



Open-source dog breed identification using CNN

Explanation of the development & underlying technological specifications

Samuel Granvik

Degree Thesis

Information Technology

2023

Degree Thesis

Samuel Granvik

Open-source dog breed identification using CNN. Explanation of the development & underlying technological specifications

Arcada University of Applied Sciences: Information Technology, 2023.

Identification number:

8745

Abstract:

This thesis aims to develop a deep learning model that uses the Fast.ai library and transfer learning to create a convolutional neural network model. The model will identify dog breeds in digital images and evaluate the developed model's performance with regards to loss and accuracy. The thesis also aims to explain the underlying technological specifications of a deep learning model. The model has been trained on dog breed images from the Stanford Dogs dataset and tested against similar dog breed identification models and apps. Open source is a cornerstone of the thesis, and the code of the developed model will be hosted publicly on the GitHub platform under an open-source license for further expansion by others. The motivation for this research comes from the lack of accurate open-source dog breed identification tools alongside the growing popularity of dogs, as well as the author's passion for deep learning and computer vision. The research questions include the development of an accurate CNN model, the suitability of Fast.ai for image classification tasks, the selection of a pre-trained model to use with transfer learning, and the comparison of the developed model against similar models and apps. Overall, this thesis aims to create a well-functioning, well-documented and modern model that accurately identifies dog breeds, and the thesis and developed code are meant to inspire others to learn and work in the fields of computer science such as artificial intelligence, and deep learning.

Keywords:

fast.ai, convolutional neural network, computer vision, transfer learning, artificial intelligence, machine learning, deep learning, open source

Lärdomsprov

Samuel Granvik

Öppen källkods hundrasidentifiering med faltningsnätverk. Förklaring av utvecklingen & dess underliggande teknologiska specifikationer.

Yrkeshögskolan Arcada: Informationsteknik, 2023.

Identifikationsnummer:

8745

Sammandrag:

Detta examensarbete fokuserar på utvecklingen och utvärderingen av en djupinlärningsmodell byggd med överföringsinlärning som använder sig av programmeringsbiblioteket Fast.ai för att utveckla ett faltningsnätverk. Modellen ska identifiera hundraser från digitala bilder. Examensarbetet har också som mål att förklara de underliggande teknologiska specifikationerna av en djupinlärningsmodell. Modellen har tränats med bilder från Stanford Dogs datasetet, och testats mot likartade utvecklade modeller och appar. Öppen källkod är en nyckedel av arbetet, koden för den utvecklade modellen finns offentligt tillgänglig på tjänsten GitHub licenserad med en öppen källkodslicens, så att andra kan ta del av och modifiera koden. Bakgrunden till arbetet kom från bristen av träffsäkra öppna källkodsverktyg för att identifiera hundraser, den ökande populariteten av hundar, samt skribentens passion för djupinlärning och datorseende. Forskningsfrågorna sammanfattar utvecklingen av en noggrann CNN modell, Fast.ais förmåga att lösa bildklassificering problem, valet av en färdig tränad modell för att möjliggöra överföringsinlärning, och jämförelsen av den utvecklade modellen gentemot likartade modeller och appar. Detta arbete fokuserar på att skapa en välfungerade, väldokumenterad och modern modell för att så noggrant som möjligt identifiera hundraser. Detta arbete och dess kod kunde förhoppningsvis inspirera andra till att lära om och arbeta med ämnen relaterade till datorseende så som artificiell intelligens och djupinlärning.

Nyckelord:

fast.ai, faltningsnätverk, datorseende, överföringsinlärning, artificiell intelligens, maskininlärning, djupinlärning, öppen källkod

Contents

Figures	3
Abbreviations	3
Explanations.....	4
1 Introduction	6
1.1 Aim of the study.....	6
1.2 Goals.....	6
1.3 Background	7
1.3.1 Motivations	9
1.4 Research Questions	9
1.5 Hypothesis.....	9
1.6 Limitations.....	9
1.7 Related Work	10
2 Technical Specifications	11
2.1 Computer Vision	11
2.1.1 Object Localization	11
2.1.2 Object Recognition.....	11
2.1.3 Image Classification.....	12
2.2 Artificial Intelligence	12
2.3 Machine Learning.....	13
2.4 ML Algorithms, Frameworks & Models	13
2.4.1 Algorithms	13
2.4.2 Artificial Neural Networks.....	14
2.4.3 Models.....	14
2.4.4 Supervised Learning	15
2.4.5 Unsupervised Learning.....	15
2.4.6 Reinforcement Learning.....	15
2.4.7 Frameworks.....	16
2.5 Transfer Learning	16
2.6 Deep Learning	17
2.7 Fast.ai	18
2.7.1 Hyperparameters	18
2.8 Pre-Processing & Data Augmentation	18
2.9 Recurrent Neural Networks.....	19
2.9.1 Long Short-Term Memory.....	20
3 Methods.....	20
3.1 Convolutional Neural Network	20
3.2 CNN Architecture	21
3.2.1 Feature Extraction.....	22
3.2.2 Overfitting & Underfitting.....	23
3.2.3 Activation Functions.....	24

3.2.4	Pooling.....	26
3.2.5	Fully Connected Layer	27
3.3	Backpropagation	27
3.4	Open Source.....	28
3.5	Development Process	28
3.5.1	Setup	28
3.5.2	Exploratory Data Analysis	29
3.5.3	Hyperparameters & Data Augmentation.....	30
3.5.4	Data Loading.....	30
3.5.5	Training.....	31
3.5.6	Post Training Analysis.....	31
3.5.7	Export	32
4	Results.....	32
4.1	Training Stats	33
4.2	Trained Model Performance.....	33
4.3	Analysis	37
4.3.1	Similar Models.....	38
4.3.2	Apps.....	40
4.3.3	Test Dataset	41
5	Discussion & Conclusions.....	42
5.1	Hardware & Software Limitations	42
5.1.1	Python	43
5.2	Stanford Dataset	43
5.3	Architectures.....	44
5.4	Open source.....	45
5.4.1	License.....	45
5.5	Readability	46
5.6	Shareability	46
5.7	The Thesis.....	47
5.8	Future Goals.....	47
6	References	48
7	Appendices	52
7.1	Appendix 1: Summary in Swedish.....	52

Figures

Figure 1. An ad covering an app on launch.

Figure 2. The relative search volume of pet, dog, and cat adoptions between 2018 – 2020, culmination of search results when WHO declared the COVID-19 pandemic.

Figure 3. Machine Learning is a type of Artificial Intelligence. Deep learning is an especially complex part of Machine Learning.

Figure 4. Shows the three main learning methods, the data they use, and in what areas they are applied.

Figure 5. Visualization of a traditional neural network and a deep neural network.

Figure 6. The structure of a Convolutional Neural Network.

Figure 7. Visualization of an image's colour values forms a pixel matrix.

Figure 8. The structure of convolution operation using a kernel convolution on an image.

Figure 9. Fast.ai's training statistics show decreasing train and validation loss, whilst accuracy increases.

Figure 10. Activation functions.

Figure 11. Visualization of both max pooling and average pooling operations.

Figure 12. The optimally trained model is shown in the .json stats file.

Figure 13. A table created by Liu et al. (2022) shows the stats of several pre-trained models tested against two popular image datasets.

Figure 14. Screenshot showing the final trained models' confidence on two dog images.

Figure 15. Screenshot of the prediction made on one image on a dog breed identification web app.

Figure 16. Three screenshots from three different dog breed identification apps and their accuracy predicted on one image.

Figure 17. Shows the fast.ai source code where the predict method uses the get_preds method to make predictions.

Figure 18. Google Trends search results on common pre-trained models.

Abbreviations

AI – Artificial Intelligence

ANN – Artificial Neural Network
API – Application Programming Interface
App – Software application
BS – Batch Size
CL - Convolutional Layer
CNN – Convolutional Neural Networks
CPI - Central Processing Unit
CV – Computer Vision
DL – Deep Learning
DNN – Deep Neural Network
EP – Epoch
FCL - Fully Connected Layer
JSON – JavaScript Object Notation
LR – Learning Rate
LSTM - Long Short-Term Memory
ML – Machine Learning
NLP – Natural language processing
PC – Personal Computer
PL – Pooling Layer
RNN – Recurrent Neural Network
ReLU - Rectified Linear Unit
SGD - Stochastic Gradient Descent
SQL – Structured Query Language
TPU – Tensor Processing Unit
UI – User Interface

Explanations

API – The way for one piece of software to access another piece of software.

Bounding box – A computer-drawn box that encapsulates different types of objects in an image or video.

Convergence – A state during training in machine learning when the loss reaches a minimum, additional training won't improve the current model.

Dataset – A structured collection of data, often containing a multitude of data.

Fast.ai – A software library that allows the development and training of DL models.

Framework – Software that contains tools and components designed to aid in the development of new software.

Git – A version control system used to manage and keep track of developed software.

GitHub – A web platform using the Git version control system to store current and past versions of code, also called repositories, either publicly or privately.

Google Play Store – An app platform for downloading apps to an Android-based smartphone.

Hardware – Physical and tangible components such as computers and servers.

Hyperparameter – A value set before the training of a ML model, that controls the learning process.

JSON – A file format to store information in a pair structure, called key-value, JSON is easily read by both humans and computers.

Kaggle – A website aimed towards data and computer science, it hosts educational content and competitions.

Metadata – Any additional information about data, but not the actual content of the data.

Model – The outcome of training a ML algorithm, used to make predictions on new data.

Notebook – A web app or software to ease the development and shareability of primary computer and data science code.

Paywall – Content that is restricted by the need to pay.

Programming library, dependency and/or package – An addition to a programming language to extend the programming languages features and capabilities.

Open-source – Types of work that can be viewed, shared, and modified due to it intentionally being publicly available, often used in software development.

Python – A high-level interpreted programming language.

PyTorch – A Python framework for working with ML.

Transfer learning – The use of pre-trained models as the starting point for new training.

UI – The visuals used by a user to interact with some form of software.

Web app – A website made to look like a traditional software application or mobile app.

Zip file – A type of file where the original content has been compressed.

1 Introduction

1.1 Aim of the study

This thesis aims to explain the development and underlying technological specifications of a DL model that uses the Fast.ai library to develop a CNN to solve an image classification problem by identifying dog breeds in images.

This thesis will also explain and evaluate the performance of the developed model concerning loss and accuracy. The developed model will be tested against similar dog breed identification models and apps. The trained model will also be assessed on how well it predicts, and how confident it is with its predictions, against single dog images and entire datasets of images unseen by the trained model.

1.2 Goals

The end goal of the thesis is to have a well-functioning, well-documented and modern model that accurately identifies dog breeds.

When working with classification problems there are two values, we focus on evaluating the performance of our trained model, these values are loss and accuracy. Loss measures the error of a model's predictions against the actual values, while accuracy is a percentage value that measures the model's ability to correctly classify inputs. The overall goal is to minimize the loss and achieve high accuracy. (Paperspace, 2020-a)

The model will be trained on a dataset containing dog breed images with corresponding dog breed names as labels. The dataset in question is the Stanford Dogs Dataset (Khosla et al. n.d.). The images in the Stanford dataset are taken from the much larger ImageNet dataset (ImageNet, 2021). The use of additional images outside the Stanford dataset will be used to evaluate the trained model.

The research is also meant to be further expanded upon by others, therefore open source is a cornerstone of the thesis. The code of the developed model, and all its past revisions, will be hosted publicly on GitHub under an open-source license.

The code has been meticulously documented throughout its development process, using the open-source notebook software called Anaconda notebooks. Notebooks utilizes the easy-to-read structure of cells that aids in the readability and documentation of the code. The combination of the well documented code, the technical specifications explained in this thesis, and the results gathered creates a unified understandable solution to the tasks of accurately identify dog breeds in digital images.

1.3 Background

The idea to develop a dog breed identification model came along as my fiancée and I brought home our first dog. I downloaded a lot of apps to easily scan and identify dogs that we saw. But the apps that I tried to use were slow, contained paywalls, gave inaccurate results, were filled with ads, and did not work as intended.

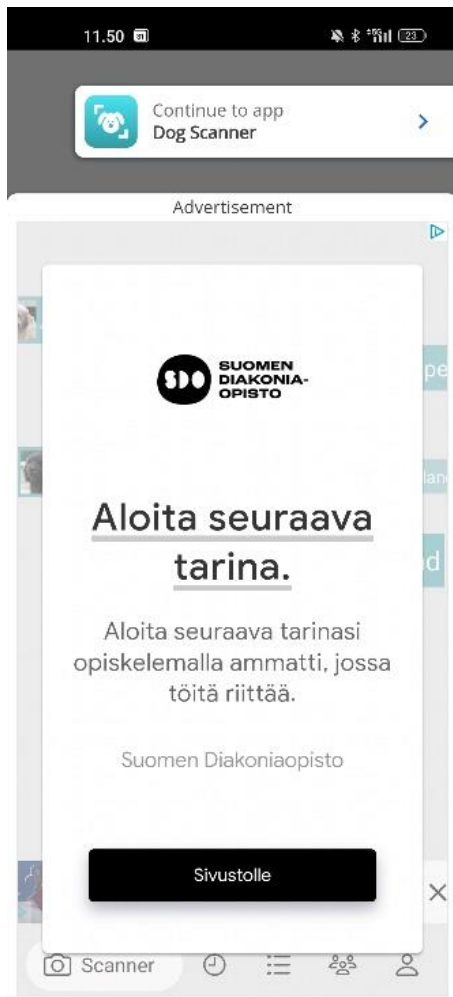


Figure 1. An ad covering an app on launch.

During the covid-19 pandemic, a great deal of people bought and adopted dogs. “From 2015 to 2020, the worldwide relative search volume (RSV) for dog adoption and cat adoption peaked in April 2020, the early epidemic phase of the COVID-19 pandemic.” (Ho et al., 2021)

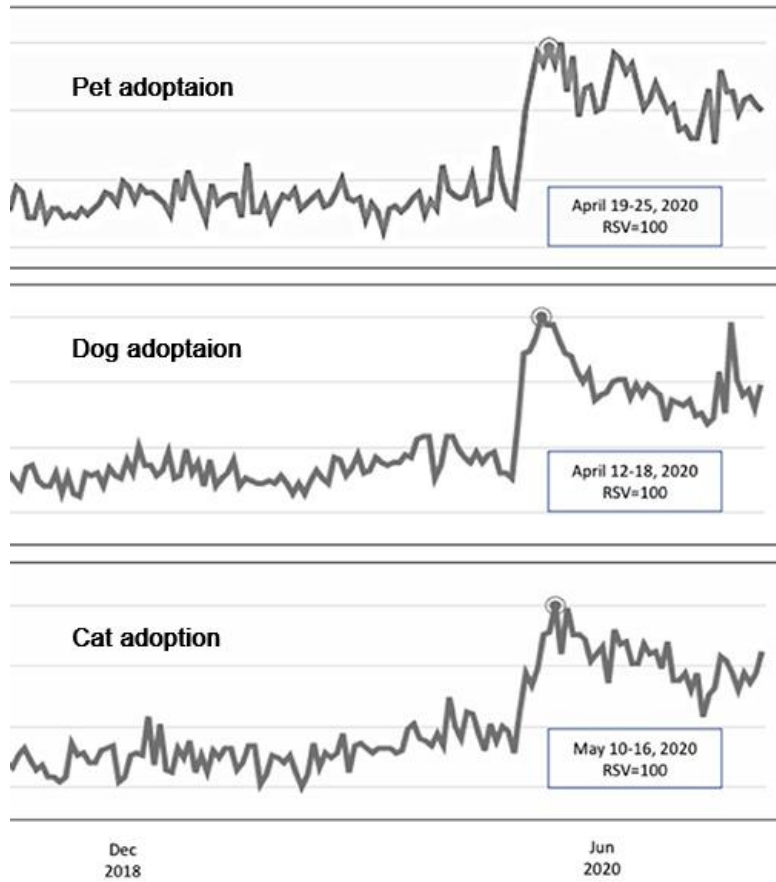


Figure 2. The relative search volume of pet, dog, and cat adoptions between 2018 – 2020, culmination of search results when WHO declared the COVID-19 pandemic. (Ho et al., 2021)

Computer science subjects regarding neural networks and CV can be quite difficult to navigate and learn. The idea to create a unified solution by combining the well-documented code and this thesis was formed. This was done to teach and inspire others based on my work.

During my university studies, I was taught AI, ML, and DL among other subjects. So the combined lack of accurate dog breed identification tools, the growing popularity of dogs, my passion to help others, and my continued interest in the subjects of DL gave me the idea to further hone my skills and develop a model of my own.

1.3.1 Motivations

ML grows in popularity and usage every year, with an annual growth forecast of 39.4% between 2021 – 2026 (BCC Research, 2022). Based on the statistics the need to understand and utilize ML is important for both corporations and individuals alike.

DL and especially CV is for me a fascinating subject. The idea that with just a few lines of code, and a relatively small dataset, be able to train and run all the code on a cloud-hosted computer and achieve state-of-the-art results seems almost too good to be true. To also be able to take part in large communities of like-minded individuals, share my work, and have the support from my previous studies aids and motivates me to keep on developing, learning, and writing.

1.4 Research Questions

- *How does one develop an accurate CNN model?*
- *Is Fast.ai well suited for image classification tasks?*
- *What pre-trained models are most suitable for dog breed identification?*
- *How well does my developed model compare to similar models and apps?*

1.5 Hypothesis

I hypothesize that the model will outperform existing dog breed identification models and apps.

The final model will have an accuracy of $\geq 90\%$. The Related Work chapter goes into more detail about the performance of similar developed models and research.

1.6 Limitations

The thesis will focus on only one area of ML, which is image classification when it comes to identifying dog breeds from digital images. AI, ML and DL are vast areas to study, and they contain many algorithms, frameworks, libraries, and different ways to develop.

The comparison against similar models and apps will only be against models and apps that identify dog breeds.

One programming language will be used to develop the model, Python. Python is one of the most used programming languages when it comes to computer science and all the subfields of AI. (Raschka & Mirjalili, 2019, p. xiv)

I will also only use a set amount of additional programming libraries, packages, and dependencies to manage the creation of the model and any additional functionality.

The model will be developed and trained on my personal computer's hardware, and by using the Google Colab platform. The training of a ML/DL model is computationally intensive and thus the use of external software and or products can be beneficial to achieve the desired results. (Neil et.al., 2020). Google Colab enables users to leverage the computing power of Google's hardware, making it effortless to perform computer science tasks through a web-based interface. (Google, n.d.)

1.7 Related Work

A participant from the Kaggle challenge created a Dog Breed identifier with Fast.ai. The code is quite old, but the accuracy on the validation dataset is ~89%. (Langenbach, 2019)

Another participant in the Kaggle challenge, with a newer submission, achieves an accuracy of ~80% during the training. The developer states that “This couple of lines of codes is still capable of achieving ~0.64 score on the leaderboard (~80% of accuracy). The model is far from being state-of-the art but it is perfect to use as a baseline to which further improvements could be compared.” (Angyalfold, n.d.)

Another developer built a web app combined with a trained model to enable the use of a web browser to predict dog breeds images. The model was trained on the same Stanford dataset, and also add images of dog breeds not present in the Stanford dataset. (Willjobs, n.d.)

The use of a CNN and transfer learning was applied to the data of a type of brain cancer, in the paper by Deepak & Ameer. (2022). The authors show that the use of a CNN model results in better performance compared to the traditional ML model when it comes to the detection and classification of brain tumours.

2 Technical Specifications

This chapter will go over the technical specifications of the thesis. It will cover the necessary parts needed to understand CV, and any relevant fields related to CNNs. The development of the CNN will be explained in detail in the methods chapter.

2.1 Computer Vision

CV is a field within AI and computer science that focuses on the ability of computers to be able to use and analyze images and videos like humans. The main goal is for computers to be able to view, interpret and understand the world as human eyes do. One part of CV is to find features and patterns in images. (IBM, n.d.). The following chapters outline the CV tasks most important to this thesis.

2.1.1 Object Localization

Localization will locate an object in an image and create a rectangle box around the object, called a bounding box. Localization is a bit lacklustre in what it can output, as it only handles the metadata, or metrics of an object in an image like width, height, and position. “Here, an object proposal comprises an object category (e.g., “dog”), coordinates of a bounding box centre, and the bounding box’s width and height.” (Brownlee, 2017)

2.1.2 Object Recognition

Recognition is the broad term that encompasses all the different tasks linked to identifying objects, patterns, or relationships within an image. Such as object detection, image segmentation and image classification among others. So, recognition is the process and broad term of analyzing an image to find patterns and other meaningful information. (Brownlee, 2017)

2.1.3 Image Classification

Image classification is a sub-set and specific task of image recognition. Image classification takes an image as an input and outputs a classification of an object in the image. For example, the input is an image of a dog, and the classification model labels the object as a dog. The output will contain certain additional properties such as the: probability and accuracy that the object in the image is what it is labelled as. This thesis will fall under the image classification task as its goal is to classify dog breeds in images. (Brownlee, 2017)

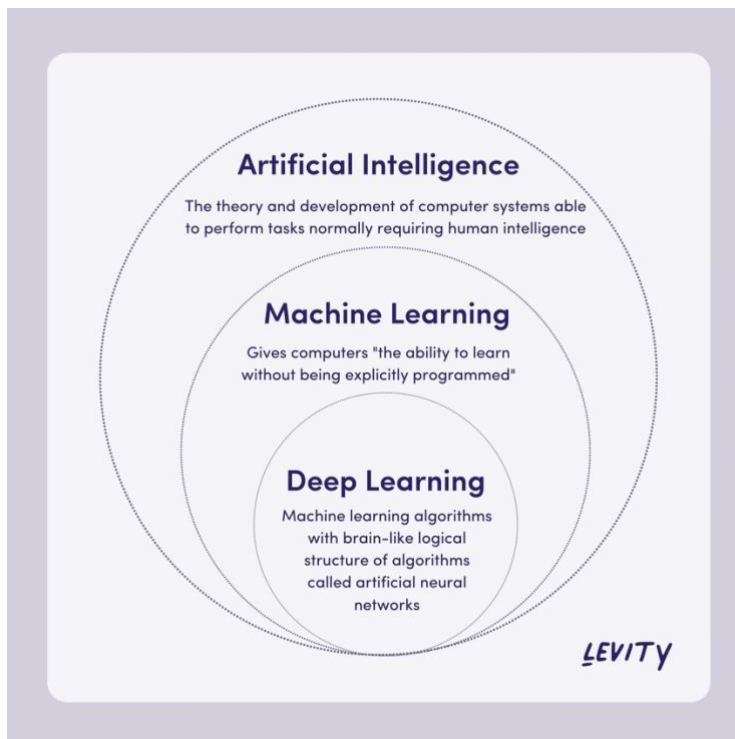


Figure 3. Machine Learning is a type of Artificial Intelligence. Deep learning is an especially complex part of Machine Learning. (Wolfewicz, 2023)

2.2 Artificial Intelligence

AI is the term used for machines, or more commonly computer programs being able to simulate human intelligence. AI focuses on human tasks such as understanding languages, object detection, and predictions among others. (McCarthy, 2007)

The integration of AI is prevalent across various industries, as it has the potential to improve the quality and effectiveness of products and services. AI is utilized by both individuals and corporations. Corporations can use AI to e.g., Enhance their products and

services, take Google's parent company Alphabet for example: "Alphabet understands the potential of AI and is set to use it across its businesses, from improving internet searches, to self-driving cars, automated homes, intelligent virtual assistants, language translation and life-saving medical science." (Marr, 2019)

AI is the broader concept of several sub-sets and fields that focus on solving specific problems related to human intelligence. (Raschka & Mirjalili, 2019, p. 1) The following chapters will outline certain fields of ML, CV, and DL.

2.3 Machine Learning

ML is a subfield of AI, that focuses on the ability of computers to learn from data and apply that knowledge to make predictions or decisions on new unseen data, without the need for explicit programming. ML is a way for computers to learn from experience, on their own. It differs from AI in the sense that ML focuses on learning patterns, features, and relationships in data, whereas AI is the broader field that encompasses a range of techniques and computer science fields including ML. (Raschka & Mirjalili, 2019, p. 1-2)

ML can work with both structured and unstructured data. Structured data can be thought of as data that has been correctly formatted and labelled based on its content. Think of structured data as an image of a dog, and the label is the breed of the dog, or as the data in a SQL database or spreadsheet with rows and columns correctly labelled, structured, and formatted. Unstructured data refers to data that is unlabeled and has no predefined structure or schema such as videos, images, and text. (IBM, 2023)

2.4 ML Algorithms, Frameworks & Models

2.4.1 Algorithms

When we want to use ML, we need to use an ML algorithm. In ML, algorithms get fed input data, process the input, and arrive at a desired output, in this thesis case a prediction of a dog breed. (Alpaydin, 2010, p. 1) There are plenty of ML algorithms such as decision

trees, k-nearest neighbor, and neural networks. They are all suited towards solving different types of problems.

2.4.2 Artificial Neural Networks

A neural network or ANN is a type of ML algorithm and is often used in DL. Its structure is often inspired by biological neural networks like the human brain. (IBM, 2023) ANNs consist of an input layer, one or more hidden layers, and finally one output layer. The layers are interconnected nodes, or "neurons", that process and transform input data to produce output data. ANNs can be used for a variety of ML tasks, such as classification, regression, and clustering. (Alpaydin, 2010, p. 273)

2.4.3 Models

When we feed an ML algorithm data as input, and the algorithm processes the input data we often say that the algorithm is learning or training on the data. When the algorithm has processed all the data we are left with trained data in a file called a model. The patterns, features, relationships, and settings of the trained data are stored in the file. We can use this model on similar, but new and unseen data. (Microsoft, 2023)

In ML, when we refer to the process of training a model to recognize features or patterns in data, we often refer to it as learning. There are three main learning methods: supervised, unsupervised and reinforcement learning, differentiated by where they are used and how they process input and output data. (Raschka & Mirjalili, 2019, p. 2) A core part of these methods is that they either use labelled or unlabeled data. Labelled data just means that the object in the data is named, categorized, or tagged after what it contains. E.g., We have an image of a dog, if the label says dog, then we have a correctly labelled image. (IBM, 2023)

All three methods have their unique advantages and are used in various applications. The methods can be applied to various areas of data in AI and ML including speech recognition, NLP, and most importantly regarding this thesis, CV.

“In the 1990s, computer speech recognition reached a practical level for limited purposes. Thus United Airlines has replaced its keyboard tree for flight information by a system using speech recognition of flight numbers and city names” (McCarthy, 2007)

2.4.4 Supervised Learning

With supervised learning, we train the model using labelled data as input and get labelled data as output. We find the relationship between the input data and the output label. The models are then tuned to as accurately as possible predict the labels of new unseen data. This type of learning requires the most amount of human interaction since we need to label the data before training it. Classifications tasks, such as image classifications fall under supervised learning. We use supervised learning in spam detection, dog breed identification, and weather forecasting among others. (Raschka & Mirjalili, 2019, p. 3)

2.4.5 Unsupervised Learning

With unsupervised learning, we use unlabeled data as input, but the output data will be analyzed data either grouped, clustered and/or characterized. We use unsupervised learning to find patterns, hidden structures, and trends in a dataset, or to cluster similar data together. Recommender systems use this type of learning in for example e-commerce websites, and movie recommendations on popular streaming services. (Raschka & Mirjalili, 2019, p. 7)

2.4.6 Reinforcement Learning

Reinforcement learning is different from supervised and unsupervised learning as its approach to learning is based on a reward structure. It uses an agent, and its goal is to maximize its reward in an environment. If the agent makes wrong actions, it loses points, and if it makes good decisions, it is rewarded points. Reinforcement learning is applied in areas such as self-driving cars, robotics, gaming, and other areas where the agent must learn to make decisions in a dynamic environment. (Raschka & Mirjalili, 2019, p. 671-674)

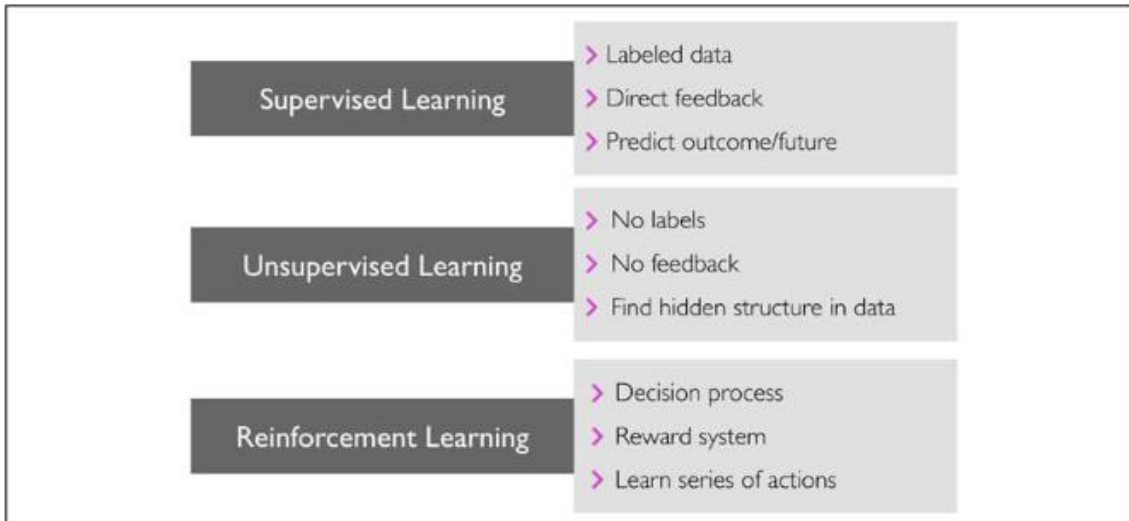


Figure 4. Shows the three main learning methods, the data they use, and in what areas they are applied. (Raschka & Mirjalili, 2019)

2.4.7 Frameworks

We often use ML and DL frameworks or libraries such as Fast.ai, PyTorch, Keras, or TensorFlow among others, to aid the development of models. A DL framework is a platform or software that lets us develop, train, and deploy ML models by providing pre-built tools, methods, and APIs to load process and output data. These tools ease the development by letting us utilize the special APIs built to interact with the underlying hardware we run the framework on. This makes the development and training of an ML algorithm far easier than doing all the programming, calculations, and development manually from scratch. (Nvidia Developer, n.d.-a)

2.5 Transfer Learning

Transfer learning is a technique in ML and DL where we leverage a pre-trained model as the starting point for a new model. Instead of training our model completely from scratch, learning all the features and patterns of each image, we are building upon the knowledge gained by an already trained model. The pre-trained model can have been trained on several types of objects such as cats, clothes, cars, or even more detailed objects or features such as mouths, flower types, and brands among others. We use transfer learning to heighten accuracy, lower loss, reduce the amount of input data, and lower the computational resources and time needed to train a model. (Yosinski et.al., 2014, p. 1-2?)

2.6 Deep Learning

DL is a subset of ML that uses a multi-layered ANN structure known as a DNN, used to solve complex problems. The use of multiple hidden layers creates complexity, lets the network learn detailed features, and improves the accuracy of the final output. (Raschka & Mirjalili, 2019, p. 383-388)

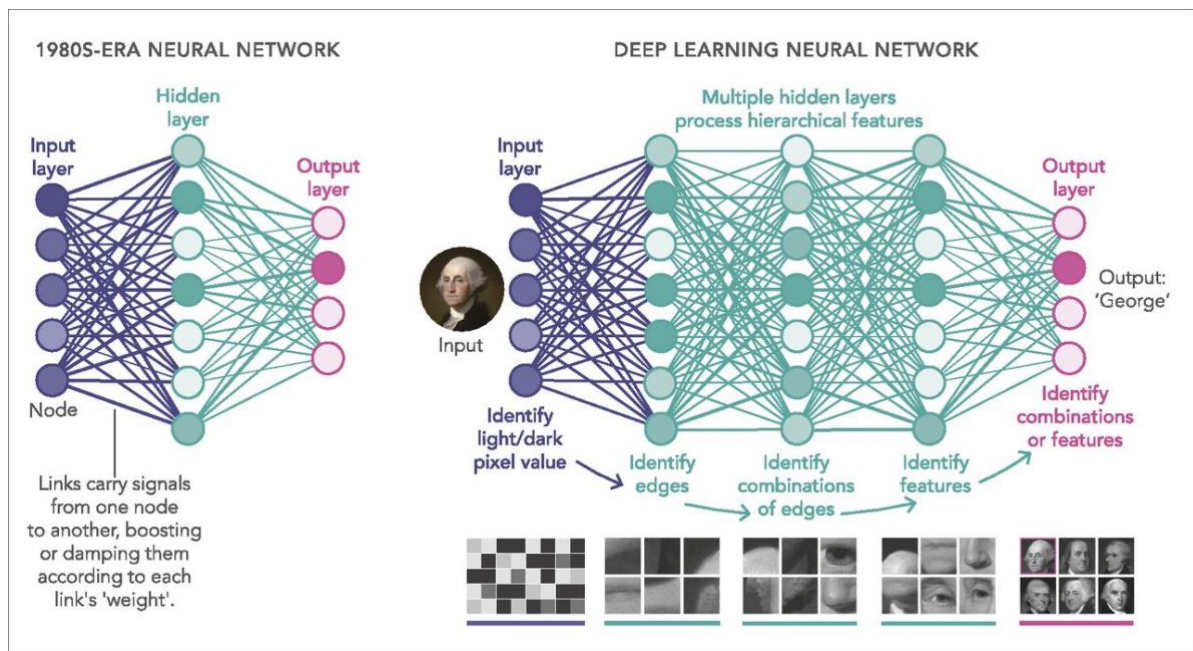


Figure 5. Visualization of a traditional neural network and a deep neural network. (Vogel & Bowers, 2019) Image credit: Lucy Reading-Ikkanda (artist).

DL is particularly useful for tasks that involve large amounts of data, such as NLP, autonomous driving, speech recognition, and image classification. DL algorithms can perform tasks that other ML algorithms cannot, such as learning from and making accurate predictions on previously unseen data. DL often requires more training data, to begin with, and that requires larger computational resources. But DL requires less human intervention compared to traditional ML algorithms. (Paperspace, 2020-b)

“Popular examples of the products in our everyday life that are powered by deep learning are Google's image search and Google Translate—an application for smartphones that can automatically recognize text in images for real-time translation into more than 20 languages.” (Raschka & Mirjalili, 2019)

2.7 Fast.ai

This thesis uses the open-source DL library Fast.ai, which is built on top of the PyTorch framework. Fast.ai utilizes a high-level API that connects with the PyTorch framework to develop and train ANNs. (Fastai, n.d.)

Fast.ai follows best practices and uses a clear syntax, making it more efficient to use and allowing users to train models more quickly compared to similar frameworks and libraries. Fast.ai takes advantage of PyTorch's many well-written functions, methods, and capabilities. Many of Fast.ai's functions and classes are PyTorch's with additional functionality or tweaks. Being built upon PyTorch also means that Fast.ai can leverage many of the underlying functions and methods found in PyTorch. (Howard, J., & Gugger, S. 2020)

2.7.1 Hyperparameters

When we are using a DL to create a model, we often have certain hyperparameters available to tweak. This lets us change the processing data and the subsequent training of our model. The values we change can impact our trained model's accuracy and loss, but they also affect the speed and efficiency of our training. (Chollet, 2021, p. 263)

Some common hyperparameters found in DL algorithms are EPs and LR. An EP refers to one full pass of the entire dataset during training. (Raschka & Mirjalili, 2019 p. 25). LR is the step size used for a model to reach the minimum loss function during the optimization of the model during training. (Chollet, 2021, p. 49)

We often find other tweakable hyperparameters such as batch size BS. BS is a set amount of training examples/images used during one iteration of the training of the model. (Paperspace, 2020-c)

2.8 Pre-Processing & Data Augmentation

Pre-processing and data augmentation are common techniques used to prepare data for use in ML and other fields related to data processing. Pre-processing is used to transform raw data into a format that is more suitable for algorithms to handle. This can involve

resizing images and converting data among others. The goal of pre-processing is to generalize and normalize the data to make it easier to analyze and work with. (Raschka & Mirjalili, 2019, p. 12-13)

Data augmentation is a technique used to artificially enlarge a dataset by creating new, versions of existing data. This is done to make the dataset more diverse and varied, which in turn can improve the performance of a trained model. Data augmentation techniques include random cropping, zooming, rotation, flipping, and changes to colour channels among others. (Raschka & Mirjalili, 2019, p. 552)

Regarding image classifications with CNNs, we want to resize or downsample images before feeding them to an algorithm. This is done to reduce the number of pixels that the algorithm needs to handle, which may improve the efficiency and speed of the training. It is also done since most CNNs require a fixed image size of all the images it works with. (Rosebrock, 2019, p. 37, 69)

When we use a DL framework or library like Fast.ai we split the entire dataset into three different datasets, train, test, and validation. We often use an 80/20% split on the entire dataset, so the training becomes 80% of the entire dataset, and 20% is left for the testing dataset, but other common split ratios are available. (Raschka & Mirjalili, 2019, p. 124) We additionally use a subset, often 20%, of the training dataset for a validation dataset. This way we get one dataset to train the model on by feeding it labelled images. One validation dataset to assess the performance of the model during training. Finally, we have a testing dataset that is used to test the trained model on unseen images. (Raschka & Mirjalili, 2019, p. 196)

There are different types of DNNs aimed towards solving different types of problems, these following chapter will go into more detail about RNNs.

2.9 Recurrent Neural Networks

RNNs are used for tasks that involve sequential data, i.e., data that has a specific order or sequence, such as speech recognition and language translation. RNNs process inputs of

different lengths and use a sort of memory and so-called feedback loop to retain the information about previous inputs. (Raschka & Mirjalili, 2019, p. 567-569)

2.9.1 Long Short-Term Memory

LSTMs are a sub-type of RNNs that are designed to better handle the vanishing gradient found in RNNs. LSTMs can handle long-term dependencies in the same type of data as RNNs, sequential data, i.e., LSTMs remember things better than normal RNNs. (Raschka & Mirjalili, 2019 p. 581-584)

The type of DL network that this thesis revolves around and will be going into more depth are CNNs. CNNs are commonly used in CV tasks, such as image, recognition, and classification. They are designed to identify patterns and features in images by analyzing and finding features in regions of an input image. (Raschka & Mirjalili, 2019, p. 515-519)

3 Methods

3.1 Convolutional Neural Network

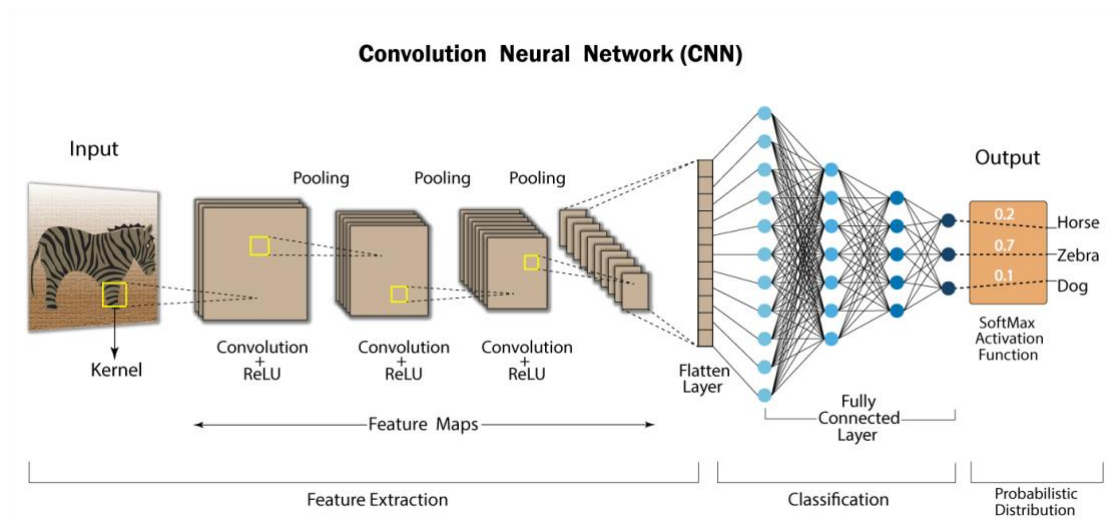


Figure 6. The structure of a Convolutional Neural Network. (Swapna, n.d.)

CNNs are a type of DNN, CNNs are currently one of the most effective ways for computers to perform image classifications. (Raschka & Mirjalili, 2019, p. 518) CNNs can work with other data than specifically image data such as speech recognition, and NLP among others. (Nvidia, n.d.-b) Moving forward this thesis will focus on image data.

CNNs view images as matrices of numbers, also called colour channels. These numbers refer to the image’s pixel data/information. Pixel data tells us the colour intensity of a pixel. CNNs can work with both greyscale and RGB images. Each channel in an image gets one matrix. Therefore, an RGB image will have three matrices for each colour channel red, green, and blue. When moving further down in the layers of a CNN the RGB matrices get combined into one unified matrix. (Raschka & Mirjalili, 2019, p. 532-533)

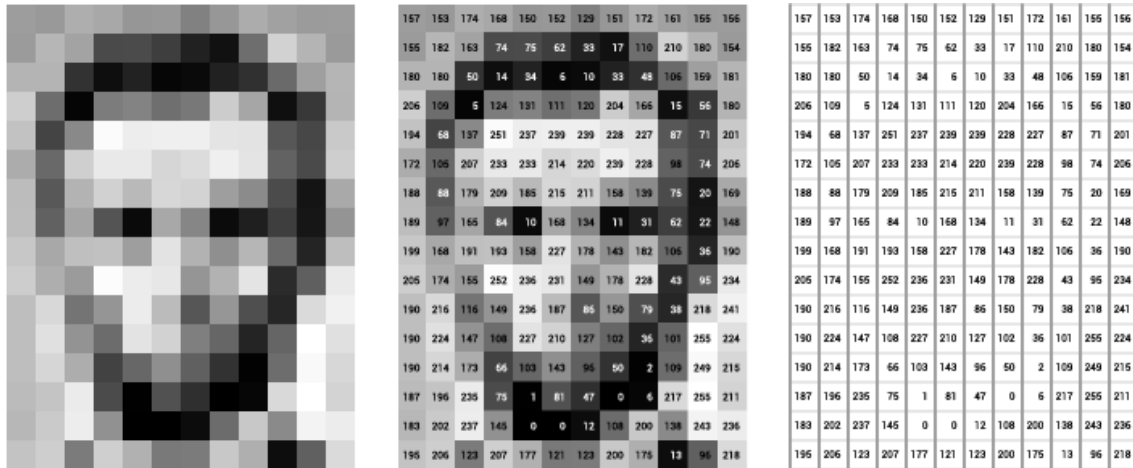


Figure 7. Visualization of an image’s colour values forms a pixel matrix.

3.2 CNN Architecture

“Typically, CNNs are composed of several convolutional and subsampling layers that are followed by one or more fully connected layers at the end.” (Raschka & Mirjalili, 2019) This process is called forward pass or forward propagation, and it’s used to pass each input image through the network to find its features. The final prediction is made by the FCL, and then backpropagation is used to tweak the learnable parameters to try to reduce the loss. (Rosebrock, 2019, p. 137-140)

CNNs are composed of several layers that work together to extract features from an input image. First, there is the feature extraction stage, which contains CL which uses multiple convolutional operations to process the image and extract features. The output of a CL is a feature map. Activation functions are then applied to introduce non-linearity, followed by pooling operations to reduce the spatial dimensionality of the feature maps. After the feature extraction stage is complete, the feature maps are flattened and passed on to a FCL, which is a regular ANN. (Nvidia Developer, n.d.-b)

3.2.1 Feature Extraction

The first layer in a CNN is a CL and it is the process that differentiates a CNN from a traditional neural network, or multilayer perceptron. CLs perform the computational heavy lifting, using convolutional operations or more commonly referred to as kernel convolutions. (Rosebrock, 2019, p. 181) Kernel convolutions use a kernel, i.e., a small matrix of numbers, called weights, to extract features from an input. The weights inside the kernel are one of two learnable parameters that get optimized during the training of the model. Bias is the second learnable parameter that gets added to the output of the CLs. This is done to help the network recognize features even if they are not centred around zero. It also adds flexibility and makes the model more expressive. (Chollet, 2021, p. 46)

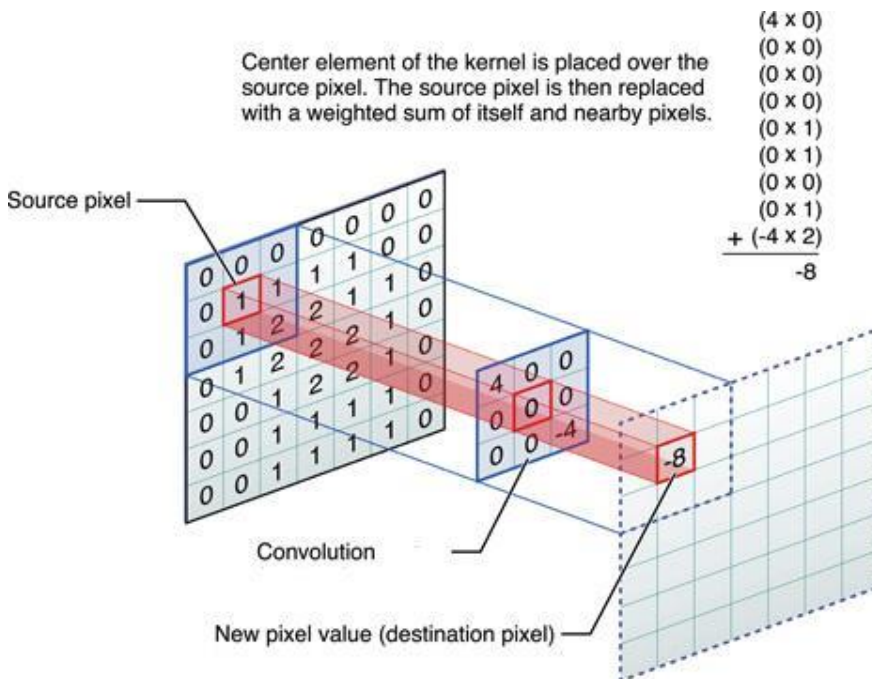


Figure 8. The structure of convolution operation using a kernel convolution on an image. (Zafra, 2012)

During the kernel convolution, the kernel moves over the data in a sliding window motion, and at each position, it performs multiplication of the kernel weights and the input pixel values and adds them together. This process is repeated for every position in the input data, producing an output matrix known as a feature map. (Chollet, 2021, p. 124)

The kernel has a set of hyperparameters that determines the behaviour of the convolution operation, these parameters can affect the performance of the CNN. Larger kernels may

capture complex features, but they require more computational power. Smaller kernels have the opposite effect. The hyperparameters of a kernel include its size, stride, and padding. The size of the kernel determines the shape of the window that is moved across the input data, it is a matrix, for example, 3x3. The stride determines the number of steps that the matrix window moves over the input data. The padding determines the number of zeros that are added to the borders of the input data to make sure that the kernel can be used at the edges of the input data. (Rosebrock, 2019, p. 181-185)

When working with a CNN it uses shared weights and biases across all kernels in the hidden layers, this is often called parameter-sharing. This means that the layers can detect features wherever it is in an image, even if the image gets warped, changed and/or transformed. This also reduces the total number of parameters that need to be learned by each kernel. (Raschka & Mirjalili, 2019, p. 519)

3.2.2 Overfitting & Underfitting

Generalization refers to the ability of models being able to perform well on both the training data and new unseen data. (Paperspace, 2020.-d) There are two main issues when training a model that disrupts the generalization abilities of a model. Overfitting occurs when a model does too well on the training data i.e., it learns the noise of the data as well as the features it's meant to extract. It becomes too complex hence it struggles to perform accurately on new unseen data. Underfitting is the opposite, meaning that the model is too simple and can't find the complex patterns in the training data, and due to that it performs poorly on new data as well. Techniques such as regularization and data augmentation are often used to mitigate overfitting. These methods can help improve the generalization ability of the model. (Raschka & Mirjalili, 2019, p. 75-77)

Fast.ai incorporates several built-in regularization methods and functions. Dropout is a technique that randomly drops neurons in a neural network, it works well to avoid overfitting. By doing this the remaining neurons are forced to learn new features and not focus solely on a couple of neurons. Another technique to reduce the risk of overfitting is a form of L2 regularization often called weight decay. It works by reducing the impact of large weights in a neural network. During training the weight decay penalty gets added

to the loss, which results in the shrinking of the weight parameters. This is done using an optimization algorithm like Adam or SGD. (Raschka & Mirjalili, 2019, p. 75-58)

“Small networks, or networks with a relatively small number of parameters, have a low capacity and are therefore likely to underfit, resulting in poor performance, since they cannot learn the underlying structure of complex datasets. However, very large networks may result in overfitting, where the network will memorize the training data and do extremely well on the training dataset while achieving a poor performance on the held-out test dataset.” (Raschka & Mirjalili, 2019)

Based on the explanations above we can assume that if both the training and validation loss is decreasing during our training, then we can assume that the model can fit the data and make accurate predictions. If the training loss is decreasing, and the validations are increasing or plateauing, then we can assume that the model is overfitting. Underfitting can be assumed if both the training and validation losses are high, and the accuracy is low.

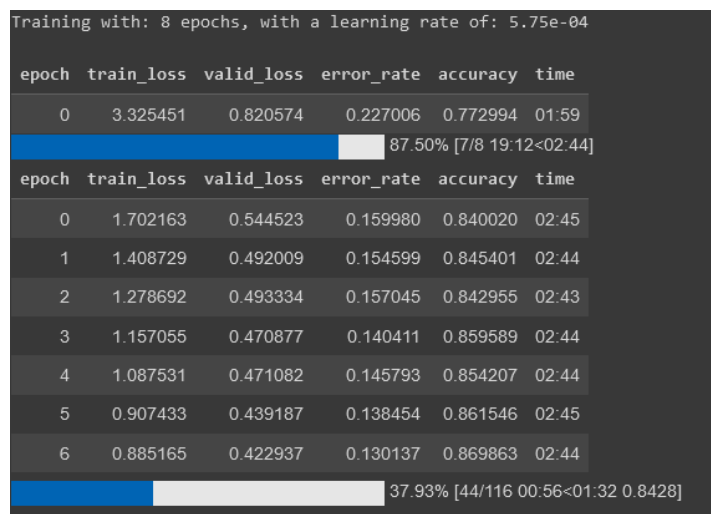


Figure 9. Fast.ai’s training statistics show decreasing train and validation loss, whilst accuracy increases.

3.2.3 Activation Functions

A non-linear function is applied to the output of a CL, using an activation function. This is done to add non-linearity to the data and allow the model to learn more complex

features. Some activation functions are Sigmoid, Tanh and ReLU. ReLU is a common and favorable non-linear activation function $f(x) = \max(0, x)$. The ReLU function effectively sets all negative values to zero, and all positive values stay as they were. Bias helps to shift the activation function towards the positive or negative side. (Raschka & Mirjalili, 2019 p. 468-469)

Table 3. Activation function

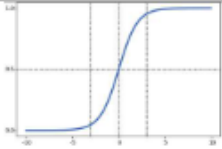
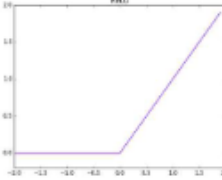
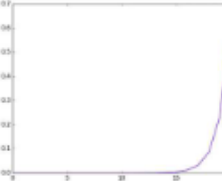
Activation Function	Plot	Equation
Sigmoid[8]		$f(x) = \frac{1}{1 + e^{-x}}$
ReLU[9]		$f(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases}$
Softmax[10]		$f(x_j) = \frac{e^{x_j}}{\sum e^{x_i}}$

Figure 10. Activation functions (Yosinski et.al., 2014)

Activation functions, such as ReLU, have larger derivatives than Sigmoid and other activation functions, and it is a non-saturating activation function. Therefore, ReLU can help mitigate the problem of vanishing gradient, which can arise during the training of DNNs. It makes it difficult to update the weights in the lower layers during training. During each iteration of the training process, backpropagation is used to update the learnable parameters from higher to lower layers. However, the gradient used with the optimization algorithm becomes progressively smaller for each layer, making it difficult to update the weights in the lower layers. This can cause convergence problems where the lower layers may not be updated effectively and may fail to converge to a satisfactory solution. Other solutions such as batch normalization can also aid in counteracting the vanishing gradient. (Lzubaidi et.al., 2021)

Batch normalization is used to improve convergence (Raschka & Mirjalili, 2019 p. 397), and the stability of an ANN by normalizing the input of each layer. This is done by subtracting the mean and dividing the standard deviation of the activations within each batch. (Lzubaidi et.al., 2021)

3.2.4 Pooling

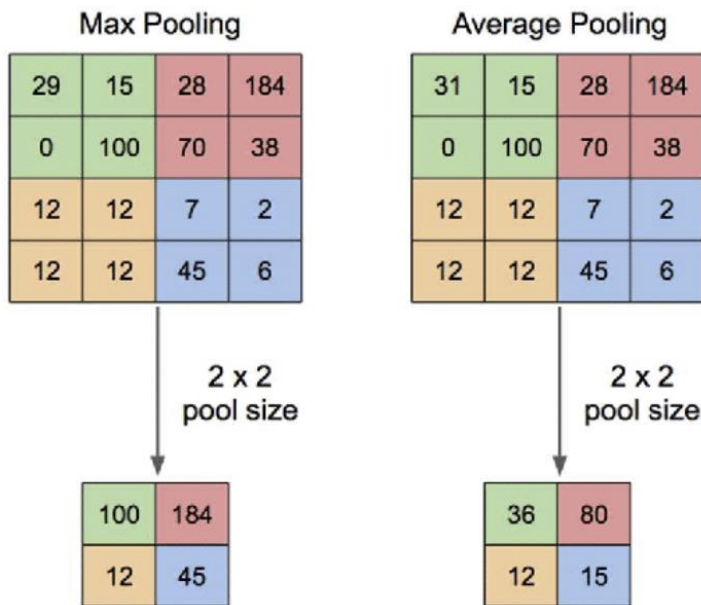


Figure 11. Visualization of both max pooling and average pooling operations. (Yani et.al., 2019)

Lastly, PL helps us reduce the computational stress of the image on the network, by reducing the spatial dimensions of the image, without losing any meaningful data. We do this by downsampling the feature maps of the CLs. There are many different types of PL functions, such as weighted average, L2-norm, and the most common max pooling function. Max pooling takes the highest value from a region covered by the pool size on the feature map. Pooling layers have no learnable parameters. (Raschka & Mirjalili, 2019, p. 520-530)

3.2.5 Fully Connected Layer

The final part of a CNN is the FCL, also called a Multi-Layer Perceptron, which is just a normal DNN, but the provided input of the FCL is not an image but the final output data of the feature extraction stage of the CNN. (Raschka & Mirjalili, 2019, p. 519)

In the FCL we begin by taking the last feature extraction layer values and flattening them, creating a one-dimensional vector. This means arranging the values in a vertical row, each value on top of one another. (Raschka & Mirjalili, 2019, p. 546)

Each layer in the FCL applies its weights and biases to the data and tries to predict the correct class of the input data. (Raschka & Mirjalili, 2019, p. 520)

The final layer of the FCL is a Softmax or Sigmoid activation function which calculates the probability distribution on the processed input image. Softmax is often used for multi-class classifications and Sigmoid for single/binary class classifications. After the probability distribution, we can run different types of loss calculations to find error rates and loss functions. (Raschka & Mirjalili, 2019, p. 539)

3.3 Backpropagation

When the FCL has made its prediction, we use a loss function such as Cross-Entropy Loss (Raschka & Mirjalili, 2019, p. 539), Mean Squared Error among others (Raschka & Mirjalili, 2019, p. 488), to calculate the loss between the actual label and the predicted one. (Alpaydin, 2010, p. 41)

After computing the loss, it is propagated in reverse through the network i.e., backpropagation. The gradient of the loss error updates the learnable parameters i.e., the weights and biases, (Raschka & Mirjalili, 2019, p. 391), with an optimization algorithm such as Adam, or SGD. The optimization algorithm calculates the gradient of the loss concerning the current weights and biases, this allows the model to update its parameters to reduce the loss. (Raschka & Mirjalili, 2019, p. 483)

The updated learnable parameters are then used as the starting point for the next feature extracting loop i.e., the next forward propagation. This is done until the loss reaches a

minimum or the set number of epochs has been reached. (Raschka & Mirjalili, 2019, p. 391)

Two popular optimization algorithms are the SGD and Adam optimizers. SGD computes the gradients of the model's parameters concerning the loss function for a small subset of the training data, called a mini-batch, and uses these gradients to update the parameters. Adam uses adaptive learning rates for each parameter, this aids in faster convergence and functions well when training especially in deep networks. (Kingma & Ba, 2015)

DNNs on their own are usually less efficient than CNNs when it comes to image classification. This is due to many reasons. CNNs use feature hierarchy, meaning that CNNs combine lower-level detected features to form complexity in the later layers. CNNs also use parameter-sharing, meaning that the same weights are used for different areas of the input image. (Raschka & Mirjalili, 2019, p. 519).

3.4 Open Source

Open-source software refers to code or software that has been made public for other people to work, alter and/or view. The importance of open-source code and software cannot be overstated in today's world. Open source enables the development and distribution of stable and well-maintained solutions, and it challenges us to collaborate and work towards a unified goal. Open-source projects can spark ideas and bring people together to work towards a common objective. (Red Hat, 2019)

3.5 Development Process

This chapter will go over the development process of the final model, split up into development chapters. The entire codebase can be easily viewed on GitHub (Krullmizter, n.d.).

3.5.1 Setup

The initial setup consists of importing required Python packages, this is all done in. These packages expand the usability of the Python language such as packages to interact with

the underlying OS, perform HTTP requests and most importantly enable the use of the Fast.ai library.

We also check that the notebook utilizes the GPU instead of the CPU. When it comes to the development and training of Fast.ai, and other DL models, the preference for using a GPU vs. a CPU boils down to the calculations that the algorithm performs on large datasets. These operations are large-scale matrix operations. These calculations can be done in parallel on a GPU, so it is much more efficient, and it will speed up both the development and training time of the model.

We can also use TPUs, they were developed by Google and are used for DL applications. TPUs are well suited towards ANNs, and often use fewer resources than GPUs. However, during this development, the choice of using GPUs came down to availability and usability. (Openmetal, n.d.)

One cell is dedicated to holding: settings and global variables such as direct file paths to frequently used files and directories. This is done to easily and efficiently be able to tweak each setting and variable in the entire notebook.

The loading of the dataset is handled in one cell. It will check if the dataset path exists in the root folder of the project. If it doesn't exist, it will use the `gdown` package to download a public .zip file containing the entire Stanford dataset, and then unzip the package to be used later. This is done to easily supply the entire dataset to any development environment. Using a .zip file will also save a lot of time when download the dataset. There is also an option in the settings cell that lets you change the public Google Drive file URL to another custom URL, this way another dataset can be used.

3.5.2 Exploratory Data Analysis

EDA lets us explore the relationships, and statistics and analyze the dataset overall. The use of the Pandas package aids in this task. We can for example view the dog breeds with the most and least amount images per breed. EDA lets us more easily understand and visualize the data that we work with.

A cell is used to view the distribution of the width and heights of the unprocessed images in the dataset in two scatter plots. The plots are handled by the Matplotlib package. This is done to easily view the spread of the resolutions of each image and find the average image resolution to be used later in the hyperparameters.

3.5.3 Hyperparameters & Data Augmentation

Fast.ai and other DL libraries, algorithms and frameworks let us define and tweak certain hyperparameters: EP, BS, SZ and LR among others. The functionality of these hyperparameters has been gone over earlier in this thesis. The tweaking of these parameters will affect both the computational load, output, and the speed of training, but they are the main factors, aside from the dataset, that affect the outcome of the trained model.

I tried to mix the hyperparameters a bit, trying to heighten the EP whilst lowering the BS and vice versa. Using 224x244px as the SZ/image resolution was used since a lot of the images in the dataset were centred around that resolution. I also tried to change up the pre-trained model, also called arch in Fast.ai, trying out both the well know ResNet, but also other models like EfficientNet, and GoogLeNet among others.

LR is one of the more important hyperparameters we can change. Using Fast.ai's `lr_find()` methods lets us easily find the best starting LR. During the different training runs different types of LRs were calculated.

The functionality of data augmentation was gone over before. Developing the model, the use of Fast.ai's `item_tfms`, and `batch_tfms` are methods available to add data augmentation to the training dataset images such as: resize, rotate, normalize, blur and so on. This is done to create variation of the original dataset images, and artificially enlarge the training dataset. This in turn improves the accuracy of the developed model.

3.5.4 Data Loading

Fast.ai provides a `DataLoaders` class, which aids in loading and applying data augmentation and pre-processing of the data. The `DataLoaders` class can load data from

CSV files, data frames, and directories. We assign a variable to the created `DataLoaders` object, `dls`, to be used later in the `Learner` object.

The `Learner` object combines the `dls` object, our chosen pre-trained model/arch, and any additional metrics we want to receive during the training process. The `Learner` object is later used to fine-tune or fit i.e., train the model.

3.5.5 Training

Fast.ai provides an easy-to-use method when using transfer learning, called `fine_tune()`. The `fine_tune()` method allows us to use transfer learning to train or fine-tune our model based on a pre-trained model with new data. `fine_tune()` only works with the last layer of the pre-trained model, as the rest of the pre-trained layers are frozen. One could use the `fit_one_cycle()` method, but it is more suited for training a model from scratch.

The trained model can then be used to make predictions on new data using the `get_preds()` method. We can view the performance of the model during the training with the `validate()` method.

3.5.6 Post Training Analysis

The use of the `ClassificationInterpretation` object allows us to view and analyze the results of the trained model. Based on its output, we can determine what needs to be tweaked. The use of methods such as `plot_confusion()` and `most_confused()` allows us to easily view which classes were the worst and most frequently misclassified.

Another useful statistic for analysis of the trained model is the F1-score. Fast.ai incorporates this metric using the method `print_classification_report()`. We use the F1-score to measure the model's accuracy for both binary/single or multi-class classifications. It uses a weighted average on the precision and recall of a trained model, $F1 - score = 2 * (precision * recall) / (precision + recall)$. F1-scores range

between 0 – 1. 1 meaning perfect precision and recall, and 0 meaning that the model is performing worse than random chance.

3.5.7 Export

The `Learner` class has an easy-to-use export method to export the trained model as a `.pkl` file. This lets anyone use the final trained model with its weights and hyperparameters set. The exported model file can't be further developed, just used for predictions. We can also save the state of the `Learner` object by using the `Learner.save()` method. It lets us easily use the used parameters, optimizations, and state of the last `Learner` object. The save is often used to continue the training, whereas the export method is used for a finalized trained model. The notebook code also incorporates a cell which explains the process of importing a trained model and using that model to predict an image.

4 Results

This chapter goes over the results of working with the developed code. The final model, with the best performance, will be gone over.

The model with the best performance returned an accuracy of 96.33% on the validation dataset. This is a good percentage, and it ties in with the hypothesis goal of $\geq 90\%$ accuracy overall. This model has been exported as a `.pkl` file and can be used on new unseen data to make predictions. This model is also the base on which the following results and analysis are based on.

```

],
"20987027ad1dae18fbae27a636b3278dd293f37143015275af4a73578302e643": [
  {
    "Time Created": "08/05/2023 - 10:52:58",
    "Training Time": "00:25:53",
    "Arch": "convnext_large",
    "Opt. Func.": "Adam",
    "GPU": "NVIDIA A100-SXM4-40GB",
    "Train/Val. Split(%)": 20,
    "EP": 10,
    "BS": 85,
    "SZ": 224,
    "LR": "6.92e-04",
    "Transforms?": true,
    "Loss(%)": 15.14,
    "Error(%)": 3.67,
    "Accuracy(%)": 96.33
  }
]

```

Figure 12. The optimally trained model is shown in the .json stats file.

4.1 Training Stats

One of the cells in the notebook is a function that handles the logging of the metrics taken from the `validate()` method. The logging cell also incorporates the hyperparameters and some additional statistics such as GPU, and the pre-trained model used, as well as the date and time of the training into the complete log file. The combined training metrics, hyperparameters and additional statistics are then exported to a .json file in a trained directory for easy-to-view post-training analysis of the most important metrics and used parameters, and for longer-term statistical storage. To avoid any duplicates of data in the .json file, a hash function is used to create a unique id for each training stat. It combines some of the used hyperparameters, and training metrics to achieve this uniqueness.

4.2 Trained Model Performance

Various tests have been used to evaluate the performance i.e., the loss and accuracy of the trained model. Using Pytorch's and Fast.ai's F1-score, `validate()`, `predict()`, and `get_preds()` methods. They will be gone into more depth following this chapter.

The `validate()` method's outputs are the main form we will measure the performance of the trained model. `Validate` is a method of the `Learner` class. The method calculates the average loss and other metrics, such as the accuracy of the trained model against the validation dataset. The validation dataset is a sub-set of the entire dataset that gets used during the training to evaluate the performance.

To the research question: *Is Fast.ai well suited towards image classification tasks?* We can conclude with the achieved accuracy of 96% that Fast.ai do function well for image classification tasks. We can also look at the number of submissions that uses Fast.ai for dog breed identification by Kaggle (n.d.)

The training was tweaked and changed to find out which hyperparameters, and settings worked based on Fast.ai and the smaller Stanford dataset. All the past training statistics and metrics are stored in the stats file, this made it easier to look over what hyperparameters worked with what arch, and hardware setup.

The initial training statistics were gathered without any data augmentation, transforms, or ludicrously large BSs and LRs. This was done to create a baseline to know how well the training did on the most basic of hyperparameters and just the normal dataset. The results from the baseline training were quite a high accuracy with a low loss. The accuracy was around the 85% mark, with a loss of approx. 50.

After creating some baseline statistics, I moved on and trained the model with the EPs, BSs and LRs, which gave the best results from the baseline training. I also added more transformations and data augmentations to the data.

Both during the baseline and optimized training I tweaked by using different types of pre-trained models, called archs in Fast.ai. Using the same amount of hyperparameters and just changing the arch gave me results that pointed towards larger networks like `ResNet-101` and `ConvNext Large`. They proved to not be too small nor too big of a network to train on. With these larger networks, we have more parameters to train, allowing for more details, and aiding in regularization, preventing overfitting. The smaller networks, like the `ResNet-18` and `ResNet-34` gave suboptimal results, even with tweaked

hyperparameters to both test against smaller and larger hyperparameter values. The larger arches were often too big to run, I used up all the GPU memory available even before the training had run one EP, even with a small dataset.

The training time using the ConvNeXt network was quite a lot longer than the previously used networks like ResNet, and GoogLeNet among others. The tweaking of the additional hyperparameters such as EP, BS and LR yielded smaller boosts in the performance and output of the trained mode, this can be viewed in the stats file (Krullmizter, n.d.) We can with that conclude that the biggest factor during my training was the choice of pre-trained models.

To answer the research question: *What pre-trained models are most suitable for dog breed identification?* We can look at the accuracy received from training several different pre-trained models, to get a clearer pictures of which ones were better look at the stats file. One table by Liu et al. (2022) shows us that ConvNeXt is one of the top performers when it comes to image classifications on similar-sized images as the Stanford dataset.

model	image size	#param.	FLOPs	throughput (image / s)	IN-1K top-1 acc.
ImageNet-1K trained models					
● RegNetY-16G [54]	224 ²	84M	16.0G	334.7	82.9
● EffNet-B7 [71]	600 ²	66M	37.0G	55.1	84.3
● EffNetV2-L [72]	480 ²	120M	53.0G	83.7	85.7
○ DeiT-S [73]	224 ²	22M	4.6G	978.5	79.8
○ DeiT-B [73]	224 ²	87M	17.6G	302.1	81.8
○ Swin-T	224 ²	28M	4.5G	757.9	81.3
● ConvNeXt-T	224 ²	29M	4.5G	774.7	82.1
○ Swin-S	224 ²	50M	8.7G	436.7	83.0
● ConvNeXt-S	224 ²	50M	8.7G	447.1	83.1
○ Swin-B	224 ²	88M	15.4G	286.6	83.5
● ConvNeXt-B	224 ²	89M	15.4G	292.1	83.8
○ Swin-B	384 ²	88M	47.1G	85.1	84.5
● ConvNeXt-B	384 ²	89M	45.0G	95.7	85.1
● ConvNeXt-L	224 ²	198M	34.4G	146.8	84.3
● ConvNeXt-L	384 ²	198M	101.0G	50.4	85.5
ImageNet-22K pre-trained models					
● R-101x3 [39]	384 ²	388M	204.6G	-	84.4
● R-152x4 [39]	480 ²	937M	840.5G	-	85.4
● EffNetV2-L [72]	480 ²	120M	53.0G	83.7	86.8
● EffNetV2-XL [72]	480 ²	208M	94.0G	56.5	87.3
○ ViT-B/16 (⊠) [67]	384 ²	87M	55.5G	93.1	85.4
○ ViT-L/16 (⊠) [67]	384 ²	305M	191.1G	28.5	86.8
● ConvNeXt-T	224 ²	29M	4.5G	774.7	82.9
● ConvNeXt-T	384 ²	29M	13.1G	282.8	84.1
● ConvNeXt-S	224 ²	50M	8.7G	447.1	84.6
● ConvNeXt-S	384 ²	50M	25.5G	163.5	85.8
○ Swin-B	224 ²	88M	15.4G	286.6	85.2
● ConvNeXt-B	224 ²	89M	15.4G	292.1	85.8
○ Swin-B	384 ²	88M	47.0G	85.1	86.4
● ConvNeXt-B	384 ²	89M	45.1G	95.7	86.8
○ Swin-L	224 ²	197M	34.5G	145.0	86.3
● ConvNeXt-L	224 ²	198M	34.4G	146.8	86.6
○ Swin-L	384 ²	197M	103.9G	46.0	87.3
● ConvNeXt-L	384 ²	198M	101.0G	50.4	87.5
● ConvNeXt-XL	224 ²	350M	60.9G	89.3	87.0
● ConvNeXt-XL	384 ²	350M	179.0G	30.2	87.8

Figure 13. A table created by Liu et al. (2022) shows the stats of several pre-trained models tested against two popular image datasets.

During the training, it was found that the training time increased substantially when changing from ResNet to ConvNeXt. The local training was not possible using the ConvNeXt network as the GPU memory limit was met long before the training was completed. The purchase of additional compute units on Google Colab was necessary to test and train the final model, with the more advanced, and deeper networks. Only one training run was possible using the lower tier of Google Colab with lower hyperparameters and with pure luck. The later additions to the stats file using ConvNeXt were able to run with a shorter training time thanks to the additional Google Colab compute units that were purchased.

4.3 Analysis

This chapter goes over the analysis of my final developed model, against similar models and apps that identify dog breeds. This way we can evaluate how well the predictions of my developed model, and similar developed models compare. The performance of each model is measured in two key values: correctly predicted class and confidence percentage of the predicted class.

Before we move on towards the single image predictions using a pre-defined set of dog images I want to bring up one cell/method in the notebook that offers the testing of dog images retrieved from a random Google search, against the trained model. The method downloads a random dog breed image retrieved from a Google search, with the search phrase: dog. The prediction method then predicts the class and confidence of the prediction of the retrieved dog image. This way one can easily and quickly evaluate the trained model against new unseen images without the need to manually download each image.

During the evaluation of similar single image predictions and/or dog breed identification apps, I used two images. One image of my dog Laban, who is a Labrador retriever, and an image of a Bernese Mountain dog. This way each single image prediction, whether being a trained model, app, or other dog breed identification program, has been evaluated on the same set of images. The single image predictions have been tested using the `Learner.predict()` method in Fast.ai. It takes an image path and returns a tuple, the first value is the predicted class, and the second is the confidence of the predicted class. (Warner, 2021)

My final developed model successfully predicted the breeds for both Laban and the Bernese Mountain dog. Not only were the predictions accurate, but they also carried a high level of confidence. Laban's prediction had a confidence level of 94.34%, while the Bernese Mountain dog's prediction boasted an impressive 99.33% confidence level.

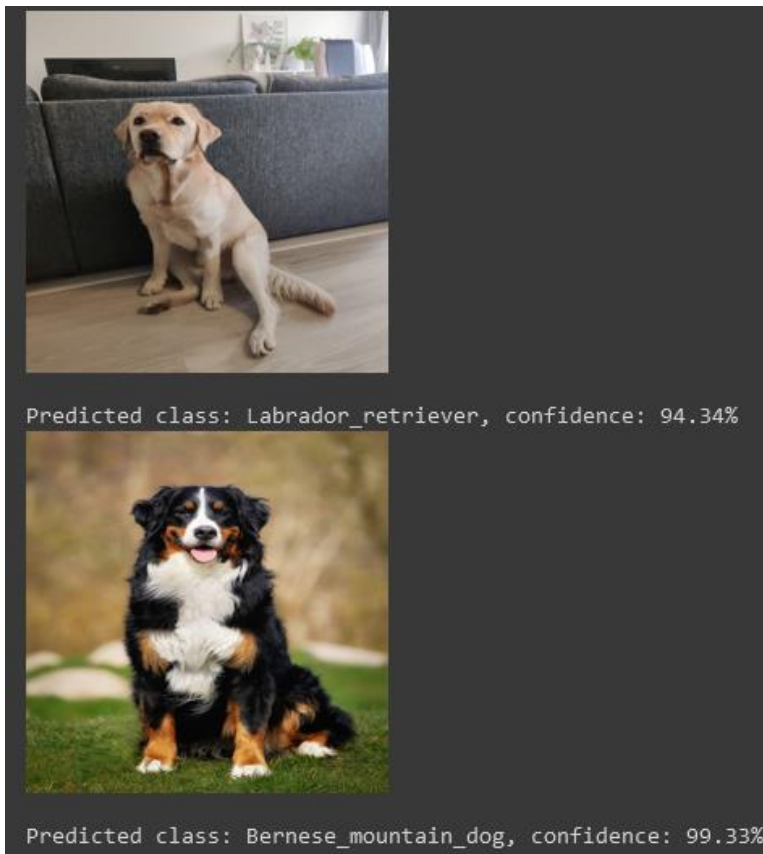


Figure 14. Screenshot showing the final trained models' confidence on two dog images.

4.3.1 Similar Models

The evaluation of similar models has been done using the same `predict()` method used in the evaluation of my own developed model. The chosen similar models have all been trained using CNNs, and some sort of transfer learning. I've selected a few developed models, run their training, and evaluated their predictive abilities on the images of Laban and the Bernese Mountain dog.

As mentioned earlier in the previous research chapter one individual created a web app that incorporates a UI with their developed model. (Willjobs, n.d.) Their developed model is like mine; it uses Fast.ai, the same dataset and techniques. It, however, uses the mid-sized pre-trained model: ResNet-50. The developed model by Willjobs, n.d., can't accurately identify Laban as a Labrador retriever, with the confidence of Labrador at 58.45%. The model is, however, 99.38% confident that the image of the Bernese Mountain dog is a Bernese Mountain dog.



What the model thinks

Breed	Confidence	Example
Labrador Retriever	58.45%	
Carolina Dog	16.84%	
Anatolian Shepherd Dog	10.87%	
Collie	3.04%	
Chinese Shar-Pei	1.50%	

Figure 15. Screenshot of the prediction made on one image on a dog breed identification web app.

The Stanford dataset, which this thesis is built upon, has been used in many competitions and challenges. One of these challenges can be found on the Kaggle website and was the inspiration for this thesis. It holds the submission of other developers' models and code.

(Kaggle, n.d.) The next evaluation will be a submission taken from the challenge from Kaggle.

The developed model by Angyalfold, 2021, correctly identifies the breeds of both Laban and the Bernese Mountain dog. The model is 99.26% confident in the image of Laban, and 99.25%. They used the ResNet-34 pre-trained model for their transfer learning.

4.3.2 Apps

I used three different apps gathered from the Google Play store to evaluate their predictions and confidence percentage.

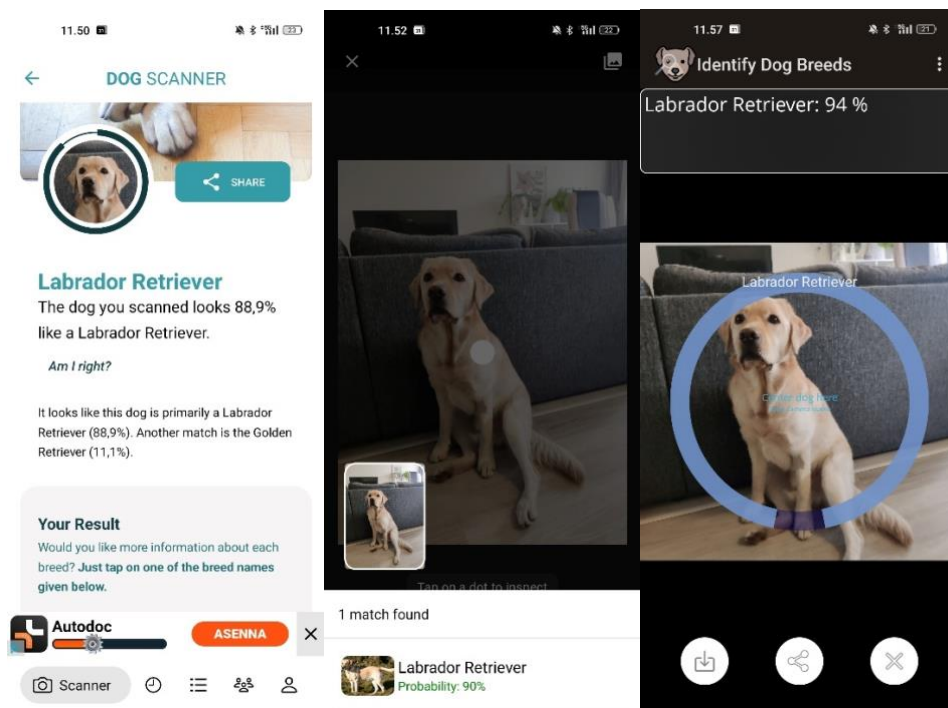


Figure 16. Three screenshots from three different dog breed identification apps and their accuracy predicted on one image.

When using the apps on the image of Laban, they all predict his breed correctly, and the average confidence is 90,6%.

Using the same apps, with the image of the Bernese Mountain dog the average confidence of the apps is 75%.

We can see that even with my limited training dataset the accuracy of my model is up to par with published dog breed identification apps. However, since the apps I've evaluated are not open-source, I cannot find out what type of training they have used to build their models.

4.3.3 Test Dataset

The Stanford dataset contains two predefined split datasets, train, and test. The test dataset contains images of dog breeds not seen by the model during training. We can predict these images. Using the Fast.ai `get_preds()` method we can perform so-called batch predictions on the entire dataset. We begin by creating a `dataloader` based on the test dataset and adding it to the `get_preds()` method. It returns a tuple containing predictions and targets/labels. Our test dataset images have no labels, so we will ignore the second output of the `get_preds()` method. The output we are left with is the predictions that the trained model has done without the aid of any labels. The predictions are in the shape of a matrix or tensor. Each value in the matrix represents the model's confidence in an image predicted class, but instead of predicting one image at a time, we predict an entire dataset. (Warner, 2021)

This prediction matrix data is useful for us as we can view how confident the model is in its predictions without relying on the correct labels for verification. This can also be seen as the process of predicting multiple unseen images like the similar `predict()` method explained previously on single image predictions. The `predict()` method uses the `get_preds()` method for single image predictions. (Fastai, n.d.)

The developed notebook also contains a cell that adds the label from the labels.csv file to the best value/breed for each image that the `get_preds()` method outputs. We get the best breed for each image outputted from the `get_preds()` method using the `argmax()` method, it finds the highest value aka. The highest predictions, in this case in a matrix.

```

319 v | def predict(self, item, rm_type_tfms=None, with_input=False):
320     dl = self.dls.test_dl([item], rm_type_tfms=rm_type_tfms, num_workers=0)
321     inp,preds,_,dec_preds = self.get_preds(dl=dl, with_input=True, with_decoded=True)
322     i = getattr(self.dls, 'n_inp', -1)
323     inp = (inp,) if i==1 else tuplify(inp)
324     dec = self.dls.decode_batch(inp + tuplify(dec_preds))[0]
325     dec_inp,dec_targ = map(detuplify, [dec[:i],dec[i:]])
326     res = dec_targ,dec_preds[0],preds[0]
327     if with_input: res = (dec_inp,) + res
328     return res

```

Figure 17. Shows the fast.ai source code where the predict method uses the get_preds method to make predictions.

5 Discussion & Conclusions

The Fast.ai library has proven itself capable of performing dog breed identifications by using transfer learning and implementing CNNs. The smaller dataset of the Stanford dataset has not made the developed model any worse than any of the other models, and apps that it has been tested against. The following chapters discuss the developed model and thesis.

5.1 Hardware & Software Limitations

One big problem that emerged when training the model was the problem of the amount of computational resources required to train the model efficiently. (Neil et.al., 2020) My PC could train some models but whenever the pre-trained model was too large, the BS or EP was set too high my PC couldn't handle the computations and ran out of GPU memory.

The Google Colab platform was a great way to take the computational load of my PC. It also made it possible for me to work on other tasks alongside the training of the model since the training was done in the cloud, and training on my PC consumed most of my PC's resources. However, the Colab instances often timed out and didn't save, and when I upgraded to a Colab Pro account (11€/month), the computational hours I were granted were consumed in less than two days of training and testing. Google offers several levels of Colab plans to suit one's needs. The Colab Pro offers 100 compute units, pay as you go, and Colab Pro+ with 500 compute units. (Google, n.d.). The need to purchase

additional compute units was required to train with the larger pre-trained models, and finally achieve a good accuracy and low loss.

When training with larger EPs, and especially larger BSs a common error encounter both on my PC and the free Colab version was a GPU memory error. `OutOfMemoryError: CUDA out of memory`. A self-explanatory error that points to the fact that the GPU has run out of memory. Several fixes can be applied. Upgrade the GPU in question, lower the BS, and free up non-essential allocated memory using `torch.cuda.empty_cache()`, and/or split the images into smaller, less memory-intensive chunks, by setting a max split size using the `max_split_size_mb` in the data loader.

5.1.1 Python

Python, and other high-level programming languages, i.e., languages that are designed to be more readable and understandable, are often considered slower compared to compiled languages like C or Swift since Python is an interpreted language. This means that each time a Python program is run, the interpreter needs to translate the Python code into machine code. This often requires more computational time compared to compiled languages where the code is translated only once and executed directly on the computer. Additionally, Python is dynamically typed, which means that data types are determined at runtime. Dynamically typed languages offer more flexibility and ease of use but come at a cost of performance, as the interpreter needs to spend extra time verifying and converting data types.

Jeremy Howard, the creator of Fast.ai says that Python is one of the bottlenecks of the Fast.ai library, and that “Python is not the future of machine learning” (Weights & Biases, 2022).

5.2 Stanford Dataset

The Stanford dataset is quite small, it holds only 120 different dog breeds, ~150 images per dog breed, with a total of 20,580 images. This limited amount of data can be a reason for suboptimal results when training a model on this dataset.

A common issue with smaller datasets is the issue of overfitting. When the model has less data to train with overfitting can occur. This means that the trained model may perform well on its trained data but perform poorly on new unseen data.

Additionally, smaller datasets may perform poorly when trying to optimize the models' parameters, as the amount of available data may not be sufficient to train the model effectively.

The choice of model architecture when dealing with smaller datasets is also good to key. More complex models may overfit the training data, while simpler models may underfit and fail to capture important patterns and features in the data. Therefore, choosing a suitable model architecture that balances both complexity and performance is key, especially when working with smaller datasets. In my case the larger models did perform better than the smaller ones. A key part of larger pre-trained models are their capabilities of capturing many more different features and patterns due to larger number of parameters they can take, tweak and train.

5.3 Architectures

Thanks to PyTorch Fast.ai lets us easily import and utilize various popular architectures or pre-trained models such as ResNet, VGG, and EfficientNet among others. It was a challenging decision to select the most suitable pre-trained model for conducting the transfer learning. The choice of using ConvNeXt came down to the fact that it performed best when I tweaked all the hyperparameters during the training. However, in the results chapter, one can see that my notebook and code enable the use of different types of pre-trained models. One can easily view the difference in performance based on the chosen pre-trained model using the stats file.

The use of transfer learning was a good choice, and it improved the development experience, cut down on the computational resources and time to train. To train a

complete model from scratch is most likely a rewarding experience.



Figure 18. Google Trends search results on common pre-trained models. Google Trends (n.d.)

5.4 Open source

The entire codebase is hosted publicly on the GitHub platform, along with the stats file. The repo also holds the past versions of the developed code. This makes it easy for anyone to clone/download the base files needed to run and train a model of their own. The hope is that other developers would use my code as a baseline to further improve their work and results.

5.4.1 License

The code is licensed under the open-source Gnu Public License v3 (GPL3). It was chosen to make the project available to others, be further developed upon, but to also guarantee the reference of the original author. (Fossa, 2019)

There exist various open-source licenses, each with specific goals and guidelines for how the licensed assets should be used and distributed. The choice between the GPL3 and the Mozilla Public License 2 (MPL2) came down to the fact that MPL2 is a weak copyleft license and GPL3 is a strong copyleft license. Using GPL3 means that the subsequent

code derived from my development also falls under the GPL3 license, this way the code and the project remain open source. (Choose a License, n.d.)

The addition of the GPL3 license to the code was done quite late in the development. This could raise issues regarding the code of the git commits that were added to the GitHub repository before the addition of the license.

5.5 Readability

The development and documentation of the code for this thesis were done with Anaconda Notebooks. The use of a notebook is ideal when working with computer and data science as they allow for an interactive environment to both collaborate, share, and visualize data. Anaconda notebooks additionally utilize a virtual environment, and pre-installed packages and libraries to ease the development in areas such as data, and computer science. All notebook software utilizes cells, they hold either text or code. When using a notebook, we run these cells one after another. The developed notebook utilizes plenty of text cells to enhance the documentation and readability of the code. This is also done to complement the written thesis to create a unified and informative solution.

5.6 Shareability

Shareability and readability are important parts of both the code and this thesis. The code of this project has been developed in such a manner to make it as readable and understandable as possible, as well as supporting shareability.

A key part of this thesis was the aspect of shareability and further development by others. The code and this thesis are meant to function as a base for further development, and hopefully inspire others to develop new ideas. The code was from the beginning developed in such a way as for others to easily read and grasp. The use of comments, descriptions and the joint functionality of this thesis creates a unified understandable solution. The idea is to publish this thesis and code to several forums and platforms that hold like-minded individuals in areas such as AI, ML and DL. The choice of open-source code will aid in the aspects of shareability.

5.7 The Thesis

This thesis and the subsequent development process have been a rewarding and interesting experience. A way for me to further explore relevant subjects in AI, ML and DL. When I reflect on the choice of this thesis and its subject, I can honestly say that a thesis subject by the request of a company or individual would have been a more optimal choice. I am proud of the developed code and this thesis, but the workload and the need for self-discipline have been tough.

5.8 Future Goals

One way of predicting dog breeds from images is to export the developed model and use the predict method or any other similar method to predict images from the code editor, like the predict method I used during the development and testing. This is however a tiresome task and can be improved. A future goal of mine would be to create an open-source mobile app that incorporates my open-source developed model to identify dog breeds easily and accurately. It would also be a goal to further extend the developed model to train it on more images, add more breeds, and even try to train the model from scratch.

6 References

- Alpaydin, E. (2010). *Introduction to Machine Learning* (2nd ed.). MIT Press.
<https://mitpress.mit.edu/9780262012119/introduction-to-machine-learning/>
- Angyalfold, T. (n.d.). *Fastai ImgClass - Dog Breeds - Step by Step*.
<https://www.kaggle.com/code/angyalfold/fastai-imgclass-dog-breeds-step-by-step>
- BCC Research. (2022). *Machine Learning: Global Markets to 2026*. BCC Publishing.
- Brownlee, J. (2017). *A Gentle Introduction to Object Recognition With Deep Learning*.
<https://machinelearningmastery.com/object-recognition-with-deep-learning/>
- Chollet, F. (2021). *Deep Learning with Python* (2nd ed.). Manning Publications.
- Choose a License. (n.d.). *Licenses*. <https://choosealicense.com/licenses/>
- Deepak, S & P.M, Ameer. (2019). Brain tumor classification using deep CNN features via transfer learning. *Computers in Biology and Medicine*, 111(C).
<https://doi.org/10.1016/j.compbiomed.2019.103345>
- Fastai, (n.d.). *Welcome to fastai*. GitHub. <https://github.com/fastai/fastai>
- Fossa. (2021). *Open Source Software Licenses 101: GPL v3*.
<https://fossa.com/blog/open-source-software-licenses-101-gpl-v3/>
- Google. (n.d.). *Choose the Colab plan that's right for you*.
<https://colab.research.google.com/signup>
- Google Trends. (n.d.) *Interest over time*.
<https://trends.google.com/trends/explore?date=today%205-y&q=ResNet,VGG,EfficientNet,Convnext,AlexNet&hl=en-GB>
- Ho, J., Hussain, S., & Sparagano, O. (2021). Did the COVID-19 Pandemic Spark a Public Interest in Pet Adoption?. *Frontiers in Veterinary Science*(8).
<https://doi.org/10.3389/fvets.2021.647308>
- Howard, J., & Gugger, S. (2020). *Deep Learning for Coders with Fastai and PyTorch: AI Applications Without a PhD 1st Edition (1st)*. O'Reilly Media.
- IBM. (n.d.-a). *Structured vs. Unstructured Data: What's the Difference?*.
<https://www.ibm.com/cloud/blog/structured-vs-unstructured-data>
- IBM. (n.d.-b). *What are neural networks?*. <https://www.ibm.com/topics/neural-networks>
- IBM. (n.d.-c). *What is data labeling?*. <https://www.ibm.com/topics/data-labeling>
- IBM. (n.d.-d). *What is Computer Vision?*. <https://www.ibm.com/topics/computer-vision>

- ImageNet. (2021). *ImageNet*. <https://www.image-net.org/>
- Kaggle. (n.d.) *Dog Breed Identification*. <https://www.kaggle.com/competitions/dog-breed-identification/code?competitionId=7327&searchQuery=fastai>
- Khosla, A., Jayadevaprakash, N., Yao, B., & Fei-Fei, L. (n.d.). *Stanford Dogs Dataset* [Data set]. <http://vision.stanford.edu/aditya86/ImageNetDogs/>
- Kingma, D. P., & Ba, J. L. (2015). Adam: A Method for Stochastic Optimization, *Published as a conference paper at ICLR 2015*
<https://arxiv.org/pdf/1412.6980.pdf>
- Krullmizter, (n.d.). *Dog Breed Identification built with Fast.ai's CNN using transfer learning* [Source code]. GitHub. <https://github.com/krullmizter/dog-breed-id-fastai/>
- Langenbach, S. (2019). *Dog Breed Test with Fastai*.
<https://www.kaggle.com/code/stefanbuenten/dog-breed-test-with-fastai>
- Liu, Z., Mao, H., Wu, C.-Y., Feichtenhofer, C., Darrell, T., & Xie, S. (2022). *A ConvNet for the 2020s*. IEEE <https://arxiv.org/pdf/2201.03545.pdf>
- Lzubaidi, L., Zhang, J., Humaidi, A. J., Al-Dujaili, A., Duan, Y., Al-Shamma, O., Santamaría, J., Fadhel, M. A., Al-Amidie, M., & Farhan, L. (2021). Review of deep learning: concepts, CNN architectures, challenges, applications, future directions. *Journal of Big Data* 8(53). <https://doi.org/10.1186/s40537-021-00444-8>
- Marr, B. (2019). *Artificial Intelligence in Practice: How 50 Successful Companies Used AI and Machine Learning to Solve Problems*. Wiley.
- McCarthy, J. (2007). *What is artificial intelligence?*. <https://www-formal.stanford.edu/jmc/whatisai.pdf>
- Microsoft. (2023). *What is a machine learning model?*. <https://learn.microsoft.com/en-us/windows/ai/windows-ml/what-is-a-machine-learning-model>
- Nvidia Developer. (n.d.-a). *Deep learning frameworks*.
<https://developer.nvidia.com/deep-learning-frameworks>
- Nvidia Developer. (n.d.-b). *Convolutional Neural Network (CNN)*.
<https://developer.nvidia.com/discover/convolutional-neural-network>
- Openmetal. (n.d.). *TPU vs GPU: Pros and Cons*. <https://openmetal.io/docs/product-guides/private-cloud/tpu-vs-gpu-pros-and-cons/>

- Paperspace. (2020-a). Accuracy and loss. <https://machine-learning.paperspace.com/wiki/accuracy-and-loss>
- Paperspace. (2020-b). Data Science vs Machine Learning vs Deep Learning. <https://machine-learning.paperspace.com/wiki/data-science-vs-machine-learning-vs-deep-learning>
- Paperspace. (2020-c). Epochs, Batch Size, & Iterations. <https://machine-learning.paperspace.com/wiki/epoch>
- Paperspace. (2020-d). Overfitting vs Underfitting. <https://machine-learning.paperspace.com/wiki/overfitting-vs-underfitting>
- Raschka, S., & Mirjalili, V. (2019). *Python Machine Learning* (3rd ed.). Packt Publishing.
- Red Hat. (2019). *What is open source?*. <https://www.redhat.com/en/topics/open-source/what-is-open-source>
- Rendyk. (2021). *Tuning the Hyperparameters and Layers of Neural Network: Deep Learning*. <https://www.analyticsvidhya.com/blog/2021/05/tuning-the-hyperparameters-and-layers-of-neural-network-deep-learning/>
- Rosebrock, A. (2018). *Deep Learning for Computer Vision*. PyImageSearch.
- Swapna, K. E. (n.d.). *Convolution Neural Network*. <https://developersbreach.com/convolution-neural-network-deep-learning/>
- Thompson, N. C., Greenewald, K., Lee, K., & Manso, G. F. *A Cognitive Framework for Analyzing Innovation Ecosystems*. MIT Innovation Initiative. <https://ide.mit.edu/wp-content/uploads/2020/09/RBN.Thompson.pdf>
- Warner, B. (2021). *Inference with Fastai*. <https://benjaminwarner.dev/2021/10/01/inference-with-fastai/>
- Weights & Biases (2022). *fast.ai's Jeremy Howard on Why Python is not the future of machine learning - Gradient Dissent Clip*. [Video]. YouTube. <https://youtu.be/4I1ejhQqD4c?t=33>
- Willjobs (n.d.). *Dog Classifier*. <https://github.com/willjobs/dog-classifier>
- Wolfewicz, A. (2023). *Deep Learning vs. Machine Learning – What's The Difference?*. <https://levity.ai/blog/difference-machine-learning-deep-learning>
- Yani, M., Irawan, B., & Setiningsih, C. (2019). Application of transfer learning using convolutional neural network method for early detection of Terry's Nail. *Journal*

of Physics: Conference Series, 1201. doi:10.1088/1742-6596/1201/1/012052.

<https://iopscience.iop.org/article/10.1088/1742-6596/1201/1/012052>

Yosinski, J., Clune, J., Bengio, Y., & Lipson, H. (2014). How transferable are features in deep neural networks?. *NIPS'14*, 2, 3320–3328.

<https://arxiv.org/pdf/1411.1792.pdf>

Zafra, G. K. (2012). *Design and development of a FROG instrument for the characterization of femto-second laser pulses*. [Master's thesis, Universidad Nacional Autónoma de México] DOI:10.13140/RG.2.1.2957.9762

7 Appendices

7.1 Appendix 1: Summary in Swedish

Mitt examensarbete handlar om djupinlärning med fokus på datorseende, bildklassificering och faltningsnätverk. Arbetet har sin grund i att när jag och min sambo skaffade vår första hund så laddade jag ner diverse mobilapplikationer för att lätt och enkelt kunna fotografera och se vad för olika hundraser vi träffade på i hundparken och när vi var ute på promenad. Jag tyckte att de appar jag laddade ner var dåliga, gav felaktiga resultat, inga av dem hade öppen källkod, och applikationerna innehöll mycket annonser. Jag tyckte att det behövde finnas ett alternativ till att lätt och enkelt kunna identifiera hundraser från ett digitalt fotografi. Då fick jag idén att utveckla ett eget program som kunde identifiera hundraser på ett lätt och träffsäkert sätt. Jag sammankopplade mina erfarenheter från mina studier, tillsammans med mitt intresse för djupinlärning, programutveckling, och hundar.

Mitt arbete kan ses som uppdelat i två delar, en del berättar om den tekniska bakgrunden till just faltningsnätverk, djupinlärning och datorseende, den andra delen beskriver utvecklingen av min modell och analysen av dess resultat.

Datorseende är en form av datorvetenskap som fokuserar på datorers egenskap att använda sig av visuella data, antingen för att analysera, vidareutveckla eller i mitt fall klassificera. Bildklassificering är ett ämne som syftar på att kunna klassificera vad en bild innehåller för objekt.

Artificiell intelligens är huvudkategorin som både maskininlärning och djupinlärning faller under. Artificiell intelligens är huvudämnet som sammansluter alla underliggande kategorier av datorvetenskap kopplat till datorers förmåga att kognitivt fungera samt att ta beslut och lära sig som den mänskliga hjärnan. Artificiell intelligens är ett modernt ämne som oftast används för att syfta på alla de underliggande kategorierna som diverse teknologier kopplat till mänsklig intelligens. Artificiell intelligens används i spel, sociala nätverk, appar, självkörande bilar med mera. Maskininlärning är en underkategori till artificiell intelligens som syftar på förmågan av en dator att kunna

lösa problem med så lite mänsklig inblandning som möjligt. Dessa problem är oftast problem kopplat till att lära mönster i olika data. Mitt arbete faller under djupinlärning, som är en underkategori till maskininlärning. Djupinlärning fokuserar på att arbeta med så kallade neurala nätverk. Dessa nätverk replikerar den mänskliga hjärnan. Neurala nätverk kan arbeta med mycket data på en gång, och kan hitta mönster i flera olika datatyper, så som bilder, ljud, text med mera. Detta gör djupinlärning och neurala nätverk en bra lösning för bildklassificering. Faltningssnätverk är en version av neurala nätverk som fungerar utmärkt för att hitta mönster i just visuella data så som bilder.

Faltningssnätverk fungerar utmärkt till problem som har med bildklassificering att göra. Faltningssnätverk är en typ av neurala nätverk, men fungerar djupare än enkla neurala nätverk. Vi sammankopplar diverse lager av noder som har som uppgift att hitta mönster i bilderna vi för in. Dessa mönster sammankopplas för att kunna identifiera, eller klassificera vad en bild representerar eller innehåller. Den första lagren av ett faltningssnätverk lär sig den mest grundläggande kännetecknen så som kanter, texturer, och former. Dessa kännetecken byggs vidare på desto djupare, eller längre, vi förflyttar oss framåt i nätverket. Till sist har vi hela objekt så som, hundar, kläder, personer med mera som vi kan urskilja.

En bild byggs upp av pixlar, dessa pixlar har värden som syftar på färgvärdet i en pixel, pixel värdet kan vara mellan 0 – 255, värdet är styrkan av färgen i pixeln. Vi kan ha svartvita bilder då bara en färgkanal, eller RGB bilder som syftar på färglagda bilder som då har tre färgkanaler med värden mellan 0 – 255, alltså tre kanaler, för varje bas färg: röd, grön och blå.

Den grundläggande teknologin som särskiljer faltningssnätverk gentemot normala neurala nätverk är faltningsslagren. Faltningsslagren består av så kallade faltningsskärnor, eller filter. Dessa kärnor är matriser med siffror, siffrorna kallas för vikter, som vi för över bilden som vi vill klassificera. Vi multiplicerar ihop pixelvärdena, med matrisens tal, och sedan summerar vi dessa värden och det siffervärdet vi får applicerar vi på ett nytt lager, som heter särdragskarta. Genom att justera matrisens värden, vikterna, så kan vi hitta olika mönster, eller kännetecken i bilderna. Vi kan då välja att föra särdragskartorna vidare i nätverket för att hitta mera avancerande kännetecken i bilden,

eller så använder vi oss av ett aktiverings lager, eller ett pooling lager. Ett aktiverings lager är ett lager, eller egentligen en matematisk formel som skiftar särdragskartans värden mot antingen 0 eller 1. Vi har också diverse pooling lager som minskar på storleken av bilden som vi skickar vidare i nätverket, detta gör vi för att minska på bilden, och i så fall ökar prestandan av nätverket. Det sista lagret i ett faltningsnätverk är ett vanligt neuronnät. Här så använder vi oss av ett vanligt neuralt nätverk för att försöka klassificera bilden vi förde in i nätverket på basen av de mönster som hittats. Det vi vill få ut av det sista neuronnätet är en klassificering, och en probabilitet av hur säker den klassificeringen är.

Då vi tränar en faltningsnätverk att lära sig kännetecknen så behöver vi mata in massvis med bilder till den för att den skall lära sig att urskilja olika objekt, eller klasser. Då vi har skickat en bild framåt i nätverket och kommit fram till en klassificering så mäter vi hur bra modellen var att klassificera bilden korrekt. Detta görs genom att se vilken klassificering som var bäst i neuronnätet, och jämför den klassificeringen gentemot den äkta klassificeringen av bilden som fördes in. Sen flyttar vi oss bakåt i nätverket och uppdaterar vikterna och diverse andra parametrar för att få en bättre klassificering. Detta gör genom att använda oss av diverse matematiska formler för att mäta nätverkets felaktighet och uppdaterar då nätverkets parametrar för att nästa gång klara av att göra en bättre klassificering. Dessa parametrar är vikterna i kärnorna som används för att hitta mönster.

Datasetet jag valde att träna min modell med är Stanford dogs datasetet. Det är ett dataset fyllt med bilder av hundar. Datasetet innehåller ca. 20 500st bilder av olika hundar, och raserna för varje hundras finns lagrade i en .csv fil. På detta sätt kan vi föra in bilderna under träningen, och modellen kan göra sin klassificering, sedan mäter vi hur bra klassificeringen blev gentemot det faktiska värdet av hundbilden. Stanford datasetet är ganska litet, gentemot likartade dataset som används inom faltningsnätverk och djupinlärning. Mina resultat visade att man kan skapa en träffsäkermodell utan enormt stora tränings dataset.

Detta arbete tar också upp utvecklingen av ett faltningsnätverk med hjälp av överföringsinlärning. Detta betyder att vi bygger ett faltningsnätverk på basen av en

färdig tränad modell. Dessa färdig tränade modeller har tränats att hitta diverse allmänna former, objekt, mönster med mera. Detta gör vi för att öka träffsäkerheten av vår modell, minska på träningstiden, samt att teoretiskt uppnå bättre resultat.

Fast.ai biblioteket valdes p.g.a. dess popularitet, och förmåga att lätt kunna arbeta med bildklassificering. Diverse andra bibliotek och program har ganska svår syntax, eller kod för att utveckla djupinlärnings modeller. Jag anser att Fast.ai är ett lättare alternativ då det kommer till utvecklingen och dess syntax. Utveckling av en modell med Fast.ai visade sig vara mycket givande. Fast.ai bygger på PyTorch ramverket, man kan därför helt och hållet använda sig av PyTorch metoder, funktioner och kod. Fast.ai sitter ovanpå PyTorch, och kan utnyttja PyTorch med en lättanvänd API. Python är också det programmeringsspråk som används för utvecklingen.

All kod har skrivits med ett så kallat jupyter notebook programvara. Det är en form av utveckling lämpat för datorvetenskap. En kombination av diverse celler spjälkar upp koden i ett enstaka dokument med både text och kodceller för att göra det lätt tillgängligt. Anaconda notebooks är tjänsten jag valde att utveckla med. Anaconda använder sig av Python, och diverse färdiginstallerade programmeringsbibliotek och verktyg för att göra utvecklingen lätt då det kommer till datorvetenskap. Min utveckling berörde inte bara utvecklingen av bildklassificeringsmodellen, utan också diverse andra metoder och uträkningar som visar hur bra modellen presterar, hur data är strukturerad och andra visualiseringar för att göra det lättare att förstå och greppa min kod.

Mina mål var att utveckla en så träffsäker modell som möjligt, detta uppnåddes med min slutgiltiga modell som hade en allmän träffsäkerhet på ca. 96%. Min hypotes i början var att ha en modell med ca. 90% träffsäkerhet. Den allmänna träffsäkerheten kalkylerades genom att mäta prestandan av en färdigtränad modell mot en del av hela hund datasetet. Jag testade min modell mot diverse appar och modeller som identifierar hundraser, samt som liknar min utvecklade modell och jag fick där resultat som var likartade och på flera ställen bättre än de modeller och appar jag testade mot. Jag valde att testa min modells träffsäkerhet genom att ge den två bilder, en på min hund Laban, som är en Labrador retriever, och en bild på en Berner sennenhund. När jag testade min modell på dessa bilder så klassificerade min modell bilderna rätt, med en träffsäkerhet

på ~94% och ~99%. De olika apparna jag testade emot fick en allmän träffsäkerhet på ca. 90% på Laban, och på bilden av Berner sennenhund 99%. På detta sätt så fick jag bevisat att min modell, som använder sig av ett relativt litet dataset av hundbilder kunde effektivt och träffsäkert klassificera hundraser lika bra som publicerade applikationer. Diverse andra modeller testade jag, och min modell fungerade lika bra, och i många fall bättre än de andra modellerna.

Öppen källkod var ett mål med mitt arbete. All kod jag utvecklat finns offentligt publicerat på tjänsten GitHub. GitHub är en av många tjänster som tillhandahåller lagring av kod för både privat och offentligt bruk. Man kan också se revisioner av koden sen den första uppladdningen. Då jag laddade ner diverse mobilapplikationer för att identifiera hundraser så hittade jag inte några populära som föll under öppen källkod. Jag ville skapa ett alternativ för hundrasidentifiering med öppen källkod. Jag ville också att andra kunde ta del av mitt arbete, modifiera det och arbeta vidare med det. Jag licensierade hela mitt arbete under GPL3, som kräver att all diverse kod som utvecklas på basen av min kod faller under samma licens, alltså så stannar alltid min kod under öppen källkod.

Läslighet och delbarhet var två viktiga mål då jag valde att påbörja skriv- och utvecklingsprocessen. Jag fokuserade på att utveckla min kod och skriva mitt examensarbete på ett så läsligt sätt som möjligt. Jag ville skapa en enhetlig lösning som personer kunde använda sig av för att lätt förstå sig på bildklassificering, hundrasidentifiering, faltningsnätverk och Fast.ai. Den slutgiltiga koden var en fungerande helhet tillsammans med det skrivna examensarbetet. Det var viktigt för mig att koden lätt kunde appliceras på nya ändamål, personer skall lätt kunna byta ut Stanford datasetet mot något annat dataset, men kunna använda min kod för att göra samma bildklassificeringar.

Slutligen så vill jag reflektera över mitt examensarbete som en helhet. Det har varit en intressant upplevelse, och jag har lärt mig mycket. Det har varit intressant att få arbeta med de ämnen och saker vi lärt oss under studierna vid Arcada. Arbetet har också väckt ett vidare intresse för bildklassificering och djupinlärning hos mig, som jag vill vidareutveckla. Jag har som mål att vidareutveckla min modell genom att träna

modellen med flera bilder, flera raser och få lika bra eller bättre resultat. Jag vill också utveckla en mobilapplikation med öppen källkod för att sammankoppla min utvecklade modell och publicera den så att andra kan ta del av en träffsäker och effektiv öppen källkodslösning för att identifiera hundraser med en mobilapplikation.