Master in Software Development and Technology

Thesis Project

# Object Detection and Classification of Canine Hip Dysplasia with Convolutional Neural Networks

Submitted by

**Johan Bender Koch**

June 3, 2018

# Contents

# Abstract

All over the world, canine breeding individuals are required to have their genetic disposition to hip dysplasia rated. These evaluations are standardized by organizations such as World Canine Organization (FCI) who rate the health of hip on a scale from A to E. Given the size of our acquired hip dysplasia dataset, we utilize transfer learning and feature extraction of x-ray images in order to develop a CNN capable of hip dysplasia classification. To accomplish this, we test if classification of grayscale x-rays (one-channel) converted into RGB format (three-channel) can take advantage of pre-trained models on a RGB dataset. First, we pre-process the dataset and transform the x-rays. Then, we train a network for feature extraction on a dataset of annotated hips by using a pre-trained model on ImageNet. Finally, we perform different experiments to optimize the classification accuracy of a pre-trained classifier. In our feature extraction, we obtain a mAP of 1.00 in Pascal VOC format and an average IoU of 0.92. Based on an unbalanced dataset of our feature extractions, consisting of 7,444 observations, a f1-score of 0.71 is obtained on five-class classification on the test set. Further, we show that by separating the dataset in subsets of [A, B, C] and [D, E], it is possible to obtain a f1-score of 0.96 on the test set. Thus, it is concluded that transfer learning with CNN models, trained on RGB image datasets, can be successfully applied to grayscale x-rays of hip dysplasia for classification.

# Acknowledgements

I would like to extend special thanks to Helle Friis Proschowsky from Dansk Kennel Klub (DKK) for allowing access to the data and taking the time to explain the challenges within canine hip dysplasia. Through Proschowsky, I was introduced to Dorte Hald Nielsen, lecturer at University of Copenhagen (KU). Hald Nielsen is responsible for hip dysplasia diagnostics and has been evaluating thousands of cases over the years. I would like to thank Hald Nielsen for her patience in explaining the HD evaluation process. Without her evaluations, the thesis work would not have been possible. In fact, since all the training data is labelled by Hald Nielsen, the model is in effect aiming to replicate her evaluation process. I also want to thank Fintan McEvoy, Professor of Veterinary Imaging in the Department of Veterinary Clinical and Animal Sciences. McEvoy's understanding and interest in the intersection of computer vision and veterinary sciences has been critical in pushing the project forward. Not only did McEvoy transfer all the raw image data, he also provided guidance with regards to feature extraction and how to work with DICOM files. Lastly, I would like to thank my supervisor, Jes Frellsen, Associate Professor at the Department of Computer Science of IT University of Copenhagen, for his support when the project faced challenges.

# Preface

As a Msc Design-track student in Software Development and Technology with a B.Sc. degree in Economics, I have 1.5 years of formal education in computer science prior to the thesis work. My electives have been focused on data science and hence for my thesis I opted for a project in which I could extend my experience with traditional machine learning techniques such as clustering, decision trees, etc. into deep learning applications. During the thesis project, an object detector and classifier for analysis of canine hip dysplasia were developed in python. The object detector is able to detect canine femoral heads, the highest part of the thigh bone, based on x-ray images from 186 different dog breeds. In order to train the object detector, a custom dataset was generated by semi automating the annotation process. Based on the hip extraction, an image classifier was subsequently developed to classify the hip. Both models take advantage of transfer learning and neural network architectures that have been proven to perform well in other object detection and classification tasks.

# 1 Introduction

Hip dysplasia (HD) is a common health concern for dogs worldwide. The condition is heritable, and therefore national canine breeding organizations evaluate the hip condition of breeding individuals. These evaluations have been standardized among countries, with the most commonly used evaluation approach based on grayscale x-ray images. Although the evaluation process is standardized, there is still a debate among national breeding organizations about the extent to which the standard is upheld. With an accurate classifier, however, it will be possible to objectively review thousands of HD evaluations from different breeding organizations and detect potential biases. Supervised models in computer vision have recently facilitated many breakthroughs within the medical field. However, in order for the supervised models to perform on par with human experts, lots of labeled data are required. In this thesis project, an unbalanced dataset of almost 21 thousand grayscale observations, before preprocessing, is gathered for classification. Given the size of the dataset, we will explore if transfer learning on pre-trained red, green and blue (RGB) images from ImageNet can be applied to grayscale medical images. The scope of the thesis includes image preprocessing and implementing the CNN architecture. To limit noise, areas of interest will be extracted from the canine x-rays before classification. The provided datasets contain canine x-ray images and evaluations that the Danish Kennel Klub (DKK) has contracted to experts from University of Copenhagen (KU) between 2012 and 2017. The x-ray images are in DICOM format, which will be converted and standardized for the neural network architecture.

## 1.1 Methodology

The methodology of the thesis can be divided into two parts: Exploring the problem domain of HD and researching the field of computer vision. For the problem domain, information about HD has mainly been collected through on-site communication with Lecturer Dorte Hald Nielsen and Professor Fintan McEvoy from University of Copenhagen. The main objective was to understand how the evaluations were conducted and identify potential bias and issues within the dataset. For the field of computer science, information has been collected via state of the art analysis presented in research papers and blog posts. This approach was used to take advantage of the latest research from technology companies such as Microsoft, Google, Facebook, and Baidu, as well as leading academic research institutions. From this analysis, it was discovered that the effectiveness of transfer learning on RGB images seldom was described in relation to medical grayscale images, hence this approach was explored. In order to determine the best suited deep learning implementations for the thesis, their effectiveness and complexity were assessed based on their performance on public image datasets. As a foundation to understand the architectures, a general understanding of neural networks was obtained from online literature. Among the different object detection models, YOLO v2 was chosen. In order to implement the object detector, a dataset of annotated images was generated and customized to the architecture of the object detector. The results from the object detector were subsequently used in various classification experiments with the classifier model known as ResNet-152.

## 1.2 Datasets

As explained in the introduction, the datasets used in this thesis includes medical x-ray images in DICOM format from KU and medical evaluation scores, in a csv file, from Danish Kennel Klub. Both databases cover the evaluations conducted between 2012 and 2017 with an average of 3,483 per year (totaling 20,898 evaluations).

### DKK Dataset

In the DKK database, the most important attribute is the hip evaluation scores - ranging from A to E - of the left and right hip. The final score per evaluation is based on the worst score of the two hips. Depending on the breed, a different threshold score is required in order for the dog to be approved for breeding. That said, the evaluation criteria is the same across breeds (Hald Nielsen pers. comm.). The evaluation scores from A to E should be interpreted the following way.

- A: Excellent hip
- B: Few signs of concern
- C: Mildly dysplasia form
- D: Moderate dysplasia form
- E: Severe dysplasia form

Given the goal of developing a classifier based on the results of the object detector, it is imperative to analyze the dataset distribution across the different hip scores. If for example there are very few instances of B observations, it can be difficult to train a robust neural network. In fact, the model might learn that it is a bad idea to predict B purely based on probabilities as opposed to learning a proxy for the evaluation procedure. The risk of working with unbalanced datasets can to some extent be mitigated by adjusting the weights of the classes in the loss function of the model, but is still not an ideal situation. Below is the summarized distribution of the hip scores in the dataset before preprocessing:

- A: 70 percent
- B: 14 percent
- C: 9 percent
- D: 5 percent
- E: 2 percent

From the distribution above, it is evident that there is a clear skew toward hips classified as 'A and B' which accounts for 84 percent, compared to 'C, D and E' which accounts for just 16 percent. Outside of hip evaluations, the DKK database also provides a strong data-foundation to investigate the heritability of the hip condition (breed, mother's score, father's score, etc.). Here is an overview of all the attributes included in the DKK dataset:

*Table 1: Description of Attributes from DKK Database*

| **Attribute Name** | **Description** |
|---|---|
| Regnr | number referencing to country region |
| Idnummer | unique id given to each dog |
| Race | specific breed of the dog |
| Navn | name of the dog |
| RontgenDato | date that the image was taken |
| Type | hip or elbow evaluation |
| Sagsnummer | the case number |
| H | the score for the right hip |
| V | the score for the left hip |
| Stat | worst score of the two hips (the final score) |
| Kvalitet | the score for the quality of the image |
| Position | score on how symmetric the dog is placed |
| Far | name of the father |
| St | hip score of the father |
| Mor | name of the mother |
| St | hip score of the mother |
| KU | if evaluation was made by KU |
| Udenlandsk | if a foreign evaluator was used (only when an evaluation is appealed) |

### KU Dataset

Through a database extraction at KU, image files totalling 192 GB were collected. A csv dataset gathered from the meta-data contained in the DICOM images was generated by running a custom python script that extracted the meta-data from the individual images. Since DICOM contains a standardized list of meta-data, of which many are not relevant to this study, the script only extracted relevant meta-data for the thesis-work:

- Filename
- Width of the image
- Height of the image
- PatientID
- Accession Number (equaling to sagsnummer in the DKK database)

The two databases are linked together with two different unique ids: Accession number (from KU dataset) and sagsnummer (from DKK dataset) and patientID (from KU dataset) and idnummer (from DKK dataset). The patientId is a 15 digit long id that uniquely identifies every dog as well as the country that it is from. An analogy to this would be our passport codes. The reasoning behind using the patientID as a unique identifier is the fact that dogs only are evaluated for their

breeding potential once. In the rare exceptions, in which the results are appealed, the new evaluation will overwrite the old in the DKK database and the x-ray will be distributed to another HD expert in Scandinavia to avoid an evaluation by the same evaluator.

# 2 Canine Problem Domain

As the thesis is conducted in computer science, it is not expected that the reader will have an understanding of the domain of canine science. This chapter seeks to give the reader an understanding of the importance of HD and how the evaluations are conducted. This will serve as a foundation for assessing the rationale behind the implementation of an object detector and classifier further in the report, as well as understanding the limitations of the dataset. The information from this chapter has been obtained from conversations with Proschowsky from DKK and Hald Nielsen and McEvoy from KU.

HD is a genetic condition in which the head of the thigh bone does not fit with the hip socket. Dogs with HD often have difficulties with standing up or have issues with walking straight on their back legs. In addition to heritage, environmental factors such as food and lifestyle are also important factors to consider. To limit the risk of inheriting HD, Federation Cynologique Internationale (FCI), World Canine Organization in English, has created global standards for HD evaluations. FCI is the largest organization that standardizes HD screenings, however, there are also alternative HD schemes including Orthopedic Foundation for Animals (OFA) program and Pennsylvania Hip Improvement Program (PennHIP) [1]. For the scope of this thesis, these programs will not be considered as the foundation of their evaluation approach is different. Instead, we will first aim to understand the rationale used for evaluating our dataset. Of the 360 canine breeds that FCI acknowledges [2], 70 breeds are mandatory to have HD screening in order to be registered in the national breeding registry. Further, 17 breeds are recommended screening (Proschowsky pers. comm.). In general, if both parents receive an approved HD score, the offspring will be registered by the national canine club. The registration is made in a national centralized database that provides the genetic history and proof of health of the parents.

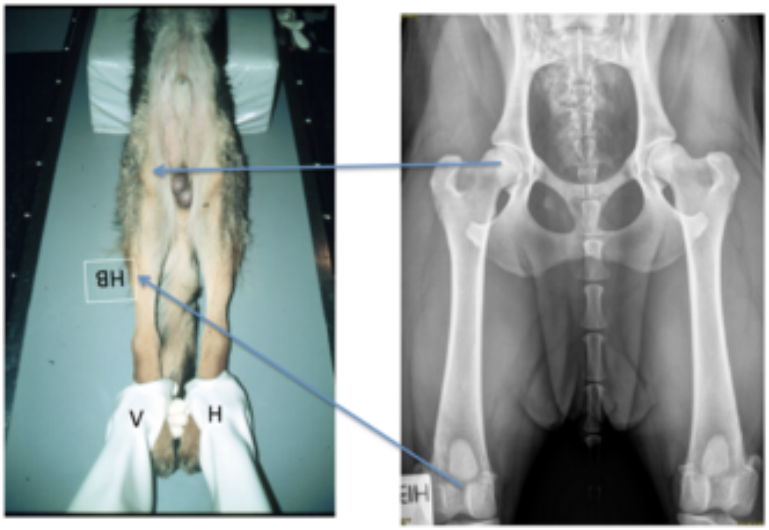



*Figure 1: The image shows how a HD x-ray is taken. Image from Hald Nielsen.*

In figure 1, a dog, in anesthesia, is placed under the x-ray machine to prepare it for the HD scan 1. Anesthesia ensures that the dog is relaxed and the hips can be placed in the correct position. In Denmark, dogs that are screened for HD often also have other joints such as elbows and shoulders checked. However, this is not a requirement on the part of FCI, and therefore is solely a decision made on behalf of DKK. That said it explains why miss-labeled elbow evaluations did appear in the database extractions of HD evaluations. Of the 3.483 HD evaluations (on average) made yearly in Denmark, some breeds are over-represented (see table 2). However, since the evaluations are made on the same criteria, no matter the type of breed, the evaluations can be used jointly to train a classifier. From table 2 it is clear that certain breeds are more disposed to HD; For example 13.8 percent of the breed Berner sennenhund received an evaluation of D or E, as opposed to only 4.3 percent of Golden retriever received an evaluation of D or E (see table 2).

*Table 2: Overview of HD evaluations of the most frequent breeds. Numbers presented are averages from 2013 to 2016.*

| Breed | Numbers | % evaluated * | A in % | B in % | C in % | D in % | E in % |
|---|---|---|---|---|---|---|---|
| Labrador | 567 | 24.5 | 80.1 | 8.3 | 6.9 | 3.5 | 1.2 |
| Schaeferhund | 464 | 27.6 | 64 | 17 | 10.8 | 6.5 | 1.7 |
| Golden retriever | 320 | 29.3 | 67.5 | 18 | 10 | 3.4 | 0.9 |
| Berner sennenhund | 123 | 55.9 | 63.4 | 9.8 | 11.4 | 11.4 | 2.4 |
| Ruhaaret hoensehund | 121 | 24.6 | 74.4 | 12.4 | 4.1 | 4.1 | 0 |
| Border collie | 115 | 38.9 | 73.9 | 5.2 | 3.5 | 3.5 | 1.7 |
| Rottweiler | 111 | 29.2 | 70.3 | 10.8 | 9 | 9 | 0 |
| Broholmer | 72 | 52.9 | 65.3 | 11.1 | 9.7 | 9.7 | 0 |

** The percentage evaluated refers to how many of the registered dogs of the given breed also has been evaluated for HD.*

In 2016, a joint study of KU and DKK aimed to evaluate whether screening for HD helped improve the genetic base by analyzing several generations of the Schaeferhund breed (Proschowsky pers. comm.). The study found that evaluations have generally improved year over year as the ratings of D's and E's have decreased (see figure 2).

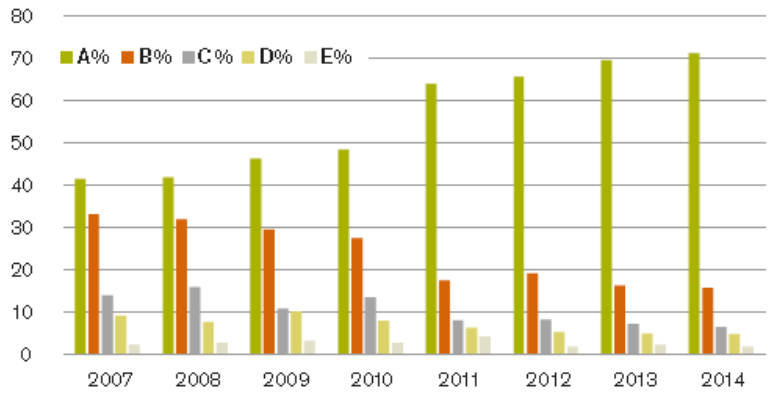



*Figure 2: HD evaluations of Schaeferhund from 2007 to 2014. Graph provided by DKK*

## 2.1 Hip Evaluation Process

With the intention of building a classifier, it is useful to understand how the x-ray images are collected and evaluated. Not only does this provide a fundamental understanding to some of the challenges in the pre-processing step, it also provides insights about the optimal training input of the classifier. Based on information from Hald Nielsen, the following step-by-step description of the evaluation process was provided. First, once dog-owners decide to have their bitches (female dogs) or male dogs approved for breeding, they need to take an x-ray at a veterinary clinic. In order to capture the x-rays, every veterinarian needs to get a certification teaching them the correct positioning during the scan. The certificate ensures a standardized way of capturing the x-rays and hence a fair way of evaluating HD. In relation to machine learning, this is an important characteristic of the dataset as standardized data make it easier for a model to generalize. The intuition is that standardized data has less noise, thereby making it easier for the model to detect the relevant patterns.

After taking the x-rays, the veterinarians will send the images to the Veterinary Imaging Department of Veterinary Clinical Science at University of Copenhagen (Hald Nielsen pers. comm.). This part of the analysis is contracted out by the Danish Kennel Klub (DKK). Upon evaluating for HD, it is first necessary that the positioning of the dog is symmetric and the legs are parallel. If the legs are not symmetric and parallel, the hip bone inside the hip socket will rotate and make it challenging to properly analyze (Hald Nielsen pers. comm.). Figure 3 shows a HD x-ray scan in which the dog has parallel legs.

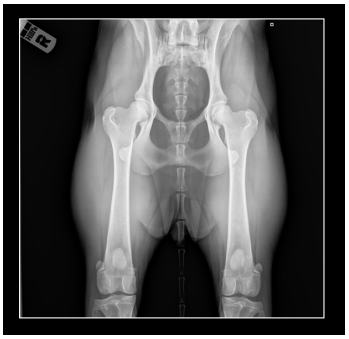


*Figure 3: X-ray of hip region from dog positioned with symmetrical pelvis and parallel hind legs. This represents optimal condition for HD evaluation. 'R' annotates the right side. Image from DICOM file.*

Hald Nielsen, who has conducted lectures in HD evaluations, described the technical evaluation procedure of HD as follows:

1. First, we define the center of the femoral head of each hip bones by using a transparent sheet of plastic with concentric rings engraved. The following angles should be depicted: 80, 90, 100, and 105 degrees respectively as shown in figure 4. The center of the femoral head, the highest part of the thigh bone, is defined by the arc of the dorsal and medial circular sectors respectively.

2. Then we localize the intersection between the cranial and the dorsal acetabular edge which is called the cranial effective acetabular rim (see figure 5).

3. Lastly, we determine the Norberg angle formed by a line connecting the cranial effective acetabular rim and the center of the femoral head and the line connecting the centers of the femoral heads (see figure 4).

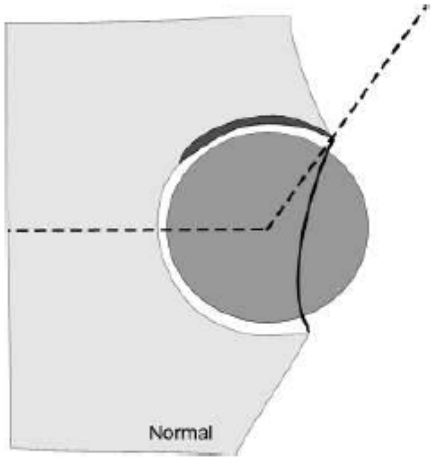



*Figure 4: Defining the Norberg angle of a normal hip*

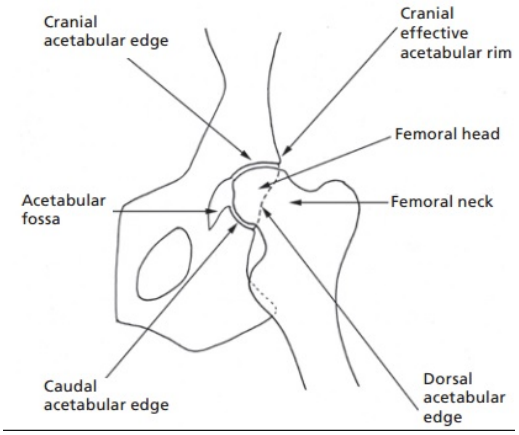



*Figure 5: Overview of the technical terms of the hip bone*

*Figure 6: Illustrations depicting the evaluation of HD. Images provided by Hald Nielsen*

The smaller the Norberg angle, the less encapsulated the femoral head is and the more severe the HD is. For a hip to be classified A or B, the angle should be around

105 degrees. In figure 7, we see different grades of hip health. In addition to the Norberg angle, other radiographic findings are also taken into account that are less well defined and can vary based on what is revealed in the x-ray (Hald Nielsen pers. comm.). The HD evaluation is, therefore, a subjective evaluation based on a set of certain principles. The degree of the condition cannot be confirmed like a binary case of cancer or no cancer, which can be assessed over time. Therefore, within reason, it is possible to argue that a C hip is actually a B hip. As such, in the Nordic countries, it is possible to request a re-evaluation. In these cases, the original x-ray image is sent to another Nordic country for analysis. For example a re-evaluation of an x-ray image from Sweden could be sent to KU. After the evaluation, the result will be sent to the Swedish database. Therefore, a few images from the KU database are missing from the DKK evaluation. However, these cases are rare and within Denmark only 10 HD cases were submitted for re-evaluation in 2016 (Proschowsky pers. comm.). In the case of regular HD evaluations by KU, the results are sent to DKK, and the images are stored in the KU database.

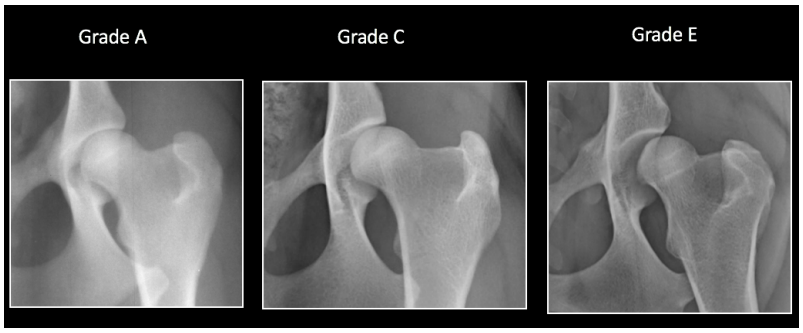



*Figure 7: X-rays images illustrating different Norberg angles and evaluation scores. Image from Hald Nielsen*

In terms of increasing chances for a better evaluation, it is believed that age can play a factor as younger dogs do not always show as strong signs of HD as when they are fully grown (see figure 8). As per the rules of FCI, it is possible to have screenings made when the dog is between one and 18 months, depending on the breed (Hald Nielsen pers. comm.). Knowing this, professional breeders have a tendency to get their dogs evaluated as early as possible because it is favorable for the evaluation.


*Figure 8: X-rays from the same dog at one and two years of age. Not all dogs exhibit such a drastic change from year one to two. Image from Hald Nielsen*

Within radiology, it is a rule that images of patients are always presented as if the patient was facing the doctor face-to-face. This ensures that the orientation of left and right of the patient is not confused. To further ensure the correct orientation, radiologist and other medical professionals are trained to place a tag, demonstrating the right and left of the patient during a scan. However, as seen in the hips in figure 9, doctors are not always consistent with this procedure and some images are miss-oriented. This problem was discovered during image annotation for the object detector.

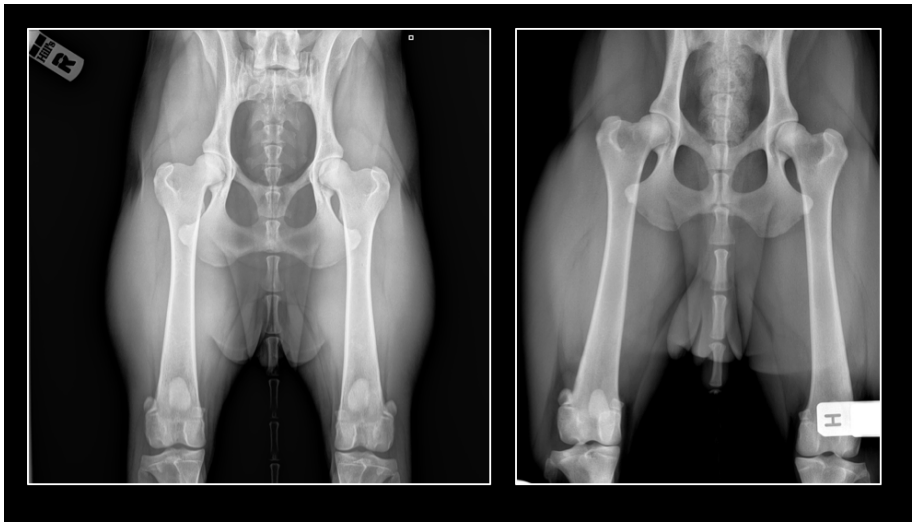


*Figure 9: On the left, we see an x-ray with correct orientation - 'R' on the correct side when facing the doctor. On the right, we see an x-ray with incorrect orientation - 'H' on the wrong when facing the doctor.*

In the x-ray scans, the orientation tags appear with different abbreviations using both Danish and English terminology. There are no accepted international standards for the orientation tags and in Sweden, for example, Latin terms are frequently used. In our dataset, left is usually labeled as VB, V, or L and right is usually denoted with

H, HB or R. As the tags appear with different font-style and the images also can include other text, it can be challenging to define a simple detector for. Therefore, although it is easy for a human evaluator to identify, it becomes a problem for a computer vision system, unless it learns how to differentiate between the different tags and other text. From a sample of 100 x-ray images, 81 percent were with tags indicating the correct orientation, 11 percent were with the tags indicating the wrong positioning and 8 percent were without tags. In the cases where there were no visible tags, the evaluator from KU will assume correct orientation and hence the evaluation in the database will match the individual hips. The issue for a classification model only occurs when the orientation is wrong and when the score of the two hips differ. From an analysis of the DKK dataset, just 17 percent of the evaluations included different ratings of the two hips. Therefore, it can be concluded that a classification model, can be built based on 98 percent accurate data [1] , given that the study of 100 images is representative. Another issue that was identified was the fact that the veterinary clinics would sometimes submit multiple images and the evaluator at KU would choose the image of the highest image quality without deleting the other images from the database. Therefore, certain cases appear as overrepresented in the database and it is not possible to identify with certainty which image was the basis of the evaluation. To avoid this issue, cases with multiple images were removed in the preprocessing step.

## 2.2 Bias in Dataset

In regards to the data quality, we also need to examine risks related to certain biases in the dataset. Although not a distinct sign of bias, we learned from conversations that Hald Nielsen, the KU evaluator, is provided information that does not appear in the datasets such as the age of the dog. Further, we learned that Hald Nielsen has an intuition on how the hips of a young dog will develop for a specific breed. Although she is not allowed to take the age-factor into considerations for the evaluation, it could subconsciously impact the final evaluations. In other words, if two nearly identical hips are between a B and C, but one dog is 1 year old and the other is 2 years old, it is possible that they receive a different grade. Without age information, it can be more difficult for the classifier to predict these edge-cases.

## 2.3 Research Goals of DKK and KU

Related to the design of the classifier, both McEvoy from KU and Proschowsky from DKK have expressed greater interest in a classifier capable of detecting the health of the individual hips rather than simply the final score (the worst of the two hips). Further, an object detector for the femoral head was also of interest as it would enable other HD experiments to be investigated by McEvoy.

[1] Calculation based on the following: (1 - (0.11 images with wrong labeling * 0.17 evaluations with different HD scores))

# 3 Brief History of Artificial Neural Networks

The term Artificial Neural Network (ANN) is used to describe computer systems that are vaguely inspired by the biological networks of the brain [3]. One of the common ANN tasks aims to replicate our ability to process image inputs. In fact, developing computer vision systems on par with human capabilities have challenged researchers for decades [4]. Since recognizing objects is a task that requires you to identify abstract object patterns from different perspectives and in different light-settings, it has historically been challenging to develop a rule-based system [5]. ANNs, however, do not rely on rules-based code, but instead relies on sufficient data to iteratively develop the pattern-recognition (rules) which correctly maps input to output. Over the last few years, major improvements in terms of computer vision have been achieved thanks to deep neural networks (DNNs), a type of ANN, larger datasets and utilizing multiple GPUs for training [5].

Before diving into the technicalities of these architectures, I would like to introduce the inner-workings and history of ANNs first. Although ANNs have recently become a hot topic within both academia and industry, the fundamental ideas have been around for decades. The first contribution to ANN is considered to be a paper published by McCulloch and Pitts in 1943 [6]. In their work, they tried to understand how the brain can compute highly complex behaviours, using processing units as simple as neurons. This paper proposed a mathematical design of calculating an output based on neurons with several weighted inputs. However, the implementation by McCulloch and Pitts was limited as the model only included a single layer and was missing an algorithm for learning.

That said, the artificial neuron designed by McCulloch and Pitts, is the foundation of neurons in modern ANNs (see figure 10). In modern ANN designs, the computation inside a neuron is to multiply the input by a weight, then add bias and finally transformed the input via an activation function. The activation function is what ensures that the network learns non-linear mappings between the inputs and outputs. There are different activation functions, but what they have in common is that the functions are non-linear [7]. This will be explained in more detail in the chapter about CNNs 4.

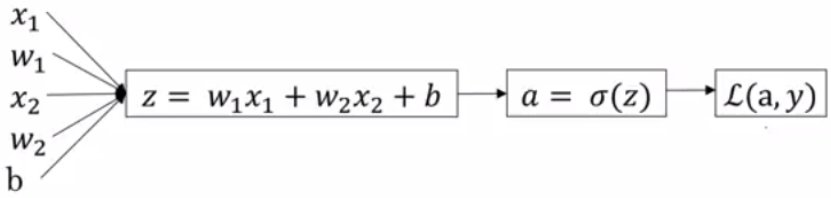



*Figure 10: Illustration of computation inside a neuron of modern ANN. Inputs are multiplied by weights, added a bias, and then transformed via an activation function. Image from DeepLearning.ai*

## 3.1 Rosenblatt's Multilayered Network

In 1958, Rosenblatt developed a neural network classifier, based upon the ideas of artificial neurons by McCulloch and Pitts. The network was a binary classifier, mapping a series of inputs to a single binary output:

$$f(x) = \begin{cases} 1, & \text{if } w * x + b > 0 \\ 0, & \text{otherwise} \end{cases} \tag{1}$$

Rosenblatt's implementation was called Mark I Perceptron [8], a network composed of one layer of trainable parameters randomly connected to the input layer and an output of 8 binary neurons. Although Rosenblatt also argued for implementing a multi-layered architecture, like we see in image classifiers today, he was not able to produce noteworthy results with a multi-layered approach due to the lack of an efficient way of tuning the parameters. In effect, without a learning algorithm, the hype about artificial intelligence and neural networks imploded and research suffered a drastic cut in funding during the seventies to mid-eighties [9].

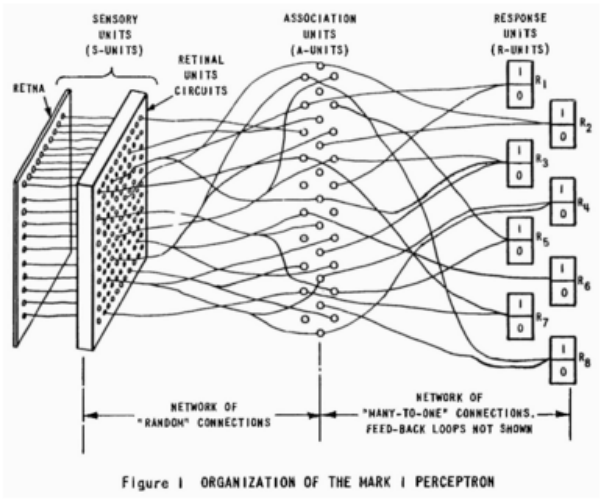



*Figure 11: Illustration of Rosenblatt's Perceptron in which neurons are sparsely connected and the model has binary outputs. Illustration from [8]*

## 3.2 Backpropagation

It was not until Rumelhart, Hinton and Williams in 1986 proposed utilizing backpropagation and stochastic gradient descent [10] that neural networks began to show promise again. In short, backpropagation works like an iterative cycle in which input is passed forward through the layers of the neural network and transformed via linear and non-linear computations 10. Once the input has reached the last layer of the network, the prediction is then compared against the ground truth label. By comparing the prediction with the ground truth, an error is calculated by a loss function. The computed loss is then used to update the weights of multiple layers of the network in order to minimize the total loss (see figure 12). To do so, backpropagation uses the chain rule to compute the partial derivative (the direction of

the error in relation to the loss-function), given that the derivative depends on the functions of the previous layers.

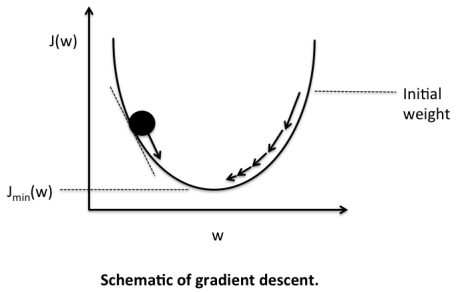



*Figure 12: Illustration of Stochastic Gradient Descent. The gradient of the loss, J(w), in relation to the weight is computed. For every iteration, a step is made towards the negative direction of the gradient in order to minimize the loss. Image from Medium*

Traditionally, the fine-tuning of the weights and bias have been computed using an optimization algorithm called Stochastic Gradient Descent (SGD). SGD iteratively estimates the best direction of the weights and biases using a subset of the whole dataset to minimize the loss function, hence an incremental improvement. SGD is different from traditional gradient descent in that the weights and biases are updated after analysing a data subset, known as batch size, which is much less computing intensive than computing the gradient for every data point. Since SGD, several alternatives have been proposed - most notably are the Adaptative Learning Methods such as ADAM. Without going into too many details, ADAM works by computing the momentum of gradient descent by taking the previous weight updates into account. ADAM has proven to converge faster than SGD 13.

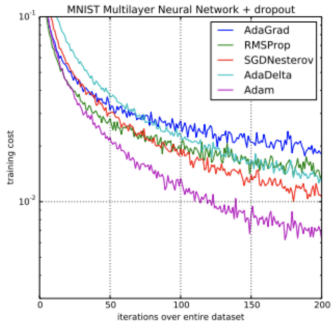



*Figure 13: ADAM compared to other learning algorithms on the MNIST dataset. Image from 'Adam: A Method for Stochastic Optimization, 2015'.*

To avoid changing the direction of the weights and biases too much at each batch, also known as over-fitting, a learning rate is introduced to decrease the direction of the weights by a specific factor. Careful tuning of this parameter is required for optimal training. Other techniques such as batch normalization can be used to ensure that small changes early in the network do not amplify deeper in the network [11]. Later in the section, batch normalization will be explained in greater detail.

### 3.3 Revival of AI Research

With the introduction of these new training techniques, research on ANNs became active again. In 1989, Yann LeCun, now Director of AI Research at Facebook, [11] developed a digit recognition system using data from the US Postal Service, and proved that ANNs could be used to solve a complex practical computer vision problem [12]. The particular kind of neural network applied by LeCun is known as LeNet and is a type of convolutional neural network (CNN). In practice, LeNet is not used anymore. Instead, it is remembered as a historic model within the field of computer vision that showed the potential of CNNs. In fact, CNNs have since outperformed all other vision algorithms in academic competitions such as ImageNet [5]. In the next chapter, we will examine CNNs and why they are so efficient at computer vision applications.

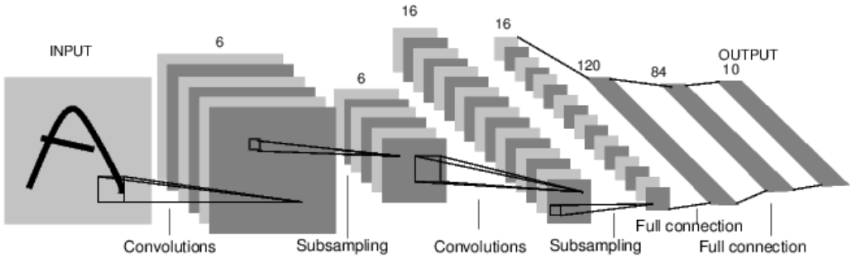



*Figure 14: The LeNet architecture - illustrated with a letter instead of a digit. As depicted, the number of feature maps increases deeper in the network, whereas the resolution decreases. Image from Research Gate*

# 4 Convolutional Neural Networks

One of the challenges with training a neural network is the number of parameters (mainly weights) that needs to be tuned. The deeper the network (number of layers) and the larger the input size, the more parameters need to be trained. In a fully connected ANN, every neuron in each layer is fully connected with every other neuron in the previous layer. Since every connection includes a weight, the number of trainable parameters increases significantly by increasing the number of initial inputs and network layers. LeNet, and other CNNs, are similar to the perceptron architecture (see figure 11) in that every neuron does not fully connect to the neurons in the previous layer. Instead, a CNN is built on layers of kernels, also known as filters, which are weight matrices that are applied across the inputs of the previous layers (see figure 15).

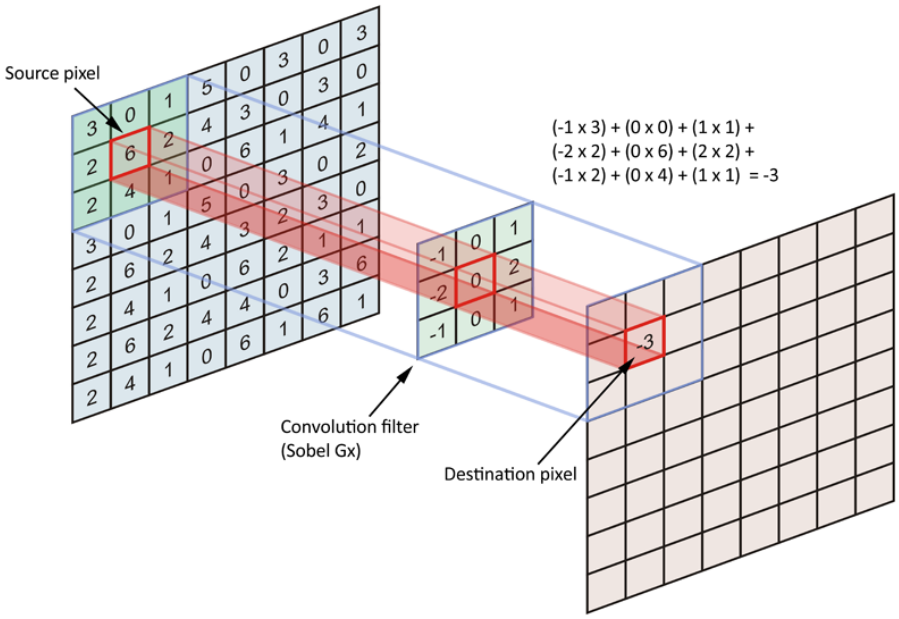



*Figure 15: Illustration of the operations inside a CNN. The weights in the kernel are applied to the input (the blue square) which produces a single output (the red square). This example is with a kernel of size 3x3. Image from Joseph Redmon.*

The architecture of CNNs drastically reduces the number of trainable parameters when compared to a fully connected network. As seen in 15 the weights in the kernel (also known as a filter) are applied to a specific region of the inputs space and producing a single output. By updating the weights inside the kernel through backpropagation, the kernel learns to detect certain general features from the input vector. This ability is particularly useful for image data as multiple filters learn to detect different features of various complexity (lines, objects, etc), which can be used across the entire image. Using a filter across an image is referred to as a sliding operation. The CNN characteristic of using a set of weights across an entire image is commonly known as parameter sharing [13]. After applying a filter across the entire image, the output is referred to as a feature map. Usually, as the inputs move deeper into the CNN, the number of feature maps increase while the resolution size decrease. This happens as multiple filters are applied at every layer and the use of max-pooling, stride or no padding decreases the input size. The reason behind the dimensionality reduction, is straight-forward given a basic understanding of how filters and pooling works. Below is an introduction to the parameters of CNN filters including window size, stride and padding:

- Window size determines the size of the filters. So in other words the number of weights. In figure 15, the window size is 3x3.
- Stride is a parameter that determines the space between the inputs as the filter is applied as seen in figure 16 and in figure 17.
- Padding allows the corners of an image to be analyzed by a filter as padding adds null values around the existing image as seen in figure 18.

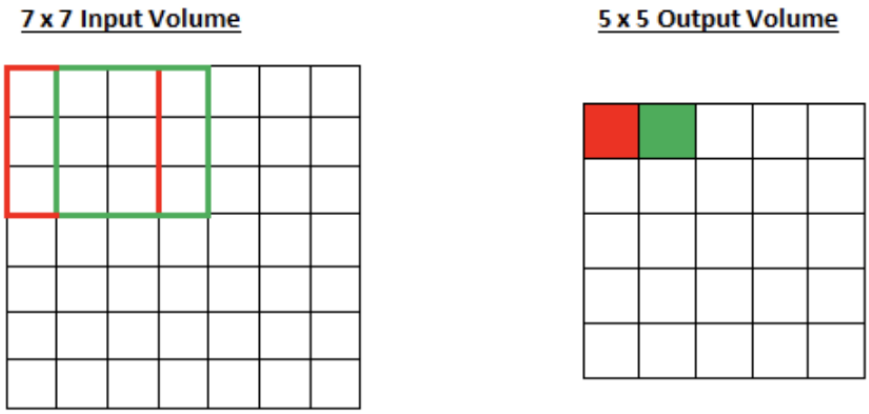



*Figure 16: Example of input and output volume with a stride = 1. Image from [14]*

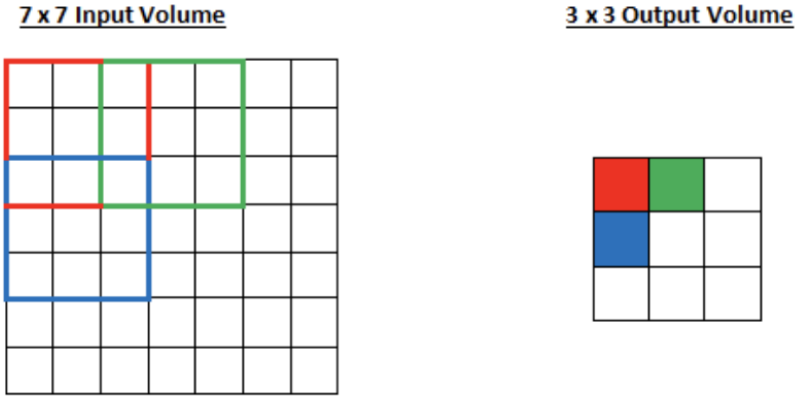



*Figure 17: Example of input and output volume with a stride = 2. Image from [14]*

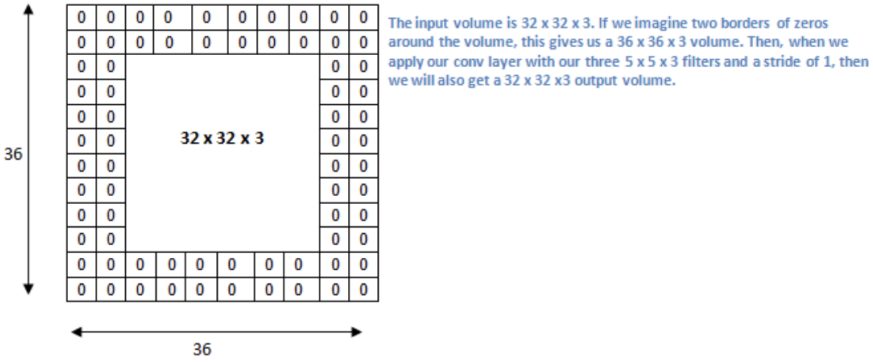



*Figure 18: Example of zero-padding of size 2. Image from [13]*

Based on the values of the input size, stride and padding, it is possible to calculate the output size. The formula is as follows:

$$OutputSize = \frac{\text{Input size} - \text{Kernel Size} + 2 * \text{Padding}}{\text{Stride}} \tag{0}$$

Instead of activating the feature map with convolution filters it is also possible to use a technique called pooling. Unlike filters, pooling does not rely on any weights to be tuned. Max pooling, for example, simply outputs the maximum input value of the window size as illustrated in figure 19. Whereas, another type of pooling called average pooling outputs the average value of the window size. Common to both is that the operation of pooling reduces the input size and that no parameter tuning is required.

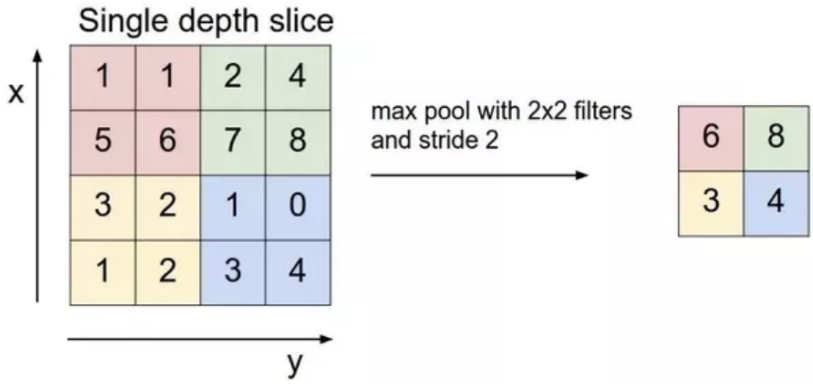



*Figure 19: Illustration of Max Pooling. As seen, the highest value from the coloured grids (left) are extracted to the grid on the right. Image from [13]*

For CNNs, the activation function called ReLu and Leaking ReLu are often used and therefore should also be briefly explained. ReLU is an example of a simple activation function; In short, it takes the input value and outputs zero if the input is zero or negative or the input if the value is positive (see left in figure 20). Leaking ReLu (on the right) is a variant of ReLU that avoids that the output is zero for inputs less than zero. The reason why it can be important to avoid zero as an output is because the value zero can subsequently turn off the next layers.

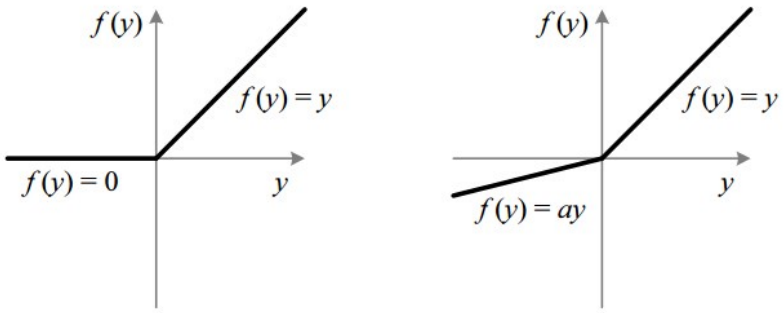



*Figure 20: Illustration of the ReLu (left) and leaking ReLu (right) activation function. Image from Towards Data Science.*

The fact that the model will ignore parts of the network during training when the output value is zero can be used to reduce over-fitting. The technique, known as dropout, aims to ensure that certain parts of the network is not over-used and thereby causing over-fitting to the training data. It works by randomly turning off neurons during the training process (setting the output to zero), which prevents the

model from over-relying on certain patterns during previous training and instead learn to identify new features that hopefully generalize better.

With all the previously described components, many different convolutional neural network architecture can be designed. Certain patterns of layers have been shown to perform particularly well, particularly in image related tasks, and have become a sort of standard. For more general purpose classifiers, the first influential CNN architecture was defined by Krizhevsky, Sutskever, and Hinton, in 2012, and has become known as AlexNet [26]. In the original LeNet architecture illustrated in figure 14, the input size was 28x28 pixels per image. However, an image size of higher resolution is required to visualize more complex objects and scenes. To give an intuition of how the parameter complexity of the input layer scales with the resolution size, consider that LeNet, in 1985, was trained on the simplest dataset for computer vision - 28x28 pixel grayscale images of handwritten digits [12]. Given this input size, the input layer of LeNet was of 784 inputs (28x28). AlexNet, on the other hand, was trained on 224x224 pixel RGB images, making the input layer consist of 150,528 inputs (2224x224x3). Not only does the input layer contain more parameters, the following layers also become more complex as the inputs are convoluted and reduced.

Today, our computing resources and data repositories far exceed the days of LeNet in 1985, and the highest image resolution applied for pre-trained networks computer vision networks of general applications are around 512x512 pixels RGB images. This includes various object detectors such as DeNet and SSD [15], which have a total of 786,432 inputs in the input layer. To put that into perspective, the input size for a state of the art object detector today is more than 3,000 times larger than the input size of LeNet.

Although CNNs have been trained on higher resolution images than 512x512, in particular in the medical field, it is a rarity as aggressive use of max-pooling is required to compress the input as used in the papers by Geras [16] and Wang [17]. However, in both cases, the data size exceeded 100.000 images, which is well-beyond my available dataset on hip dysplasia.

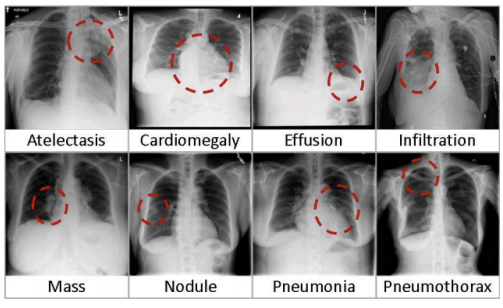



*Figure 21: Areas of interest in chest illness analysis. Image from [17]*

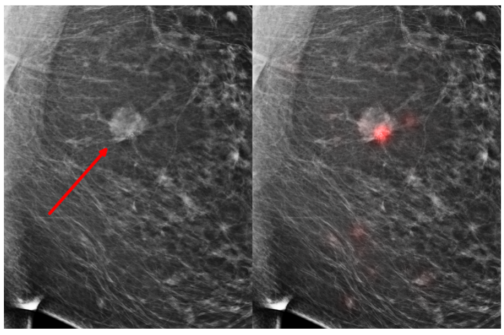


*Figure 22: Areas of interest in cancer analysis. Image from [16]*

Among classification models, they are usually designed for 224x224 pixel input. There are several reasons for this; Large public databases such as COCO [18] and Imagenet [19] contain images of a certain size. Further, it has been proven to be

a resolution that is sufficient for effective classification, and finally larger resolution sizes require more computing power, which was limited during the creation of these datasets. In table 3, some of the most known classifiers are listed as well as the resolution size they are designed for.

*Table 3: Overview of famous classifiers and their designed input size.*

| Architecture | Input Size | Year Published |
|---|---|---|
| VGG | 224x224 | 2012 |
| ResNet | 224x224 | 2015 |
| Inception V3 | 224x224 | 2014 |

To determine the efficiency of the architectures, they are tested on the same public datasets so that the results can be compared on the same foundation. One of the best-known image classification dataset is known as Imagenet, which consists of more than a million images spread across thousands of classes [4]. Since 2012, and the introduction of AlexNet [20], the improvements in classification accuracy has steadily improved to the point in which we have reached accuracy on par with humans as seen in figure 23. This is partly due to the fact that AlexNet showed the potential of CNNs in image classification and other contestants started to develop models inspired by its architecture. In fact, by 2014 all competitors at ImageNet used CNNs [4]. It should also be noted that AlexNet also introduced a novel training approach using multiple GPUs to speed up the process [20].

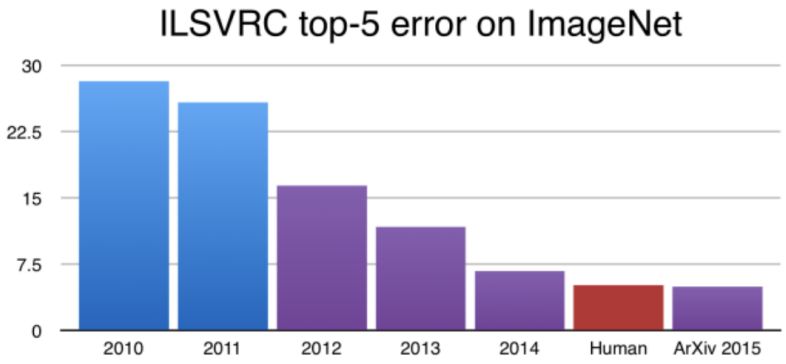



*Figure 23: Best results of the ImageNet competition from 2010 to 2015. Ever since the introduction of AlexNet in 2012, the error rate fell year over year until it reached human level in 2015. Image from Nvidia Developer Blog.*

## 4.1 Classification

On a high-level, computer vision can be divided into classification and object detection. Classification is the task of analyzing the entire image in relation to categories or a certain regressional score. One example use-case of classification includes detecting the presence of a cat in an image. Due to the ease of access to training data of cat images on the internet, and the fact that neural networks are known for requiring a lot of data, this was one of the easier implementations during the early days. Another example use-case of classification, which is relevant to this thesis, is

determining if cancer is present in a medical image. To determine the accuracy of a classifier, we analyse the precision *see formula* (1), recall *see formula* (2) and the f1 score *see formula* (3) of a model. Ideally for a classifier, it has both a high precision and recall, which is illustrated in the formula for f1-score.

- Precision is calculated as the number of correct predictions out of all the predictions of a certain class. In other words, if a model is good at accurately predicting a certain class.

  Formula for precision calculation:

$$\frac{\text{predicted instances} \cap \text{relevant instances}}{\text{predicted instances}} \tag{1}$$

- Recall is calculated as the number of correct predictions out of all the cases of a certain class. In other words, if a model is good at identifying instances of all the examples of a certain class.

  Formula for recall calculation:

$$\frac{\text{predicted instances} \cap \text{relevant instances}}{\text{relevant instances}} \tag{2}$$

- The traditional F-measure or balanced F-score (f1 score) is the harmonic mean of precision and recall.t

  Formula for f1 score:

$$F1 = \frac{2}{\frac{1}{\text{recall}} + \frac{1}{\text{precision}}} \tag{3}$$

**ResNet**

One of the famous classifier networks is called ResNet and was published in the paper Deep Residual Networks for Image Recognition on December 2015 by Kaiming He from Microsoft Research [21]. The idea behind residual networks is that the accuracy of image classification should improve in deeper networks as more complex abstractions are made possible by additional convolutional layers. However, one challenge with training deeper networks is that the gradients move towards 0 or infinity also known as vanishing and exploding gradients. This happens if consecutive weights are especially large or small, thereby making it difficult for the chain-rule in gradient descent to update the weights of the model. One solution is to more carefully initialize the random weights inside the network to limit the chances of consecutive large or small weights [22]. However, this only works to some extent. As seen in the training graph in figure 24, the accuracy of the traditional classifier does not keep improving by adding more layers. In the paper Deep Residual Learning for Image Recognition, the authors propose an architecture that includes residual blocks, allowing inputs to skip layers during training and thereby improve results with deeper networks.

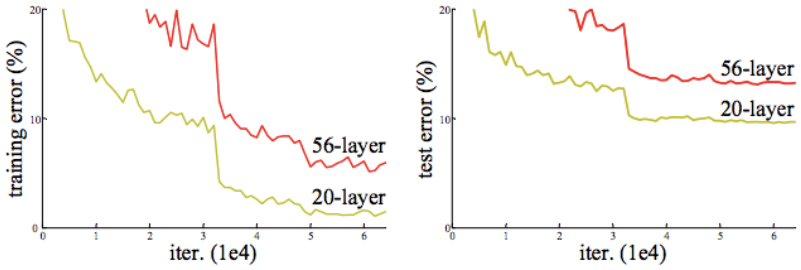



*Figure 24: Classification accuracy with shallow and deeper CNN as a function of number of iterations. The training error does not improve by training with 56 layers instead of 20 layers. Image from [21]*

The core idea of residual blocks is to introduce a so-called identity shortcut connection that learns when to skip one or more layers, as shown in figure 25. To do so, the model needs to learn a mapping of an identity function, which helps determine when the inputs skip a layer. The approach to this is to change the output function of the neurons to be computed as y = F(x) + x, rather than simply y = F(x). By making this change, the model can learn when F(x) is 0, and hence when the input is directly passed through the neuron as the output. The reason why this small tweak works mathematically, is because it is easier for the model to learn to compute a relative change (Frellsen pers. comm.).

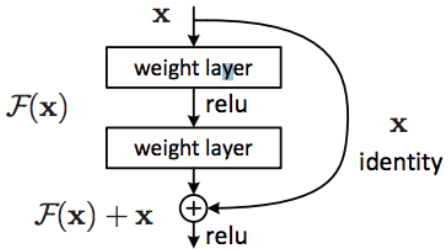



*Figure 25: Illustration of a residual block showing the concept that the input, x, passes through the block directly as output and are then added to the computation of the block, F(x). Image from [21]*

By designing a CNN with residual blocks 25, Kaiming He and his team were able to successfully train much deeper networks than previously attempted. Compared to VGG-19, a network of 19 layers, ResNet proved to be efficient at a depth up to 152 layers. Further, although ResNet was up to 8x deeper than VGG, it would still have lower complexity (number of trainable parameters) than VGG due to its network design [21]. From the ResNet experiments on various depth, the intuition that deeper networks are able to identify more complex features were proven as seen in figure 26.

| method | top-1 err. | top-5 err. |
|---|---|---|
| VGG [41] (ILSVRC'14) | - | 8.43† |
| GoogLeNet [44] (ILSVRC'14) | - | 7.89 |
| VGG [41] (v5) | 24.4 | 7.1 |
| PReLU-net [13] | 21.59 | 5.71 |
| BN-inception [16] | 21.99 | 5.81 |
| ResNet-34 B | 21.84 | 5.71 |
| ResNet-34 C | 21.53 | 5.60 |
| ResNet-50 | 20.74 | 5.25 |
| ResNet-101 | 19.87 | 4.60 |
| ResNet-152 | **19.38** | **4.49** |

*Figure 26: ResNet at different depth compared to other efficient classifiers. As seen the deepest network, ResNet-152 performs with the highest accuracy. Image from [21]*

## 4.2 Object Detection

Although classification is a powerful technique in analyzing images, it is limited to identifying a single object in an image i.e. the class with the highest confidence for a particular image. This implies that no information about the location of the class is estimated. Object detection, on the other hand, enables the detection of multiple objects in an image as well as their location. Hence, this is a more complicated task compared to classification as more information needs to be estimated: Rather than just identifying the presence of a single class, an object detector is detecting multiple classes and their respective location. Whereas classification models are able to perform on par with humans (see figure 23), we have still not reached human levels in terms of object detection, which is used for applications such as self-driving cars (in which model speed is critical) and surveillance technology. Before explaining the different approaches to object detection, it is crucial to have an understanding of how we compute the accuracy of object localization. In object localization, the goal is to improve the area of overlap between the predicted boundary box of the object and the ground truth boundary box illustrated in figure 27. To calculate the accuracy we use the formula for IoU, which stands for intersection over union. As illustrated in figure 27, an IoU score of around 0.4 is considered poor, whereas 0.7 is considered good and 0.9 is excellent. There are no exact numbers for what constitutes a good IoU score, but renowned Deep Learning professor Ng stated that a score above 0.6 is considered good [23].

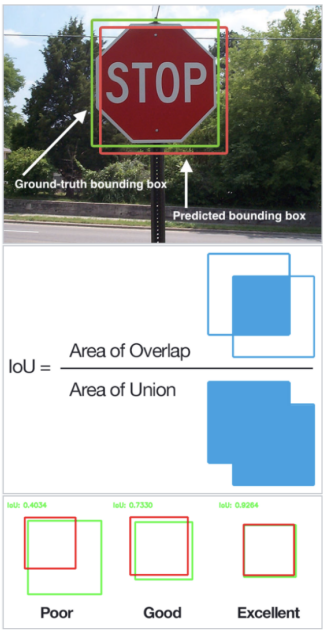



*Figure 27: Illustration of how IoU in object detection is calculated. As depicted, the greater the overlap between the ground truth and the predicted box, the greater the IoU. Image from pyimagesearch.com*

Another important object detection measurement is the mean average precision (mAP). The mAP is calculated differently for PASCAL VOC and COCO datasets as the ground truth is annotated differently. In our implementation of an object detector, we will use annotation in the PASCAL VOC format and therefore we will focus on the mAP calculation in relation to this type of annotation. Average Precision (AP) with Pascal VOC is calculated as the number of correct class predictions with an IoU over 0.5 out of a certain class. Hence, mAP is simply the average AP of all the classes. mAP is particularly interesting when building object detectors that need to distinguish many classes, of which some might appear very similar. In our implementation of HD, the number of classes are constrained to detecting left and right hips and therefore if our estimated IoU is generally above 0.5, the mAP should also be high if the model can distinguish between left and right hips.

Just like ImageNet was a public dataset that allowed researchers to train, compete and improve their classification models, Pascal VOC and COCO are the equivalent public datasets used to train object detection models. The two datasets vary in their format and object annotation detail. Whereas Pascal VOC is a dataset of images with annotated boundary boxes of objects stored in xml format (see figure 29), COCO is a dataset which identifies the objects pixel by pixel (see figure 28). Therefore, the COCO dataset is much more expensive to generate as it requires more precision and time to annotate every object.

*Figure 28: Example of COCO annotated image file. Image from Facebook Research*

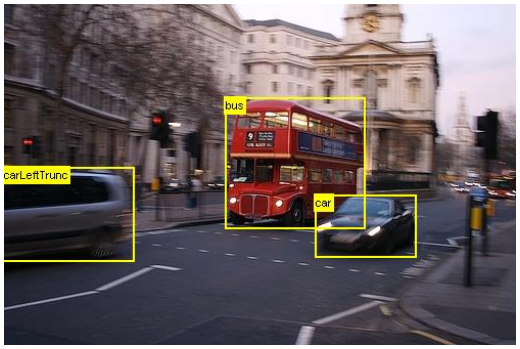



*Figure 29: Example of Pascal VOC annotated image file. Image from University of Oxford.*

Now that we know the goal of object detection, we can explore how some different object detection architectures work. Historically, the first attempt to build object detectors was to repurpose classifiers by slicing the image into smaller cells and perform detection over the individual cells [24]. This was done by sliding the classifier with a given window size over the cells as seen in figure 30.

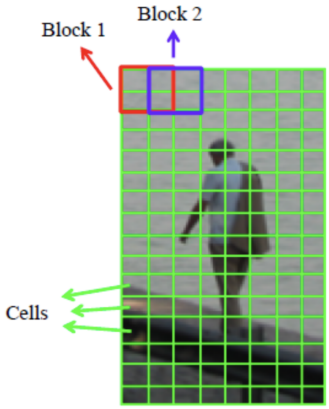



*Figure 30: Illustration of how a classifier can be used in object detection divided into a grid of cells. Image from StackOverflow.*

At each window step, you run the classifier to predict the most likely object inside the current window. As such, this approach can give several hundred predictions for an image depending on the input and cell size. To limit the number of predictions, a confidence threshold is used so predictions with low confidence are removed. This approach of a re-purposed classifier has the disadvantage of not being able to detect multiple objects inside one cell and also have a disadvantage when it comes to detecting objects that are separated by two cells. Further, the computational resources for this approach are intensive as inputs are passed to the classifier multiple times. Lastly, the classifier does not specify the location of the object - solely which object was detected in which cell. In other words, this approach is a rather brute force approach to object detection, which is both slow and inaccurate. A more intelligent alternative than using a classifier over every cell of an entire image is by algorithmically identify areas that are likely to contain objects. This is referred to as generating region proposals.

### R-CNN and SSD

R-CNN is one of the most famous object detection model built on the idea of region proposals [25]. Although, R-CNN was an improvement to the brute force approach previously explained, it was still slow as it relied on separate models: One for region proposals and one for classifications (see figure 31). Having two separate models also made training computer resource intensive. Since the inception of R-CNN, multiple improvements to the original version has been introduced such as Faster R-CNN. However, the approach with region proposal had a lower ceiling in terms of computing speed, given the constraints of the network design.

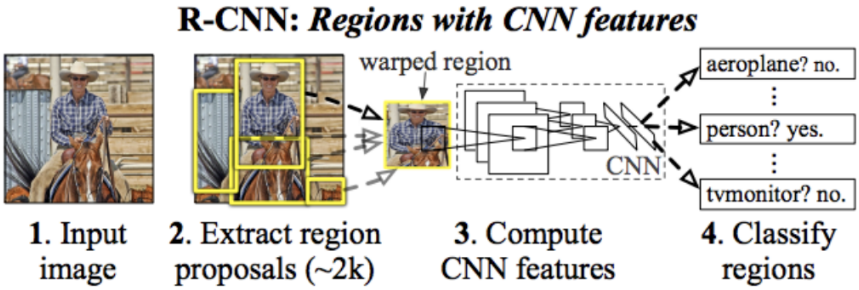



*Figure 31: Object detection with R-CNN is composed of two steps: First, the model identifies region proposals. Then, a classifier is used on top of those region proposals. Image from Towards Data Science.*

Today, two of the most efficient models for object detection are YOLO and SSD. SSD, also known as Single Shot MultiBox Detector, is developed by UNC, Google and University of Michigan [15]. The model is inspired by the successfully CNN classifier architecture known as VGG-16, but rather than the fully connected layers in the end for object classification, the architecture has been revised to detect class-agnostic boundary boxes based on feature maps from different layers in the network. This approach is known as a single shot approach. The illustrations in figure 32 and figure 33 show how boundary boxes are detected from multiple layers in the network.

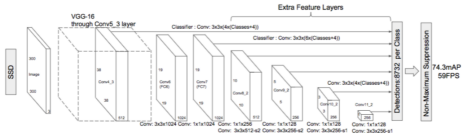



*Figure 32: SSD Network Architecture*

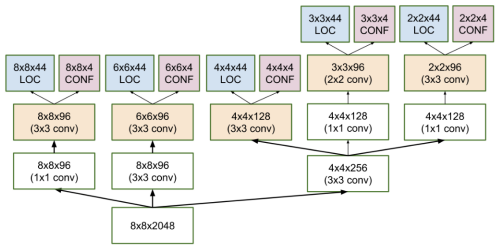



*Figure 33: How multi-detection is performed at different layers*

*Figure 34: Illustration of SSD architecture and single shot approach. Predictions for object detection are made from different layers in the network. Images from Towards Data Science*

To optimize the predictions of the boundary boxes, the SSD loss function computes both confidence and location loss. The confidence loss measures how confident

the network is at predicting the specific class. Categorical cross-entropy is used to compute this loss. Location Loss measures how far away the predicted bounding boxes are from the ground truth from the training set. L2-Norm is used to compute the location loss.

SSD has become known as one of the most accurate approaches to object detection. When SSD was released in November 2016 its performance was 75.1 percent mAP on 500x500 pixel input on the VOC dataset [15]. When trained on the same dataset but with a resolution of 300x300 pixel input, the accuracy was 72.1 percent mAP showing that higher resolution can positively impact detection.

### YOLO

YOLO, also known as You Only Look Once, was published by Joseph Redmon from University of Washington in collaboration with Facebook AI in June 2015 [24]. Compared to the other object recognition architectures previously mentioned, YOLO relies on a single end-to-end neural network with outputs at the end of the network. As such, YOLO is not a traditional classifier that has been re-purposed for object detection. Instead, YOLO is a single shot approach designed to solve a single regression problem - straight from pixel inputs to proposed bounding boxes and class probabilities. This makes YOLO perform faster when compared to the other object detectors, yet despite the speed upside YOLO also pertain a high accuracy on boundary box predictions as seen in figure 35.

| Model | Train | Test | mAP | FLOPS | FPS |
|---|---|---|---|---|---|
| Old YOLO | VOC 2007+2012 | 2007 | 63.4 | 40.19 Bn | 45 |
| SSD300 | VOC 2007+2012 | 2007 | 74.3 | - | 46 |
| SSD500 | VOC 2007+2012 | 2007 | 76.8 | - | 19 |
| YOLOv2 | VOC 2007+2012 | 2007 | 76.8 | 34.90 Bn | 67 |
| YOLOv2 544x544 | VOC 2007+2012 | 2007 | 78.6 | 59.68 Bn | 40 |
| Tiny YOLO | VOC 2007+2012 | 2007 | 57.1 | 6.97 Bn | 207 |

*Figure 35: YOLO and YOLO v2 results on Pascal VOC datasets compared with different designs of SDD. YOLO v2 performs at the same accuracy as SSD 500, but is significantly faster. Image from Joseph Redmon.*

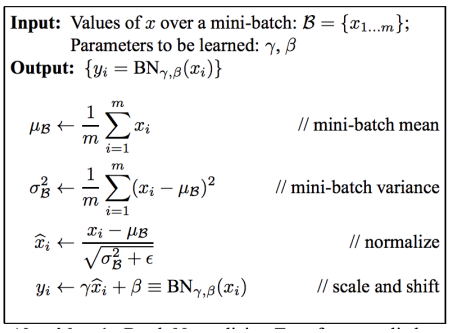
Since the original YOLO paper was released, a newer version known as YOLO9000 or YOLO v2 was released on December 25th 2016. The updates to YOLO included techniques to improve the accuracy, increase the number of object detection per image and extend the number of classes for categorization. As seen in figure 35, YOLO v2 is just as accurate as SSD500, but it is much faster as it computes 67 frames per second instead of 19. For applications such as self-driving cars or surveillance cameras, it is important that the object detection model is fast and does not require too much computing resources, so it can be completed on-device. In the case of HD, we are only interested in accuracy.

The building blocks of YOLO's architecture is simply using convolutional layers with a 3x3 kernel and max-pooling with a 2x2 kernel. For every filter (3x3), the padding parameter is set to 'same' ensuring that the output feature map has the same dimensions as the input. The only operation in which the size is compressed is during the max-pooling (2x2) effectively halving the image size every time. While the resolution decreases deeper into the network, the number of feature maps increases. Before the last layer, there are 1024 feature maps of 13x13 pixels. The resolution of each feature maps is exactly 32 times smaller than the original input of 416x416. The number of feature maps is finally reduced to 125 by using 125 of 1x1x1024 filters (see figure 36).

| Layer | kernel | stride | output shape |
|---|---|---|---|
| Input | | | (416, 416, 3) |
| Convolution | 3×3 | 1 | (416, 416, 16) |
| MaxPooling | 2×2 | 2 | (208, 208, 16) |
| Convolution | 3×3 | 1 | (208, 208, 32) |
| MaxPooling | 2×2 | 2 | (104, 104, 32) |
| Convolution | 3×3 | 1 | (104, 104, 64) |
| MaxPooling | 2×2 | 2 | (52, 52, 64) |
| Convolution | 3×3 | 1 | (52, 52, 128) |
| MaxPooling | 2×2 | 2 | (26, 26, 128) |
| Convolution | 3×3 | 1 | (26, 26, 256) |
| MaxPooling | 2×2 | 2 | (13, 13, 256) |
| Convolution | 3×3 | 1 | (13, 13, 512) |
| MaxPooling | 2×2 | 1 | (13, 13, 512) |
| Convolution | 3×3 | 1 | (13, 13, 1024) |
| Convolution | 3×3 | 1 | (13, 13, 1024) |
| Convolution | 1×1 | 1 | (13, 13, 125) |

*Figure 36: The layers in the Yolo v2 network. Image from Kaggle*

Aside from max-pooling and convolution layers, batch normalization and leaking relu are frequently used in YOLO. These concepts have been previously explained. However, Batch normalization works a bit different with CNNs, and hence a more in-depth understanding is required. As previously explained, batch normalization is simply the operation of adjusting the input, so that they represent a similar distribution at each individual layer of the network. This can help speed up training results and stabilize training of deeper networks [11]. More specifically, batch normalization changes the inputs of the mini-batch before activation at each layer so it has unit standard deviation and zero mean before it is scaled and shifted by a beta- and gamma value which is learned during training (see figure 37). Beta is the scale factor, while gamma is the variance factor. By using batch normalization it causes the values to become more stable in the network, which improves the accuracy [11]. Specifically with CNNs, batch normalization works across the multiple filters of each layer, meaning that the normalization is computed across multiple means of the feature maps. Batch normalization can be applied at different stages of a CNN, however, in the original implementation it is used before the activation function [11].



**Input:** Values of $x$ over a mini-batch: $\mathcal{B} = \{x_{1...m}\}$;
Parameters to be learned: $\gamma, \beta$
**Output:** $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma^2_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma^2_{\mathcal{B}} + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma \widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

*Figure 37: The steps to compute batch normalization. Image from [11]*

To detect the boundary boxes, YOLO v2 down-samples the images of 416x416 pixels by 32 into 13x13 grids. For each grid cell up to 5 boundary boxes are predicted. Therefore YOLO v2 is able to detect up to 160 (32 times 5) objects inside an image. The output of this operation is a one-hot vector containing the predicted confidence, center and dimensions of the boundary box as well as the predicted classes. To optimize the predictions of the class prediction and boundary boxes, YOLO's loss function is divided into five parts that together compute the total loss from which the gradients are tuned (see figure 38).

$$\lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[ (x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right]$$

$$+ \lambda_{\textbf{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[ \left( \sqrt{w_i} - \sqrt{\hat{w}_i} \right)^2 + \left( \sqrt{h_i} - \sqrt{\hat{h}_i} \right)^2 \right]$$

$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left( C_i - \hat{C}_i \right)^2$$

$$+ \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{noobj}} \left( C_i - \hat{C}_i \right)^2$$

$$+ \sum_{i=0}^{S^2} \mathbb{1}_{i}^{\text{obj}} \sum_{c \in \text{classes}} (p_i(c) - \hat{p}_i(c))^2$$

*Figure 38: The total loss function of YOLO is composed of five terms. The expressions are explained individually below. Image from Towards Data Science*

1. A term to penalize for bad localization of the center of the boundary box
2. A term to penalize for bad detection of in-accurate height and width dimensions
3. A term to predict whether an object is present there based on IOU of the ground truth
4. A term to calculate the confidence close to 0 when there is no object in the cell

5. A term to compute the classification loss from the class category predictions.

As mentioned, YOLO v2, compared to the original YOLO, is more accurate in predicting boundary boxes and also is able to classify from more nuanced subclasses such as certain canine breeds instead of just dog. The exact differences between YOLO and YOLO v2 in regards to prediction accuracy can be seen in figure 39. As seen in figure 39, several design changes impact the increase from 63.4 mAP to 78.6 mAP. The specific details regarding the improvements can be found in the paper [26].

| | YOLO | | | | | | | YOLOv2 |
|---|---|---|---|---|---|---|---|---|
| batch norm? | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| hi-res classifier? | | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| convolutional? | | | | ✓ | ✓ | ✓ | ✓ | ✓ |
| anchor boxes? | | | | ✓ | ✓ | | | |
| new network? | | | | | ✓ | ✓ | ✓ | ✓ |
| dimension priors? | | | | | | ✓ | ✓ | ✓ |
| location prediction? | | | | | | ✓ | ✓ | ✓ |
| passthrough? | | | | | | | ✓ | ✓ |
| multi-scale? | | | | | | | ✓ | ✓ |
| hi-res detector? | | | | | | | | ✓ |
| VOC2007 mAP | 63.4 | 65.8 | 69.5 | 69.2 | 69.6 | 74.4 | 75.4 | 76.8 | **78.6** |

*Figure 39: The figure illustrates the design improvements of YOLO v2 with the original YOLO. As seen, the addition of several new characteristics to YOLO v2 improves the mAP on Pascal VOC 2007 from 63.4 to 78.6. Image from [24]*

## 4.3 Transfer Learning

Lastly, before diving into the implementation of the object detector and classifier for HD, it is important to cover the idea of transfer learning. One of the challenges with training neural networks is the fact that the models require a lot of training data before they perform well. For example within the field of image recognition, ImageNet contains more than a 1 million training images distributed equally over a thousand classes. As explained in previous chapters, a CNN learns to detect simple lines and shapes in its first layers, but deeper in the network more complex levels of abstractions are learned. Since some of the same patterns inside the filters (curves, lines, eyes, etc.) can be relevant together at detecting similar classes, it makes sense to train large datasets of various classes.

*Figure 40: Notice that the first-layer weights are very nice and smooth (image on left side), indicating nicely converged network. Images from [13].*

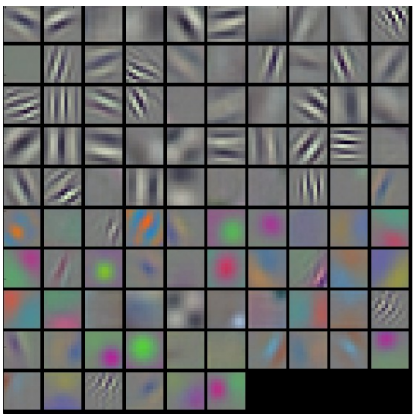


*Figure 41: Typical-looking filters on the first CONV layer of a trained AlexNet. In this layer, it appears as if the filters both learn RGB and grayscale patterns.*

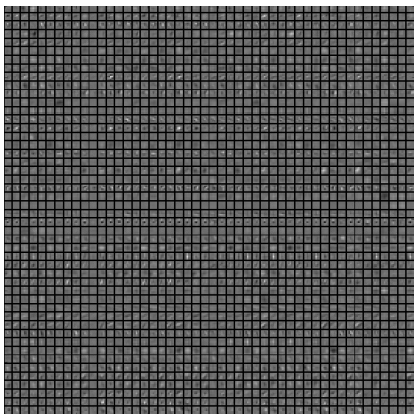


*Figure 42: Typical-looking filters on the 2nd CONV layer of a trained AlexNet. As seen, the number of filters increases in order to detect more complex features.*

Additionally, the fact that a CNN learns general rules, can be utilized to re-train a network for new classes. This type of training is known as transfer learning, as learned patterns from one domain are applied to another domain. As an example, it is possible to learn a model pre-trained on ImageNet to recognize raccoons, a class it has not been trained on, with relatively sparse data (200 examples). This is due to the fact that the model already knows how to recognize features like eyes, faces, fur, etc, from training on other classes on ImageNet. In relation to the dataset for this thesis, the data size is below 20,000 examples (after preprocessing) and not balanced across the different category classes. In fact, only 873 and 333 instances are available of hip calls D and E respectively. Given that previous network such as LeNet, which was designed for a less complex task, still requires 60,000 instances to be trained from scratch, albeit on an inferior network design, it is fair to assume that our HD dataset will not suffice without a pre-trained model. That said, the degree to which a pre-trained model is better (or perhaps worse) would have to be tested before a proper conclusion can be made.

Since transfer learning is mainly applicable due to the general filters learned in the earlier layers of the network, it can be debated if a deeper network like ResNet is advantageous. That said, today's number 1 ranking Data Scientist on Kaggle, known as Bestfitting, usually begins with a ResNet like architecture when starting a new project [27].

# 5 Considerations Regarding Classification Input

Given the sparse amount of HD data, it has been explained that transfer learning most likely is a requirement. However, since the dataset of x-ray images consist of one-channel images i.e. grayscale, a pre-trained model on grayscale images would have to be used or the x-ray images would have to be converted to a three-channel format. If not then the input would not fit the weights of the pre-trained model.

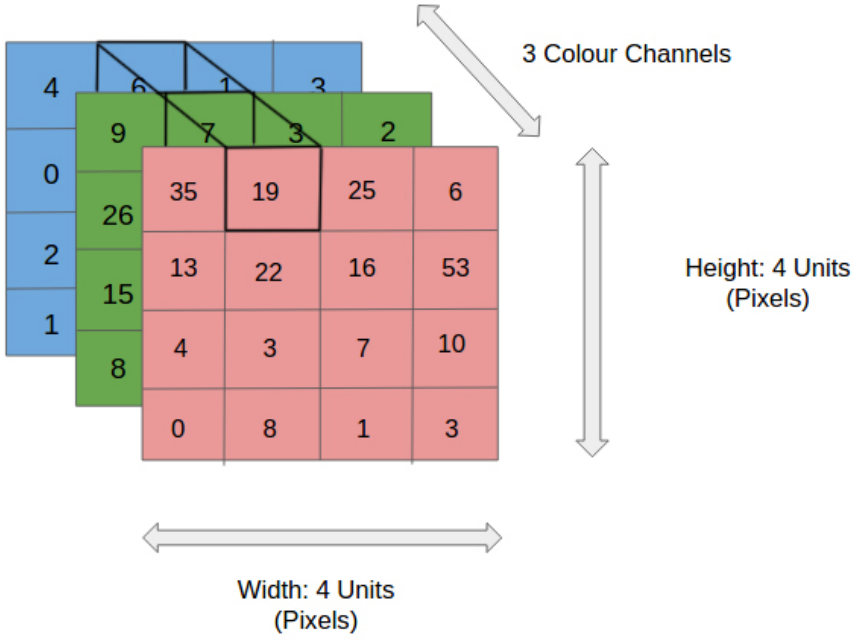



*Figure 43: Illustration of the input layer of a CNN trained on RGB images. Since the inputs are three-channel, the corresponding weight matrix will be of depth 3 as illustrated with the black outlines of a 1x1x3 weight matrix. Image from XRDS.*

Among grayscale public datasets, MNIST is the most famous which consist of 28 by 28 pixel images of hand-written digits. Both the resolution and the complexity is deemed too low for a HD classifier. Hence, another option would be to find a pre-trained model in which all images of ImageNet had been converted from three-channel images into grayscale. Such a pre-trained model could not be identified somewhere online and hence would have to be trained with the available resources which were deemed too costly in terms of money and time. Therefore, it was decided to convert the grayscale images into RGB format. The risk related to this approach is that the pre-trained model has learned colour sensitive patterns (see figure 41) and although the grayscale images are converted into three-channel images, they are still black and white and hence do not reflect the training data of the pre-trained model. Whether this risk is relevant will have to be examined through actual experiments.

From the conversation with Proschowsky from DKK and McEvoy from KU, it became clear that they were more interested in a model that could predict the score of the individual hip than simply the worst score of the hips (the final score). We also learned that in the evaluation process Hald Nielsen would both analyse the totality of the x-ray for proper positioning as well as zoom in on the hip socket. To replicate this process, we could either feed the network the full x-ray for classification of the final score (see figure 44) or for classification of the individual hips (see figure 45).

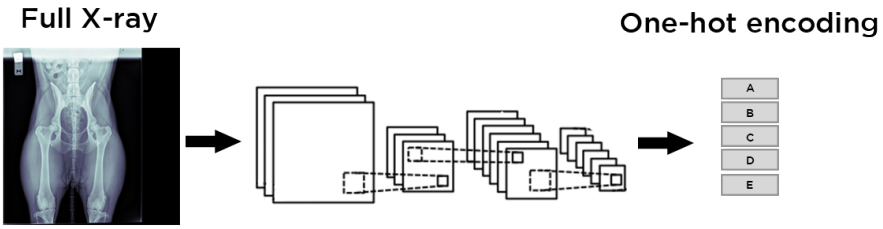



*Figure 44: Sketch of CNN classifying straight from x-ray to the worst score of the two hips i.e. a one-hot encoded vector that is 5 long*

The latter approach, depicted in figure 45, is more challenging as the model is required to predict from an output-space five times as large as in figure 44. When the model is required to predict the two hips based on one image, the output space is 25 long (5x5) in a one-hot encoded vector instead of just 5, making the task more difficult. Given the limited data-size, noise in the full x-rays might challenge the model in identifying the correct mappings. Therefore, figure 44 is preferred for experimenting with classification of a full x-ray.

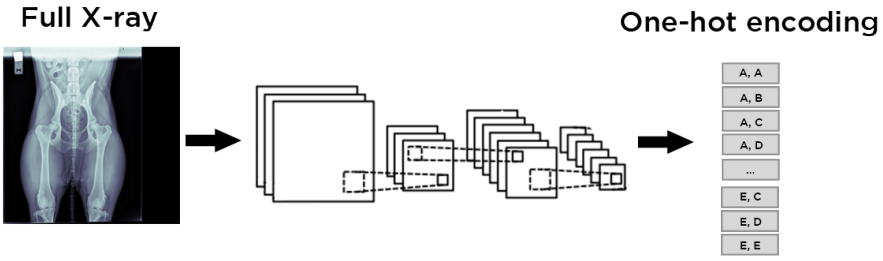



*Figure 45: Sketch of CNN classifying straight from x-ray to two hip scores i.e. a set of 25 combinations*

If a model was to be trained from scratch i.e. not rely on transfer learning, it would be possible to design the network for two image inputs. In this case, the input could be a version of the entire x-ray, combined with a hip extraction as seen in figure 46. These multi image models are not supported by available pre-trained models and this design is hence discarded is a viable option.

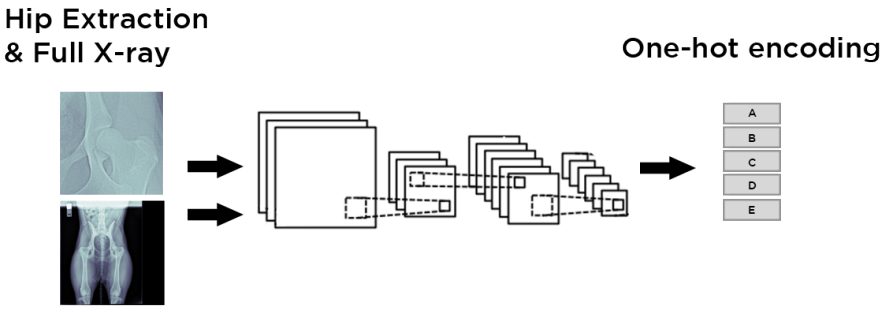



*Figure 46: Sketch of CNN classifying straight from full x-ray and hip crop to the worst hip score i.e. a one-hot encoded vector that is 5 long*

After analyzing the collected evaluations from DKK, it was discovered that the positioning, rated from 1 to 3, was only rated 3, the worst score, in less than 1 percent of the cases and rated 2 in 15 percent of the cases. Further, Hald Nielsen separately mentioned that solely focusing on the hip socket should provide enough data for a proper evaluation. In other words, she did not deem the positioning score very important. Hence, the desired architecture of the classifier was chosen to be based on individual hip extractions as illustrated in figure 47.


*Figure 47: Sketch of CNN classifying straight from x-ray hip crop to the hip score i.e. a one-hot encoded vector that is 5 long*

# 6 Data Preprocessing

Before implementing any object detector or classifier, the datasets must be examined and prepared. The data pre-processing for this thesis can be distilled into the following steps:

1. Converting DICOM into jpg images and storing DICOM meta-data in a separate csv file.
2. Cleaning datasets from KU and DKK by matching unique ids.
3. Separating images based upon evaluation scores.
4. Transforming images so that they are ready for an object detector.

As the first step, we need an understanding of the DICOM format. In short, DICOM is a standard for storing and transmitting medical images that is used worldwide in radiology and hospitals [28]. The file format includes an image-array, similar to a png or jpg file, as well as descriptive information regarding the patient, image, etc. To view a DICOM image, a special program is required. Therefore, it was decided to extract the image-arrays from the DICOM files, so standard python image libraries could be applied. Although DICOM is a file standard, different x-ray machines sometimes use slightly different notations (McEvoy pers. comm.). This can cause some challenges when reading DICOM files, especially if a program is designed to read multiple files consecutively. To extract information from the DICOM files, two separate python script were written; One to extract the meta-data from the files and one to convert the images. Both used the python library pydicom [29], which allows to read and write DICOM files in python. It was observed that not all veterinarians included the standard required information. To avoid that the script crashed while iterating through the files and extracting the meta-data, attributes were set to 0 values whenever the data was missing. The extracted information was ultimately saved inside a csv file. From the csv file observations with 0's were then later deleted. Further, images from which the width exceeded the height were deleted as well. This step was conducted because we know that the images should be taller than wide, as that reflects the build of a dog. One way to avoid this issue would be to rotate the images, however, the direction of the flip would have to be manually supervised to avoid upside-down images.

Although the DICOM format is commonly used in the medical field, it is practical to save the image information separately as a PNG or JPEG so standard python image libraries could be used on the dataset. To store the images the fromarray function from pydicom was used to read the image array and save it. Since some of the DICOM files are duplicates, saved in separate sub folders, the shutil.copy function from pathlib was applied in case that the path already existed. The reason why the same DICOM file would reside inside multiple folders is because the files received from KU were extracted from a database (PACS) that is not tailored for bulk downloads. After all the images from the DICOM files and the corresponding meta-data were extracted, a script was used to ensure that the two datasets (from KU and DKK) were matching. In this process, images with corresponding unique ids were moved into a new sub folder. Files that were not moved were ultimately removed. After this data cleaning process, the resolution sizes of the images were visualized to better understand the resolution variance across the dataset.

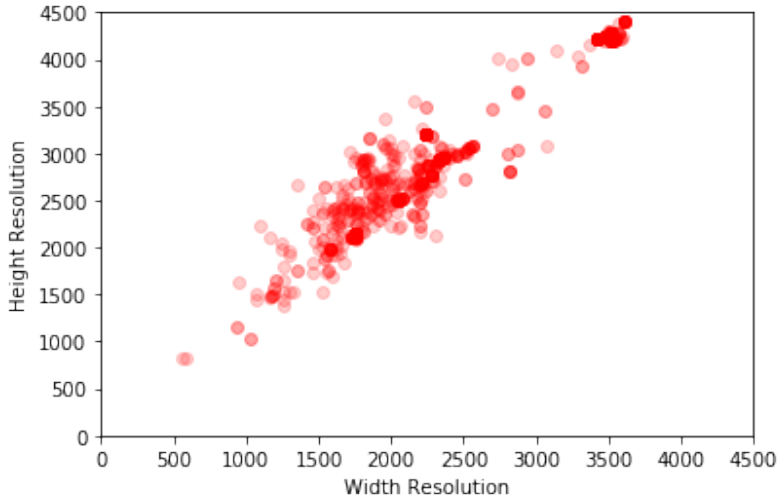



*Figure 48: The graph shows the variance in resolution sizes of the HD dataset. From the images, we can identify a linear relationship between width and height.*

As seen in figure 48, the observations approximate a linear relationship between width and height. In other words, although the resolution differs, the ratio between the width and height is somewhat consistent. When using neural networks it is required to standardize the inputs ie. having the same resolution size for the entire dataset. Clearly, the resolutions of the images vary a lot - spanning from approximately 800x500 to 4500x3500 pixels - however when the ratio is similar it is possible to resize while avoiding drastic augmentations. To avoid any distortions to the images, it was decided to add padding to all images, giving them an equal dimension in width and height. This was decided because most pre-trained models, which can be used for transfer learning, were trained on square images. Inside the matrix of an image array, the order begins from the top left corner. Therefore, padding was added to the right side of the image to keep the same order of the original digits inside the image array 49. This is useful as the images also require rescaling, and since padding is added to the end of the array, the position can easily be re-calculated of the original image without having to take the variance of padding into account.

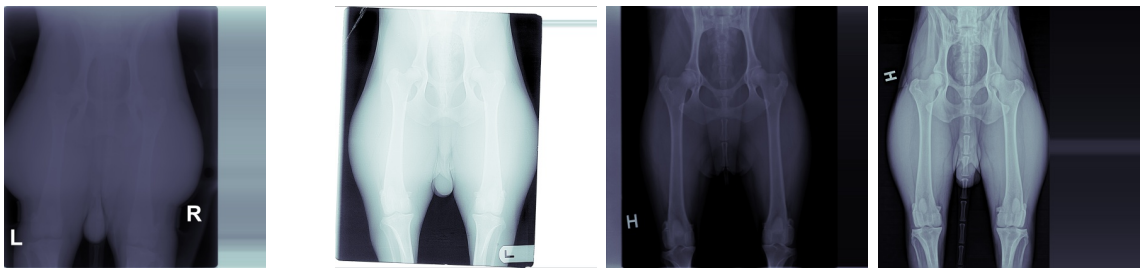



*Figure 49: Illustration showing how padding were added to x-rays to standardize the image size.*

For neural networks, there is an upper limit for how large you can design the input layer and still be able to effectively train the network. The limit is a bit of an art as it depends on the architecture such as the number of layers, types of layers and amount of data. Generally, for object detectors, they are rarely trained on images

exceeding 512 pixels in width and height. For an image of 512x512 pixels extended into a one-hot encoded vector the length of the vector is 786,432 inputs long. [2]. If the image is in grayscale, which is a format ranging between white and black, the length of the vector would be divided by three. However, since pre-trained models for transfer learning are almost exclusively trained on RGB images, it makes sense to take the three colour channels into account. In relation to our problem domain, one important question is if transfer learning on RGB dataset of dogs, cats, cars, etc. can be applied to medical grayscale images converted into RGB format. To examine this, all grayscale images were saved as RGB jpg files. Further, since YOLO v2 was chosen for object detection, all images were downscaled to 416x416, the input size of YOLO v2, after padding was added. Lastly, image files were separated into folders matching their evaluation so that a classifier could utilize the dataset as well.

## 6.1 Annotating Boundary Boxes

In our dataset, we have dogs of various breeds and sizes. Therefore, the appearance of the hip varies quite a bit. Normally this is a challenge in object detection, but with CNNs it is feasible as long as sufficient data representing this variance is collected the model will learn to identify a specific object regardless of the variance and the placement inside the image. This is due to the fact that a CNN applies numerous filters across the image input that detect different features. So in order to build a hip detector with a supervised CNN, a lot of training data is required. With object detection, as explained in earlier chapters, the image file and related boundary boxes of the objects inside the image is required. As object detection is a technique with a wide range of applications, scripts to outsource the annotation work-load on Amazon Mechanical Turk are publicly available. Further, open sourced programs can be downloaded to have a graphical UI for annotating. One of the popular solutions available on Github, named Labellmg, was initially tested because it saves the annotations of the objects in xml PASCAL VOC format and has a user-friendly interface. In short, PASCAL VOC is a xml standard with a specific order in which the image path, object label and location are stored. Labellmg works by drawing labelled rectangles on top of images and saving corresponding xml files with the objects. However, the program was unstable on my computer and it was also time-consuming to draw the squares. So, instead of Labellmg, a custom python script was developed that allowed to open images from a directory and extract the hip location from just two mouse clicks. The program simply requires the user to click on the center of the two hip bones from which the coordinates of the centers are saved 50. By utilizing the fact that the diameter of each hip can be calculated from the distance between the centers, given a constant ratio of 2.5, the location of each hip was calculated (McEvoy pers. comm.):

$$Diameter = \frac{\text{left center x} - \text{right center x}}{2.5}$$

[2]The reason why the length is (512x512x3) is because it is assumed that the image is saved in RGB format and containing the three colour channels

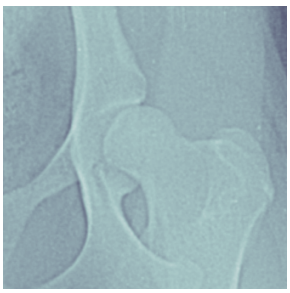


*Figure 50: An example of the hip coordinates after clicking on the center of the femoral head.*

Based on the diameter calculations, the coordinates of the hips were calculated and stored in a csv file. A total of 2,264 hips were annotated in this process. Since miss-clicks would happen during the process (for instance a double click), the script was amended in that the annotations would only be stored if the coordinates of the two hips varied sufficiently. So by clicking close to the same coordinates twice would skip to the next image without saving the annotation information. This feature was also useful when duplicate images from time to time occurred (due to multiple submission by veterinarians) by avoiding getting additional samples of the same image. Finally, all the annotation information was saved inside a csv file. In figure 51, a sanity test was made by drawing the boundary boxes on top of the original image to ensure that the coordinates saved were correct.

*Figure 51: Checking that the boundary boxes are saved correctly in the csv file*

In the annotation process, a variance in exposure and contrast were noticed in the hip extractions. This could potentially be a challenge for the object detector and classifier if the variance is not equally distributed across the different classes. The examples in figure 52 were chosen to display the differences in image quality among the hip extractions. Figure 52 is one of the few examples in which the femoral head is not in the center of the image.

*Figure 52: Illustrating differences between x-rays*

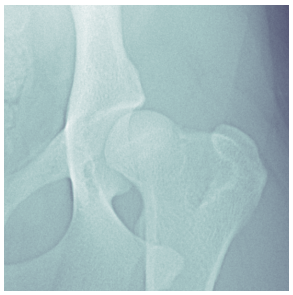


*Figure 53*

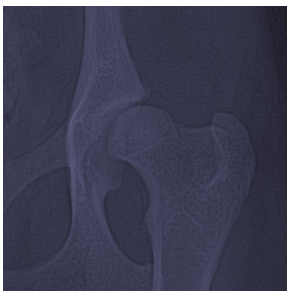


*Figure 54*

*Figure 55*

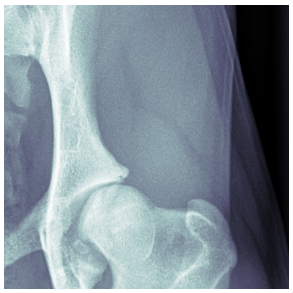


*Figure 56*

*Figure 57*

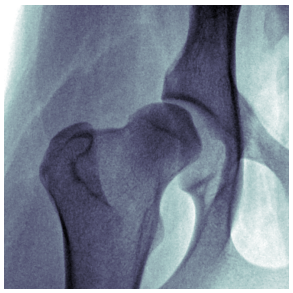


*Figure 58*

# 7 Object Detection with YOLO

With hip annotations saved in csv format, a script was created to convert the data into xml files in Pascal VOC format for implementation of the YOLO algorithm. Instead of implementing YOLO from scratch, open source models were found on Github in different languages (Keras, Tensorflow, etc.). As mentioned, there have been two YOLO releases and since YOLO v2 performs better, this algorithm was prioritized over the original YOLO when searching for models. After testing out different repositories, a library of YOLO v2 in Keras was chosen by Huynh Ngoc Ahn [30]. Keras is a high-level language that operates with both Tensorflow and Theano. While not as flexible as Tensorflow, it is a faster tool for building prototypes as less code is generally required. The YOLO v2 repository is composed of the following scripts:

*Table 4: YOLO v2 Repository Description*

| **Script Title** | **Description** |
|---|---|
| backend.py | Architecture of YOLO to initialize all the layers |
| frontend.py | Initialize YOLO with or without weights, define loss, training and prediction function |
| gen_anchors.py | K-Means algorithm to generate boundary box suggestions for the beginning of training |
| predict.py | Script to build the YOLO model and run prediction on an image or directory |
| preprocessing.py | Prepare and augment data on a given dataset |
| utils.py | Support functions for reading labels, bounding boxes, weights, drawing boxes and calculating the overlap between ground truths and predictions. |
| config.json | Define the parameters of the YOLO model |

The step-by-step use of the code is the following (utils is used for most of the steps):

1. Calculate the anchors used for the boundary box prediction by analysing the annotation data (gen_anchors.py)
2. Define configurations of images size, batches, etc. (config.py)
3. Prepare the dataset including images and annotations based on the configurations in training and validation sets (preprocessing.py)
4. Construct the YOLO network (backend.py)
5. Load the pretrained YOLO weights into the network (frontend.py)
6. Run training (train.py)
7. Test model on unseen data (predict.py)

## 7.1 Challenges with development environment

To run the YOLO model there are a few requirements described by the author. One of them is to use python version 2.7 together with some image packages (CV2 and Imgaug). On my own computer, the script ran without any major obstacles. That said, the training would take several days at the recommended number of epochs and therefore GPUs were required. However, the environment setup ran into some challenges with external GPUs. Since ITU had issues with getting their GPU setup ready, I reached out to Barcelona Supercomputer Center (BSC) in October 2017 to potentially get access to their system. After three meetings, they agreed

to help with access to one of their GPU clusters called MinoTauro, consisting of 8 Tesla K80 GPUs. Although, the resources at BSC were adequate, I was not aware that their GPU clusters were not connected to the internet and only the support team was able to make edits to the installed packages and environments. Given their setup with Python 2.7 at BSC, Keras 1.1 was the only version available - not sufficing for the script which according to the author requires keras 2.0.8. Further, the image augmentation library imgaug and CV2 were not installed either. On the other hand, a more up to date library of keras 2.0.6 was available with python 3. Therefore, I began to convert the code to python 3 and importing the imgaug library. Unfortunately, issues would keep occurring and after a week of testing alternatives ie. other YOLO implementations and as well as augmentations to the code, I decided to set up a GPU cluster with AWS instead. AWS offers a service called AML (short for Amazon Machine Learning) that comes with a package of libraries for machine learning pre-installed. One challenge with the configuration of the environment was getting CV2 also known as OpenCV installed. OpenCV is an image library based on C++, but can be installed with python given the right python wrappers. It has been downloaded more than 14 million times [31], and hence is one of the most popular image libraries. However, no matter the configurations (python version, Ubuntu, Linux, etc.) on the AML environment, the library was missing essential paths at run-time to execute. The solution was to side-step AML and create a new environment on the server from scratch in python 3. Therefore, the code of the original library was amended as well in order to accommodate the change. After contacting AWS, they have now made updates to the environment so OpenCV works again on their AML environments for python 3.

## 7.2 Training with YOLO

Rather than training YOLO on the canine x-rays to begin with, a Jupyter notebook was used to get an intuition of the model as well as ensure the model worked on simple use-cases with the pre-trained weights. This allowed me to get familiar with the code and solve occasional bugs. From the YOLO weights provided by the original author [24], a detector for giraffes and zebras was pre-trained and hence it was tested if the Jupyter Notebook model was able to detect them (see figure 59).

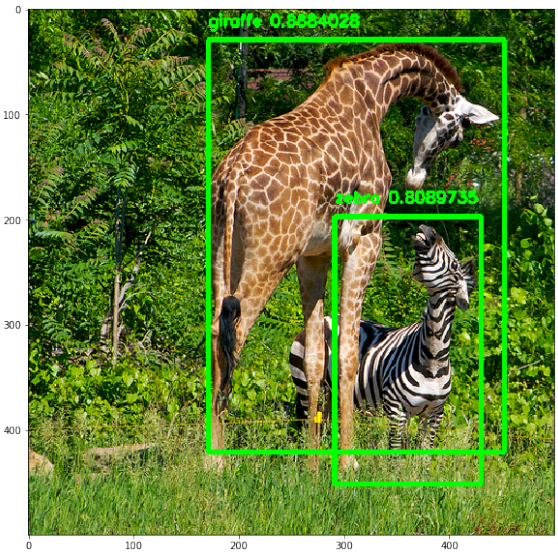



*Figure 59: Testing the pre-trained YOLO Weights and classes it has already learned to identify. Confidence scores are listed above the predicted boundary boxes. Image from [30]*

After the test of the weights in Jupyter notebook, training with transfer learning was tested with a command-line interface. To do so a dataset consisting of 200 labelled raccoon images was used with annotations saved in xml Pascal VOC format. At that time, March 2018, only CPU resources were available and hence testing was very time-consuming. The training was divided into two phases (warm-up training and final training). At the warm-up phase the five detected boundary boxes in each cell are forced to match the sizes of the 5 suggested anchors. According to the author, this trick improves precision empirically [30]. Before the actual warmup training, the pre-trained dataset is loaded and the output vector is updated to only include one class (raccoon). Then after 3 epochs of warm-up training, the weights are then saved. The second training phase includes up to 50 epochs and is initialized with the saved warmup weights. During training, early stopping is set to 3, meaning that if the validation accuracy does not improve after three consecutive training epochs, the training is terminated and the best weights based on the validation accuracy is saved. After several experiments, and a GPU setup with AWS, a reliable raccoon detector was developed. Given these results with transfer learning, the next step was to generate a hip detector. Below are the details of the configuration file prior to training of the hip detector:

```
{
"model" : {
"architecture":         "Full Yolo",
"input_size":           416,
"anchors":              [2.55,2.07, 3.10,2.51, 3.58,2.86,
4.22,3.19, 5.16,4.09],
"max_box_per_image":    6,
"labels":               ["right-hip", "left-hip"]
},

"train": {
"train_image_folder": [path to image folder],
```

```
"train_annot_folder": [path to annotation folder],

"train_times":          10,
"pretrained_weights":   "",
"batch_size":           16,
"learning_rate":        1e-4,
"nb_epoch":             10,
"warmup_epochs":        0,

"object_scale":         5.0 ,
"no_object_scale":      1.0,
"coord_scale":          1.0,
"class_scale":          1.0,

"saved_weights_name":   "full_yolo_hip.h5",
"debug":                true
},

"valid": {
"valid_image_folder":   "[path to image folder]",
"valid_annot_folder":   "[path to annotation folder]",

"valid_times":          1
}
}
```

As previously mentioned, YOLO v2 is able to detect up to five objects in each of the 32 cells that the image is split into. However, the configuration file allows to specify a maximum of detected objects. This parameter does not change the limit of the YOLO boxes per cell. Instead it selects only the predictions of the highest confidence, which are then used to calculate the total loss. In our use-case, we are only interested in detecting up to two objects per image (the left and the right hip). That said, the model could still detect two left-hips. Therefore, the maximum of objects per image was set to 6. Because no fine-tuning of the YOLO algorithm was completed, a test dataset was not required. Finally, anchors were calculated based on the gen_anchor.py file.

### Image Augmentation

As explained earlier, image augmentation is a tool that can be useful when training object detectors or classifiers with sparse data. The intuition is that small datasets can in some cases lead to over-fitting, as the model learns to detect unwanted correlations. An example of this could be an x-ray image of a B hip with the tag 'Left' in the top right corner. If the model does not train on many different B hips it could learn that the tag 'Left' in the top right corner means a B hip, which is certainly not the correlation that we are looking for. Another example, which is very relevant for this data-set, is if the training set of a certain category happens to be darker than the other categories. In that case, the model might learn to associate the brightness with a certain result. To overcome some of these challenges, it is therefore useful to

slightly alter the images during training, so that the data in every epoch is not the same. Here are some examples of techniques that can be used:

- rotate
- noise
- blurring
- sharpening
- contrast normalization

While no academic papers on the impact of image augmentation on the accuracy for object detection, a few papers examining the effects on classification accuracy were examined. The literature described that the tuning image augmentation depends on the underlying dataset and problem domain. For instance if the dataset contains images including a lot of noise, then only a small amount of noise can be applied without rendering the image useless. The same considerations for classifiers, should also be made to object detectors. In the papers by Wang [32] and Lemley [33], the authors examine how a Generative Adversarial Network can be trained to learn the correct measure of augmentation. Although, an interesting approach, the implementation of a GAN to support the augmentation is outside the scope of the thesis and hence visual intuition was used on the tuning instead. To apply augmentation during YOLO-training a library for image augmentation called imgaug was used. The script was configured to randomly apply the following augmentations to 50 percent of the data inside the batch:

- different kinds of blur
- gaussian noise
- change brightness of images
- change hue and saturation
- improve or worsen the contrast
- sharpen images

Although, the Keras library has a built-in image augmenter, the augmentation options are limited and mainly include functions for rotations, flipping, etc, which for obvious reasons are not useful when dealing with object detection of boundary boxes. To test the augmentation of an image, the same image was applied the augmenter function six times and compared in figure 60.

*Figure 60: Image of a hip and five versions of the hip after being applied the augmentation function. As mentioned, augmentation is only applied 50 percent of the time.*

*Figure 61: Original hip*

*Figure 62: Experiment 1*

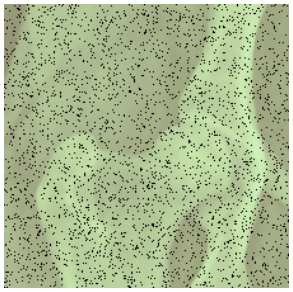


*Figure 63: Experiment 2*

*Figure 64: Experiment 3*

*Figure 65: Experiment 4*

*Figure 66: Experiment 5*

As seen from the augmentations in figure 60, the original image is randomly applied noise, blurring and a change of exposure. The experiment is also a sanity check to ensure that the important visual features (the hip bone and socket) are clearly visible on all of the examples. A visual test of the extreme values of the augmentations was conducted as well.

## 7.3 YOLO Results

The supervised annotation dataset, consisting of 2,264 left and right hips, was split into 80 percent training data and 20 validation data. As mentioned, no test was generated as no hyperparameter-tuning was used. Instead, the approach was to simply follow the YOLO v2 implementation. Based on the configurations explained, the model trained for 14 epochs before early stopping. The results were at this point a mAP (mean average precision) of 1.00 on the validation set, indicating a high precision when it comes to classification and an IoU above 0.5 for every prediction. This shows that the model has a high lower-bound prediction of the localization for all the examples in the validation set. Also, it proves that transfer learning on object detectors trained on RGB images can be efficient on medical grayscale x-rays that have been converted to RGB images. By using Tensorboard the following graph was generated to illustrate the validation error as a function of the number of the epochs (see figure 67).

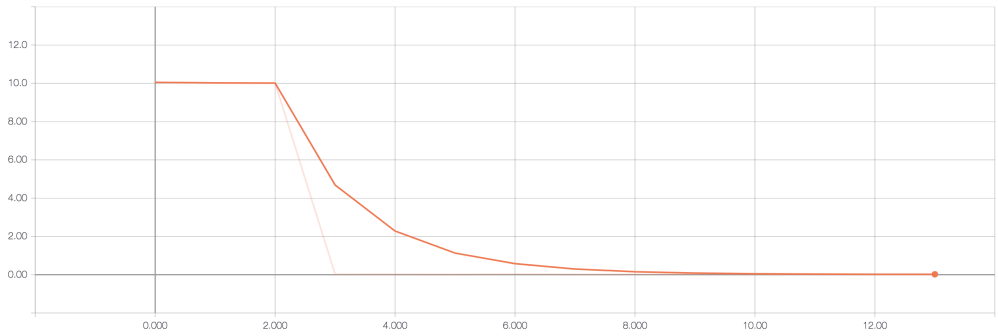



*Figure 67: Graph of the YOLO validation loss as a function of epochs.*

One explanation for the high precision obtained is the fact that all the x-ray images are taken at the same angle, whereas in many object detection datasets the positioning varies greatly. Also, the model is only required to detect two classes, which are distinctively different (horizontal opposites). Lastly, it should be noted that the work we conducted in the annotation process paid off.

*Figure 68: Comparing a hip extraction on the validation set from YOLO with the ground truth. The example shows a very positive result.*

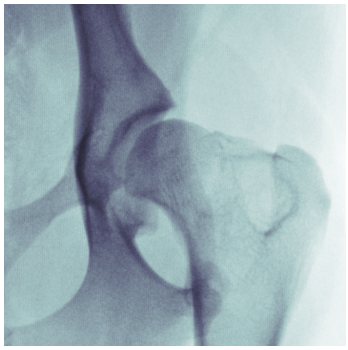


*Figure 69: Ground truth hip extraction*

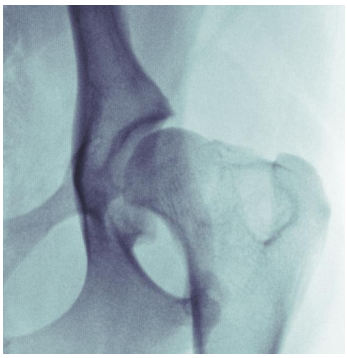


*Figure 70: Predicted hip extraction*

Given a maximum mAP score of 1.00, the more interesting measure of the accuracy of the model is to examine the IoU of the predicted boundary boxes. By analysing the accuracy of the training and validation set, it is possible to determine how well the model generalizes. To examine this, changes were made to the predict.py code to analyse an entire directory and save the predicted boundary boxes in a separate script. Further, a script was written to calculate IoU based on the csv predictions. The changes to the predict.py script recognises if the argument given is a path for an image or a directory. Therefore, no changes to the original command are required to initiate the prediction:

```
python predict.py -c config.json -w [path to weight file]
-i [path to image file or to directory]
```

Further, since we know that every picture contains a left and a right hip, the prediction script was updated so that only one detection of the left and right hip were saved in the csv file. The selection criteria being the detection with highest predicted confidence for each class.

*Table 5: Average IoU for training and validation dataset. For the YOLO v2 model no test set was required as no tuning of hyperparameters was conducted*

| Dataset | Average IoU | mAP |
|---|---|---|
| Training | 0.9184 | 1.00 |
| Validation | 0.9286 | 1.00 |

The average IoU was computed for both the training and validation dataset, in which the validation set consisted of 464 hip annotations. As seen in table 5, the average IoU is over 90 percent for both datasets. In fact, the IoU score is almost the same showing strong generalization of the model. The IoU results tell us that the predicted boundary boxes on average cover more than 90 percent of the ground truth boundary box. This is excellent results. Outliers from the validation set were analysed individually to detect potential patterns (see figure 71). Interestingly, the worst predictions by the models are all on the right hip. Also, it seems that the predictions (in blue) are all a bit larger than the ground truths (in green). However, even the worst predictions are still quite accurate (above 50 percent), which is a good sign for the object detector.

*Figure 71: IoU outlier prediction from validation set. Green boxes are the ground truths and blues boxes are the predicted boundary boxes.*

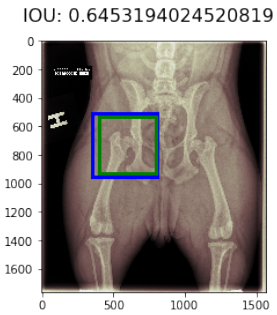



*Figure 72*

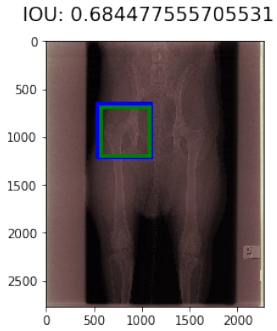



*Figure 73*

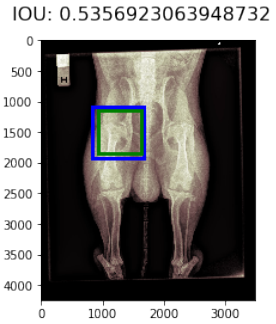



*Figure 74*

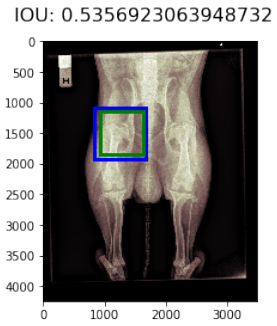



*Figure 75*

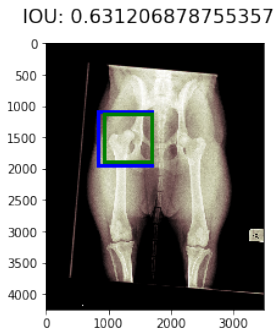



*Figure 76*

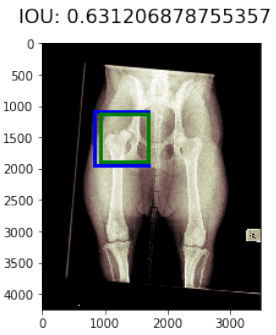



*Figure 77*

Based on the boundary box predictions, the resolution sizes of the hip extractions were plotted in figure 78 to identify the variance. Most of the observations are in

the range of 400x400 pixels and 1000x1000 pixels. This means that almost all of the images will be re-sized as the pre-trained ResNet model will use a resolution size of 224x224 pixels.

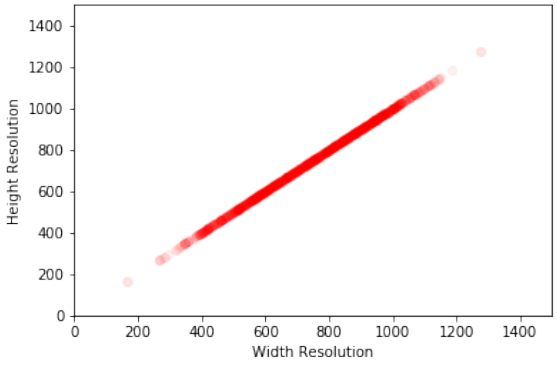



*Figure 78: The resolution sizes of the YOLO v2 hip extraction. All the outputs are of square format.*

# 8 Classification with ResNet

Based on the extracted data from the YOLO algorithm, the next step is to test if indeed a classifier is able to predict the hip score based on hip extractions. To do so, we will use pre-trained weights on Imagenet with the ResNet-152 architecture, as previously explained in the CNN theory chapter. Similar to the implementation approach of YOLO, an open-sourced ResNet repository was applied. In this case, the repository was developed by Felix Yu [34]. Similar to the YOLO repository, the code was implemented so that the training only is performed on the last fully connected layer also known as fine-tuning.

*Table 6: ResNet Repository Description*

| **Script Title** | **Description** |
|---|---|
| johan_resnet_152.py | ResNet architecture, training and predictions |
| load_hip.py | dataset generator for balanced and unbalanced training sets |
| scale_layer.py | function for learning the sets of weights and biases |

## 8.1 Experiment Details

The goals of the classification experiments are to answer the following:

- Can transfer learning on RGB images be successfully applied to HD classification on grayscale images converted to RGB.
- Does feature extraction with YOLO improve accuracy of classification as opposed to using the entire x-ray as input for the classifier.

- Are there certain hip scores that the model has difficulties distinguishing between.
- Is it possible to partially overcome over-fitting, given the limited data size, by adjusting the hyper-parameters.
- Does an unbalanced dataset with adjusted class weights perform better than a balanced dataset.

To examine this, three separate datasets were generated. Since the dataset of hip extractions were generally quite unbalanced, additional instances were created in which the left hips of score 'D' and 'E', were flipped horizontally to generate a more balanced dataset. The datasets are as follows:

1. Full x-ray images of 416x416 resolution divided into folders according to the hip final hip score. See class distribution below:
    - A: 7989 instances
    - B: 2002 instances
    - C: 1350 instances
    - D: 873 instances
    - E: 333 instances
2. Right hip extractions (with some flipped left hips for class D and E) of various resolutions divided into folders corresponding to the grade of the hip. See class distribution below:
    - A: 2988 instances
    - B: 1766 instances
    - C: 1026 instances
    - D: 1132 instances
    - E: 492 instances
3. Left and right hip extractions of various resolutions divided into two separate folders (A, B, C) and (D, E) to generate a binary dataset of "approved" and "not approved" evaluations. See class distribution below:
    - Approved (A, B, C): 3955 instances
    - Not approved (D, E): 1622 instances

For the experiments on both five class and binary classification, a script, hip_load.py, was written to load balanced and unbalanced datasets, and ensuring that the test sets for classification of similar experiments were the same. When conducting the experiments the test set was balanced, so that every class was equally represented. Given that I had to pay for resources on AWS, only a limited amount of experiments were conducted. Below is a list of the total experiments with ResNet-152 on 5-class classification, binary individual hip classification and 5-class full hip classification:

- 5-class individual hip classification
  - Experiment 1: Baseline results trained on balanced dataset
  - Experiment 2: Adjusted learning rate
  - Experiment 3: Adjusted batch size
  - Experiment 4: Training on unbalanced dataset
- Full hip classification
  - Experiment 5: Training on balanced dataset

After training models to classify the 5 hip scores, ranging from A to E, the dataset was transformed to solely distinguish between sets of [A, B, C] and [D, E]. As a simplification, the two sets were denoted "approved" and "not approved", although the actual threshold depends on the canine breed.

- Binary individual hip classification
  - Experiment 6: Training on unbalanced dataset

For all the experiments conducted with a balanced dataset, the least represented class set the baseline for the number of instances per class. All of the experiments are a split into 80 percent training data, 10 percent validation data and 10 percent test data.

### Experiment 1: Baseline ResNet Model

*Table 7: Experiment 1: Hyper parameters*

| Learning rate | Epochs | Batch size | Optimizer | Class weights |
|---|---|---|---|---|
| 0.001 | 8 | 20 | SGD | none |

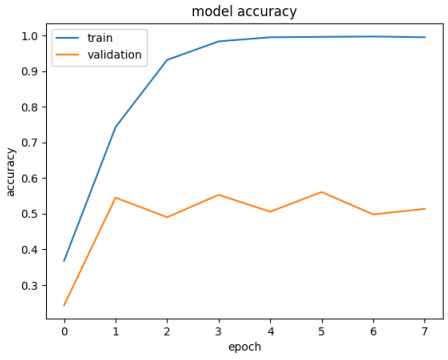



*Figure 79: Experiment 1: Model accuracy as a function of the number of epochs*

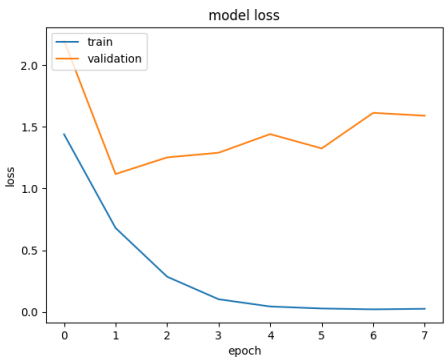



*Figure 80: Experiment 1: Model loss as a function of the number of epochs*

As seen in figure 79, the model is over-fitting as accuracy on training data improves continuously, and peaks around 93 percent, while the accuracy of the validation hovers around 53 percent after the 2nd epoch. In the classification report (table 8), the average accuracy on the test data is 55 percent, which compared with the baseline accuracy of 20 percent, shows that the model is able to predict better than random guessing, yet still has difficulties with distinguishing between certain classes. One reason for this could be that learned features from the trained weights on ImageNet does not generalize themselves well to distinguishing between the classes. After all, in our experiments, we only perform fine-tuning on the last fully connected layer. Another explanation is that we have not defined the hyper-parameters optimally. From the classification report in table 8, we observe that the recall for class C is an outlier at 27 percent. This is a clue to which classes that the model is not able to define well.

*Table 8: Experiment 1 classification report of the test data*

| Class | Precision | Recall | F1-score | Support |
|---|---|---|---|---|
| A | 0.60 | 0.55 | 0.57 | 51 |
| B | 0.47 | 0.41 | 0.44 | 51 |
| C | 0.45 | 0.27 | 0.34 | 51 |
| D | 0.44 | 0.76 | 0.56 | 51 |
| E | 0.79 | 0.67 | 0.72 | 51 |
| avg / total | 0.55 | 0.53 | 0.53 | 255 |

In order to better identify which classes the model is not able to generalize well, we implemented a confusion matrix based on the results on the test dataset (see figure 81). In the confusion matrix, it is possible to see where the miss-classifications occur. Ideally, a confusion matrix, with color-intensity reflecting the number of predictions, shows a clear diagonal line from the top-left corner to the bottom-right corner.

In the classification report in table 8, we notice that the model is relatively stronger at predicting E, D and A hips compared to predicting B and C hips. However, we do not know which classes the model confuses the predictions with. In the confusion matrix, however, we can see that the model is more likely to confuse a C hip with a D hip. Given the dataset size, it is possible that the model has not learned to detect the slight nuances required for those classes. One positive signal of the confusion matrix is that when the model predicts wrong it is more likely to guess neighbouring classes as opposed to more distant classes (see confusion matrix 81). For instance, the model does not predict any As as Es or vice versa.

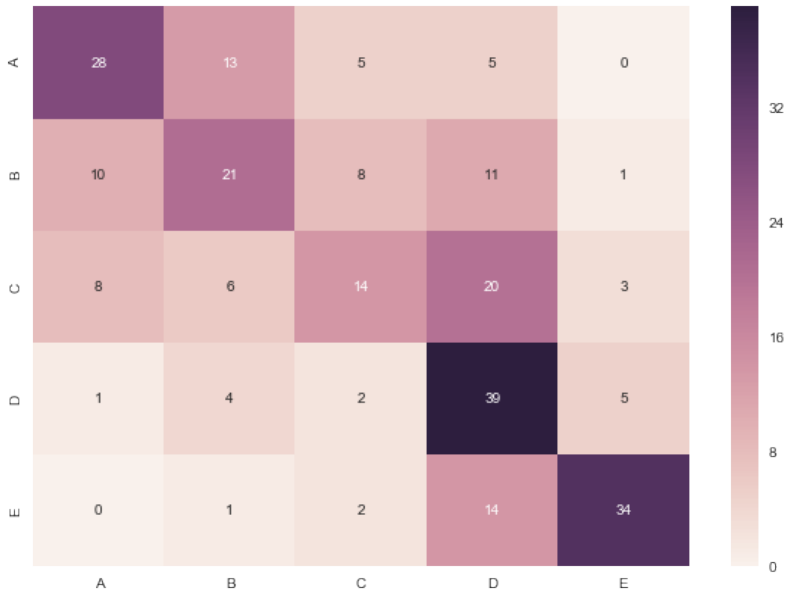



*Figure 81: Experiment 1: Baseline model - Confusion Matrix from the test data*

To limit the amount of over-fitting, the next experiment was conducted with a learning rate of one order of magnitude lower. The reason why a smaller a learning rate can decrease over-fitting is due to the fact that the weights are updated by a smaller factor to fit each batch, which can avoid local minima.

### Experiment 2: Adjusted learning rate

*Table 9: Experiment 2: Training parameters*

| **Learning rate** | Epochs | Batch size | Optimizer | Class weights |
|---|---|---|---|---|
| **0.0001** | 8 | 20 | SGD | none |

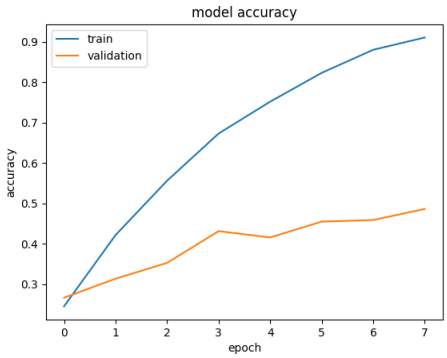



*Figure 82: Experiment 2: Adjusted learning rate - Model accuracy as a function of the number of epochs*

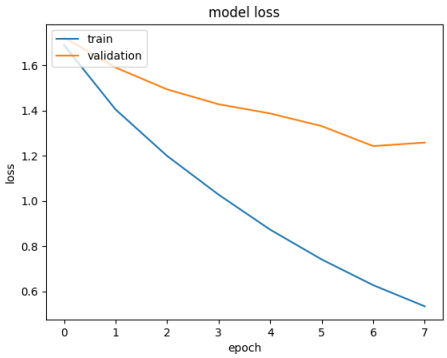



*Figure 83: Experiment 2: Adjusted learning rate - Model loss as a function of the number of epochs*

With the adjusted learning rate, the training, as seen in figure 82 is smoother and the accuracy improvements are slower, as expected, when compared to the previous

experiment in figure 79. It is worth noticing that the curve for both training and validation accuracy is upward trending until the last epoch, signalling that the model potentially could learn more from further training on more epochs. As seen in the classification report 10, the results of experiment 2 are worse across all average statistics when compared to experiment 1.

*Table 10: Experiment 2: Adjusted learning rate - Classification report*

| Class | Precision | Recall | F1-score | Support |
|---|---|---|---|---|
| A | 0.45 | 0.39 | 0.42 | 51 |
| B | 0.24 | 0.24 | 0.37 | 51 |
| C | 0.32 | 0.43 | 0.37 | 51 |
| D | 0.50 | 0.43 | 0.46 | 51 |
| E | 0.67 | 0.65 | 0.66 | 51 |
| avg / total | 0.44 | 0.43 | 0.43 | 255 |

The worse results are also evident in the confusion matrix (see figure 84). Here, we see that the model is now sometimes predicting an A as E and vice versa.

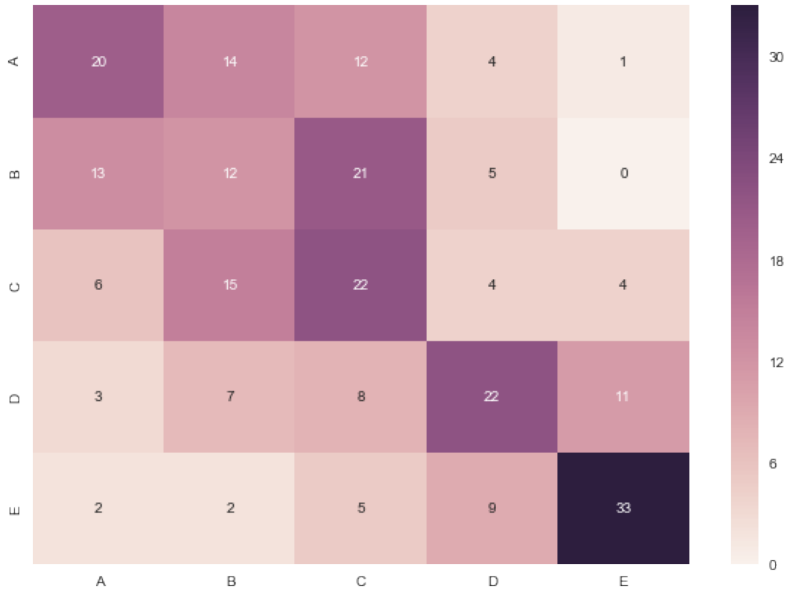



*Figure 84: Experiment 2: Adjusted learning rate - Confusion Matrix from the test data*

Based on these results, even though training perhaps should have been set up with more epochs, the adjusted learning rate is not used in the following experiments. Instead, as another strategy to avoid over-fitting, the batch size was examined. In the paper, 'On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima' [35], the authors argue that large batch size can fail to generalize, while small batch sizes can be too noisy. In the paper, the general batch size is considered to be between 32 to 512, from which the authors suggest a larger batch size. However, in the paper 'Revisiting Small Batch Training for Deep Neural Networks' [36], the authors argue that on datasets such as CIFAR-10 best results are achieved with batch sizes between 2 and 32. Given the inconclusiveness of the papers, a smaller batch size was tested to see if better results could be obtained this

way. Similar to the adjusted learning rate, a smaller batch size would decrease the total loss computed from every batch.

### Experiment 3: Adjusting batch size

*Table 11: Experiment 3: Training parameters*

| Learning rate | Epochs | Batch size | Optimizer | Class weights |
|---|---|---|---|---|
| 0.001 | 8 | **10** | SGD | none |

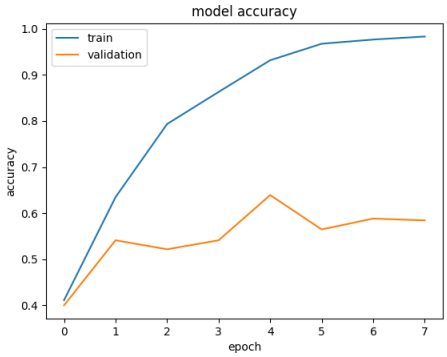



*Figure 85: Experiment 3: Adjusted batch size - Model accuracy as a function of the number of epochs.*

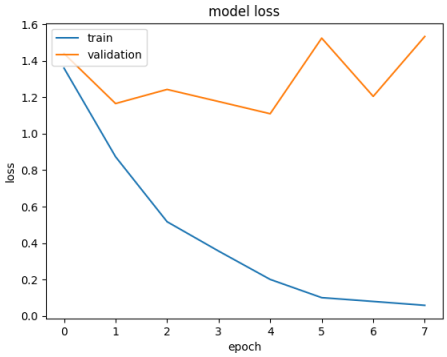



*Figure 86: Experiment 3: Adjusted batch size - Model loss as a function of the number of epochs.*

In figure 85, training accuracy plateaus at 8 epochs, while validation accuracy is a bit unstable and falls from the peak at epoch 4, slightly above 63 percent, to just below 60 percent at epoch 8. One way of understanding the development, after epoch 4, is that the model keeps optimising for features that improve the accuracy of the training samples, but does not generalize well to the domain of HD. The results from the test set are slightly better than the best results from the validation data (see table 12). The average precision rate is at 70 percent and the F1 score is at 0.68, which is a 15 percent increase in both statistics from the baseline experiment, which was the previous best results. One clear outlier from the classification report is the recall value for class B. A recall of 22 percent tells us that the model does not predict B for a lot of the actual Bs in the test data.

*Table 12: Experiment 3: Adjusted batch size - classification report on test data*

| Class | Precision | Recall | F1-score | Support |
|---|---|---|---|---|
| A | 0.50 | 0.86 | 0.63 | 51 |
| B | 0.69 | 0.22 | 0.33 | 51 |
| C | 0.63 | 0.61 | 0.62 | 51 |
| D | 0.80 | 0.76 | 0.78 | 51 |
| E | 0.87 | 0.90 | 0.88 | 51 |
| avg / total | 0.70 | 0.67 | 0.65 | 255 |

The results from the confusion matrix, in figure 87, also show strong classification performance for every class except B just as the classification report in table 12 revealed. In fact, the model is more than twice as likely to predict a B as A.

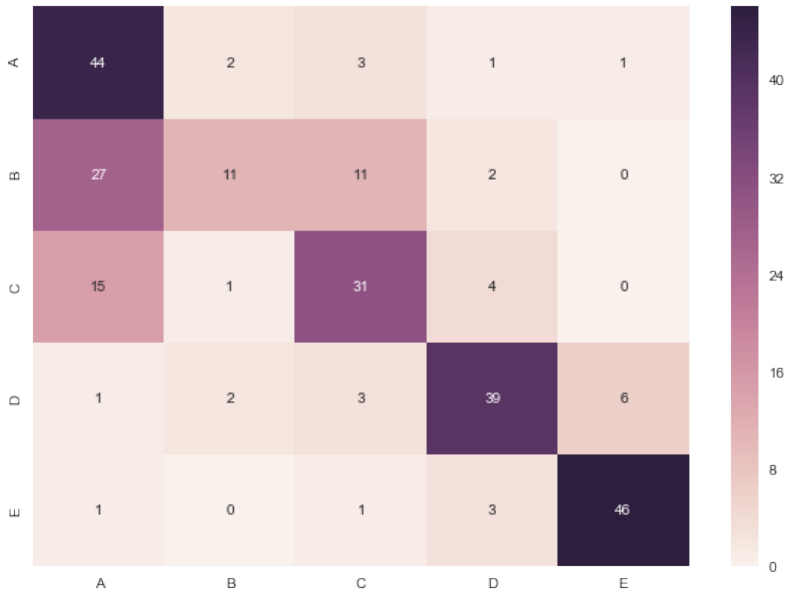



*Figure 87: Experiment 3: Adjusted batch size - Confusion Matrix from the test data*

Overall, 70 percent average precision shows promising results for a five-class classification. Especially, given that only 492 instances were used per class and divided into train, validation and test data. In the following experiments, we will test if better results can be obtained by training on a larger unbalanced dataset, but adjusting the weights of the classes for the loss function.

### Experiment 4: Unbalanced dataset

*Table 13: Experiment 4: Training parameters*

| Learning rate | Epochs | Batch size | Optimizer | Class weights |
|---|---|---|---|---|
| 0.001 | 8 | 10 | SGD | **yes** |

For this experiment, rather than balancing the number of instances across all classes, all instances are used in training. The size of the validation and test are the same as in previous experiments. However, to avoid the model from simply guessing the over-represented classes during training, the classes are weighted in the loss function in relation to their representation in the entire dataset.

```
class_weight = {0:1.0, 1:1.7, 2:2.9, 3:2.6, 4:6}
```

Class A, which equates to 0 in the dictionary, is over-represented and hence when the class is not predicted correctly the loss is computed with a factor of 1. However, for the other classes that are under-represented in relation to class A, the loss is multiplied by the factor based on the inverse ratio to class A. Using the class weights and unbalanced training set, the following training graphs were generated

(see figure 88 and 89). The results show an accuracy of 70 percent on the validation set, which is the highest so far. Also, we notice that the validation loss stabilizes after the 4th epoch.

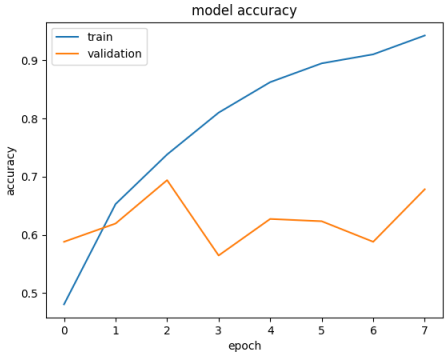



*Figure 88: Experiment 4: Unbalanced dataset - Model accuracy as a function of the number of epochs.*

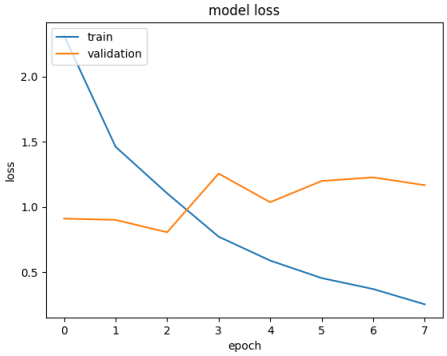



*Figure 89: Experiment 4: Unbalanced dataset - Model loss as a function of the number of epochs.*

When analysing the test data results in the classification report 14, we notice that the average results are slightly better than in experiment 3, which was the previous best results. More noticeably, the statistics of the test set in the classification report with the unbalanced dataset (see table 14) does not include as many outliers. For instance, the lowest recall is at 57 percent with the unbalanced dataset, compared to 22 percent with the balanced dataset.

*Table 14: Experiment 4: Unbalanced data - classification report on test data*

| Class | Precision | Recall | f1-score | Support |
|---|---|---|---|---|
| A | 0.77 | 0.65 | 0.70 | 51 |
| B | 0.51 | 0.67 | 0.58 | 51 |
| C | 0.69 | 0.57 | 0.62 | 51 |
| D | 0.80 | 0.73 | 0.76 | 51 |
| E | 0.84 | 0.94 | 0.89 | 51 |
| avg / total | 0.72 | 0.71 | 0.71 | 255 |

The balanced predictive power of the unbalanced dataset is also evident in the confusion matrix in figure 90, which has a clear diagonal line and very few miss predictions with far-off neighbours. It is interesting to note that the two most accurate classes are D and E, in which E is significantly more accurate. On the other hand, the model has difficulties with separating between class B and C (see figure 90) which was discussed with Hald Nielsen, who was not surprised by those results. During the conversation, Hald Nielsen mentioned that she would find class B and C more challenging to evaluate and, hence, in some cases would ask for a second opinion. Given less well-defined evaluation rules for class B and C, it is understandable that the model has a lower predictive power as the supervised dataset might not reflect clear patterns. Although, the weights have been balanced in respect to the class representation, further fine-tuning could be tested by lowering the weight for class E and to examine the results.

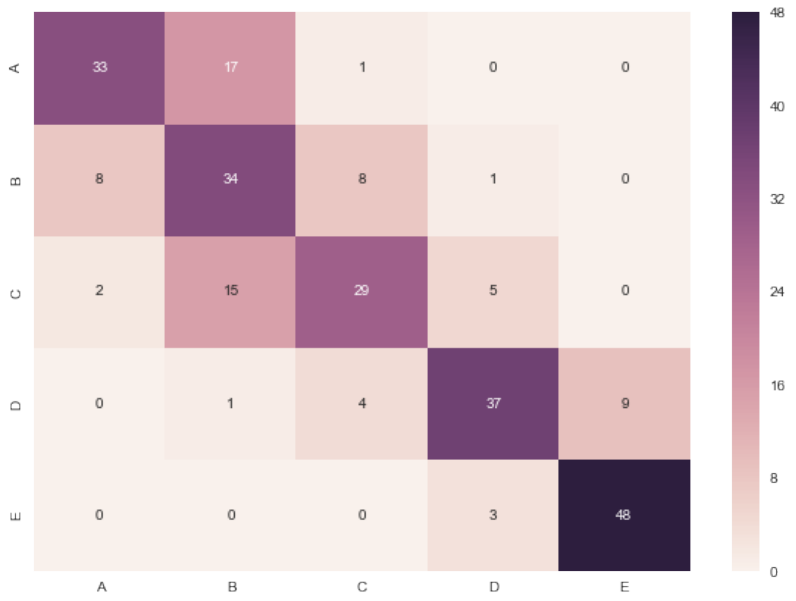



*Figure 90: Experiment 4: Unbalanced dataset - Confusion Matrix from the test data*

In order to determine if the effort of building an object detector for hip classification, and thereby reducing the amount of noise in the input is worthwhile, the next experiment will determine if a model is able to classify the final hip score (the worst of the two hips), based on the full x-ray image.

**Experiment 5: Classification of full x-ray image**

*Table 15: Experiment 5: Training parameters*

| Learning rate | Epochs | Batch size | Optimizer | Class weights |
|---|---|---|---|---|
| 0.001 | 8 | 10 | SGD | none |

From the graphs in figure 91, it can be concluded that training the neural network with the full x-ray image indeed is not very promising. In fact, the validation accuracy ends at around 30 percent, which only is slightly better than random guessing and the worst result so far. Because we use the entire x-ray images it is not possible to generate extra data by flipping the individual hips. Therefore, the dataset is smaller than in the previous experiments, which also could impact performance.

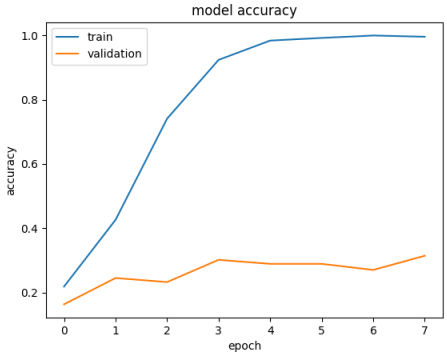



*Figure 91: Experiment 5: Full size x-ray - Model accuracy as a function of the number of epochs.*

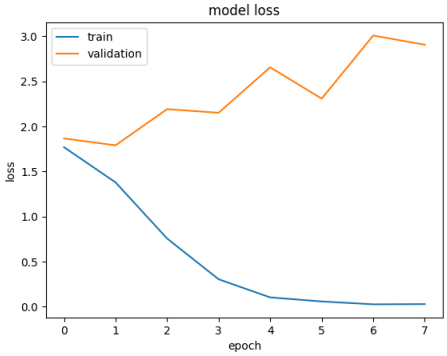



*Figure 92: Experiment 5: Full size x-ray - Model loss as a function of the number of epochs.*

The results of the test set in the classification report (see table 16) are significantly above the validation results. Given this variance and the fact that the validation and test set are quite small, it is more likely the true accuracy lies in between the results of the two datasets.

*Table 16: Experiment 5: Full size x-ray - classification report on test data*

| Class | Precession | Recall | f1-score | Support |
|---|---|---|---|---|
| A | 0.40 | 0.59 | 0.48 | 34 |
| B | 0.53 | 0.29 | 0.38 | 34 |
| C | 0.30 | 0.26 | 0.28 | 34 |
| D | 0.54 | 0.44 | 0.48 | 34 |
| E | 0.51 | 0.65 | 0.47 | 34 |
| avg / total | 0.45 | 0.45 | 0.44 | 170 |

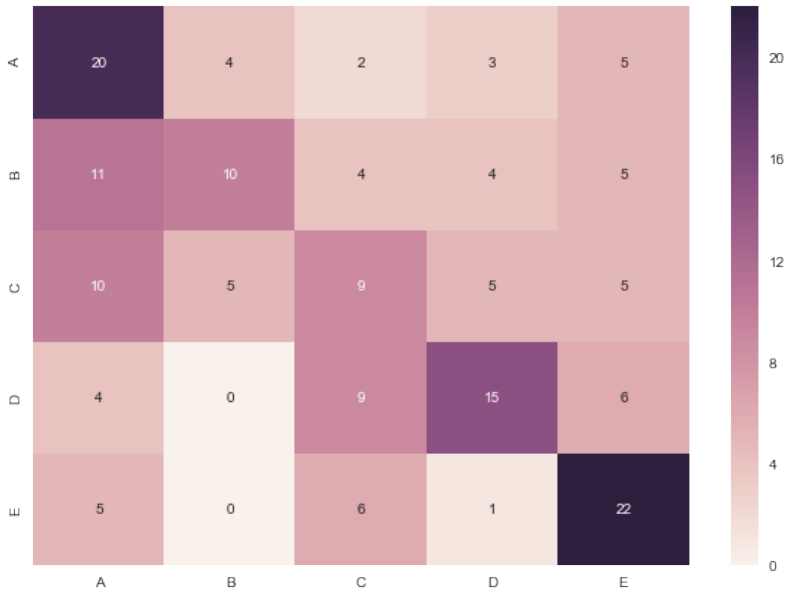



*Figure 93: Experiment 5: Full size x-rays - Confusion Matrix from the test data*

As seen in figure 93, the model is not as good at separating classes that otherwise are from the opposite sides of the spectrum. Both the confusion matrix in figure 93 and classification report in table 16 supports the argument that using the full x-ray images, given the dataset size, will not yield strong results. As we have seen so far the models have in general had a difficulty with learning to categorize class B and C. For the next experiment, we will train a model on a binary data set. Given that the score of A, B, C means approved for a range of breeds (the cut-off is distinct for every breed), the dataset was split into two binary classes: 'Approved' and 'not approved'.

### Experiment 6: Binary classification unbalanced dataset

*Table 17: Experiment 6: Training parameters*

| Learning rate | Epochs | Batch size | Optimizer | Class weights |
|---|---|---|---|---|
| 0.001 | 8 | 10 | SGD | **Yes** |

Since the best results were achieved with batch size of 10 and an unbalanced dataset with weighted classes, these learnings will be applied to binary classification as well. Given the distribution of the dataset, the weights were set to the following:

```
class_weight = {0:1, 1:2.4}
```

As seen in the training graphs 94, the validation and training accuracy ends up at almost the same accuracy in the end, although there is a major drop during in validation accuracy at epoch 4. At this epoch, the model improved training accuracy at the cost validation accuracy.

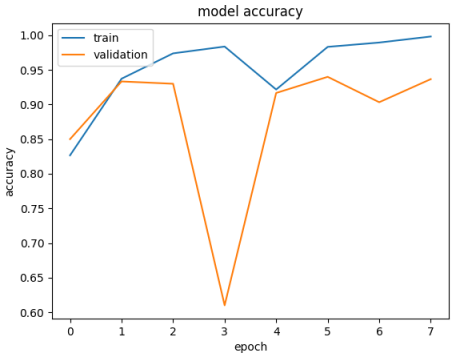



*Figure 94: Experiment 6: Unbalanced binary dataset - Model accuracy as a function of the number of epochs*

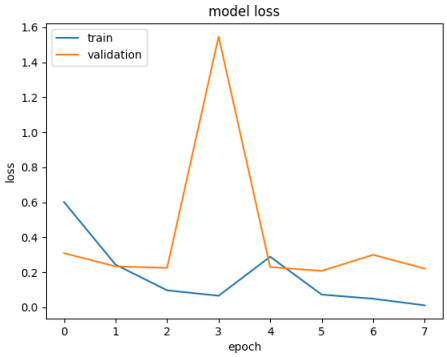



*Figure 95: Experiment 6: Unbalanced binary dataset - Model loss as a function of the number of epochs*

Similar to the validation accuracy in figure 94, the predictions on the test data performs with a very high accuracy. In the classification report (see table 18), the average precision, recall and f-1 score are all 96 percent, showing that the model is highly capable of differentiating between [A, B, C] and [D, E].

*Table 18: Experiment 6: Unbalanced binary dataset - classification report on test data. Note that the test data set is slightly unbalanced (150 vs. 140).*

| Class | Precision | Recall | f1-score | Support |
|---|---|---|---|---|
| Approved | 0.94 | 0.97 | 0.96 | 150 |
| Not approved | 0.97 | 0.94 | 0.95 | 140 |
| avg / total | 0.96 | 0.96 | 0.96 | 290 |

To get a better understanding of the images that are miss-classified, a sensitivity analysis was conducted. The idea is to assess how confident the model is when it is wrong. Ideally, with binary classification, the confidence is as close to 50 percent as possible for the miss-classification. If this is the case it is possible to build a model that correctly flags images that need human attention. Three miss-classifications are illustrated below (the difference in exposure should be overlooked - this is due to using two different python image libraries). For every miss-classification the original full size image is illustrated next to it. As mentioned in the dataset description section, some x-ray scans have not been conducted with a correct orientation at the clinic and hence right and left are switched. In those cases, it could make it challenging for the model to predict correctly if the hips have different scores.

*Figure 96: Confidence for this miss-classification was 84 percent. As seen in the full image, the x-ray scan is oriented correctly (see 'L')*

*Figure 97: X-ray of miss-classified hip from experiment 6*

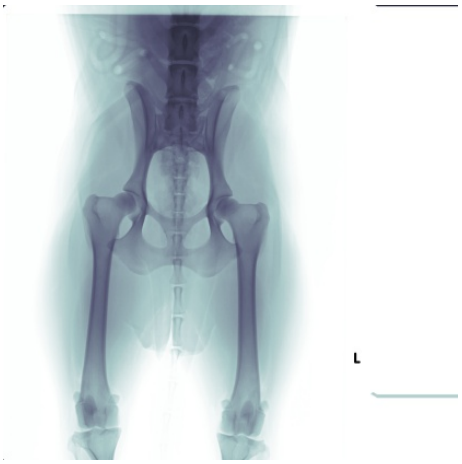


*Figure 98: Full x-ray corresponding to figure 97.*

*Figure 99: Confidence for this miss-classification was 76 percent. As seen, the orientation of the x-ray scan is wrong (see 'R'). Further, the images are very dark.*

*Figure 100: X-ray of miss-classified hip from experiment 6*


*Figure 101: Full x-ray corresponding to figure 100.*

Given that the hips are oriented incorrectly in figure 100, it was not possible for the model to predict on the correct mapping of input and output. However, upon opening the DKK evaluations, it was discovered that both hips for this observation received the same score. Hence, this argument is discarded as a possible reason. Therefore, it is likely that the darkness of the image is the reason for the prediction difficulties.

*Figure 102: Confidence for this miss-classification was 99 percent. The orientation of the image is correct.*

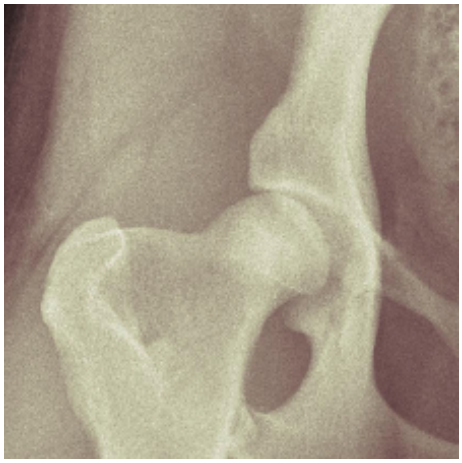


*Figure 103: X-ray of miss-classified hip from experiment 6*

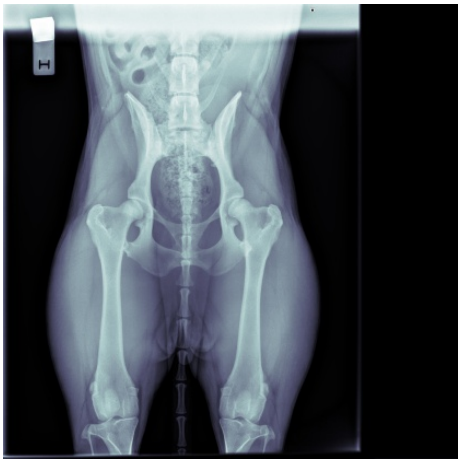



*Figure 104: Full x-ray corresponding to figure 103.*

This miss-classification is more concerning as the confidence is very high (99 percent), yet the prediction is wrong, despite not having any immediate problems with the image data.

## 8.2 Summary of Experiments

In table 19 is a summary of the classification reports from the conducted experiments. All statistics represent the averages from the hip score results of all the classes.

*Table 19: Summary of the classification reports of the conducted experiments*

| Experiment | Precision | Recall | f1-score | Batch Size | Learning Rate | Weights |
|---|---|---|---|---|---|---|
| #1 - Baseline | 0.55 | 0.53 | 0.53 | 20 | 0.001 | No |
| #2 - Learning rate | 0.44 | 0.43 | 0.43 | 20 | 0.0001 | No |
| #3 - Batch size | 0.70 | 0.67 | 0.65 | 10 | 0.001 | No |
| #4 - Unbalanced | 0.72 | 0.71 | 0.71 | 10 | 0.001 | Yes |
| #5 - Full hip images | 0.45 | 0.45 | 0.44 | 10 | 0.001 | No |
| #6 - Binary | 0.96 | 0.96 | 0.96 | 10 | 0.001 | Yes |

As seen from the summary table 19, the best performing experiment for five-class classification was conducted with a batch size of 10 and with an unbalanced training set (experiment 4). The fact that the unbalanced training set helped improve the precision by 2 percent (compared to experiment 3), shows that more data is able to improve the accuracy of the model. Other experiments such as further fine-tuning and using ensemble models could improve the accuracy further. That said, as the data-size of the experiments are very modest, compared to other medical experiments of observations exceeding 100.000 instances, a larger dataset is certainly worth gathering. With a sufficiently large, balanced dataset a model could be trained from scratch. During conversations with McEvoy and Proschowsky, it has become clear that other kennel clubs and researchers are interested in participating in a broader data-collection. At the current data-size, the model is very capable of distinguishing between [A, B, C] and [D, E] as binary classification reaches 96 percent precision.

# 9 Conclusions

Based on the work on hip dysplasia with CNNs, the following can be concluded:

- Transfer learning with CNN models trained on RGB image datasets, such as ImageNet, can be successfully applied to a small dataset of grayscale medical x-ray images of hip dysplasia.
- Within object detection, an average IoU of 92 percent and mAP of 100 percent on the test data (consisting of 464 examples) was achieved by implementing YOLO v2 on a dataset of 2,264 annotated hips.
- Within classification, promising results were generated based on the hip extraction from the object detector using a ResNet-152 model pre-trained on ImageNet. Using an unbalanced dataset of 7,404 observations, a f1-score of 71 percent was achieved on 5-class classification. Further, on a binary dataset of 5,577 observations, a f1-score of 96 percent was achieved with an unbalanced

dataset. Given the limited data-sizes, it is likely that better results can be produced with a larger and more balanced dataset.

- Using an object detector to extract features, and thereby excluding noise, was an efficient strategy to clean the data for the classifier. When using the entire x-ray image as input only a 30 percent validation precision accuracy and 44 percent precision test accuracy was achieved.

- Some classes were more difficult to distinguish between for the classifier. In particular, the experiments showed that the model had difficulties generalizing rules for class B and C in the confusion matrices. From conversations with the evaluator, the problem of generalizing these classes could arise from the fact that the classification rules for B and C are less well-defined. Thus, given the current supervised dataset, there might be an upper-limit for the accuracy of these classes. This is also reflected in the increase of accuracy in the f1-score from 72 percent to 96 percent once the dataset is divided into a binary set of [A, B, C] and [D, E].

## 9.1 Future Research

Based on the fact that a prototype classifier has reached promising results on a limited dataset from Denmark, here are some areas that could be interesting to further examine in the domain of hip dysplasia and CNNs:

- Collect more data from other national kennel clubs. This has already been discussed and the kennel club from Finland has shown interest. By gathering a larger and more balanced data-set it would be possible to train a model from scratch. For such a model, national bias could be tested by making one of the inputs correspond to the nationality of the rating. In addition, the impact of transfer learning could be measured.

- Conduct a study among multiple evaluators to examine to which degree they agree on evaluations. The results would be a base-line from which the results of a CNN would be compared to.

- Collect a labelled dataset consisting of majority voting from multiple evaluators and test if better results can be produced this way.

- Since ensemble models achieve the highest accuracies in most data science competitions [27], an ensemble model could be tested based on different architectures (VGG, Inception, etc.) for further improvements in accuracy.

# Bibliography


[1] Heather J. Chalmers, Stephanie Nykamp, and Assaf Lerer. The Ontario Veterinary College Hip Certification Program. *The Canadian Veterinary Journal*, 54(1):42–46, January 2013.

[2] List of dog breeds recognized by the FCI, February 2018. Page Version ID: 826981781.

[3] Artificial neural network, May 2018. Page Version ID: 841745464.

[4] Emerging Technology from the arXiv. The Revolutionary Technique That Quietly Changed Machine Vision Forever, May 2018.

[5] Dave Gershgorn. The data that transformed AI research and possibly the world, May 2018.

[6] Walter Pitts, April 2018. Page Version ID: 839028442.

[7] Anish Singh Walia. Activation functions and types, May 2017.

[8] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[9] AI winter, April 2018. Page Version ID: 838824526.

[10] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, October 1986.

[11] Sergey Ioffe and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *arXiv:1502.03167 [cs]*, February 2015. arXiv: 1502.03167.

[12] MNIST database, May 2018. Page Version ID: 840500375.

[13] CS231n Convolutional Neural Networks for Visual Recognition.

[14] Adit Deshpande. Beginner Guide To Understanding Convolutional Neural Networks Part 2, May 2018.

[15] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single Shot MultiBox Detector. *arXiv:1512.02325 [cs]*, 9905:21–37, 2016. arXiv: 1512.02325.

[16] Krzysztof J. Geras, Stacey Wolfson, Yiqiu Shen, S. Gene Kim, Linda Moy, and Kyunghyun Cho. High-Resolution Breast Cancer Screening with Multi-View Deep Convolutional Neural Networks. *arXiv:1703.07047 [cs, stat]*, March 2017. arXiv: 1703.07047.

[17] Xiaosong Wang, Yifan Peng, Le Lu, Zhiyong Lu, Mohammadhadi Bagheri, and Ronald M. Summers. ChestX-Ray8: Hospital-Scale Chest X-Ray Database and Benchmarks on Weakly-Supervised Classification and Localization of Common Thorax Diseases. pages 3462–3471. IEEE, July 2017.

[18] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollar. Microsoft COCO: Common Objects in Context. *arXiv:1405.0312 [cs]*, May 2014. arXiv: 1405.0312.

[19] Adrian Rosebrock. ImageNet: VGGNet, ResNet, Inception, and Xception with Keras, March 2017.

[20] AlexNet, April 2018. Page Version ID: 835991175.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. *arXiv:1512.03385 [cs]*, December 2015. arXiv: 1512.03385.

[22] NG Andrew. Vanishing / Exploding gradients - Practical aspects of Deep Learning | Coursera, May 2018.

[23] Ng Andrew. Intersection Over Union - Object detection.

[24] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. *arXiv:1506.02640 [cs]*, June 2015. arXiv: 1506.02640.

[25] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *arXiv:1311.2524 [cs]*, November 2013. arXiv: 1311.2524.

[26] Joseph Redmon and Ali Farhadi. YOLO9000: Better, Faster, Stronger. *arXiv:1612.08242 [cs]*, December 2016. arXiv: 1612.08242.

[27] Profiling Top Kagglers: Bestfitting, Currently #1 in the World, May 2018.

[28] DICOM, May 2018. Page Version ID: 840967966.

[29] pydicom: Read, modify and write DICOM files with python code, May 2018. original-date: 2013-10-31T02:58:52Z.

[30] Huynh Ngoc Anh. keras-yolo2: Easy training on custom dataset. Various backends (MobileNet and SqueezeNet) supported. A YOLO demo to detect raccoon run entirely in brower is accessible at https://git.io/vF7vi (not.., May 2018. original-date: 2017-03-22T15:08:29Z.

[31] OpenCV library, May 2018.

[32] Jason Wang, Serra Mall, and Luis Perez. The Effectiveness of Data Augmentation in Image ClassiïňĄcation using Deep Learning. page 8, 2017.

[33] Joseph Lemley, Shabab Bazrafkan, and Peter Corcoran. Smart Augmentation - Learning an Optimal Data Augmentation Strategy. *IEEE Access*, 5:5858–5869, 2017. arXiv: 1703.08383.

[34] Resnet-152 pre-trained model in Keras.

[35] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On Large-Batch Training for Deep Learning: Generalization Gap and Sharp Minima. *arXiv:1609.04836 [cs, math]*, September 2016. arXiv: 1609.04836.

[36] Dominic Masters and Carlo Luschi. Revisiting Small Batch Training for Deep Neural Networks. *arXiv:1804.07612 [cs, stat]*, April 2018. arXiv: 1804.07612.