Jani Kurkinen

On practicalities of identifying and implementing a suitable software architecture for a typical deep learning data science project

Master's Thesis in Information Technology

July 8, 2019

University of Jyväskylä

Faculty of Information Technology

Author: Jani Kurkinen

Contact information: jani.o.kurkinen@student.jyu.fi

Supervisors: Jussi Hakanen, and Vesa Ojalehto

Title: On practicalities of identifying and implementing a suitable software architecture for a typical deep learning data science project

Työn nimi: Ohjelmistoarkkitehtuurin suunnittelu ja toteutus tyypillisessä syväoppimisprojektissa

Project: Master's Thesis

Study line: Software engineering

Page count: 60+0

Abstract: This thesis describes what phases a typical deep learning project has and what tools can be used to implement it. The aim is to explore how to get results with certain software tools with existing data. In addition, theory behind deep learning will be briefly introduced. The practical part of this thesis demonstrates how a neural network can be built, trained and used for image classification with selected tools. Image classification can be done with various tools and the ones used in this thesis proved to be good choices because of their ease of use and feature richness.

Keywords: AI, artificial intelligence, CNN, deep learning, framework, image classification, machine learning, SaaS

Suomenkielinen tiivistelmä: Tutkielmassa tarkastellaan, minkälaisia vaiheita tyypillinen syväoppimista hyödyntävä projekti sisältää ja minkälaisilla työkaluilla se voidaan toteuttaa. Tarkoituksena on selvittää, miten tietyillä ohjelmistotyökaluilla saadaan tuloksia aikaan valmiiksi kerätyllä datalla. Lisäksi kerrotaan lyhyesti syväoppimiseen liittyvästä teoriasta ja demonstroidaan, miten valituilla työkaluilla voidaan rakentaa ja kouluttaa neuroverkko sekä käyttää sitä kuvantunnistukseen. Kuvantunnistusta voi tehdä useilla eri työkaluilla, ja tähän tutkielmaan valitut työkalut osoittautuivat hyviksi vaihtoehdoiksi helppokäyttöisyytensä ja

monipuolisuutensa ansioista.

Avainsanat: AI, CNN, koneoppiminen, kuvantunnistus, ohjelmistokehys, SaaS, syväoppiminen, tekoäly

Glossary

ADL	Architecture description language
CNN	Convolutional neural network
CPU	Central processing unit
FFNN	Feedforward neural network
GAN	Generative adversarial network
GPU	Graphics processing unit
MLP	Multilayer perceptron
ReLU	Rectified linear unit
RNN	Recurrent neural network
SaaS	Software as a service
SOA	Service-oriented architecture
UML	Unified modeling language

List of Figures

Contents

1 Introduction

Deep learning can provide solutions for complex problems. Previous research has shown that it can be an effective method to solve problems that used to be very difficult for computers. Building of neural networks have become more and more user-friendly with high-level deep learning libraries such as Keras. In addition, various machine learning platforms have been developed to make machine learning accessible for data scientists and engineers. Deep learning technology is now more accessible than ever before and many companies are looking for opportunities on how to benefit from it. Achieving good results requires a capable set of tools and knowledge on how to use them. However, choosing suitable deep learning tools from a wide range of options can be a difficult and time-consuming task.

This thesis offers a brief overview on practicalities of a typical deep learning project and shows an example implementation on how some modern deep learning tools can be utilized for a typical image classification problem. Theoretical aspects of deep learning are explained and two different neural network architectures are mentioned. As software as a service (SaaS) has become a common way of delivering software products to customers, advantages and disadvantages of it are examined. A pre-built CNN is trained with image data and used to classify test images. In addition, transfer learning is examined by using a pre-trained convolutional neural network (CNN) to predict classes for testing data. These chapters should provide a basic understanding on what is deep learning, what are important things to consider when working with deep learning projects and how to actually get results with specific tools.

Literature has been searched from IEEExplore, ACM Digital Library and Google Scholar with keywords such as "SaaS", "MLaaS", "framework", "machine learning", "deep learning" and "software architecture". The source material has not only been gathered from scientific publications but also websites that have been considered useful for the topic.

Chapter 2 focuses on theoretical aspects of machine learning and SaaS, with the focus being on deep learning. Chapter 3 introduces a general dataset of flower images and some practicalities on building neural networks for image classification tasks. In chapter 4, com-

mon technologies for scalable deep learning applications are mentioned. Chapter 5 shows an example implementation of a deep learning image classifier model using some of the technologies mentioned in Chapter 4. In Chapter 6, some observations that were made during the writing process are discussed. Finally, Chapter 7 summarizes the contents of this thesis.

2 Background

This chapter introduces some basic concepts about artificial intelligence, machine learning and neural networks. As various machine learning tools are offered as services over the Internet, Service Oriented Architectures (SOA) and SaaS are also explained. In addition, some aspects of collecting and storing data are introduced. The goal is to provide a basic understanding about these topics without going too deeply into details. Utilizing neural networks in practical applications requires knowledge on the deep learning theory but many modern machine learning libraries have abstracted complex algorithms behind simple APIs. This thesis aims to discuss practicalities of how to use some of these tools for a typical deep learning problem.

2.1 Artificial intelligence & machine learning

The concept of artificial intelligence (AI) has been talked about even before the first computers were built. Problems that are difficult for humans to solve can be solved quite easily with computers. However, some problems, such as understanding speech or recognizing faces, can still be rather difficult to solve by a computer. The difference between these problems is that some of them are more difficult to present in a formal way than others. Teaching a computer to be a master in the game of Chess is actually not so complicated because the game has rules everybody are required to follow. Recognizing a spoken language is a very different kind of task. During the recent years, computers have began to catch up to humans in these more intuitive tasks where humans used to have an advantage. (Goodfellow, Bengio, and Courville 2016, Chapter I)

Dictionary definition of artificial intelligence is

> "the study of how to produce machines that have some of the qualities that the human mind has, such as the ability to understand language, recognize pictures, solve problems, and learn"
> `https://dictionary.cambridge.org/dictionary/english/`

Therefore, if a machine can solve problems in a way similar to how the human mind works, we are talking about AI. For example, mathematical models such as statistical inference, and expert systems that operate with hard-coded knowledge are types of AI (Pilli-Sihvola 2010). Figure 1 shows the relations between some types of AI.

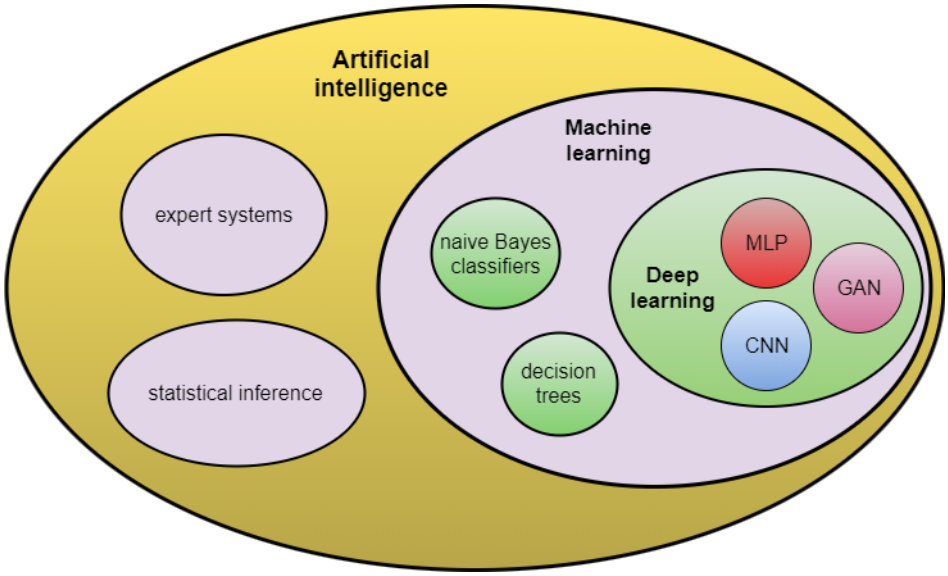

Figure 1. Artificial intelligence. This diagram shows only some types of AI, machine learning and deep learning.

During the recent years, machine learning has become one of the hottest topics in the field of data science. Advances in data processing, training algorithms and hardware have made this possible. Machine learning happens when computer learns to do tasks based on experience. But what does it mean when a computer learns to do something? Mitchell (1997) defines it as follows:

> "A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at tasks in T, as measured by P, improves with experience E."

Many problems that are difficult for humans to solve manually can be solved with machine learning. Classification, regression, anomaly detection and machine translation are some examples of tasks where machine learning has been utilized successfully (Goodfellow, Bengio,

and Courville 2016). Nowadays it is used in various areas ranging from offering personalized video recommendations to medical diagnosis. One interesting example from the medical field is the detection of early signs of Alzheimer's disease by analyzing patients' MRI images (Moradi et al. 2015).

As demonstrated by Ribeiro, Grolinger, and Capretz (2015), Figure 2 shows that machine learning can be divided into three categories: supervised learning, unsupervised learning and reinforcement learning.

Supervised learning associates certain features in testing data to a specific label. The training data has to be labeled. For example, we want to know what type of car is in a picture. Our machine learning model has been trained with a dataset consisting of labeled pictures of various types of cars (label "0" for hatchback, "1" for convertible, "2" for SUV etc.) and we want to know what kind of car is in our new image. Based on visual features of the car, the machine learning model will determine a label for our input image. In short, applications that utilize supervised learning extract labels from certain features.

In **unsupervised learning**, the data in a training set is unlabeled. One example of this kind of dataset could include information about customers and their buying habits. A company selling products on their web shop may have lots of data about their registered users and transactions. A machine learning model can map similarities between different inputs using clustering techniques. For example, it could analyze customer behavior on online stores and find out that people aged 25-30 years old are more likely to spend more time on searching for electronics than people who are less than 25 years old. Thus, these customers can be classified into same segment.

In **reinforcement learning**, the aim is to teach a machine learning model to improve its performance by dynamically giving it feedback from the environment. Reinforcement learning is widely used in the field of robotics ("Johdatus tekoälyn taustalla olevaan matematiikkaan - TIM" 2016). For example, if we want to teach a bipedal robot to stand up, we can control its joints and their positions to balance it. Moving one leg in wrong direction at the wrong time will have a negative effect, and we can use this learning experience to perform better next time.

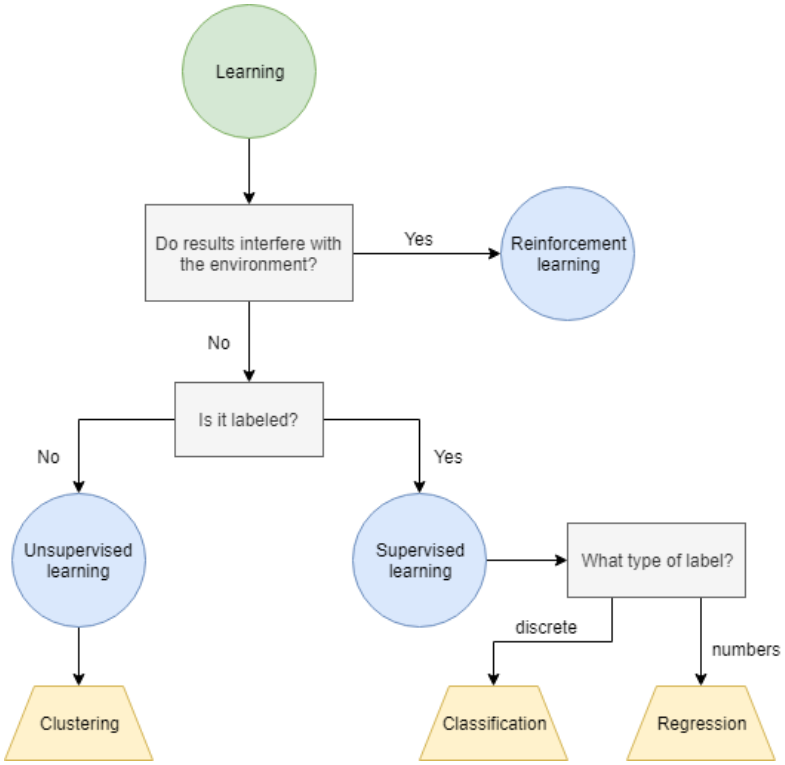

Figure 2. Machine learning methods categorization (Ribeiro, Grolinger, and Capretz 2015)

Machine learning can be done with many different algorithms such as decision trees, naive Bayes classifiers and K-nearest neighbors algorithm ("Johdatus tekoälyn taustalla olevaan matematiikkaan - TIM" 2016). Deep learning is one type of machine learning that utilizes more than one layer of neural networks to solve complex problems (Gersey 2018). In deep learning, a computer is taught a hierarchy of concepts and it is made to learn from experience (Goodfellow, Bengio, and Courville 2016, Chapter I).

2.2 Artificial neural networks

Human brains have neurons that are connected to each other with synapses. Our decision making is based on signals that travel between the neurons. Artificial neural networks mimic the functionality of human brains ("Johdatus tekoälyn taustalla olevaan matematiikkaan - TIM" 2016). As seen in Figure 5, a neural network consists of an input layer, an output layer and one or more hidden layers. The layers have neurons that process an input and produce an output as seen in Figure 3. The neurons in hidden layers determine the output value for an

input. Neural network is called feedforward neural network (FFNN) when the connections between neurons always go towards the output layer. The connections are called weights and each neuron has a bias. Bias can be thought of as a value that tells how easily the neuron "activates". The output value of a neuron i can be written as follows:

$$z_i = \sum_{i=1}^{n} (w_i x_i) + b$$

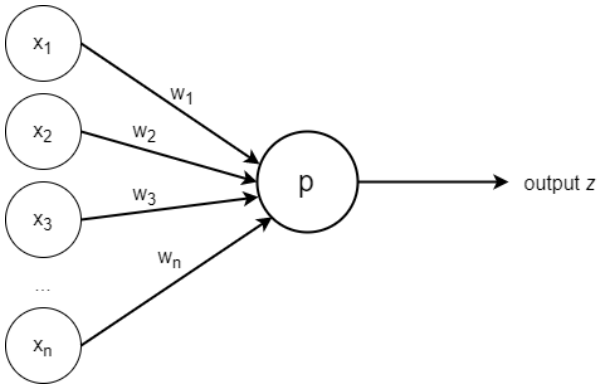

Figure 3. Neuron and its inputs

Output is passed on to an activation function before it gets passed on to a next neuron. Different layers of a neural network can use different activation functions. Their purpose is to bring nonlinearity to a linear input. In other words, they make sure that a small change in neuron's weight or bias doesn't cause too dramatic consequences to its output value. ("Johdatus tekoälyn taustalla olevaan matematiikkaan - TIM" 2016)

Sigmoid, hyperbolic tangent and ReLU are commonly used activation functions ("Johdatus tekoälyn taustalla olevaan matematiikkaan - TIM" 2016). With an activation function, the final output value of a neuron would be:

$$\varphi(\sum_{i=1}^{n} (w_i x_i) + b)$$

where the activation function φ is the sigmoid function

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

This produces nonlinearity to the result as seen in Figure 4

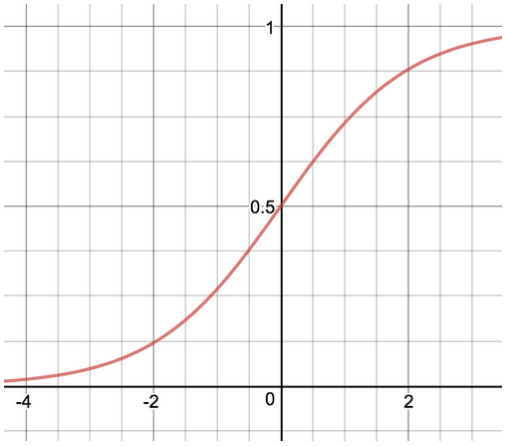

Figure 4. Sigmoid function (“Johdatus tekoälyn taustalla olevaan matematiikkaan - TIM” 2016)

In order to make a neural network to learn in supervised learning environment, it has to bee trained with input data and expected output values from the training data. Each input x has an expected output y, so they form an input-output pair (x, y). For example, an input could be an image of a cat. To put it more precisely, it is the pixel values of an image of a cat. An output is the class where the said animal belongs to, so in this case, ’cat’. Loss function (sometimes called a "cost function") tells how far the actual output was from the expected output. Therefore, minimizing the loss function is the key to training and learning. Gradient descent and backpropagation algorithms minimize the loss function by changing weights and biases of neurons in hidden layers. These algorithms can be computationally expensive. (“Johdatus tekoälyn taustalla olevaan matematiikkaan - TIM” 2016; Goodfellow, Bengio, and Courville 2016, Chapter 8.3.1)

Training a neural network with gradient descent and backpropagation algorithms can be described as putting a ball on a bumpy surface and watching it to roll to the lowest position. When the ball is set to its starting position, it will start rolling towards some direction. When

it stops moving, it has found the lowest point of the surface, the local minimum. The position of the ball can be thought of as the loss function value with weights and biases. Backpropagation algorithm then changes these parameters in a way that the loss function decreases on the next update cycle (epoch). Choosing the right initialization parameters is important but in a typical neural network training process, the biases for each unit are initially set to heuristically chosen constants, and the weights are initialized randomly (Goodfellow, Bengio, and Courville 2016, Chapter 8.4).

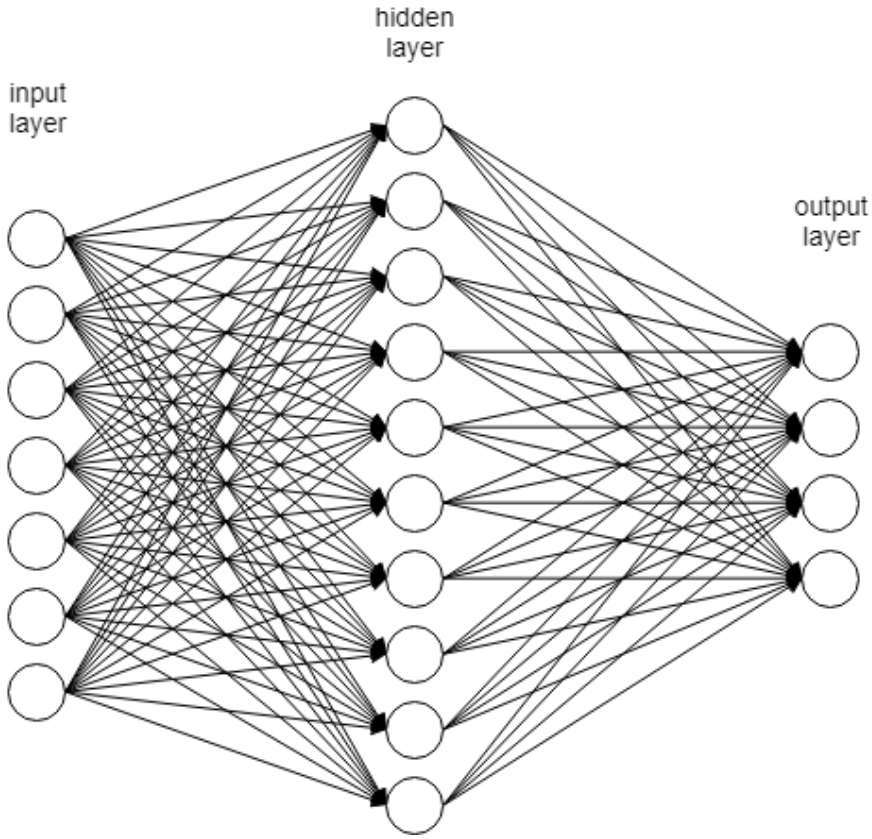

Figure 5. A fully connected FFNN with a single hidden layer.

The following subsections introduce two types of neural networks: convolutional and generative adversarial networks. The former are a popular choice for image classification tasks and the latter are interesting for their ability to generate content.

2.2.1 Convolutional neural networks

Convolutional neural networks (CNNs) are neural networks where calculation of output values are based on mathematical convolution operations. The concept of CNNs was originally introduced by Yann LeCunn in 1989. CNNs are proven to be effective for 2D image classification tasks and other similar tasks where data is presented in a grid like structure. (Gersey 2018)

CNNs usually use a 3x3 kernel matrix for calculating convolutional operations, and one typical use case is to detect edges of an image (Gersey 2018). The purpose of these operations in CNNs is to find out what kind of features a kernel matrix can find when the whole image is "scanned" with it.

In short, the idea is to find the most important features of an image by downsampling the input (Gersey 2018). A typical layer in CNN architecture consists of three stages: a convolution stage, a detector stage and a pooling stage (Goodfellow, Bengio, and Courville 2016, Chapter 9.3). Each of these stages are layers in a network. The layer in the first stage produces a set of linear activations, the second stage uses a nonlinear activation function for the outputs from the first layer, and lastly, the output is modified with a pooling layer (Goodfellow, Bengio, and Courville 2016, Chapter 9.3). Pooling creates a statistic summary of feature maps: For instance, it can calculate the average or maximum value within the area where a kernel matrix is positioned (Gersey 2018).

2.2.2 Generative adversarial networks

Generative adversarial networks (GANs) were first introduced by Goodfellow et al. (2014). Since then, GAN architecture has gained a lot of interest and it has been described as one of the most interesting advances in deep learning (Gersey 2018). Recent advances in GANs have showed promising results in various tasks such as generating images of human faces that are indistinguishable from real life samples (Karras, Laine, and Aila 2018) or depicting a model in different poses with images of 2D skeleton (Pumarola et al. 2018). GAN has also been utilized successfully to detect anomalies in network intrusion datasets with non-image data (Zenati et al. 2018).

The basic idea of GAN is that two different neural networks are set to compete with each other. A generator tries to create samples that are as close as possible to the samples in training data. A discriminator takes a generator-created sample as an input and determines its validity: does it look like a real sample?

One way to understand GAN architecture is to think of it as a race between a police and a forger: The discriminator is a police whose job is to identify fake products. The generator acts as a forger and tries to create products that are indistinguishable from genuine products. When the police receives a product, it determines whether it is a real one or a fake one. The product could be taken from a training set or it can be created by the generator. Initially, the police isn't very smart because it has no prior experience: It has no idea what a real product looks like. Same goes for the forger. When the police falsely identifies a product from a training set, it learns from its mistake and remembers this next time. The forger is happy to have fooled the police and maybe tries to fool him again with a similar product. It doesn't work anymore because the police has already learned to identify this type of fake products. The forger has to try something new next time. ("Photo Editing with Generative Adversarial Networks (Part 1)" 2017)

In short, GAN is a competition between two actors: the generator and the discriminator. Figure 6 shows the main components and functionality of GAN architecture. The original GAN architecture by Goodfellow et al. (2014) has inspired many other implementations (Dumoulin et al. 2016; Mirza and Osindero 2014; Odena, Olah, and Shlens 2016; Pumarola et al. 2018).

2.3 Image classification with neural networks

Neural networks can be utilized to solve various problems in areas such as speech recognition, medical diagnosis, machine translation and computer vision. The focus of this research lays on image classification with CNNs. CNNs were chosen as an example because they are considered to be a good choice for image classification tasks (Goodfellow, Bengio, and Courville 2016, Chapter 9).

In typical image classification tasks, a fully-connected neural network uses all the pixels of

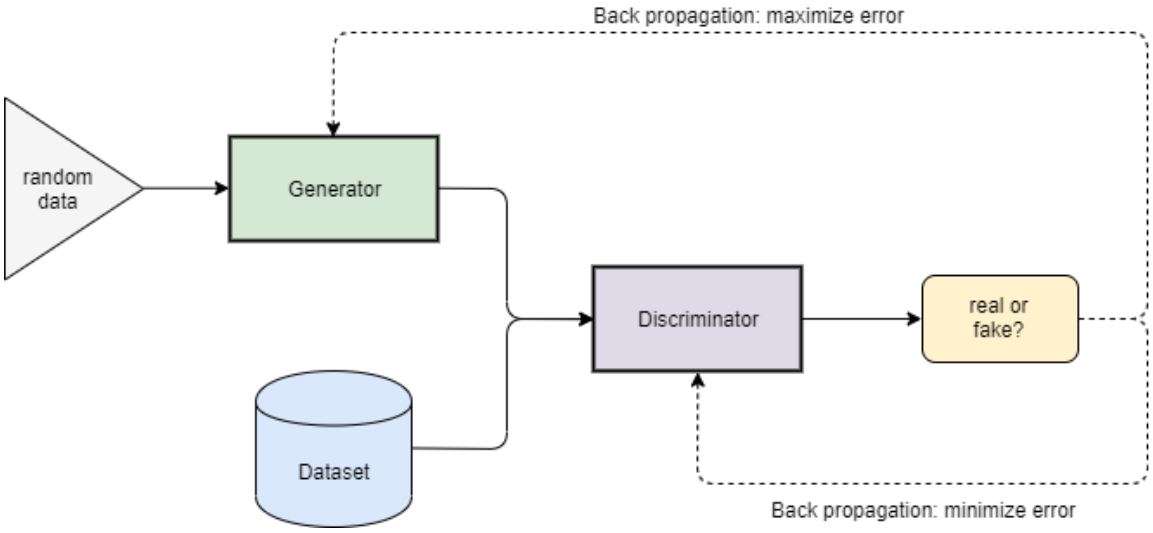

Figure 6. GAN framework ("Photo Editing with Generative Adversarial Networks (Part 1)" 2017)

the inputted image to predict a label for it. This means that the input layer has one neuron for each pixel. For example, if we have a grayscale image with width of 32 pixels and height of 32 pixels, we would have $32 \times 32 = 1024$ neurons on input layer. For colored images, the amount of neurons would be $32 \times 32 \times 3 = 3072$. Thus, the first layer alone would already have 3072 trainable weights. With CNNs, the dimensions of the input layer can be reduced by using convolution, thus reducing the amount of parameters in the network. Downsampling is important because it can extract the main features of the input data efficiently (Gersey 2018).

Training complex neural networks is computationally expensive and therefore requires powerful hardware (Goodfellow, Bengio, and Courville 2016, Chapter 8). However, training neural networks for new tasks is not always necessary because pre-trained models can be used. This process is known as transfer learning (Goodfellow, Bengio, and Courville 2016). More details about transfer learning will be introduced in Sections 3.2 and 5.4.

2.4 Connecting users with machine learning software

Distributed systems have become a common occurrence during the recent years. Many applications utilize "the cloud" in order to provide a consistent user experience between different

devices. This has led to service oriented architectures (SOA) and software as a service (SaaS) become typical terms in the field of software engineering. These two can be mixed up quite easily. Laplante, Zhang, and Voas (2008) summarize that SOA is an architecture framework that can be used to construct a software whereas SaaS is a model for delivering the software.

2.4.1 Why use SaaS? Advantages and disadvantages

SaaS is only one possible model for delivering software to clients. In contrast, a more traditional software delivery method is to deliver an application on a physical installation disc which used to be a common way for customers to install new software products. Carroll, Van Der Merwe, and Kotze (2011) list many benefits for using SaaS and have arranged them by the number of citation occurrences in literature:

1. Cost efficiency
2. Scalability
3. Flexibility
4. Agility
5. Better IT resource management and business focus
6. Efficiency
7. High reliability / availability
8. Rapid developments, deployments and change management
9. Better performance
10. Greater mobility
11. Improved automation, support and management
12. Improved security
13. Green-IT data centre

However, there are also risks that need to be considered with SaaS. According to the literature review by Carroll, Van Der Merwe, and Kotze (2011), the biggest concerns are:

1. Security
2. Third party vendors (service providers)
3. Management and control

4. Laws and regulations (compliance)
5. Portability and interoperability
6. Disaster recovery
7. Virtualization risks
8. Lack of standards and auditing
9. Maturity of technology
10. Uncontrolled viable costs

It's important to note that security is considered more as a risk than a benefit. Sending private data to cloud servers causes security and privacy concerns: Sometimes it can be impossible to know where the data will be located and how it will be handled (Valjakka 2012). On the other hand, it can be seen as a security benefit. In some cases, when data is hosted by a 3rd party service provider, it may actually be in safer hands than on a company's own systems. After all, there are many things that need to be considered in order to mitigate security risks as Carroll, Van Der Merwe, and Kotze (2011) conclude in their research.

In any case, security issues are an important factor that must be taken into account when software is provided as a service. It's important to understand that management has the responsibility of keeping the data safe, no matter where it is hosted (Carroll, Van Der Merwe, and Kotze 2011). Both options should be considered when designing software services: hosting the data locally or sending it to cloud servers.

Of course, SaaS applications have technical requirements too. According to Valjakka (2012), the important aspects to consider from the technical point of view when designing a SaaS application are

- The client application has to be easily accessible via web browser
- Customers don't install the software but they must do the initial deployment by themselves
- Customers are usually companies or organizations that provide the application to their end users
- The application has to serve multiple users simultaneously

2.4.2 Machine learning as a service

Pilli-Sihvola (2010) writes about a concept of "intelligence as a service" where intelligence is considered to be "a capability to solve problems by answering questions". His research introduces some points that need to be considered in order to provide intelligent software services. The most crucial issues are related to communication between the client and the service provider: (Pilli-Sihvola 2010)

- Mutual comprehension
- Functional representation of knowledge
- Data synchronization
- Coupling of the service consumers and providers

In short, customers need to know how to use the service and what to expect in response and service providers have to make sure that requests are received and handled in an unambiguous way. These are relevant questions that should be considered when designing MLaaS software. Pilli-Sihvola (2010) mentions web services as a one possible solution to provide intelligence as a service. According to Laplante, Zhang, and Voas (2008), web services are the best option for supporting SaaS.

During the recent years, a demand for machine learning as a service (MLaaS) has emerged, and making these services affordable for everybody has become an important goal (Ribeiro, Grolinger, and Capretz 2015). Various software products have been developed to meet this demand: Comet, Microsoft Azure ML Studio, Valohai, Floydhub, Google Prediction API, IBM Watson, MLflow and NSML are only some examples of machine learning platforms that provide machine learning services. They provide tools for monitoring and analyzing data through web UI and also access to cloud hardware resources for training and testing ML models. Figure 7 demonstrates the steps of a typical ML workflow.

Because MLaaS is basically one form of SaaS, common SOA design models can be utilized for MLaaS software design. Therefore, using web services technology to provide MLaaS applications is a feasible solution.

Figure 7. Typical workflow of a ML project (Li et al. 2017)

Since security is the number one concern in SaaS applications, it is an important requirement to be addressed when designing MLaaS software. Training ML models with confidential data in cloud insfrastructure has risks. Attackers can access the training data and results by carrying out extraction attacks. Kesarwani et al. (2018) introduces a security measure against extraction attacks that monitors the usage of the machine learning model. One way to achieve improved security in SaaS applications is to encrypt the data before it is send to a remote server. This can work well for MLaaS applications as well: Hesamifard et al. (2017) concluded that training neural networks and making predictions with encrypted data is a viable solution to increase data security.

2.4.3 Architecture description

There are many ways to visualize a software architecture. As of 2012, 135 different architecture description languages (ADLs) were identified (“Software Architecture: Architecture Description Languages” 2012). Drawing simple boxes and lines on paper is one solution to describe software architectures but in many cases this can be too ambiguous approach. ADLs were developed to tackle this problem. Singhoff et al. (2019) summarize the main purposes of ADL languages as "a way to capture architecture elements of a system and to help designers to make various analysis about it".

Bashroush et al. (2006) investigated different ADLs and noted that the reason why ADLs are not widely adopted in designing industrial applications is because of their limitations such as over-constraining and restriction to only a structural view of the system. Their research didn’t consider unified modeling language (UML) to be an ADL because of its limitations. For example, it offers only graphical notation. Their research also states that using existing ADLs for SOA may not be a good idea because they tend to abstract interfaces too much.

Another way to visualize a SaaS system architecture is SCA notation. According to Ribeiro,

Grolinger, and Capretz (2015), SCA inherits all advantages of SOA because it is built on top of it. Therefore, using SCA notation to present MLaaS architecture is a justifiable option. Figure 8 demonstrates the notation of SCA component diagram. Wires connect components to each other throughs ports that can be either reference ports or service ports. Ding, Chen, and Liu (2008) define these ports in the following way:

> "If the component provides a service through a port, the port is called service port; if the component requires a service through a port, the port is called reference port."

A more detailed presentation of SCA and its notation can be found on paper published by Ding, Chen, and Liu (2008). Figure 9 shows an example MLaaS system with SCA notation created by (Ribeiro, Grolinger, and Capretz 2015). For example, the topmost "Data Gatherer"-component provides a service for receiving a training dataset. It also requires a reference port for a dataset so that it can be used for training by the "Modeler"-component.

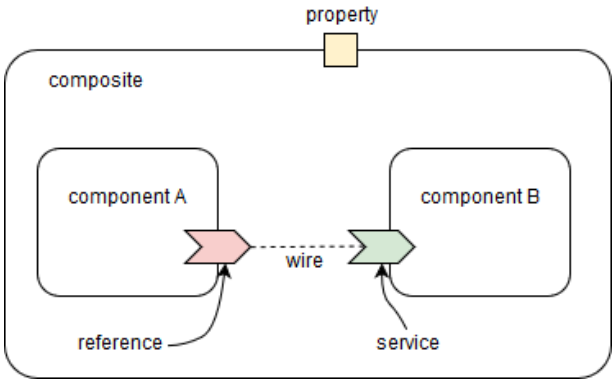

Figure 8. SCA component diagram.

2.4.4 Scalability

SaaS applications can have millions of users who are located all over the world. Providing a responsive software for everybody can be challenging, especially when the amount of concurrent users may change significantly in a short period of time. SaaS applications have to be able to adapt to these fluctuations without affecting negatively to user experience by providing enough server capacity for all users. (Valjakka 2012)

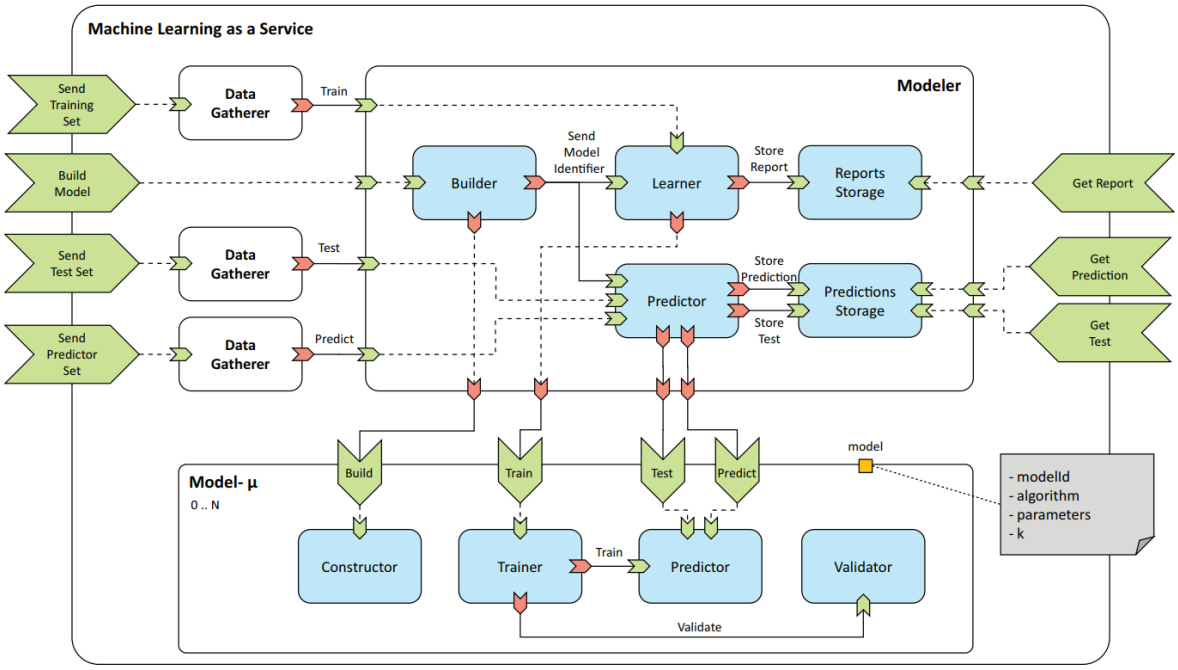

Figure 9. An architecture description of a MLaaS system using SCA notation in Ribeiro, Grolinger, and Capretz (2015).

Scaling is done with virtualization. Hardware resources such as CPUs, GPUs and storage devices can be virtualized programmatically instead of using physical resources. Applications can then run on these virtual machines. Setting up new virtual machines or stopping them based on demand is fast and simple. Containers take the scalability even further by making applications run in an easily transferrable container that includes everything the application needs to run in a single package.

2.5 Message dispatcher

SOA based applications usually consist of various components that are accessed over the Internet. For example, a web application for checking public transport schedules may use a Google Maps service for showing location information and it can get the bus schedules from a bus company's own service. A messaging system is needed to handle messages between senders and receivers because in many cases they are built to use different messaging protocols (Koskimies and Mikkonen 2005, Chapter 6.2.2). Messaging is organized with message

queues that ensure that the communication between senders and receivers is organized correctly (Schuchmann 2018). SOAP and REST are commonly used messaging protocols for accessing web services across the Internet.

2.6 Data preprocessing

The amount of digital data has exploded in the 21st century. Social media, Internet of things and digitalization of banking are some examples that generating a vast amount of data every day. Processing huge volumes of data is one of the biggest challenges in data analytics and it has to be approached with distributed computing (Garcia et al. 2016). In order to make use of the data, several steps are needed before any valuable knowledge can be obtained from it. Figure 10 shows these steps and connections between them as Garcia et al. (2016) have demonstrated in their paper. This process is a big challenge in data mining. A lot of data already exists but making valuable observations of it can be quite difficult.

In data analytics, the set of techniques used before the actual data mining is known as data preprocessing (Garcia et al. 2016). This paper won't go into details of these methods but for more information, readers can refer to Garcia et al. (2016) who have reviewed various methods for data preprocessing.

2.7 Data storage

The data in Big Data can be anything: photos, videos, charts, audio etc. Choosing the right way to store data is an important step in every data science project. It determines the data security and the process of extracting, transforming and loading of the data. This is known as the ETL process (see figure 11). It defines what data is selected from the original source, how to transform it to another format and determines how to load it into a data storage system. (Saraee and Silva 2018)

Data storage system should be scalable, fast and, of course, reliable. Extraction of data and means of storing it should be determined based on the content of the data (Saraee and Silva 2018). For example, image data has different characteristics than video data. These

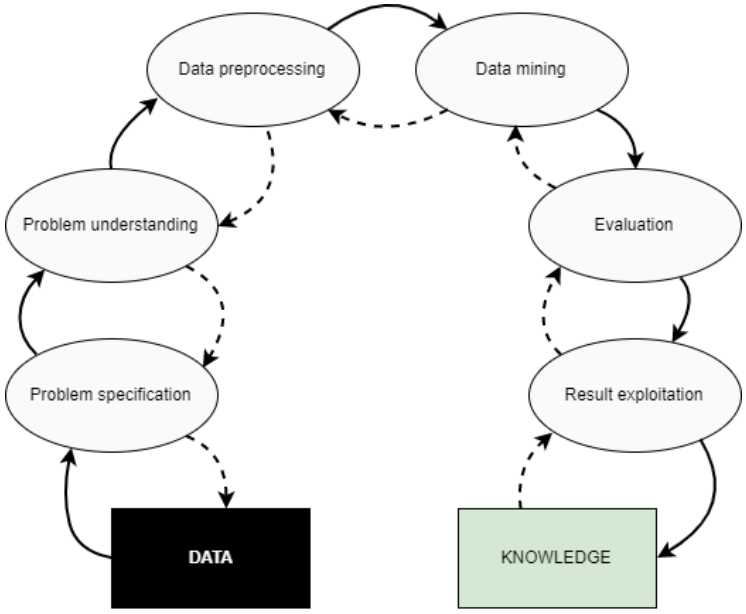

Figure 10. The steps needed to create knowledge from data as described in Garcia et al. (2016)

differences should be taken into account when selecting storage systems for the tasks. Siddiqa, Karim, and Gani (2017) have made an extensive survey on different big data storage technologies. According to their research, various fault-tolerant systems for storing big data exist. They conclude that decision has to be made between consistency and availability because according to Brewer's CAP theorem (consistency, availability, partition-resilience), all of them can't be achieved with a single system.

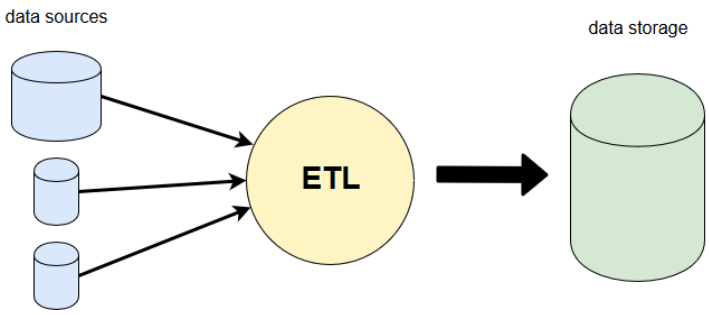

Figure 11. In the ETL process, data is selected from the sources and saved to a storage system.

3 Machine learning in practice

This chapter introduces some practicalities that are related to a typical deep learning application. A dataset consisting of flower images was chosen as an example data. In this chapter, a pre-built CNN is examined and trained with the data. Finally, the deep learning model's accuracy is evaluated.

3.1 Data

The "tf_flowers"-dataset[1] is one of the example datasets from TensorFlow, a popular machine learning framework. It consists of 3670 images of flowers. There are five classes in total: daisy, dandelion, roses, sunflowers and tulips. Flowers of each class are located in separate folders. The images are colored .jpg-files and their widths and heights are not fixed. ("Datasets | TensorFlow Datasets | TensorFlow" 2019)

The dataset was chosen for this paper because it serves the purpose of this thesis well: The data is collected from a real world example and it can be utilized easily as it is without doing much preprocessing or formatting for typical deep learning applications.

The features of the dataset look like the following:

```
FeaturesDict({
 'image': Image(shape=(None, None, 3), dtype=tf.uint8),
 'label': ClassLabel(shape=(), dtype=tf.int64, num_classes=5),
})
```

This means that each sample in the dataset is a 3-channel image. They also have a label, a `tf.int64` integer. Because there are five classes, the label is 0, 1, 2, 3 or 4.

An image can be presented as a three-dimensional array in two different ways: channels first or channels last which simply means where the number of channels are stored. A 150×200 colored jpg image as a three-dimensional array would have a shape of (150, 200, 3) when

[1] `http://download.tensorflow.org/example_images/flower_photos.tgz`

using the "channels last"-format.

3.2 Building a CNN for image classification

Designing good neural network architectures for image classification tasks can be complicated. A simple neural network architecture can work relatively well for certain image classification tasks. The biggest challenge is to find out how to create an optimal design solution for network architecture. How many layers should be used? What kind of layers and where? How about the optimal number of neurons on these layers? How to train a neural network? Choosing the right loss function is very important (Goodfellow, Bengio, and Courville 2016, Chapter 6.2.1). Training a neural network can be computationally expensive: Training process can take months even with hundreds of machines (Goodfellow, Bengio, and Courville 2016, Chapter 8). These complex training optimization algorithms are out of scope of this research. Instead, some general functions are mentioned.

The basic idea is to divide a problem into smaller sub-problems. If we want to know whether an input image has a cat or a dog in it, we can solve this by looking for certain features in the image. What physical features distinct cats from dogs? This is an easy task for humans but not so straightforward for computers to solve. We could start by examining their paws, tails or ears and see the typical characteristics for each animal. Deep neural networks can be trained to detect these differences.

Training deep learning models to make accurate predictions with new data is not a simple task. However, some common design principles for building and training neural networks exist. Szegedy et al. (2016) mention four general design principles for building CNNs:

1. Representational bottlenecks should be avoided. This is important especially in the first layers of a network.
2. In order to make training faster, increasing the activations per tile is beneficial.
3. Faster learning can be achieved when dimension reduction is done before spatial aggregation
4. The depth and the width (filters per stage) of a network should be optimized by increasing their complexity equally.

Szegedy et al. (2016) also note that using these general design principles don't necessarily work for every situation. They should rather be thought more of as guidelines for general cases where CNNs are build.

When neural networks are built for image classification, the last layer before an output layer usually uses Sigmoid activation function (see figure 4) because it maps the outputs between 0 and 1 (Gersey 2018). This can easily be intepreted as a probability of the input image belonging to a certain class.

Luckily, we don't always have to start building our own neural networks from scratch. Pre-trained models can be used for other kinds of tasks too. For example, InceptionV3 is a complex deep neural network that has been trained with images from 1000 different classes Szegedy et al. (2016). We can modify the network architecture by changing the output layer and utilize the network for other image classification tasks. This is called transfer learning. For example, Esteva et al. (2017) used a modified InceptionV3 to detect skin cancer and achieved great results.

Designing neural networks manually is not the only option. The process of building deep neural networks can be automated with evolutionary algorithms and it may become an effective option in the future (Miikkulainen et al. 2019).

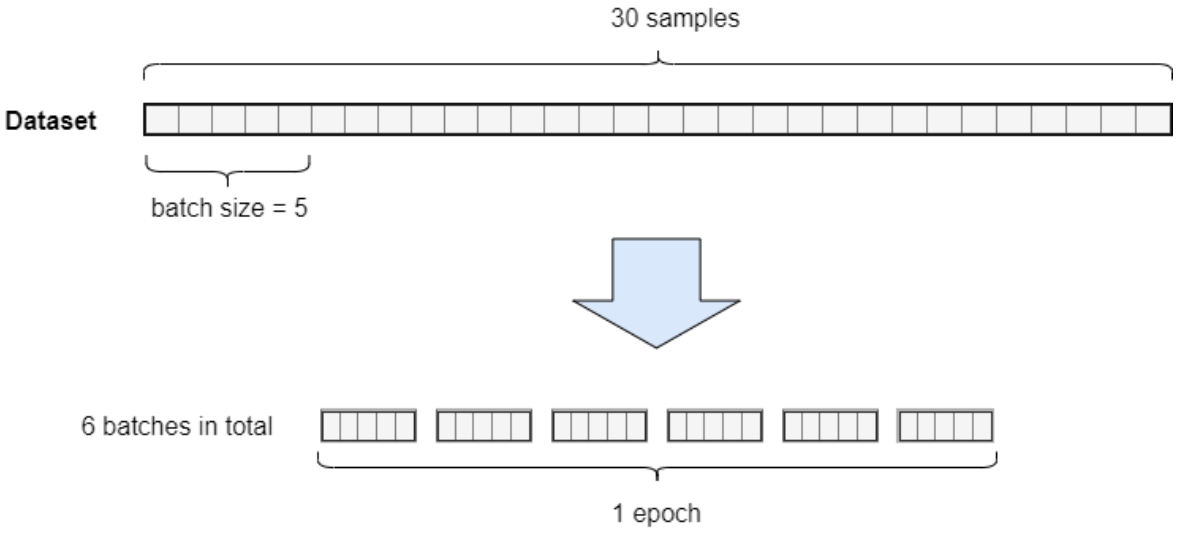

Figure 12. Example of a dataset with 30 samples divided into batches of five samples.

The following CNN is "mnist_cnn.py" an example network from Keras [2]. It was built to detect images of handwritten digits in the classic "mnist" dataset that was introduced by LeCun, Cortes, and Burges (1998). Taking a closer look of its summary reveals important details on the structure of the network:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 24, 24, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 12, 12, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 12, 12, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 9216)              0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               1179776
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                1290
=================================================================
Total params: 1,199,882
Trainable params: 1,199,882
Non-trainable params: 0
_________________________________________________________________
```

The summary includes information about the layers, their outputs and a number of parameters of the network. This CNN was slightly modified for the image classification problem of this thesis: For example, the input layer was changed to accept 100×100 images. Keras has easy-to-use functions for adding new layers to CNNs. For instance, the first layer is a convolutional 2D layer and it can be added as follows:

[2] "mnist_cnn.py" source: https://github.com/keras-team/keras/blob/d954aef2a2e33532f8725f58183cf605f5c54656/examples/mnist_cnn.py

```
model.add(Conv2D(32, kernel_size=(3, 3),
                 activation='relu',
                 input_shape=(100, 100, 3)))
```

The first parameter specifies the number of output filters in the convolution. Second, "kernel_size" specifies the dimensions of the kernel matrix. Next, "activation" is the activation function used in the layer. In this case ReLU (rectified linear unit) is used.

The output layer was also modified to be suitable for the five classes of the dataset. The altered CNN looks like the following:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 98, 98, 32)        896
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 96, 96, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 48, 48, 64)        0
_________________________________________________________________
dropout_1 (Dropout)          (None, 48, 48, 64)        0
_________________________________________________________________
flatten_1 (Flatten)          (None, 147456)            0
_________________________________________________________________
dense_1 (Dense)              (None, 128)               18874496
_________________________________________________________________
dropout_2 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_2 (Dense)              (None, 5)                 645
=================================================================
Total params: 18,894,533
Trainable params: 18,894,533
Non-trainable params: 0
_________________________________________________________________
```

3.3 Model training and evaluation

The images were split to training and testing sets. The training dataset contains 3303 and the testing dataset 367 images and both of them have samples from each classes. The images were left colored so the number of channels stayed at three.

The images were put in folders that were named '0', '1', '2', '3' and '4' which were interpreted as labels for the images. Pictures of daisies were stored in folder '0', dandelions in '1', roses in '2', sunflowers in '3' and tulips in '4'. The images were read with Keras's `flow_from_directory()`-function. For example, the training images were read as follows

```
train_generator = train_datagen.flow_from_directory(
    'train',
    target_size=(100,100),
    class_mode='categorical',
    batch_size = batch_size)
```

The "target_size"-argument resizes the images to a fixed size. The images from the testing folder were read in a similar way.

Keras' `fit_generator()`-function was used for training the model with the training data for 10 epochs. The model is validated with the testing data after each epoch. The data is split into smaller batches. Batch size determines the number of training samples that are sent to the network on each update cycle (Chollet et al. 2015). For example, when a dataset has 30 samples in total, it can be divided into 6 batches. Each of these batches have 5 samples. On each epoch, the weights of the model are updated after every batch. Thus, one epoch uses all 30 samples in the dataset and updates the model 6 times (see Figure 12).

```
model.fit_generator(train_generator,
    steps_per_epoch=train_imgs//batch_size,
    epochs=10,
    verbose=2,
    validation_data=validation_generator,
```

```
    validation_steps=test_imgs//batch_size)
```

After each epoch, the accuracy of the model is evaluated with the testing data and the results are presented after each epoch:

```
Epoch 1/10
 - 95s - loss: 1.4380 - acc: 0.3941 - val_loss: 1.2785 - val_acc: 0.4347
Epoch 2/10
 - 93s - loss: 1.2307 - acc: 0.4820 - val_loss: 1.1658 - val_acc: 0.5164
Epoch 3/10
 - 92s - loss: 1.1300 - acc: 0.5342 - val_loss: 1.1340 - val_acc: 0.5403
Epoch 4/10
 - 91s - loss: 1.0842 - acc: 0.5641 - val_loss: 1.1437 - val_acc: 0.4716
Epoch 5/10
 - 91s - loss: 1.0214 - acc: 0.5995 - val_loss: 1.0888 - val_acc: 0.5403
Epoch 6/10
 - 91s - loss: 0.9601 - acc: 0.6328 - val_loss: 1.0164 - val_acc: 0.5821
Epoch 7/10
 - 91s - loss: 0.8967 - acc: 0.6517 - val_loss: 1.0537 - val_acc: 0.6090
Epoch 8/10
 - 91s - loss: 0.8374 - acc: 0.6911 - val_loss: 1.0496 - val_acc: 0.5254
Epoch 9/10
 - 91s - loss: 0.7906 - acc: 0.7132 - val_loss: 1.0218 - val_acc: 0.5881
Epoch 10/10
 - 91s - loss: 0.7263 - acc: 0.7385 - val_loss: 0.9848 - val_acc: 0.6090
```

Now that the CNN has been trained with training data, an interesting question is how well our model performs with the testing data? The trained model was evaluated using Keras's `evaluate_generator()`-function which returns values for loss and accuracy of the testing data:

```
score = model.evaluate_generator(validation_generator, 10)
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

This gave the following results:

```
Test loss: 0.9685436024594662
Test accuracy: 0.6089552247702186
```

This shows that the trained model was able to classify images in the testing dataset with

~60.9% accuracy. This was done with a network that was originally designed for a very different task. Better results could be achieved with more complex network topology and parameter tuning but those topics are out of scope of this thesis.

4 Technologies for deep learning applications

Deep learning can be a very effective method to solve difficult problems. However, it is important that deep learning technologies are easily available to those who want to use them. Availability and ease of use are important things to consider when choosing the right tools for solving problems. Data gathering, preprocessing, building of neural networks and results analysis should be done efficiently without using too much time to learn how to use the selected tools.

This chapter mentions a few different key technologies that can be used in typical deep learning applications. Various options exist and each of the following technologies have alternatives. This chapter is not an extensive review of concurrent technologies and only some of the most popular options are mentioned.

4.1 Deep learning libraries

Building machine learning models can be a complicated task. Creating a new accurate model has many steps. The data has to presented in a way that it can be fed to a model. With neural networks this means encoding it to numerical values. The models have to be built and trained with functions that require deep mathematical understanding. Making a model to accurately predict which class a new sample belongs to can be a time-consuming process.

During the recent years, various tools for machine learning have been developed and many are currently under development. This section mentions two libraries that have become popular choices for deep learning applications. They make deep learning technology easily approachable for non-experienced users and also provide powerful tools for solving complex problems.

4.1.1 TensorFlow

TensorFlow (Martín Abadi et al. 2015) is an open source machine learning framework for numerical computing that was originally developed in Google's Machine Intelligence Research

organization. TensorFlow has APIs in Python and C languages. The software is distributed under Apache License 2.0 so it can be used for private or commercial purposes. TensorFlow supports Windows, Ubuntu, macOS and Raspbian operating systems. (“GitHub - tensorflow/tensorflow: An Open Source Machine Learning Framework for Everyone” 2019)

In TensorFlow, data is presented as multidimensional arrays. These arrays are called "tensors". Tensors are transformed with mathematical operations that can be computed with multiple CPUs or GPUs. TensorFlow’s use cases are not limited only to deep learning applications. (“GitHub - tensorflow/tensorflow: An Open Source Machine Learning Framework for Everyone” 2019)

4.1.2 Keras

Keras is an API for building neural networks with Python programming language. It is a high-level interface that simplifies the process of building and evaluating deep neural networks. Keras works as an interface between machine learning libraries and defines how deep learning models can be implemented and trained. Both CNNs and recurrent neural networks (RNNs) can be built with Keras and computation can be done on both CPU and GPU. There are three different backend engine options for Keras: TensorFlow, CNTK or Theano. Of these three, TensorFlow is the officially recommended option. (Chollet et al. 2015)

In Keras, neural networks are built by stacking layers on top of each other. The models are then compiled and fitted with the training data. Testing a model’s accuracy with testing data and making predictions with new data samples requires minimal coding. This is achieved with Keras’s sequential model. "Keras functional API" can be used for building more complex network architectures. (Chollet et al. 2015)

4.2 Cluster computing and stream processing

Cluster computing has become a common way to achieve scalable machine learning. In order to solve problems with big data efficiently, more powerful tools are needed and a demand for running application instances in a distributed way has emerged (Garcia et al. 2016).

A need for real time analysis of data is a typical requirement in many data science projects. For example, reading data from social media in real time is a common scenario. Many tools exist for handling these kind of data streams. This section mentions two popular tools for these kind of data gathering tasks.

4.2.1 Apache Storm

Apache Storm is an open source software with high scalability for online stream processing. Example use cases include real time data analytics and machine learning. It integrates with any queuing and database systems and it can work with any programming language. Apache Storm is actively developed: version 2.0.0 was released in May 2019. (“Apache Storm” 2019)

4.2.2 Apache Spark

Apache Spark is a popular cluster-computing framework used in data analytics. It has been open sourced and is being distributed under Apache License 2.0. Spark can be described as a general purpose framework that combines big data processing with analytics tools (“Apache Spark - Unified Analytics Engine for Big Data” 2019). It uses distributed file system (RAM) instead of disk-based read and write operations which makes fast computation possible. This is achieved with technology called RDD (Resilient Distributed Datasets) (Garcia et al. 2016). Spark has four main libraries: Spark SQL, Spark Streaming, MLlib and GraphX (see Figure 13). From the machine learning perspective, "MLlib" is an interesting library. It is one of the most popular machine learning library for big data (Garcia et al. 2016).

Apache Spark has competitors especially in streaming side such as Apache Storm and Apache Flink (Garcia et al. 2016). Chintapalli et al. (2016) made a comparison between Spark, Storm and Flink on their default settings and concluded that each of them have their advantages and disadvantages. They focused on streaming computation capabilities and found out that main difference is that Spark can handle higher throughput but it comes with the expense of higher latencies. Qian et al. (2016) made similar observations in their research of benchmarking Spark, Storm and Samza, which is another streaming platform for handling real-time data

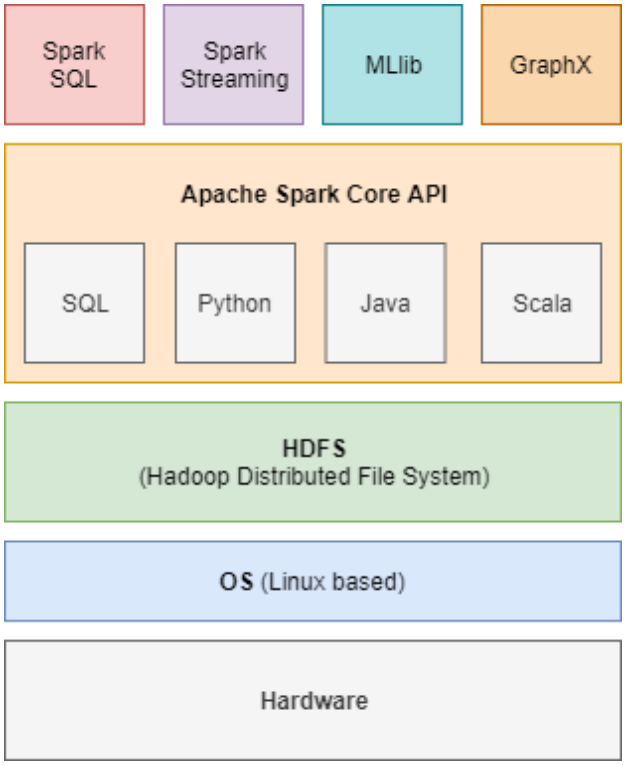

Figure 13. Apache Spark (“Apache Spark - Unified Analytics Engine for Big Data” 2019)

streams.

4.3 Messaging systems

In order to collect and analyze data from several different sources effectively, a system to handle messages between the data sources and the analyzer is needed. A common solution for this is to use a message broker. Message brokers are convenient in cases where vast amounts of data need to be sent between applications. Their main purpose is to accept and forward messages to their recipients. Throughput and reliability are the key aspects that need to be considered when deciding between message broker protocols (John and Liu 2017). For example, AMQP (asynchronous message queuing protocol) is one protocol used for communication. This protocol can be implemented with multiple software, RabbitMQ being one of them. Another popular choice in modern applications is to use Apache Kafka, which is a software that uses its own messaging protocol.

This section briefly introduces these two example software that can be used as a messaging system between data sources and target applications.

4.3.1 RabbitMQ

RabbitMQ is a popular open source message broker software that supports various messaging protocols, AMQP being one of them. It runs on Windows, Linux and MacOS operating systems and supports common programming languages such as Java, Python and PHP. (“RabbitMQ” 2019)

According to a research by John and Liu (2017), testing AMQP with RabbitMQ showed that latency is lower than Kafka’s implementation. They also note that AMQP may be the better choice for applications where security is considered an important quality attribute.

4.3.2 Apache Kafka

Apache Kafka is a streaming platform that can process and store data streams in real time. Kafka can also be used as a messaging system (“Apache Kafka” 2019). John and Liu (2017) compared Kafka and AMQP protocols and found out that Kafka has higher throughput and is well-suited for cases where performance is more important than reliability. They also concluded that if preservation of all the data is crucial (e.g. anomaly detection), then AMQP is the better option.

Kafka is based on publish-subscribe model but it also shares some benefits of a queuing model: Processing of data can be scaled as in queue-based systems but it also allows message broadcasting to multiple users simultaneously (“Apache Kafka” 2019). Figure 14 demonstrates how Kafka clusters connect producers and consumers with each other.

Kafka has four core APIs written in Java but the APIs are also available in various other languages (“Apache Kafka” 2019):

- Producer API: Publishes streams of data to Kafka cluster
- Consumer API: Allows consumers to read streams of data stored in topics
- Streams API: Allows an application to act as a stream processor that takes an input

from one or multiple topics and creates an output to another topic.

- Connectors API: Enables connection between Kafka other systems.

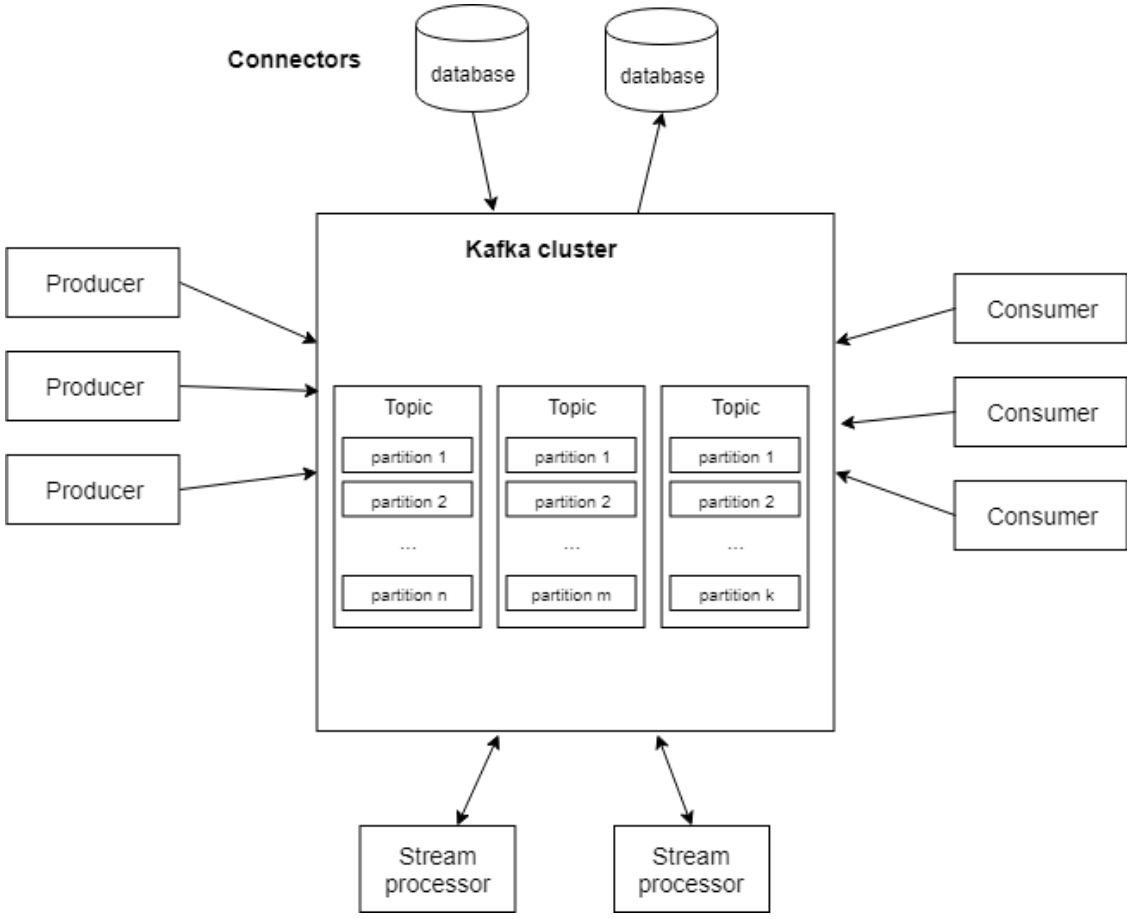

Figure 14. Kafka clusters are based on publish-subscribe pattern. (“Apache Kafka” 2019)

4.4 Deep learning platforms

Various tools are needed in order to utilize deep learning effectively. Data has to be acquired and cleansed, deep learning models have to be built, the results need to be visualized in an easily understandable way etc. Numerous platforms have been created for managing the deep learning project workflow in more efficient way. This section mentions two of these platforms: IBM Watson Studio and Databricks Unified Analytics Platform. These platforms were chosen because they offer good support not only for deep learning but also for other kind of machine learning applications. Both platforms are user friendly and have plenty of support material available so it’s easy to get started with building machine learning

applications with them.

4.4.1 IBM Watson Studio

IBM Watson Studio is a computing platform for data science and machine learning applications. It can be deployed in a cloud environment (IBM Watson Studio Cloud) or a local machine (IBM Watson Studio Desktop). The platform offers tools for building and training machine learning models and data analysis and it can be integrated with open-source deep learning frameworks such as TensorFlow and Keras. IBM Watson Studio also provides a drag-and-drop interface for building deep learning models. Figure 15 shows a view of the platforms UI. ("Watson Studio - Overview | IBM" 2019)

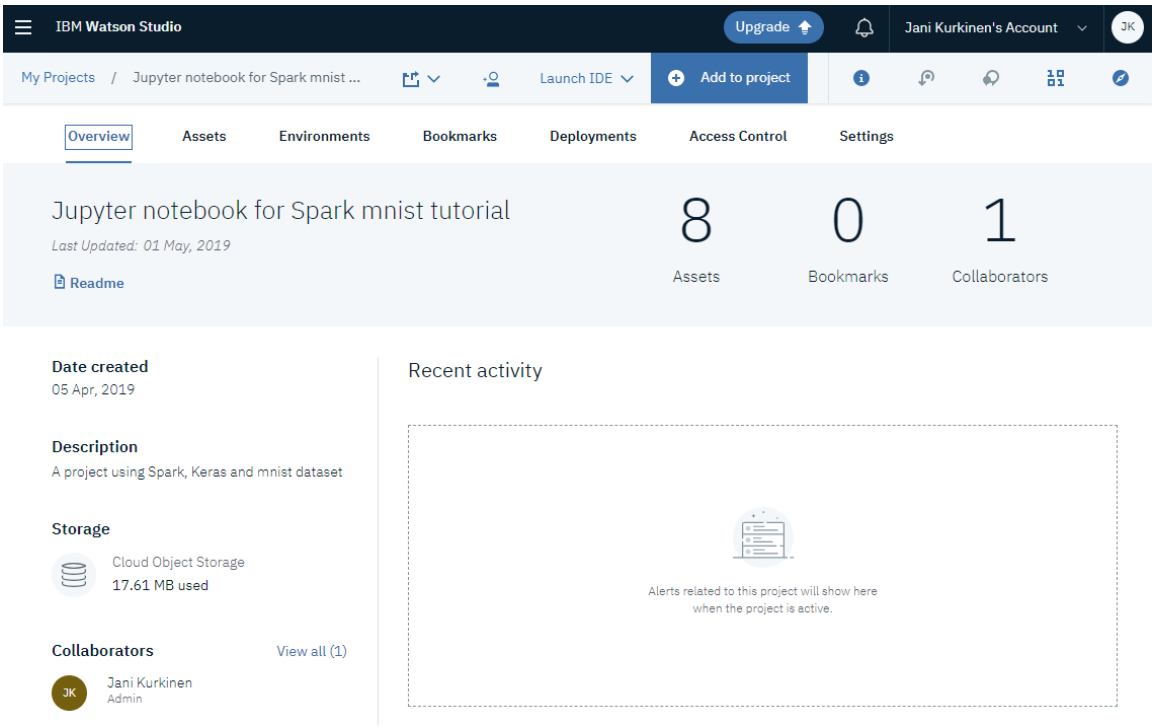

Figure 15. A screenshot of a project overview page in IBM Watson Studio Cloud.

4.4.2 Databricks Unified Analytics Platform

Databricks Unified Analytics Platform provides a platform for data analytics projects and tools for implementing deep learning models at scale. Managing libraries of computing clusters can be done with an easy-to-use UI. Getting started with Databrick's platform is

quite easy as there are plenty of training material available. The main features of the platform can be found from the dashboard view (see Figure 16).

More about the Databricks's platform will be discussed in Chapter 5 where it is used for a general image classification problem.

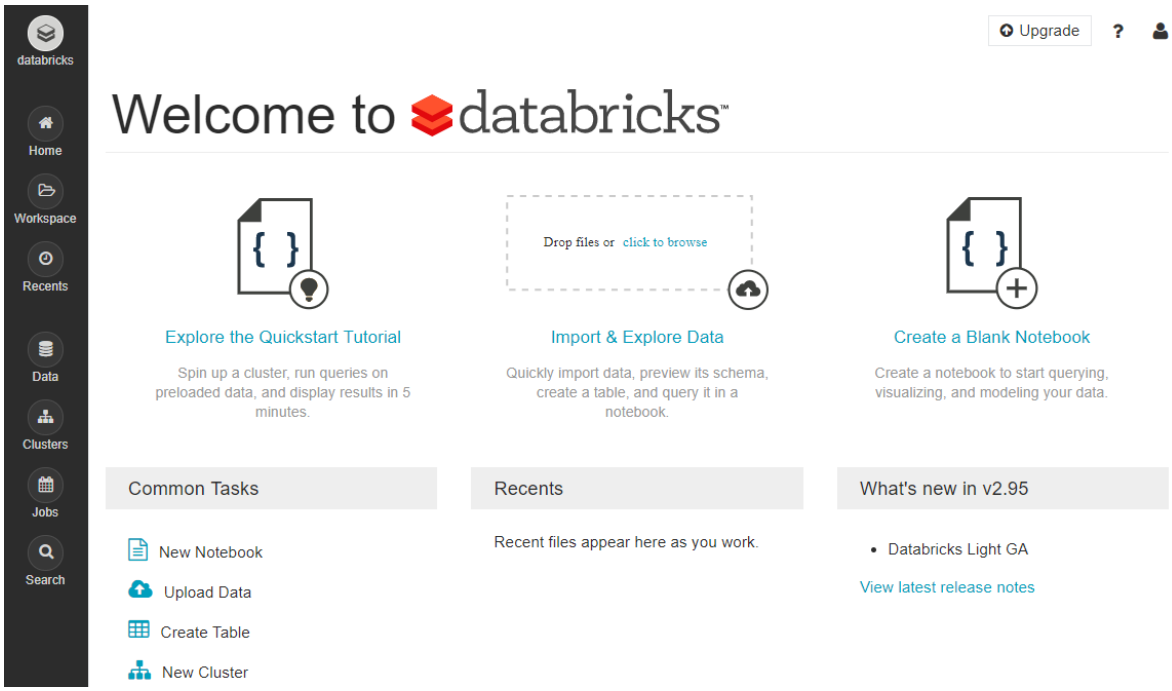

Figure 16. Screenshot of the dashboard of the Community Edition of Databricks Unified Analytics Platform.

5 Example implementation

In this chapter, some of the technologies mentioned in previous chapter are used to create an example solution for a typical image classification problem. The following example implementation largely follows the tutorials from "Deep Learning Pipelines on Databricks" (2017) and Vázquez (2018).

Community Edition of Databricks Unified Analytics Platform was chosen as a platform because of its affordability and ease of use. The codes were run on a notebook running in a 5.3 ML cluster with Apache Spark version 2.4.0 and Scala 2.11 and having 6 GB of RAM.

5.1 Setting up a notebook

In order to run codes in a notebook, it has to be attached to a computing cluster. The Community Edition of Databricks allows users to run only one cluster at a time but it is sufficient for demonstrating purposes. "Runtime: 5.3 ML (Scala 2.11, Spark 2.4.0)" -cluster was created for the notebook. An additional library, "1.5.0-spark2.4-s_2.11" -release of Deep Learning Pipelines was installed on the cluster. The notebook was then attached to the newly created cluster.

5.2 Loading images

This example implementation uses the flower data introduced in section 3.1. Images were first downloaded from TensorFlow website [1] to a local machine. After that, all the image data was uploaded to Databricks platform's filesystem (dbfs) using its "Add data" -feature. The file structure of the newly created directory can be viewed as follows

```
dbutils.fs.ls("/FileStore/tables/flower_photos/")
```

```
Out[1]:
[FileInfo(path='dbfs:/FileStore/tables/flower_photos/daisy/', name='daisy/', size=0),
 FileInfo(path='dbfs:/FileStore/tables/flower_photos/dandelion/', name='dandelion/', size=0),
 FileInfo(path='dbfs:/FileStore/tables/flower_photos/roses/', name='roses/', size=0),
```

[1] http://download.tensorflow.org/example_images/flower_photos.tgz

```
FileInfo(path='dbfs:/FileStore/tables/flower_photos/sample/', name='sample/', size=0),
FileInfo(path='dbfs:/FileStore/tables/flower_photos/sunflowers/', name='sunflowers/', size=0),
FileInfo(path='dbfs:/FileStore/tables/flower_photos/tulips/', name='tulips/', size=0)]
```

In addition, a folder for new testing images were created. Three of the images in "sample"-folder were downloaded from `https://www.publicdomainpictures.net/`. Two of the sample pictures ("dandelion1.jpg" and "rose1.jpg") are original as they were taken specifically for this thesis by the author. All of the sample images can be seen in Figure 19

After the images are loaded to dbfs, a Spark dataframe is created of the sample photos:

```
image_df = spark.read.format("image").load("dbfs:/FileStore/tables/flower_photos/sample/")
```

The images in the dataframe can be viewed with `display(image_df)`:

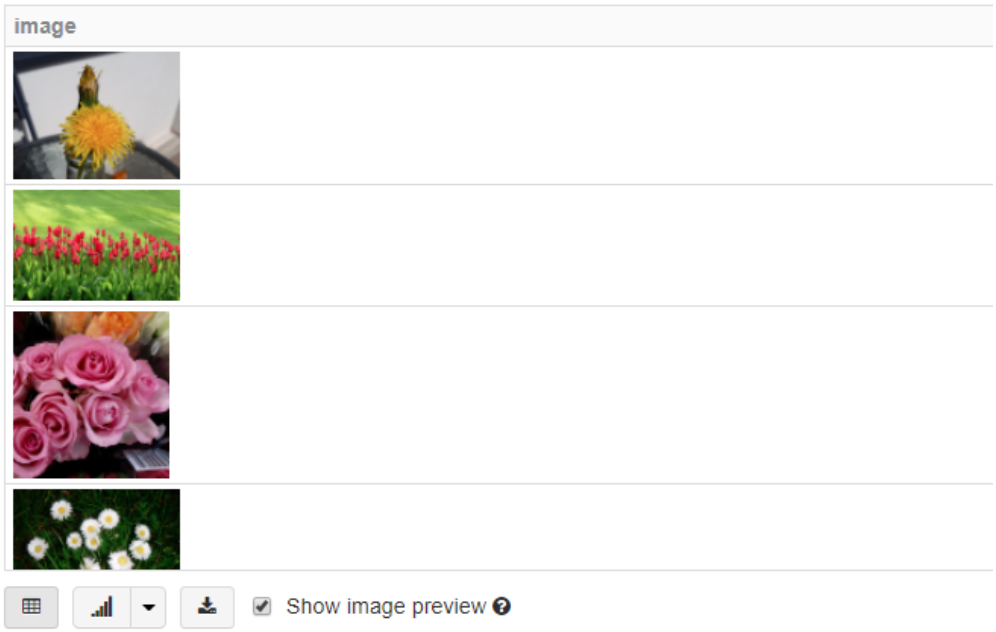

Figure 17. Sample photos viewed in a Databricks notebook.

Next, dataframes are created for each class. The classes were labeled as shown in the table below:

Flower type	Class label
daisy	0
dandelion	1
rose	2
sunflower	3
tulip	4

The images were read with "sparkdl"-package's "imageIO"-module and labeled with Spark's lit-function:

```
from pyspark.sql.functions import lit
from sparkdl.image import imageIO

daisy_df = imageIO.readImagesWithCustomFn("dbfs:/FileStore/tables/flower_photos/daisy/",
                    decode_f=imageIO.PIL_decode).withColumn("label", lit(0))
tulips_df = imageIO.readImagesWithCustomFn("dbfs:/FileStore/tables/flower_photos/tulips/",
                    decode_f=imageIO.PIL_decode).withColumn("label", lit(1))
roses_df = imageIO.readImagesWithCustomFn("dbfs:/FileStore/tables/flower_photos/roses/",
                    decode_f=imageIO.PIL_decode).withColumn("label", lit(2))
sunflowers_df = imageIO.readImagesWithCustomFn("dbfs:/FileStore/tables/flower_photos/sunflowers/",
                    decode_f=imageIO.PIL_decode).withColumn("label", lit(3))
tulips_df = imageIO.readImagesWithCustomFn("dbfs:/FileStore/tables/flower_photos/tulips/",
                    decode_f=imageIO.PIL_decode).withColumn("label", lit(4))
```

The data is then divided into training and testing data randomly. The split was chosen to be 60% to the training and 40% to the testing dataframes. The dataframes are then combined to form two dataframes with similar split.

```
daisy_train, daisy_test = daisy_df.randomSplit([0.6, 0.4])
dandelion_train, dandelion_test = dandelion_df.randomSplit([0.6, 0.4])
roses_train, roses_test = roses_df.randomSplit([0.6, 0.4])
sunflowers_train, sunflowers_test = sunflowers_df.randomSplit([0.6, 0.4])
tulips_train, tulips_test = tulips_df.randomSplit([0.6, 0.4])

train01 = daisy_train.unionAll(dandelion_train)
test01 = daisy_test.unionAll(dandelion_test)

train012 = train01.unionAll(roses_train)
```

```
test012 = test01.unionAll(roses_test)

train0123 = train012.unionAll(sunflowers_train)
test0123 = test012.unionAll(sunflowers_test)

train01234 = train0123.unionAll(tulips_train)
test01234 = test0123.unionAll(tulips_test)
```

Dataframes "train01234" and "test01234" now contain all the samples of the dataset.

5.3 Using the model to make predictions

In this section, the model that was trained in section 3.3 is used to predict classes for images located in the sample folder.

The trained model, "flower_cnn2.h5", was uploaded to dbfs in a similar way than the images in section 5.2. DL Pipelines' KerasImageFileTransformer is used to create a transformer that is used to apply the model to the images in the sample folder.

```
transformer = KerasImageFileTransformer(inputCol="uri",
                                        outputCol="predictions",
                                        modelFile='/tmp/flower_cnn2.h5',
                                        imageLoader=loadAndPreprocess,
                                        outputMode="vector")
```

A helper function was created for loading images:

```
def loadAndPreprocess(uri):
    image = img_to_array(load_img(uri, target_size=(100, 100)))
    image = np.expand_dims(image, axis=0)
    return image
```

A new predictions dataframe is created from the transformer:

```
predictions_df = transformer.transform(uri_df)
```

The dataframe can now be displayed with `display(predictions_df)`:

uri	predictions
/dbfs/FileStore/tables/flower_photos/sample/daisy1.jpg	[1,5,[],[1,0,0,0,0]]
/dbfs/FileStore/tables/flower_photos/sample/dandelion1.jpg	[1,5,[],[1,0,0,0,0]]
/dbfs/FileStore/tables/flower_photos/sample/rose1.jpg	[1,5,[],[0,0,1,0,0]]
/dbfs/FileStore/tables/flower_photos/sample/sunflower1.jpg	[1,5,[],[0,0,0,1,0]]
/dbfs/FileStore/tables/flower_photos/sample/tulip1.jpg	[1,5,[],[0,0,0,0,1]]

Figure 18. Prediction dataframe displayed in Databricks

The predicted classes are shown in the last column of the "predictions" dataframe. The model was able to classify four out of five testing images correctly. The image of a dandelion was mistaken for a daisy, possibly because of the white background of the image.

Figure 19. Images in "sample"-folder used for testing

5.4 Transfer learning with Deep Learning Pipelines and InceptionV3

Sometimes building and training neural networks from scratch is not necessary. In many cases, transfer learning can help to solve problems effectively. In this section, a pre-trained CNN is used to recognize images from two different classes. Deep Learning Pipelines and InceptionV3 are used as demonstrated in “Deep Learning Pipelines on Databricks” (2017) and Vázquez (2018).

Spark ML Pipelines is a set of APIs that make using Spark dataframes for machine learning tasks easier. An ML pipeline is a representation of a workflow that consists of several algorithms used for machine learning. It has two main components: transformers and estimators.

According to “ML Pipelines - Spark 2.4.2 Documentation” (2018), "a transformer is an algorithm which can transform one DataFrame into another DataFrame" whereas an estimator is defined as "an algorithm which can be fit on a DataFrame to produce a Transformer". A pipeline can run several of these algorithms in a sequence. (“ML Pipelines - Spark 2.4.2 Documentation” 2018)

Databricks DL Pipelines is an open-source framework for deep learning. It enables deep learning models built with Keras to be scaled out with Spark (“Deep Learning Pipelines for Apache Spark” 2019).

In this example, a simple ML pipeline is created from a DL Pipelines’ DeepImageFeaturizer and Spark MLlib’s logistic regression algorithm. A pre-trained CNN, InceptionV3, is used. It has been trained to detect 1000 different classes for images but it can be used for other types of classification tasks too. The last layer of the network can be removed with DeepImageFeaturizer, which is a class from "sparkdl" module. It pops the last layer of a pre-trained Keras model (InceptionV3, Xception, ResNet50, VGG16 and VGG19 are currently supported) (“Deep Learning Pipelines for Apache Spark” 2019). A logistic regression algorithm is used in the last layer of the network to calculate probabilities for each class.

```
from pyspark.ml.classification import LogisticRegression
from pyspark.ml import Pipeline
from sparkdl import DeepImageFeaturizer

# Combines a featurizer with a logistic regression algorithm
featurizer = DeepImageFeaturizer(inputCol="image",
                                 outputCol="features",
                                 modelName="InceptionV3")
lr = LogisticRegression(maxIter=10, regParam=0.05,
                        elasticNetParam=0.3, labelCol="label")
p = Pipeline(stages=[featurizer, lr])

# model is fitted with the training data using a Spark ML pipeline
p_model = p.fit(train_df)
```

The model has now been trained with the training dataframe.

5.4.1 Model evaluation

The accuracy of the model was evaluated with Spark's MulticlassClassificationEvaluator:

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

# model evaluation
tested_df = p_model.transform(test_df)
evaluator = MulticlassClassificationEvaluator(metricName="accuracy")
print("Test accuracy = " +
   str(evaluator.evaluate(tested_df.select("prediction", "label"))))
```

This gave an accuracy of ~0.968 for the testing set. It shows that the images in the testing set can be classified quite accurately with a model that was not specifically trained for detecting differences between tulips and daisies. However, the original InceptionV3 model was trained with images of daisies. Therefore, it can tell a difference between an image with the same label that was used in the training process and an image it has not learned to recognize.

5.4.2 Making predictions

Predicting classes for images can be done with transformers. In this section, predicting classes for three images in a sample folder is demonstrated (Figure 20 shows the images as a dataframe). DL Pipelines' DeepImagePredictor is used as a transformer in the pipeline.

```
from sparkdl import DeepImagePredictor

# sample images are read into a dataframe
image_df = spark.read.format("image")
              .load("dbfs:/FileStore/tables/flower_photos/sample2/")

predictor = DeepImagePredictor(inputCol="image",
                               outputCol="predicted_labels",
```

```
                                    modelName="InceptionV3",
                                    decodePredictions=True,
                                    topK=10)
predictions_df = predictor.transform(image_df)
```

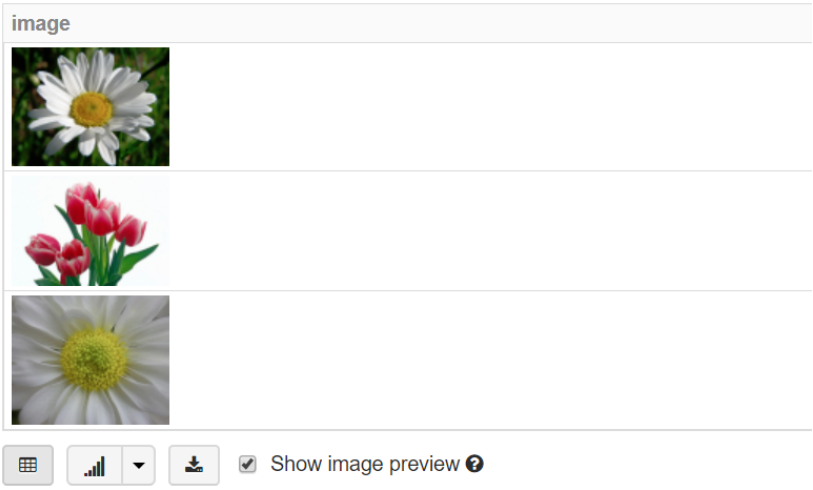

Figure 20. Images in predictions dataframe. These images are samples of TensorFlow's "tf_flowers"-dataset that was introduced in Section 3.1.

The output column was named to "predicted_labels". The results can be viewed by selecting it from the predictions dataframe:

```
predictions_df.select("predicted_labels").show(truncate=False,
n=3)
```

```
|predicted_labels
+----------------------------------------------------------------------------------------+
|[[n11939491, daisy, 0.8600176],          [n02219486, ant, 8.88716E-4],
  [n07930864, cup, 6.500126E-4],          [n02107908, Appenzeller, 6.187557E-4],
  [n03016953, chiffonier, 5.351678E-4],   [n02206856, bee, 5.322351E-4],
  [n01756291, sidewinder, 5.2060944E-4],  [n01795545, black_grouse, 5.1415246E-4],
  [n02071294, killer_whale, 5.036278E-4], [n02190166, fly, 5.01368E-4]]|
|[[n04522168, vase, 0.5255079],           [n03991062, pot, 0.07667497],
  [n07930864, cup, 0.042901672],          [n03476684, hair_slide, 0.037432313],
  [n03443371, goblet, 0.018425234],       [n03950228, pitcher, 0.015975073],
  [n04476259, tray, 0.011403176],         [n03482405, hamper, 0.009134999],
  [n03930313, picket_fence, 0.008365866], [n02909870, bucket, 0.006912277]]|
|[[n11939491, daisy, 0.89181423],         [n02219486, ant, 0.0012404543],
  [n02206856, bee, 8.130483E-4],          [n02190166, fly, 6.0380466E-4],
  [n02165456, ladybug, 6.005448E-4],      [n02281406, sulphur_butterfly, 5.320976E-4],
```

```
[n04599235, wool, 4.665371E-4],          [n02112018, Pomeranian, 4.6253452E-4],
[n07930864, cup, 4.4006197E-4],          [n02177972, weevil, 4.243419E-4]]|
```

This shows the 10 most probable classes for each image. For example, the model is ~86.0% sure that the first image in the sample dataframe has a daisy in it. The model identifies the second image as a vase with ~52.6% certainty and the third image is most likely (~89.2%) a picture of a daisy.

6 Discussion

As previous research has shown, deep learning has a lot of potential to solve difficult problems. It has already proven to be an effective method to approach certain problems in fields such as medical diagnosis or natural language processing. There is a high demand for deep learning experts and it's safe to say that people who master various machine learning methods will be in high-demand in the future. Companies and organizations have access to a vast amount of data but making use of it is always a challenge. For example, understanding and predicting customer behavior is something that many companies want to do better. Defining a problem is an important first step but deciding how to solve it is also a relevant question.

This paper demonstrated how to use Databricks Unified Analytics Platform with Keras models and Spark dataframes. These tools proved to be easy-to-use options for a simple image classification use case. Using a pre-built CNN was simple and training it with a dataset was a straightforward process. Accuracy of the model (~60.9%) is was not high but it was achieved with making only small changes to the network. Transfer learning with a pre-trained CNN also gave good results but more complex examples should be examined. The amount of data used in this example was quite small, only 3670 samples. It doesn't really show the benefits of distributed nature of Spark dataframes but it is still an adequate dataset for demonstration purposes.

An extensive research of available tools for deep learning should be done. Thorough evaluation of their features and performance would be useful for people who are interested in making use of deep learning in their projects. New tools are being developed and existing ones are being improved so keeping up with the latest developments can be challenging. It should be noted that image classification is only one type of a problem that can be approached with deep learning. Therefore, testing how easily the tools demonstrated in Chapter 5 are suited for other problems, such as spoken language recognition or anomaly detection in real-time data streams could be an interesting project.

7 Conclusion

A typical deep learning data science project workflow has many steps. Choosing tools for the task is one of them. The goal of this thesis was to provide some basic knowledge about the theoretical aspects of deep learning and examine on how to apply deep learning in practical applications with modern deep learning tools. Some technologies were introduced and a simple example of image classification with deep learning was demonstrated. A pre-built CNN was modified to handle the dataset that was used in this thesis. In addition, transfer learning with a pre-trained CNN was shown. The tools used in Chapter 5 proved effective for a simple task but they should be capable of handling much larger datasets too.

It should be noted that there are many other tools available and some of them may suit different use cases better than others. In any case, many capable options exist for data scientists and engineers to use for various deep learning applications. Many of the current tools are open source software so getting started with deep learning is now affordable and easily available for everybody. Deep learning has proven to be an effective method to solve difficult problems and it will be interesting to see what kind of new developments will emerge in the coming years.

Bibliography

"Apache Kafka". 2019. Visited on June 15, 2019. https://kafka.apache.org/intro.

"Apache Spark - Unified Analytics Engine for Big Data". 2019. Visited on June 18, 2019. https://spark.apache.org/.

"Apache Storm". 2019. Visited on June 16, 2019. http://storm.apache.org/.

Bashroush, Rabih, Ivor Spence, Peter Kilpatrick, and John Brown. 2006. "Towards more flexible architecture description languages for industrial applications". In *European Workshop on Software Architecture,* 212–219. Springer.

Carroll, Mariana, Alta Van Der Merwe, and Paula Kotze. 2011. "Secure cloud computing: Benefits, risks and controls". In *2011 Information Security for South Africa,* 1–9. IEEE.

Chintapalli, S., D. Dagit, B. Evans, R. Farivar, T. Graves, M. Holderbaugh, Z. Liu, et al. 2016. "Benchmarking Streaming Computation Engines: Storm, Flink and Spark Streaming". In *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW),* 1789–1792. doi:10.1109/IPDPSW.2016.138.

Chollet, François, et al. 2015. "Keras". Visited on June 15, 2019. https://keras.io/.

"Datasets | TensorFlow Datasets | TensorFlow". 2019. Visited on May 2, 2019. https://www.tensorflow.org/datasets/datasets#tf_flowers.

"Deep Learning Pipelines for Apache Spark". 2019. Visited on April 16, 2019. https://github.com/databricks/spark-deep-learning.

"Deep Learning Pipelines on Databricks". 2017. Visited on April 16, 2019. https://databricks-prod-cloudfront.cloud.databricks.com/public/4027ec902e239c93eaaa8714f173bcfc/5669198905533692/3647723071348946/3983381308530741/latest.html.

Ding, Zuohua, Zhenbang Chen, and Jing Liu. 2008. "A rigorous model of service component architecture". *Electronic Notes in Theoretical Computer Science* 207:33–48.

Dumoulin, Vincent, Ishmael Belghazi, Ben Poole, Olivier Mastropietro, Alex Lamb, Martin Arjovsky, and Aaron Courville. 2016. "Adversarially learned inference". *arXiv preprint arXiv:1606.00704.*

Esteva, Andre, Brett Kuprel, Roberto A Novoa, Justin Ko, Susan M Swetter, Helen M Blau, and Sebastian Thrun. 2017. "Dermatologist-level classification of skin cancer with deep neural networks". *Nature* 542 (7639): 115.

Garcia, Salvador, Sergio Ramirez-Gallego, Julian Luengo, Jose Manuel Benitez, and Francisco Herrera. 2016. "Big data preprocessing: methods and prospects". *Big Data Analytics* 1 (1): 9.

Gersey, Bálint. 2018. "Generative Adversarial Networks". Master's thesis, University of Cambridge.

"GitHub - tensorflow/tensorflow: An Open Source Machine Learning Framework for Everyone". 2019. Visited on May 2, 2019. `https://github.com/tensorflow/tensorflow`.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning.* `http://www.deeplearningbook.org`. MIT Press.

Goodfellow, Ian, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. "Generative Adversarial Nets". In *Advances in Neural Information Processing Systems 27,* edited by Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, 2672–2680. Curran Associates, Inc. `http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf`.

Hesamifard, Ehsan, Hassan Takabi, Mehdi Ghasemi, and Catherine Jones. 2017. "Privacy-preserving Machine Learning in Cloud". In *Proceedings of the 2017 on Cloud Computing Security Workshop,* 39–43. CCSW '17. Dallas, Texas, USA: ACM. doi:`10.1145/3140649.3140655`.

"Johdatus tekoälyn taustalla olevaan matematiikkaan - TIM". 2016. Visited on February 24, 2019. https://tim.jyu.fi/view/kurssit/tie/tiep1000/tekoaly-mat/moniste.

John, Vineet, and Xia Liu. 2017. "A survey of distributed message broker queues". *arXiv preprint arXiv:1704.00411.*

Karras, Tero, Samuli Laine, and Timo Aila. 2018. "A Style-Based Generator Architecture for Generative Adversarial Networks". *arXiv preprint arXiv:1812.04948.*

Kesarwani, Manish, Bhaskar Mukhoty, Vijay Arya, and Sameep Mehta. 2018. "Model extraction warning in mlaas paradigm". In *Proceedings of the 34th Annual Computer Security Applications Conference,* 371–380. ACM.

Koskimies, Kai, and Tommi Mikkonen. 2005. *Ohjelmistoarkkitehtuurit.* Talentum.

Laplante, Phillip A, Jia Zhang, and Jeffrey Voas. 2008. "What's in a Name? Distinguishing between SaaS and SOA". *IT Professional Magazine* 10 (3): 46.

LeCun, Yann, Corinna Cortes, and Christopher J.C. Burges. 1998. "MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges". Visited on March 1, 2019. http://yann.lecun.com/exdb/mnist/.

Li, Li Erran, Eric Chen, Jeremy Hermann, Pusheng Zhang, and Luming Wang. 2017. "Scaling machine learning as a service". In *International Conference on Predictive Applications and APIs,* 14–29.

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, et al. 2015. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems.* Software available from tensorflow.org. https://www.tensorflow.org/.

Miikkulainen, Risto, Jason Liang, Elliot Meyerson, Aditya Rawal, Daniel Fink, Olivier Francon, Bala Raju, Hormoz Shahrzad, Arshak Navruzyan, Nigel Duffy, et al. 2019. "Evolving deep neural networks". In *Artificial Intelligence in the Age of Neural Networks and Brain Computing,* 293–312. Elsevier.

Mirza, Mehdi, and Simon Osindero. 2014. "Conditional generative adversarial nets". *arXiv preprint arXiv:1411.1784.*

Mitchell, Thomas M. 1997. *Machine Learning.* 1st edition. New York, NY, USA: McGraw-Hill, Inc.

"ML Pipelines - Spark 2.4.2 Documentation". 2018. Visited on April 16, 2019. `https://spark.apache.org/docs/latest/ml-pipeline.html`.

Moradi, Elaheh, Antonietta Pepe, Christian Gaser, Heikki Huttunen, Jussi Tohka, Alzheimer's Disease Neuroimaging Initiative, et al. 2015. "Machine learning framework for early MRI-based Alzheimer's conversion prediction in MCI subjects". *Neuroimage* 104:398–412.

Odena, Augustus, Christopher Olah, and Jonathon Shlens. 2016. "Conditional image synthesis with auxiliary classifier gans". *arXiv preprint arXiv:1610.09585.*

"Photo Editing with Generative Adversarial Networks (Part 1)". 2017. Visited on November 23, 2018. `https://devblogs.nvidia.com/photo-editing-generative-adversarial-networks-1/`.

Pilli-Sihvola, Viljo. 2010. "Intelligence as a Service". Master's thesis, University of Jyväskylä.

Pumarola, Albert, Antonio Agudo, Alberto Sanfeliu, and Francesc Moreno-Noguer. 2018. "Unsupervised Person Image Synthesis in Arbitrary Poses". In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition,* 8620–8628.

Qian, Shilei, Gang Wu, Jie Huang, and Tathagata Das. 2016. "Benchmarking modern distributed streaming platforms". In *2016 IEEE International Conference on Industrial Technology (ICIT),* 592–598. IEEE.

"RabbitMQ". 2019. Visited on June 15, 2019. `https://www.rabbitmq.com/`.

Ribeiro, Mauro, Katarina Grolinger, and Miriam AM Capretz. 2015. "Mlaas: Machine learning as a service". In *2015 IEEE 14th International Conference on Machine Learning and Applications (ICMLA),* 896–902. IEEE.

Saraee, Mo, and Charith Silva. 2018. "A new data science framework for analysing and mining geospatial big data". In *Proceedings of the International Conference on Geoinformatics and Data Analysis,* 98–102. ACM.

Schuchmann, Marcel. 2018. “Designing a cloud architecture for an application with many users”. Master’s thesis, University of Jyväskylä.

Siddiqa, Aisha, Ahmad Karim, and Abdullah Gani. 2017. “Big data storage technologies: a survey”. *Frontiers of Information Technology & Electronic Engineering* 18 (8): 1040–1070.

Singhoff, Frank, Alain Plantec, Stéphane Rubini, Vincent Gaudel, Shuai Li, Christian Fotsing, Laurent Lemarchand, Pierre Dissaux, and Jérôme Legrand. 2019. “How architecture description languages help schedulability analysis: a return of experience from the Cheddar project”.

“Software Architecture: Architecture Description Languages”. 2012. Visited on February 19, 2019. https://www.slideshare.net/henry.muccini/software-architecture-architecture-description-languages.

Szegedy, C., V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. 2016. “Rethinking the Inception Architecture for Computer Vision”. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR),* 2818–2826. doi:10.1109/CVPR.2016.308.

Valjakka, Mikko. 2012. “Uuden SaaS-ohjelmistotuotteen suunnittelu: Case AnyCase Asianhallinta”, Tampere University of Applied Sciences.

Vázquez, Favio. 2018. “Deep Learning With Apache Spark - Part 2”. Visited on April 12, 2019. https://towardsdatascience.com/deep-learning-with-apache-spark-part-2-2a2938a36d35.

“Watson Studio - Overview | IBM”. 2019. Visited on April 29, 2019. https://www.ibm.com/cloud/watson-studio.

Zenati, Houssam, Chuan Sheng Foo, Bruno Lecouat, Gaurav Manek, and Vijay Ramaseshan Chandrasekhar. 2018. “Efficient gan-based anomaly detection”. *arXiv preprint arXiv:1802.06222.*