”. In: *CoRR* abs/1806.00069. arXiv: 1806.00069. URL: http://arxiv.org/abs/1806.00069.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville (2016). *Deep Learning*. MIT Press. URL: \url{http://www.deeplearningbook.org}.

He, Kaiming et al. (2015). “Deep Residual Learning for Image Recognition”. In: *CoRR* abs/1512.03385. arXiv: 1512.03385. URL: http://arxiv.org/abs/1512.03385.

Kristensen, Emil et al. (2020). “Fingerprinting pike: The use of image recognition to identify individual pikes”. In: *Fisheries Research* 229, p. 105622. ISSN: 0165-7836. DOI: https://doi.org/10.1016/j.fishres.2020.105622. URL: http://www.sciencedirect.com/science/article/pii/S0165783620301399.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey Hinton (Jan. 2012). “ImageNet Classification with Deep Convolutional Neural Networks”. In: *Neural Information Processing Systems* 25. DOI: 10.1145/3065386.

Larsson, P., P. Tibblin, and P. et al Koch-Schmidt (2015). “Ecology, evolution, and management strategies of northern pike populations in the Baltic Sea”. In: *AMBIO 44, 451–461 (2015)*. URL: https://doi.org/10.1007/s13280-015-0664-6.

Lecun, Yann et al. (Dec. 1998). “Gradient-Based Learning Applied to Document Recognition”. In: *Proceedings of the IEEE* 86, pp. 2278 –2324. DOI: 10.1109/5.726791.

Murphy, Kevin P. (2013). *Machine Learning: A Probabilistic Perspective*. Cambridge, Mass. [u.a.]: MIT Press. ISBN: 9780262018029 0262018020. URL: https://github.com/kerasking/book-1/blob/master/ML%20Machine%20Learning-A%20Probabilistic%20Perspective.pdf.

Pan, Sinno Jialin and Qiang Yang (2010). "A Survey on Transfer Learning". In: *IEEE Transactions on Knowledge and Data Engineering* 22.10, pp. 1345–1359.

Ronneberger, Olaf, Philipp Fischer, and Thomas Brox (2015). "U-Net: Convolutional Networks for Biomedical Image Segmentation". In: *CoRR* abs/1505.04597. arXiv: `1505.04597`. URL: `http://arxiv.org/abs/1505.04597`.

Rosenblatt, Frank F. (1958). "The perceptron: a probabilistic model for information storage and organization in the brain." In: *Psychological review* 65 6, pp. 386–408.

Ruder, Sebastian (2016). "An overview of gradient descent optimization algorithms". In: *CoRR* abs/1609.04747. arXiv: `1609.04747`. URL: `http://arxiv.org/abs/1609.04747`.

Schmidhuber, Jürgen (2015). "Deep learning in Neural Networks: An Overview". In: *ScienceDirect* 61, pp. 85–117. DOI: `https://doi.org/10.1016/j.neunet.2014.09.003`.

Schneider, Stefan et al. (2019). "Similarity Learning Networks for Animal Individual Re-Identification - Beyond the Capabilities of a Human Observer". In: *CoRR* abs/1902.09324. arXiv: `1902.09324`. URL: `http://arxiv.org/abs/1902.09324`.

Simonyan, Karen and Andrew Zisserman (2014). "Very deep convolutional networks for large-scale image recognition". In: *arXiv preprint arXiv:1409.1556*. URL: `https://arxiv.org/abs/1409.1556`.

Srivastava, Nitish et al. (2014). "Dropout: A Simple Way to Prevent Neural Networks from Overfitting". In: *Journal of Machine Learning Research* 15.56, pp. 1929–1958. URL: `http://jmlr.org/papers/v15/srivastava14a.html`.

Szegedy, Christian, Sergey Ioffe, and Vincent Vanhoucke (2016). "Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning". In: *CoRR* abs/1602.07261. arXiv: `1602.07261`. URL: `http://arxiv.org/abs/1602.07261`.