Ways to use Machine Learning approaches for software development

Nicklas Jonsson
Abstract

With the rise of machine learning and in particular deep learning entering all different types of fields, including software development. It could be a bit hard to know where to begin to search for the tools when someone wants to use machine learning for a one’s problems. This thesis has looked at some available technologies of today for applying machine learning to one’s applications.

This thesis has looked at some of the available cloud services, frameworks, and libraries for machine learning and it presents three different implementation structures that can be used with these technologies for the problem of image classification.
Acknowledgements

I want to thank C4 Contexture for giving me the thesis idea, support, and supplying me with a working station. I also want to thank Lantmännen for supplying me with the image data that was used for this thesis. Finally, I want to thank Eddie Wadbro for his guidance during this thesis and of course a big thanks to my family and friends for their support during this period of my life.
Content

1 Introduction
   1.1 The Client
      1.1.1 C4 Contexture PIM software
   1.2 The data
   1.3 Goal
   1.4 Limitation

2 Background
   2.1 Artificial intelligence, machine learning and deep learning.
      2.1.1 Artificial intelligence
      2.1.2 Machine learning
      2.1.3 Learning representations from data
      2.1.4 Deep learning
   2.2 Cloud services for Machine learning
      2.2.1 IBM Watson
   2.3 Frameworks and libraries for machine learning
   2.4 Convolutional neural networks
      2.4.1 The problem space
      2.4.1 What is an image to a CNN?
      2.4.2 Structure
      2.4.2 The convolution operation
      2.4.2 The max-pooling operation
   2.5 Using a pretrained CNN

3 Methods
   3.1 Cleaning up the data
   3.2 IBM Watson Visual recognition
      3.2.1 Create custom classifiers on Watson
   3.3 Using frameworks and libraries for machine learning
      3.3.1 Tensorflow
      3.3.2 Keras
      3.3.3 Tensorflow.js
      3.3.4 Models
      3.3.5 Retrain the models
   3.4 Prepare the imagedata for the training phase
   3.5 As applications

4 Results
   4.1 Watson Visual Recognition
   4.2 TensorFlow with Inception V3
   4.3. Keras with MobileNet
4.4 Converting the MobileNet Keras model into TensorFlow.js format 30
4.5 As applications 31
   4.5.1 Swing application using Watson 31
   4.5.2 Web application using Inception V3 32
   4.5.3 MobileNet and TensorFlow.js 33

5 Discussion 35
   5.1 Services 35
   5.2 Frameworks and libraries 35
   5.3 The applications 36
      5.3.1 Watson as service 36
      5.3.2 Own service for hosting models 36
      5.3.3 Loading models into the browser. 36

6 Conclusion 39
   6.1 Future work 39

References 41
1 Introduction

"The consequences of this sudden progress extend to almost every industry. But in order to begin deploying deep-learning technology to every problem that it could solve, we need to make it accessible to as many people as possible, including non-experts, people who aren’t researchers or graduate students. For deep learning to reach its full potential, we need to radically democratize it [8].”

- Francois Chollet, the creator of the Keras deep-learning library, about deep learning.

Artificial intelligence and in particular its subfields machine learning and deep learning are changing all types of business functions, and software development is not left unaffected by this [1]. Traditional programming is mostly combined with logic operations that will tell the program what to do. We used to be tough that computers are dumb [3], meaning they does only do exactly what they are told to do, nothing more and nothing less. But in the last couple of years, this concept has been challenged [4].

AI for businesses is today mostly made up of different type of machine learning algorithms. These algorithms are compiled in order to teach the systems to learn something from some type of data that it has been handed. This simplifies, scales and also introduces solutions for problems that previously was considered impossible to solve when just using the traditional programming approach [2].

This thesis explores how machine learning approaches can be used in software development. This thesis is gonna focus and present solutions in the machine learning field of vision recognition and image tagging, but also briefly explain where software developers can find machine learning solutions for other functionalities. The focus of this work has been to find some of the “easier” ways to implement machine learning solutions for developers that are non-experts within the field of machine learning. This report will present solutions that takes almost zero knowledge of machine learning, but also some solutions that will require some more learning within the field for a developer to be able to use them in a sufficient way.

1.1 The Client

This master thesis was made with the help of C4 Contexture. C4 Contexture is a company that expertise themselves in information logistics [5]. Their main product, also called C4 Contexture, Is a Product Information Management (PIM) software.
1.1.1 C4 Contexture PIM software

C4 Contexture PIM makes it possible for users to store all product related information. PIM is a system that stores, structure, and distributes the user's information. It makes the information searchable, easy to manage and accessible from any location. The user controls the data flow and decides who sees what [6].

PIM makes it possible to assemble:

- images, videos, and text
- templates, drawings, and documents
- product details (name, price, and supply status)
- files like PDF, InDesign, Excel, and many others.

![Figure 1.1: C4 Contexture PIM system overview. PIM supports publication of information on multiple platforms like the web, mobile devices, printed media, and more. Figure supplied by C4 Contexture [7].](image)

1.2 The data

The image data used for this thesis is supplied by Lantmänn. Lantmänn is an agricultural cooperative and they are northern Europe's leading player in agriculture, machinery, bioenergy, and food [22].

The data are images of different types of food products. The images mostly depict different kinds of bread and pastries, with both singular products on
them, but also products that are mixed together, and that are in environments (environment example: second image to the left in figure 1.2 below).

![Image](image.jpg)

*Figure 1.2: examples of some of the image data being used.*

1.3 Goal

The goal of this thesis is to present available techniques and methods that can be used to simplify machine learning approaches for the average software developer. The thesis will also show that it is possible to make different types of implementations with machine learning.

In the case of C4 Contexture, they want to implement a solution in their PIM system that automatically tags the image data that they handle for their business customers so that the tags of the images will become more continuously and structured.

1.4 Limitation

Even though the machine learning services and frameworks used in this thesis supports many different kinds of fields, this thesis is mostly gonna focus and present solutions for the field of image tagging. The reason for this is that this is the functionality that C4 Contexture wanted to have implemented for their PIM system.
2 Background

This chapter will explain the definitions of artificial intelligence, machine learning, deep learning. And it will also talk about services and frameworks that can be used for machine learning. Also, some extra explanation on the neural network called: Convolutional Neural Network (CNN) will be made since this is the main machine learning model that was used in this thesis.

2.1 Artificial intelligence, machine learning and deep learning.

So what is the difference between these “buzzwords” that there is so much talk about? How do they relate to one another? This subchapter will explain more about that.

![Figure 2.1: the relation between artificial intelligence, machine learning, and deep learning.](image)

2.1.1 Artificial intelligence

The concept of artificial intelligence (AI) is old. It originates all the way back to the 1950s. Back then, a handful of pioneers from the early field of computer science asked themselves: “can computers think?” This is a question that we still exploring today. In short, the field can be explained as follows: the effort to automate intellectual tasks normally performed by humans. As such, AI is a very general field that also includes machine learning and deep learning, but it also includes many more approaches than just learning approaches. The early approaches from the 1950s to the late 1980s were known as symbolic AI. The main example of these are the early chess programs that followed a set of hardcoded rules created by programmers and didn't qualify as machine learning since they never learned to play, they were just programed to “know” how to play chess. For fairly long this approach was believed to be the answer of reaching human-level AI. However, this way was not good to figure out explicit rules for solving more complex, fuzzy problems, such as speech- and image recognition. Hence this is where the approach of machine learning enters the picture [8].
2.1.2 Machine learning

Also back in the 1950s, Alan Turing realized his now famous paper “Computing Machinery and Intelligence [10]” which is, were the Turing test was introduced, as well as key concepts that would come to shape AI. Turing came to the conclusion that general-purpose computers could be capable of learning. This is where machine learning (ML) arises from. The question of: if computers can go beyond what we humans are capable of ordering them to perform and to learn how to perform a specific task on its own?

So instead of programmers crafting data-processing rules by hand, could it be possible to switch that approach up a bit and make the computer learn these rules by looking at the data? This is the question that made the way of a new programming paradigm. In classical programming (and also in symbolic AI), a programmer writes the rules for the software and the data will be processed according to those rules, and out come the answers. In machine learning, however, the user inputs the data, as well as the expected answer and out, comes the rules (see figure 2.2 below). These rules can then later be used on new similar data to produce new answers.

![Classical Programming vs Machine Learning Paradigm](figure2.2.jpg)

A machine learning system is trained rather than explicitly programmed. The machine learning solution used will try to find a statistical structure in the data that is being given, and from there it will come up with the rules for automating the task. Recently machine learning has rapidly become the most popular and successful subfield of AI. This success is made possible because of the availability of faster and stronger hardware, but also because of the now available amount of data that exists today [8].

2.1.3 Learning representations from data

To understand the difference between deep learning and other machine learning approaches, the idea behind what machine learning algorithms do has to be understood. As stated in previous subchapter 2.1.2, machine learning creates rules from a structured dataset that it is being trained on and then executes those rules on new similar data. The following three things are needed for machine learning:
• **Input data points:**
  ○ For image tagging: pictures (jpeg, png, etc).
  ○ For speech recognition: sound files (mp3, m4b, etc)

• **Examples of the expected output:**
  ○ For image tagging: tags like “donut”, “muffin”.
  ○ For speech recognition: human generated transcripts of sound files, for example: “heartbeats” or “bird sounds”.

• **Some type of way to measure the algorithms job performance:**
  ○ This is needed to measure the distance between the algorithms current output and the given expected output. The algorithm uses this measurement as a feedback signal to adjust the way it works. The adjustment is what is called *learning*.

A machine learning model transforms the input data into some meaningful outputs. This is a process that is being learned from the known inputs and outputs example data. Hence the main problem in machine learning and deep learning a lot of the time is to get a hold of datasets and to structure it into meaningful representations. Representations that will take the model closer to the expected output.

So finally what is representation? At its core, representation is a different way to look at data, to represent or encode data. For example, a color image can be encoded in the RGB (Red, Green, and Blue) format or in the HSV (hue, saturation, and value) format: these are two different representations of the same data. However, some tasks can be more suited to RGB, and other tasks with HSV. The point is that machine learning models are all about finding the appropriate representations for their input data [8].

### 2.1.4 Deep learning

Deep learning is a specific subfield of machine learning. This is quite a new approach when it comes to learning representations. Deep learning takes the approach of learning with the help of layers, that gets increasingly meaningful representations. The *deep* in deep learning is not referred to that deep learning models gets some kind of “deeper understanding” from the approach. What it stands for is the idea of successive layers of representations. How many layers the model contains is what's called the depth of the model. Some other appropriate name of the field could have been: layered representation learning and hierarchical representation learning. Today's deep learning models often involve tens or even hundreds of successive layers of representations and all of them learns automatically from the training data. So, normal machine learning approaches only learn on one or two layers of representations of the data, they are also sometimes called shallow learning. In deep learning, these layered representations are almost always learned via models called neural networks.
Neural network is a term that is coined from neurobiology. The main concept of deep learning was inspired by the brain (not a direct model of the brain). There is a big misunderstanding in the mainstream that neural networks are directly modeled after the human brain, but there is no such evidence that the brain implements anything like the learning mechanisms used in today's deep learning models. This is a confusing and counterproductive way for newcomers to the field of deep learning to believe. You don’t need any knowledge of neurobiology to do deep learning. The right way to see it is that deep learning is a mathematical framework for learning representations from data.

So in short, deep learning is technically a multistage way to learn data representations by mostly using neural networks. A simple idea, but when sufficiently scaled, they can end up looking like magic [8].

2.2 Cloud services for Machine learning

As of today Google, IBM, Amazon, and Microsoft are the biggest players in the fast-growing market of machine learning cloud services [11]. This thesis has taken a closer look at the machine learning services that are provided by IBM.

IBM provides an API for each of their different services. Making it so that the user doesn't have to write any complicated machine learning code on their own, hence they can focus more on their training data.

2.2.1 IBM Watson

Watson is the famous IBM supercomputer that won against the previous human champions of Jeopardy back in 2011 [13]. Today Watson powers the latest innovations in machine learning, and it has the ability to learn from quite the small amount of data. With Watson, a user can build their models from scratch if they want to, but what is even more convenient is to use their machine learning services that have been designed, built and trained with the help of machine learning experts that works for IBM. These services can be used by their corresponding APIs [11][12].

As of now (mars 2018), the available machine learning services that Watsons offers are [14]:

- **Vision** - the user can tag and classify visual content.
- **AI Assistant** - the user can build an artificial assistant for mobile, messaging platforms and robots.
- **Speech** - convert text and speech.
  - **Speech to text** - audio and voice into written text.
○ **Text to speech** - written text into audio.

- **Language** - analyze text and extract metadata from it.
  - **Language translator** - translate text from one language to another.
  - **Natural Language Classifier** - Interpret and classify natural language with confidence.

- **Empathy** - Understand the tone, personality, and emotional state.
  - **Personality insights** - predict personality characteristics through text.
  - **Tone analyzer** - understand emotions and communication style in a text.

### 2.3 Frameworks and libraries for machine learning

With the rise of machine learning in the last couple of years, plenty of frameworks and libraries that can be used for machine learning has been developed for the public to use. This thesis has looked at some popular ones called, TensorFlow, Keras, and TensorFlow.js.

### 2.4 Convolutional neural networks

Since this is the main neural network model used for the image classification part of this thesis, some more in depth explanation is made for it.

Neural networks started to grow in popularity when one was used to win ImageNet competition (basically the world cup of computer vision) in 2012[8]. It dropped the current error rate record from 26% to 15%, which were a significant improvement at the time. Ever since then, many different companies has been using deep learning at the core of their services in areas like product recommendation, search infrastructure, automatic tagging, and many more. So as of now (2018) the most popular neural network used within image processing for image classification is the Convolutional Neural Network (CNN) [9].

#### 2.4.1 The problem space

Image classification is the task of an image as the input and trying to put a class tag on the content of that image as the output, or mostly, put a probability score of multiple classes that best describes the image. This is, in most cases, an easy task for humans. One of the first skill we ever learn, and as adults are able to do this quickly and effortlessly. These skills of being able to quickly recognize patterns, generalized from prior knowledge, is a skill that is not so easy for a computer to achieve.
2.4.1 What is an image to a CNN?

CNNs sees and process images as tensors, and tensors are just a fancy word for matrices of numbers with additional dimensions. Every image can be represented as an tensor of pixel values. A standard digital color image will have three 2D-matrices in depth \((width \times height \times depth)\) containing a value of 0-255 (see figure 2.3 below) and a grayscale image is only one 2D-matrix [8][9].

![Figure 2.3: an input image and its corresponding matrix/tensor that the computer sees.](image)

So given an image input, the CNN is able to perform image classification, by looking at low level features such as edges and curves, and then building up more abstract concepts through the convolutional layers [9].

2.4.2 Structure

So a more detailed overview would be that the CNN takes an image, pass it through a series of convolutional, nonlinear, pooling, and fully connected layers, and after that, gets the predicted classification output of a single class or a probability of multiple classes (see figure 2.4 below) [9].
2.4.2 The convolution operation

Convolution layers learn local patterns. For images, patterns are found in small windows of the inputs.

This concept gives CNN’s two key properties:

- **The patterns they learn are translation consistent.** After they have learned a specific pattern in the lower-right corner of an image, a CNN can recognize this pattern anywhere else on other images. This is what makes them so data efficient for processing images, since they need fewer training samples to learn representations that have generalization power.

- **They have the ability to learn spatial hierarchies of patterns.** The first convolution layer learns small local patterns such as edges, the second layer learns larger patterns that it gets from the previous layer, and so on. This allows CNN’s to efficiently learn increasingly complex and abstract visual concepts (since the visual world is fundamentally spatially hierarchical).

Convolutions operate over 3D tensors, also called feature maps. They have the structure: \((height, width, depth/channels)\). The convolution operation extracts patches from the input feature map and applies the same transformation to all of these patches, creating an 3D tensor called output feature map. The output feature map has the same type of structure, but since the output now is a parameter of the layer, the different channels in the depth axis no longer stand for specific colors like RGB input. They now stands for filters. Filters encodes specific aspects of the input data at a high level, one filter could for example encode if an image contains a cat or a face.
Convolutions are defined by two key parameters:

- **Size of the patches extracted from the inputs**: These are typically 3x3 or 5x5.
- **Depth of the output feature map**: the number of filters computed by the convolution.

A convolution works by sliding the 3x3 or 5x5 windows over the 3D input feature map, and from that extracting the 3D patch of surrounding features. Each of these patches is then, with the help of a tensor product with the same learned weight matrix called *convolution kernel*, transformed into a 1D vector of shape: \((\text{output depth}, )\). These vectors are then put together into a 3D output map of shape: \((\text{height}, \text{width}, \text{output depth})\). Every spatial location in the output feature map corresponds to the same location in the input feature map (top-left corner of the output corresponds to the top-left corner of the input). The depth of the output feature map is not the same as the depth of the input feature map. The difference of the depth can be because of:

- Border effects, which can be countered by padding the input feature map.
- The use of strides.

**The border effect and padding**

For example, a 5x5 feature map (total of 25 cells). There are only 9 cells around which can be in the center of a 3x3 window, forming a 3x3 grid (see *figure 2.6* below). Hence, the output feature map will be 3x3. It shrinks by two cells alongside each dimension, in this case.

![Figure 2.6: possible locations of 3x3 patches in a 5x5 input feature map.](image)

To achieve the same spatial dimensions on an output feature map as the input, one can use *padding*. Padding adds an appropriate amount of rows and columns on each side of the input feature map so that the center convolution...
windows can fit around every possible cell. For the example above, there would be one row and one column added for each corresponding side (see figure 2.7 below) [8].

Figure 2.7: padding a 5x5 input to make it possible to extract 25 3x3 patches.

Convolution strides
The other factor that can influence output size is the notion of *strides*. The stride value is how many cells the patches will move each time (see figure 2.8 below) [8].

Figure 2.8: 3x3 convolution patches with 2x2 strides.

Using stride 2 means the width and height of the feature map are gonna become half the size after the convolution operation. Strides with a value larger than 1 is rare in practice. To downsample feature maps, the more common way is to use the max-pooling operation.

2.4.2 The max-pooling operation
Max pooling consists of extraction windows from the input feature maps and outputting the max value of each channel. It is similar to convolution, however instead of transforming local patches via a learned linear transformation, they are transformed via a static max tensor operation. Max pooling is often done in
2x2 windows and uses a stride value of 2, in order to downsample the feature maps by a factor of 2.

So why do max pooling? The reason to use downsampling is to reduce the number of feature map coefficients to process, as well as to induce spatial-filters hierarchies by making successive convolutional layers look at increasingly large windows.

There are other ways to achieve downsampling. However max pooling tends to work the best [8].

![Figure 2.9: example of max pooling.](image)

2.5 Using a pretrained CNN

A common and effective approach for a smaller image dataset is to use a pretrained network. A pretrained network is a network that has its stage saved from a training phase that was done on a large dataset. If this original dataset is large and general enough, then the spatial hierarchy of those features learned by the pretrained CNN, can be used as an generic model of the visual world, and hence its features can prove useful for many different other images, even on classes that it was not originally trained on. As an example, a complex CNN can have been trained on ImageNet (1.4 million labels and 1000 different classes of mostly animals and everyday objects). ImageNet has many images of different food products, hence it would work fine for re-training that CNN to classify different types of bread products [8].

This is also called transfer learning. Transfer learning is a research area in machine learning that focuses on storing knowledge that has been gained during one training phase and to use that knowledge to apply on a different but related problem [25].
3 Methods

This chapter of the thesis will explain the different methods that were used to perform image tagging on Lantmännen’s data, and also how these methods will be implemented as applications.

3.1 Cleaning up the data

Since the data that was being used was in such a mess initially, the first step has to be to “clean” it. This included:

1. Looking through it to find and pattern of what possible tags Lantmännen would want to have for their images.
2. Sorting and grouping them into folders with their correlated class.
3. Find images that could “hurt” the learning phase for classes and remove them.
4. Find classes with too few images in them and remove them.

Step 3 above, is not a simple task. This step is about trying to identify if, given the data used, there could be one or more images that would do more harm than good to have included in the class. This process does not really have an easy “do this” kind of answer to it. The process is mostly a “gut feeling” process.

However, some guidelines have been used for the cleaning process:

- **Mind the background colors.** Example: ten images of an object, nine has a blue background, one has a red background. Hence, get rid of the red one.
- **Mind the amount of noise.** Example: ten images, nine is an image of just a single loaf, one is an image of a loaf sandwich with cheese and ham on it. Hence, get rid of the loaf sandwich image.

However, these guidelines can be “skipped” if there is enough variety of images in the class. For example, ten images of an object, four has a blue background, three has a red one and three has a green one. In this case, there is still a majority of blue backgrounds, however, it is not by a lot and there is also three different kinds of background, so the CNN should not focus on the background as the main pattern to be learned from the images, and hence focus more on the object in the image [24][25].
After following the guidelines and searching for patterns, seventeen classes were created (see figure 3.1 below) with a total of 1337 images distributed among them.

![Folders containing images with class names](image)

*Figure 3.1: the folders containing the corresponding images and has the corresponding class names.*

3.2 IBM Watson Visual recognition

IBM’s Watson has a complex CNN that has been trained to recognize a vast number of objects. This service can easily be used with a simple API call.

![Hamburger bun](image)

*Image 3.2: hamburger bun.*

However, the classifiers that Watson provides, doesn't always result in the right tags that the user might want. For example, *image 3.2* above returns the following top five tags:

- *Food* - score: 0.963
- *Sandwich* - score: 0.945
- *Dish* - score: 0.945
- *Nutrition* - score 0.945
- *Hamburger* - score: 0.706

So by just using the default classifier, the service returns a total of thirteen different tags that all have a score of greater than 50%. But none of these tags are suited for what Lantmännen wants since this is not a product of any of these tags in the eyes of Lantmännen. This is an image of one of their hamburger bun products. To solve this problem, one can create custom classifiers.
3.2.1 Create custom classifiers on Watson

A custom classifier is, just as the default classifier, a group of classes that are trained against each other. However, here, Watson allows the user to train their own classifiers with their own custom-made tags for a given data.

Before the user can post the data to Watson, they have to structure it as in figure 3.3 below. Which is that each corresponding class must be compressed into zip files. The names of the zip files will also become the names of the classes for the classifier.

![Figure 3.3: the structure of the training data for Watson.](image)

To create the custom classifier with all the custom classes the following Curl command could be used:

```bash
$ curl -X POST -u "apikey:{your_api_key}" -F "baguette_positive_example=@baguette.zip" -F "ciabatta_positive_example=@ciabatta.zip" -F "croissant_positive_example=@croissant.zip" -F ...
```

Where `{your_api_key}` is the API-key that was generated when the instance was created and "name=lantman" is the name of the classifier.

As can be seen for each class is the `_positive_example=@` syntax, which is an indication to Watson that these are positive examples for the classifier, meaning the class belongs to the classifier. The user can also provide negative examples if available. An example of this could be to add the string:

```bash
"negative_example=@cats.zip" -F
```

Which would train the classifier to know that cats (which would be a zip file containing images of cats) do not belong to this classifier. A user would want to train a classifier with negative examples if their classifier is sometimes...
exposed to these kinds of images. So in the example above, we could imagine the case that Lantmännens, for some reason, would have images of cats posted to their custom classifier, but since they don't sell cats, they would want the classifier to give the images of cats as low as a score as possible, hence they train the classifier to learn that cats are negative, which will result in images of cats getting a low score. But for this thesis, there were no negative data available to use, hence the classifier was only trained on positive examples.

Watson Visual Recognition has support for *Curl*, *Node.js*, *Java*, and *Python*. So it is well supported for creating a standard application, or making the API calls from the back-end, or front-end of a web application [23]. This thesis uses an own developed Java swing application to communicate with Watson.

### 3.3 Using frameworks and libraries for machine learning

This subchapter will present the machine learning frameworks and libraries that have been used for creating custom classifiers. It will also present the pre-trained models that have been used and method for how they were re-trained for Lantmännens’s data.

#### 3.3.1 Tensorflow

TensorFlow is an open source software library used for high-performance numerical computation. Originally developed by researchers and engineers from Google, this library contains strong support for machine learning and deep learning [20]. It allows the user to train their models on both the CPU, GPU.

#### 3.3.2 Keras

Keras is a deep-learning framework for python that provides high-level building blocks for developing almost any kind of deep-learning model in a much more convenient way than to build it all from scratch. Keras also allows the same code to be run on both the CPU and GPU.

Keras does however not handle low-level operations, such as tensor manipulation and differentiation. So what it does instead is to rely on well optimized tensor libraries (like TensorFlow) to handle that part, functioning as a backend engine of Keras. Currently (April 2018) Keras can use Theano, Microsoft Cognitive Toolkit, and TensorFlow as its backend. More will probably come in the future. As of now, the Keras team recommends using TensorFlow as its backend [8].
3.3.3 Tensorflow.js
TensorFlow.js is a newly released JavaScript library that makes it possible for developers to train and deploy machine learning models in the browser and on Node.js [21].

3.3.4 Models
These are the following pre-trained models that will be used:

**Inception V3** (see figure 3.4 below): a large CNN that was trained for the ImageNet Large Visual Recognition Challenge using the data from 2012. The model is trained on millions of images with an output of 1000 classes [26].

![Inception V3 architecture](image)

*Figure 3.4: the Inception V3 architecture.*

**MobileNet** - a smaller CNN that is also trained on ImageNet and has the same output classes as Inception V3. However, it is smaller and is mostly created to be more suited for usage on mobile devices and embedded vision applications [19].

3.3.5 Retrain the models
This is where transfer learning comes in handy. Modern image recognition models have millions of parameters and training them from scratch with a large amount of images can take hundreds of GPU-hours. Transfer learning is a shortcut to this. With this technique, all that has to be done is to take the last piece of a model that has been trained on a related task and reuse it in a new model. In transfer learning, the lower layers that have been trained to distinguish between some objects can be reused for many recognition tasks.
without any alteration. This will save a lot of time and compute power for the user and will also give surprisingly good results [25].

An example can be seen in figure 3.6 above. This figure shows how transfer learning are made on the Inception V3 model. What's happening is that the last two layers is removed and replaced by new untrained ones. These layers will then be trained using the previous weight of the model, to recognize and be able to classify the new images. The reason the retraining of the final layer works on the new classes is that the old information needed to distinguish between the previous 1000 classes is often useful to distinguish between new kinds of objects.

This thesis uses the Inception V3 model that is implemented using TensorFlow and hence the retraining is done with TensorFlow. The MobileNet model used however, is the one that is implemented with Keras, so it will be retrained with the help of the Keras API.

Both of these models (and many more) has their corresponding implementation in TensorFlow and Keras, so both retraining phases can be done in one or the other frameworks if one does not want to learn both.
3.4 Prepare the imagedata for the training phase

First, the data needs to be structured in the correct way, when training the pre-trained models, there has to be a root folder that contains all the classes, and the classes are folders with the corresponding class names. The images will also be split into \textit{train}, \textit{validation}, and \textit{test} folders (see \textit{figure 3.7} below), where:

- \textit{training} contains: 1084 images
- \textit{validation} contains: 170 images
- \textit{test} contains: 83 images

Training is the main data that the models will be trained on, but during the \textit{training} phase, once in while, the model will be \textit{validated} for ensuring that the training is going the right way, and to make it easier to spot overfitting. After the training, the model will be \textit{tested} with the test images.

\textbf{Figure 3.7: Structure of the data to be trained on.}

There is also the fact that both models only have one fixed input size for the images. The inception V3 model has an input size of 299x299 pixels and the MobileNet model has 224x224 pixels. The practical part of how the models work is that if an image is not the exact size, both of these models will resize the images automatically when they are being trained on. However, it is good practice to have the images around these sizes, for else they can be resized into
something that looks very different from the initial image. An example of this can be visualized in figure 3.8 below, where the hamburgers go from a circle shape to a more oval one, this can affect the training results.

Since the provided images were between 200-340 pixels in width and height, no other reshape than that of what the model does automatically was made.

Watson however, works a bit differently, it is more of a “black box”. Meaning that the user does not have much control over the training phase, all the user does is hand over the data to Watson. So all of the images within the training and validation folders will be merged together, so 1,254 images split into 17 different classes, and compressed into zip-files for training the custom classifier. Watson is more freely with the image size, IBM only defines a minimum of 32x32 pixels. However, they recommend having at least 224x224 pixels. Since this is the same as MobileNet, and the images are approximately this size, no resizing will be made for Watson either. After the training phase, the classifier will be tested against the same 83 test images as the pre-trained models.
3.5 As applications

For IBM Watson, a standard Java swing application will be implemented with the model-view-controller design pattern, where the communication with Watson will be the model part.

**Figure 3.9:** swing application using Watson as the service to send images to, and get back the prediction(s).

For TensorFlow, a simple Python web application, using the Python flask framework as the backend service to host the retrained Inception V3, was implemented.

**Figure 3.10:** flask service hosting the Inception V3 model for the web client send the images to, and get back the prediction(s).

For Keras and TensorFlow.js, a simple web-application was implemented, and Keras will be used to retrain the MobileNet model, which will then be converted with the help of TensorFlow.js so that the model can be loaded and used to perform predictions directly in the browser.

**Figure 3.11:** a JSON API containing the retrained MobileNet model, that has been converted into TensorFlow.js format, for the web client to fetch and load into the browser so it can perform prediction(s) within the browser.
4 Results

In table 4.1 below, the different results of the image classifiers are presented.

<table>
<thead>
<tr>
<th>Model:</th>
<th>Training time:</th>
<th>Average prediction Time for one image:</th>
<th>Accuracy from test set:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Watson Classifier</td>
<td>&lt;57 min</td>
<td>2355 ms</td>
<td>86.75%</td>
</tr>
<tr>
<td>Inception V3</td>
<td>~24 min</td>
<td>3788 ms</td>
<td>77.11%</td>
</tr>
<tr>
<td>MobileNet</td>
<td>~64 min</td>
<td>126 ms</td>
<td>62.65%</td>
</tr>
</tbody>
</table>

Table 4.1: The results for each classifier.

4.1 Watson Visual Recognition

After sending up the zip files with the images, Watson takes care of all of the training for the user on IBMs servers.

Watson takes quite “long” time to train, however, it had the highest accuracy, and it were simple to use. Also, the prediction time for an image is okay when you consider that the image has to be sent over the network, to IBMs servers and then get back the prediction response.

During and after the training phase, one could not really observe how the training was going/had gone for Watson. There was no way of seeing the training history of accuracy or what not. Watson didn't really notify the user either when it was done training (hence the uncertainty of the training time). All that could be one to observe the training, was to send a request to Watson, asking for the API instance custom classifiers, and the response would be a JSON that contained the parameter "status" that either had the value “training”, for that the classifier was still training or “ready”, for that the classifier was finished training and ready to use.
4.2 TensorFlow with Inception V3

The training phase of the Inception V3 model was the shortest and had the seconds highest accuracy out of the different classifiers. However, it has the longest average prediction time for an image.

During the training, one could observe how it was going for the model (see figure 4.1 below).

```
INFO:tensorflow:2018-07-27 14:04:31.249771: Step 3970: Train accuracy = 95.0%
INFO:tensorflow:2018-07-27 14:04:31.448653: Step 3970: Validation accuracy = 56.0% (N=100)
INFO:tensorflow:2018-07-27 14:04:33.079392: Step 3980: Train accuracy = 93.0%
INFO:tensorflow:2018-07-27 14:04:33.255289: Step 3980: Validation accuracy = 55.0% (N=100)
INFO:tensorflow:2018-07-27 14:04:35.077288: Step 3990: Train accuracy = 88.0%
INFO:tensorflow:2018-07-27 14:04:35.103557: Step 3990: Cross entropy = 0.247827
INFO:tensorflow:2018-07-27 14:04:35.077288: Step 3990: Validation accuracy = 56.0% (N=100)
INFO:tensorflow:2018-07-27 14:04:36.733334: Step 3999: Cross entropy = 0.231655
INFO:tensorflow:2018-07-27 14:04:36.845283: Step 3999: Validation accuracy = 62.0% (N=100)
```

*Figure 4.1: the last output from the training steps for the Inception V3 model.*

Another way of observing the training history, both during and after, was to use TensorBoard [15]. This made it possible to observe visually with plots how the training history was/had gone (see figure 4.2 below).
4.3. Keras with MobileNet

Retrain MobileNet with Keras took the longest and it also has the worst accuracy out of the classifiers. However, it is the fastest when it comes to prediction time.

When training the Keras model one could observe the output for each of its epoch (see figure 4.3 below).
4.4 Converting the MobileNet Keras model into TensorFlow.js format

Converting the retrained MobileNet model from Keras, into TensorFlow.js format was simple. All that needed to be done was to enter command 4.1 in a bash terminal (also requires a Python environment and to have the converter installed) [16].

```
tensorflowjs_converter --input_format keras \ 
  path/to/lantman_mobilenet.h5 \ 
  path/to/tfjs_target_dir
```

**Command 4.1:** bash command for converting Keras model into TensorFlow.js format.

In the `tfjs_target_dir` the converted model withstands of a `model.json` file and a set of sharded weight files in binary format. The `model.json` file contains the...
description of the layers and how they are connected and a manifest of the weight files.

4.5 As applications
Here the application results will be presented.

4.5.1 Swing application using Watson
The Watson application was implemented in the Java Swing library. It had a simple user interface (see figure 4.1 below), where one could make all of the possible API requests that Watson Visual recognition had to offer.

![Figure 4.1: the user interface for the application built for Watson.](image)

The application in figure 4.1 can use all functionality of Watson's Visual recognition API. For this API instance, there are six different custom classifiers trained (seen in the black output area), where "lantman_1702805864" is the one that has been trained with all the given data from Lantmännen.
4.5.2 Web application using Inception V3

The Inception V3 web application was implemented with JavaScript, HTML, and CSS for the frontend (see figure 4.2 below), and the backend service hosting the Inception model was implemented in the Python flask web framework.

Figure 4.2: the start page for the application that uses Inception V3 where an image has been selected for classification.
In figure 4.2 above, the application allows the user to upload an image to the browser, and when the “Classify” button is pressed, this image will be sent to the backend service where the Inception model will classify it, and return the predicted results (as can be seen in figure 4.3 above).

4.5.3 MobileNet and TensorFlow.js

The MobileNet web application was implemented with JavaScript, HTML, and CSS for the frontend (see figure 4.4 below), and it also uses the TensorFlow.js library that will load the retrained MobileNet model that is hosted on a GitHub JSON API [17] as model.json.
The loading of the model took less than two seconds, and after that, all that has to be done is to select an image, and the classification process will begin immediately.
5 Discussion

An important thing to include for this thesis is that the effort of the training phase for each of the classifiers has been quite “low”. This means that for the framework and library part, a lot more could have been done to improve the accuracy for the retraining of the models. For example, a lot more tweaking like:

- test with different learning rates,
- test with different number of training steps,
- test with different batch sizes,
- test different combinations of layers to retrained in the models,

and more. By resorting the images, the result for the classifiers could also have been improved. However, this could also have ended up with worse results, for one or more of the classifiers.

However, the “low” effort of the training phase is an important aspect of the thesis, since the interest was to see how non-experts or peoples with low knowledge of machine learning could use these different techniques for machine learning approaches when solving their own problem(s). So this chapter will discuss how one can look at the result from this sort of aspect.

5.1 Services

For the users willing to pay, services are by far the easiest technique to use when one wants to do a machine learning approach for their problem(s). By just focusing on structuring the training data into their classes and then send them to the API instance, Watson Visual Recognition API manages to create a classifier that had an accuracy of 86.75%, and this with a relatively small amount of images too.

5.2 Frameworks and libraries

For sure, this is where one can accomplish a lot and be able to take full advantages and own control of one's own implemented models. But one can also take advantage of other machine learning developers open sourced models. That they have either pre-trained or just share their architecture descriptions of a model that can be used for own implementation.
5.3 The applications

When it comes to using these different techniques in ones application, all of them have their benefits.

5.3.1 Watson as service

When it comes to using services, like Watson, there is the luxury of not having to host your own service for the classifier, it is also easy to use and easy to learn, and it has good libraries to use for implementing all the different functionality that Watson has to offer.

However, during this thesis, the limitation of this specific API instance was noticeable. What one could achieve by using Watson is probably enough for a lot of users. However, it would also be very narrowed for others when all the trust in the training is handed over to Watson.

The user must also send Watson its data. Which means that the data is sent to IBM’s servers, so this could be a problem for users that don’t want to risk having sensitive data leaked.

5.3.2 Own service for hosting models

This could be a good substitute for Watson if one has the resources to host an service for their own costume trained models. One would be freer to use any of the all the open sourced models for their own projects. For image classification, it would be recommended to use large models, like the Inception V3 model, when going for higher accuracy and smaller models, like MobileNet, when one wants faster predictions.

However one must also, in this case, think about safety when sending sensitive data over the network to their host.

5.3.3 Loading models into the browser.

This is quite a new technique that can be used. TensorFlow.js was released in March of 2018 and has fastly become popular as a machine learning libraries for developers. Since JavaScript is the most used programming language for the web, this library now allows developers to create, and use their models directly in the browser. This solution hence has the convenient aspect of that the classification of one's data is not sent over the network to some service since it is done directly in the browser, the predictions are made locally on the users machine. Which would be a great safety solution for sensitive data.
However, this will limit the user to smaller models, since browsers, in general, has very limited space for loading in data. Hence smaller models like MobileNet are used.
6 Conclusion

The conclusion from this thesis would be that if developers want to be able to use machine learning approaches in their software, there are many multiple options for them to do so. If they, however, don’t want to have to learn a lot about machine learning, the easy way would be to just use available machine learning services. However, for those willing invest more time in learning about machine learning, and combine that knowledge with a framework like TensorFlow and/or Keras, would result in a lot more freedom to develop all kinds of application. One would also not be dependent on a cloud service, and would, therefore, be more independent.

No real conclusion of which type of techniques is the “best” (services or frameworks) to use can really be drawn from this thesis. One can get good results by using both kinds of techniques. However, using services are easier and the developer does not have to invest a lot of time into learning more about machine learning and tweaking parameters for the model. One just has to send the data to the service were it handles all of the machine learning aspects. However, this limits the developer, and also makes him/her reliable on a cloud service. Also, one important factor is that most (if not all) cloud services cost money, and the amount it will cost often depends on the amount of data that has to be processed. So for someone with a lot of data, using services can end up costing a lot. However, if the user has the financial resources and is willing to pay for the services, there is no problem. So both techniques have pros and cons.

6.1 Future work

There are plenty of other types of problems that can be approached by using machine learning, like semantic analysis, voice recognition, and more. It would be interesting to also test out using services and frameworks for solving these kinds of problems. It would also be net to see the results of a completely re-trained model, like the Inception V3 and/or MobileNet, from scratch and see how well it perform compared to just one that has been re-trained using transfer learning on a few layers.

Also, the TensorFlow community has grown significantly in the last few years, and a lot has happened just during the first half year of 2018. One can use many different pre-trained models for transfer learning for many different types of areas, and by using TensorFlow-Hub [18] one could accomplish a lot by combining different kinds of pre-trained models to solve their own tasks.
References


