CZECH TECHNICAL
UNIVERSITY
IN PRAGUE

FACULTY
OF MECHANICAL
ENGINEERING

BACHELOR
THESIS

IMAGE RECOGNITION WITH DEEP LEARNING FOR WEB SCRAPPED IMAGES

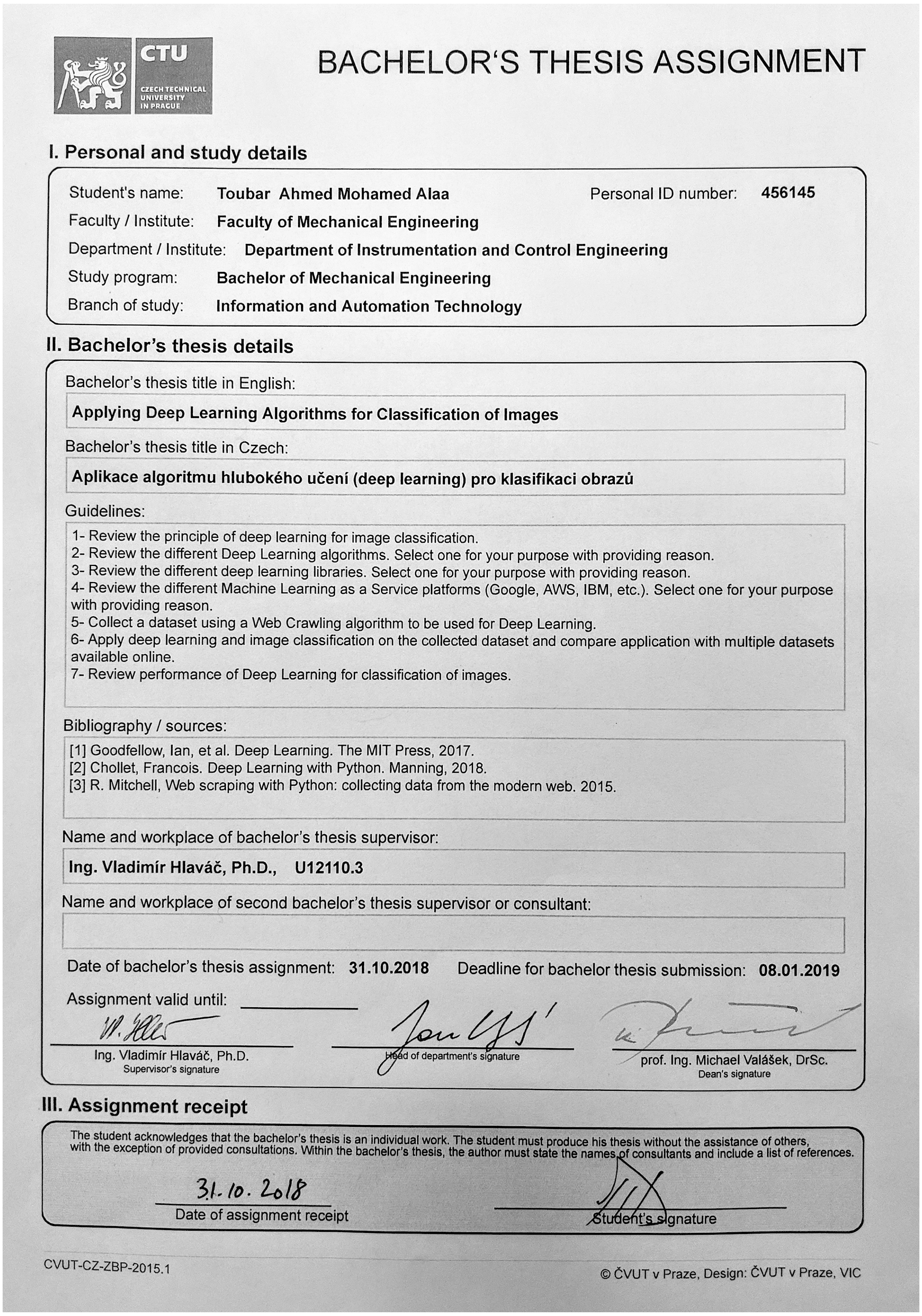
2019

AHMED
TOUBAR

Supervisor: Ing. Vladimír Hlaváč, Ph.D.,

Annotation Sheet

Name	Ahmed
Surname	Toubar
Title Czech	Aplikace algoritmu hlubokého učení (deep learning) pro klasifikaci obrazů
Title English	Applying Deep Learning Algorithms for Classification of Images
Academic year	2018/2019
Language	English
Department	Instrumentation & Control Engineering
Specialization	Information & Automation Technology
Supervisor	Ing. Vladimír Hlaváč, Ph.D.
Reviewer	
Keywords	Deep Learning, Python, TensorFlow, Keras, Computer Vision, Image Classification



BACHELOR'S THESIS ASSIGNMENT

CTU CZECH TECHNICAL UNIVERSITY IN PRAGUE

I. Personal and study details

Student's name:	Toubar Ahmed Mohamed Alaa	Personal ID number:	456145
Faculty / Institute:	Faculty of Mechanical Engineering		
Department / Institute:	Department of Instrumentation and Control Engineering		
Study program:	Bachelor of Mechanical Engineering		
Branch of study:	Information and Automation Technology		

II. Bachelor's thesis details

Bachelor's thesis title in English:

Applying Deep Learning Algorithms for Classification of Images

Bachelor's thesis title in Czech:

Aplikace algoritmu hlubokého učení (deep learning) pro klasifikaci obrazů

Guidelines:

1- Review the principle of deep learning for image classification.
2- Review the different Deep Learning algorithms. Select one for your purpose with providing reason.
3- Review the different deep learning libraries. Select one for your purpose with providing reason.
4- Review the different Machine Learning as a Service platforms (Google, AWS, IBM, etc.). Select one for your purpose with providing reason.
5- Collect a dataset using a Web Crawling algorithm to be used for Deep Learning.
6- Apply deep learning and image classification on the collected dataset and compare application with multiple datasets available online.
7- Review performance of Deep Learning for classification of images.

Bibliography / sources:

[1] Goodfellow, Ian, et al. Deep Learning. The MIT Press, 2017.
[2] Chollet, Francois. Deep Learning with Python. Manning, 2018.
[3] R. Mitchell, Web scraping with Python: collecting data from the modern web. 2015.

Name and workplace of bachelor's thesis supervisor:

Ing. Vladimír Hlaváč, Ph.D., U12110.3

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **31.10.2018** Deadline for bachelor thesis submission: **08.01.2019**

Assignment valid until: ____________

[illegible]	[illegible]	[illegible]
Ing. Vladimír Hlaváč, Ph.D.	Head of department's signature	prof. Ing. Michael Valášek, DrSc.
Supervisor's signature		Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

31. 10. 2018	[illegible]
Date of assignment receipt	Student's signature

CVUT-CZ-ZBP-2015.1

Declaration

I declare that this bachelor thesis entitled “**Applying Deep Learning Algorithms for Classification of Images**” is my own work performed under the supervision of Ing. Vladimir Hlavac, Ph.D., with the use of the literature presented at the end of my bachelor thesis in the list of references.

In Prague 12.02.2019 Ahmed Toubar

Abstract

This thesis aims to explore Image Classification using Artificial Intelligence (AI) in the general sense and Deep Learning in the more specific sense. AI has gone through a lot of phases ever since the term was first coined by the computer scientist John McCarthy in 1955[1]. This thesis will introduce the topic of Deep Learning as a branch of Machine Learning and Artificial Intelligence, then it will explore how data is collected and prepared for Deep Learning. The paper will also discuss the challenges faced when applying Deep Learning on a brand-new dataset versus using one that is available and ready for Deep Learning applications.

Furthermore, the thesis will review the different deep learning algorithms, libraries as well as the cloud services that offer Deep Learning virtual machines as a service. Finally, an evaluation of the efficiency and effectiveness of Deep Learning as a means of classification of images will be included as a conclusion to the thesis.

Table of Contents

Introduction

In science fiction films, the future that was once imagined is now part of our reality. It is undeniably true that we human beings are on the brink of an eternal revolution– the Artificial Intelligence Revolution (AI) [1].
It all started with the industrial revolution in the 1800s that changed how we interact with the world in every possible way. It has disrupted transportation, communication and raised the average standard of living by offering jobs to workers at factories that never existed before[2]. But still there were a lot of manual wearing tasks that consumed one's mind, effort and time. The valuable resources that lets a person be creative and innovate relentlessly.
Then, in 1926, Julius Edgar Lilienfeld patented the transistor. The single component that opened the gate to yet another revolution - The Computer Revolution. The invention of the computer again changed how we interact with the world and with each other. It has put an end to manual exhausting tasks and became the gateway to limitless innovation[3]. However, those computers were distant and did not communicate effectively. And so, came the Internet. Meanwhile, two remarkable gentlemen were working on a communication system experiment to replace their existing networking system at ARPANET. This experiment went out of hands and laid the foundations of what we know today as the internet[4]. The internet came to revolutionize informatics and communication. It disrupted every industry there is, created jobs, changed the world to the better and more importantly left us with enormous amounts of data. Data that can make computers learn like us humans do.
Until now in our timeline, computers were programmed to do every single task. A programmer had to explicitly instruct it, for example, to store the value of X and Y, then add them together and print the result to the screen. This was still one of our greatest inventions as humans. But it made computer scientists think about the what if – What if computers can think for themselves?
And so, John McCarthy thought about this particular question and coined the term "Artificial Intelligence" in 1955 at the world's first AI conference. He was the pioneer that imagined the possibility of computers reasoning like how humans do. That paved the way to what is currently taking place in our world – An AI Revolution[5]. In the upcoming chapter I will discuss the evolution of AI and how it is transforming how we interact with the world today.

Artificial Intelligence (AI)

Artificial Intelligence (AI) is defined as the intelligence executed by machines that take actions driven towards succeeding at a goal. It leverages self-learning systems and uses various tools such as pattern recognition and data mining. It functions in a way similar to how a normal human brain works in day-to-day tasks from common-sense reasoning to behaving in a social manner. AI has been one of science's most incomprehensible subjects in Computer Science due to its significance in our everyday lives and the massive potential it has to continue changing people's lives. It is considered to be the fourth industrial revolution where automation will be intense, and connectivity will be very abundant. The applications that AI has are vast and are in the way computers are being used in society.[6], [7]

History & Evolution

Although the term Artificial Intelligence was first used at a conference on the subject in 1956 by John McCarthy, the journey to understanding if machines can think began even before that. A system which showcases people's own knowledge and understanding of artificial intelligence was proposed by Vannevar Bush, and only 5 years later, a paper was written by Alan Turing discussing the topic of machines being able to simulate human beings and perform intelligent activities such as playing Chess. In his landmark paper, Turing noted that the term "thinking" is difficult to define and derived his famous Turing Test, which states it was reasonable to say that a machine is thinking that if it could carry out a conversation via a teleprinter that could be identified as indistinguishable compared to a conversation with a human being. Turing Test was utilized as a green light to argue that a thinking machine was at least possible and it was the first serious proposal in the study of artificial intelligence. [7] There have been numerous advances over the past 60 years mainly in search and machine learning algorithms and integrating statistical analysis. Artificial Intelligence is divided broadly into three phases: artificial narrow intelligence (ANI), artificial general intelligence (AGI) and artificial super intelligence (ASI). The first stage is restricted to only focusing on one functional area. AGI, the second stage, covers multiple fields such as abstract thinking, reasoning and problem solving and is considered an advanced level. The final stage (ASI) is where artificial intelligences outperforms and exceeds human intelligence throughout all fields. [6]

As displayed in the graph below, the transition between ANI and AGI (first and second phases) has taken a long time. Scientists believe that we are nowadays almost completing ANI, the first phase, and will slowly be completing the transition to the second phase, AGI, where the intelligence of machines is being equated to that of humans, which is a drastic achievement. It is believed that by the end of the decade, we will be entering the phase AGI and the industry of artificial intelligence is expected to start exploding and its impact on society and businesses will begin emerging.[6]

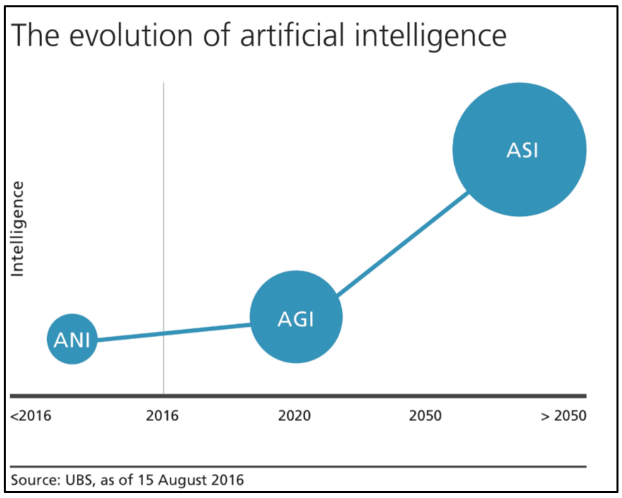


Figure 0-1 – Evolution of AI

By the end of the decade, a true autonomy will commence. AI-powered software and machines will no longer need human supervision and will embark on a new path of them becoming alert beings. Having said that, the next few years will witness tremendous industry growth and will have a remarkable expansion on its expansion on businesses and society. Estimates show that by 2020, the industry's marketplace by revenue should more than double compared to its value in 2015. As shown in the graph below, at a 20% annual growth rate, it is expected to become a USD 12.5 billion industry driven by exponential improvements. [6]

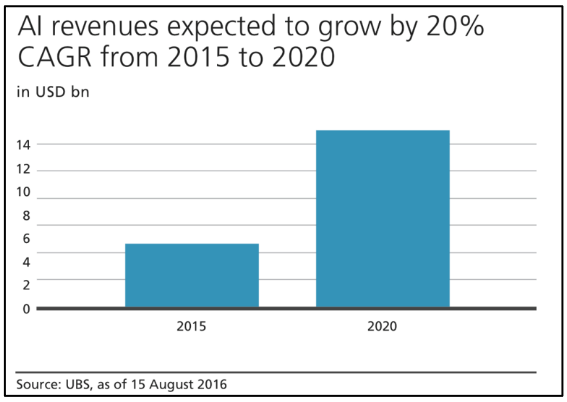


Figure 0-2 – AI Revenue growth expectations

The Increasing Value of Data

Our world today hosts an unbelievably remarkable vast amount of data in digital format and it's getting even vaster very quickly. This amount of digital information makes it possible to do things that could not be done in the past such as prevent diseases, defeat crime and identify business trends. An example includes Wal-Mart, an American retail giant, which handles more than a million customer interactions per hour and feeds "databases estimated at more than 2.5 petabytes—the equivalent of 167 times the books in America's Library of Congress" [8]

The unimaginably vast amount of data available in today's world can be utilized to unlock new sources of monetary and economic value as well as provide new insights into science if managed well. The biggest and most obvious reason for the information explosion is technology. The functions and capabilities of digital devices are massively increasing, and their prices are plummeting, which means that sensors are digitizing massive amounts of information that was historically unavailable. There are currently more than 4.5 billion mobile phone subscriptions in the world and almost 2 billion people use the internet, which opens up endless doors for data collection of all sorts. [8]

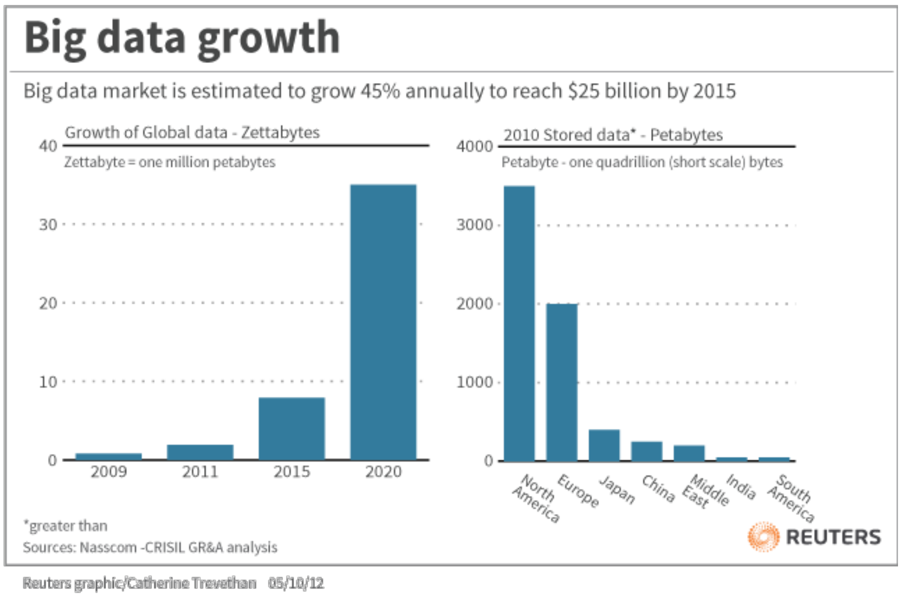


Figure 0-3- Global Data Growth

As we can see from figure 1-3, there is a global exponential increase in data growth year over year. This has led to the need of innovative brand new methods for data processing and storage. And so, the machine learning and then deep learning fields emerged.

Machine Learning & Deep Learning – Theoretical Part

With the astounding increase of the value and amount of data available for everyone, rose the need for a better means of analyzing it. That is how Machine Learning came into play.

Machine Learning

Machine Learning emerged from the theory that computers can learn without being explicitly programmed to using pattern recognition. It is an iterative process where the machine learning model adapts and learns with more data getting fed to it. This science is not new but has gained significant momentum in the recent years due to what was explained earlier of the increasing amounts of data in need for effective and efficient tools to analyze them – when there is a will there is a way. Some of the prominent applications of Machine Leaning that we encounter on a daily basis are spam filtering in emails, natural language processing in autocorrect while typing on smartphones and fraud detection.

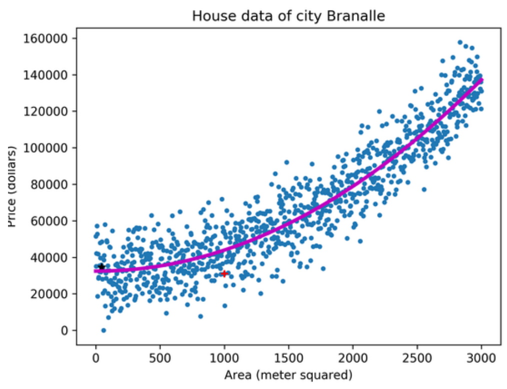
Although Machine Learning is beyond the scope of this thesis, it is essential to mention its foundations as it is the basis from which deep learning emerged.

Supervised Learning

The majority of research and algorithms available today is based on Supervised Learning. This technique is when you have X input variables and Y input variables and the algorithms is used to basically map the input to the output. And during this mapping, the algorithms is finds and learns patters of data that can then let it predict or expect what's the output should be if given only a new input X. In principle, this iterative process of learning should come to a halt when an adequate level of accuracy for the model is achieved.

The Supervise Learning category can be further drilled down to two categories: classification and regression.

- Classification is where the output is a category. That can be either binary, like a patient's data showing "negative" or "positive" for a particular disease (only 2 outputs are expected). Classification models can have several classes as output as well but of course with more classes, the more complex the model becomes.
- Regression is where the output is predicted when the model is trained on a large set of continuous data. For example, we have a dataset of areas of houses (X) against their prices (Y). When the model is trained it is expected to predict the price of a house when only given its area (as shown in figure below).



The challenges of supervised learning algorithms are the fact that attaining labelled data is extremely expensive as I will explain in the practical part of the thesis.

Unsupervised Learning

Unsupervised learning is where only input data (X) and no corresponding output variables are available. The objective of unsupervised learning is to model the underlying structure or distribution of the data to learn more about the data. These are called unsupervised learning because there are no correct answers and no teacher, unlike supervised learning above. To discover and present the interesting data structure, algorithms are left to their own devices. Unsupervised learning problems can be grouped into problems of clustering and association. Clustering:

- A **clustering** problem is where you want to discover the data's inherent groups, such as purchasing behavior to group customers.
- An **association** rule learning problem is where you want to discover rules that describe large portions of your data, such as people that buy X also tend to buy Y.

Examples of popular Unsupervised Learning algorithms are **K-means for clustering problems**.

Although supervised learning seems to be the solution to the vastness of unstructured data available easily to anyone, it is quite challenging to build unsupervised learning model with absolutely no supervision. The field itself is not quite mature as much as the supervised learning field. And it is this need that paved the way to the upcoming category of Machine Learning: Semi-Supervised Learning.

Semi Supervised Learning

In a nutshell, this field is mixture of both: Supervised and Unsupervised learning techniques. It is an attempt by data scientists to reap the fruits of both fields by reaching some kind of middle ground. The Machine Learning model is considered semi supervised when the majority of the data is unlabeled. Unlike in Supervised models.

Transfer Learning

Humans apply the concept of Transfer Learning on a daily basis. It is the process of applying the knowledge that you already know upon an entirely new task. For example, when I human that knows how to ride a bicycle attempts to learn how to ride a motorcycle, the balancing comes naturally to them since balancing on a motorcycle is quite similar (though not exactly the same) as balancing on a bicycle.
Transfer learning is a research problem in machine learning that focuses on storing knowledge gained while solving one problem and applying it to a different but related problem. I will explore Transfer Learning in this thesis later on in the practical part.

Deep Learning

Deep Learning emerged from Machine Leaning practices using Artificial Neural Networks (ANNs). ANNs were slightly inspired by the structure of neurons in our brains. The work *deep* in deep learning refers to the many hidden layers (other than the input and output layers) of an ANN. Since technology evolves fast, also its terms evolve with it. 8 years ago, an ANN with 10 hidden layers was considered *deep*. But today, that number evolved to more than hundreds of layers.
Deep Learning is an aspect of machine learning that emerged from the overlapping of signal processing, optimization, neural networks, pattern recognition and artificial intelligence. The intersection of those is now referred to as deep learning. Models of deep learning have been honored by exemplary scholars in renowned academic journals. Businesses of all sizes are now able to utilize deep learning to revolutionize real-time data analysis thanks to cheaper memory and faster computer processors. [9]
Figure 2-1 is referenced when it comes to all types of machine learning models developed. Input Data is passed through the model and is filtered out across multiple non-linear layers. The final layer of the model constitutes of a simple classifier that determines the class to which the object falls under.

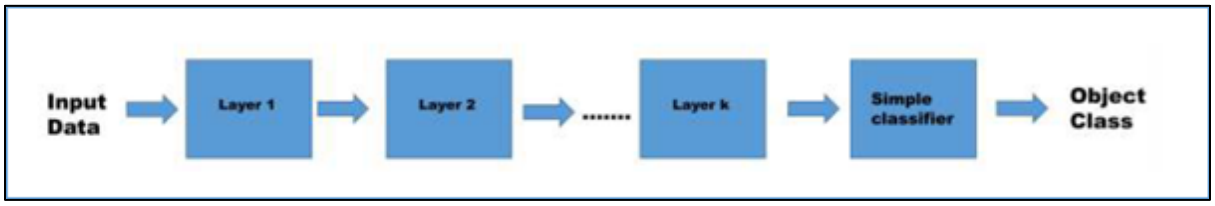


Figure 0-4- General deep learning framework

Predicting a response that is variable using given attributes is the goal of learning from data, which is quite similar to linear regression, where a linear model is used to predict a dependent variable (response) using several independent variables. Having said that, however, traditional linear regression is not considered deep as multiple layers of nonlinear transformation are not being applied with the data. In addition, data techniques such as decision trees, support vector machines and random forests are also not considered deep, although they are powerful tools. Random forests and decision trees utilize original input data with no transformations generated, same as with support vector machines which only consist of linear transformation, hence considered shallow. In addition, single hidden layer neural networks are considered shallow because they solely consist of one hidden layer. [9]

Problems that Deep Learning can solve

Deep learning models are unique in the sense that they are able to classify or predict nonlinear data using a number of parallel nonlinear steps. Deep learning models learn the features of input data hierarchy starting from raw data input all the way to the actual classification of data. Each layer uses the extract features of the output from previous layers.

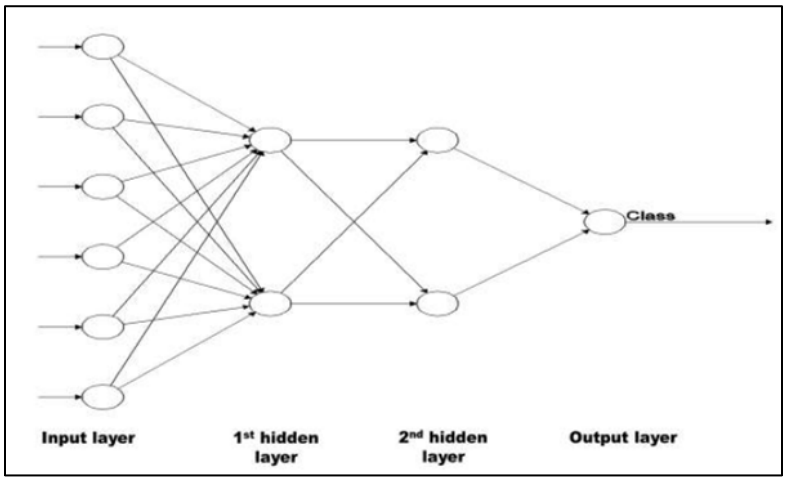


Figure 0-5 - Feed forward neural network with 2 hidden layers

Neural networks with multiple hidden layers will be the deep learning models that will be considered throughout this text. Figure 2-2 shows the simplest form of deep neural network, which contains at least two layers of hidden neurons. Each additional layer in this model produces the output by utilizing the previous layer as input. Deep multi-layer neural networks constitute of many levels of non-linearities, allowing them to represent nonlinear functions compactly. They are excellent at identifying complicated patterns from a set of data and have been utilized to improves aspects such as natural language processing and improving computer vision. In addition, it’s also been used to solve unstructured challenges that have to do with data.[9]

Deep Learning Concepts and Algorithms

The Human Brain Neuron

Deep learning was created in an attempt to mimic the complex human brain and its ability to learn and observe. And so, deep learning algorithms are structured upon the understanding of the basic building block of the human brain; the neuron. The Neuron is a simple body that, when is connected in a huge neural network, makes this network perform extremely complex tasks. A Neuron's main function is to receive and transmit chemical or electrical signals from and to other neurons connected end-to-end in the same neural network. Neuron's has Dendrites, which are responsible for receiving a signal, and Axons, which are responsible for transmission of a signal (as show in figure 2-3). [9], [10]

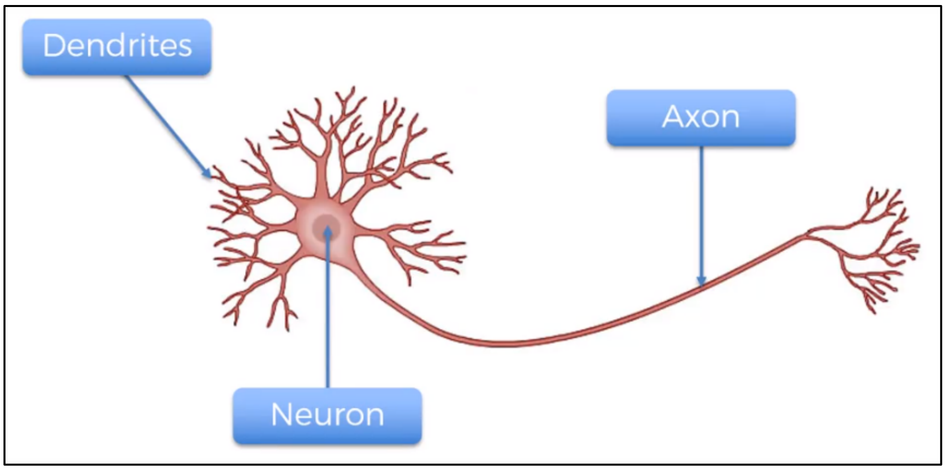


Figure 0-1- The Human Brain Neuron

Signals are passed from one neuron to another via what is called an "action potential". It is a spike in electricity along the cell membrane of a neuron. The interesting thing about action potentials is that either they happen, or they don't. That is why neurons were interesting to computer scientists. This state of being on or off, is just how computers work; 1 or 0. [10]

The Artificial Neuron

As mentioned earlier, the artificial neuron mimics the biological one. Signals come in as input, some processing occur in the neuron itself, then signals come out as output (as shown in figure 2-4). The simplest implementation of a neural network is a Perceptron. A Perceptron follows the *feed-forward* model, where inputs are sent into the neuron are processed and then result in an output. [10]

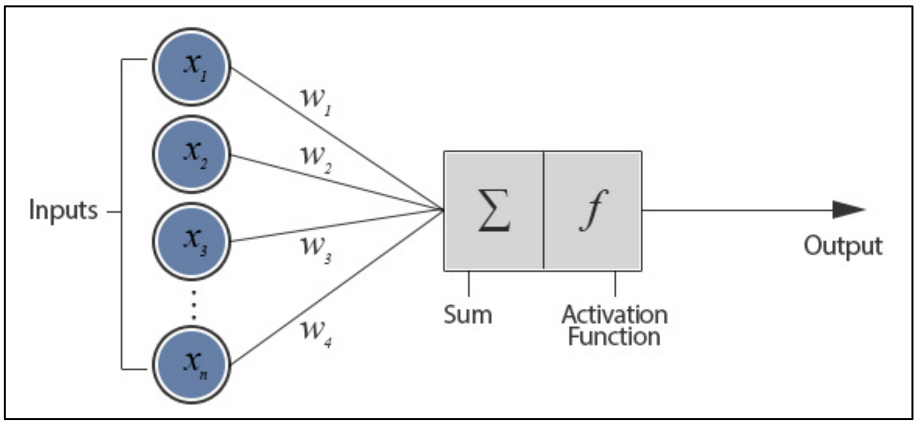


Figure 0-2 - The Simplest Neural Network - The Perceptron.

A Perceptron neural network follows the following steps:

1. Receive inputs.
2. Weight Inputs.
3. Sum Inputs.
4. Generate Output.

A Perceptron will typically begin by assigning random weights. Each input is multiplied by its weight. Those values are then summed and passed through an activation function, which is explained in the next section.

There is one more aspect to the Perceptron and that is ***bias.*** Biases are put into neural networks as one last input to avoid a result summation of zero. [9]

Activation Functions

In neural networks, activation or transfer functions create bounds for the output of neurons. Neural networks can use various activation functions.

Choosing an activation function for your neural network is an essential consideration because it can affect how the input data should be formatted.

Linear Activation Function

The linear activation function is the simplest transfer function that can be used in a neural network. It simply returns the value that the input neuron has passed to it. Regression neural networks will typically use the linear activation function to output a numerical value (figure 2-5). [10]

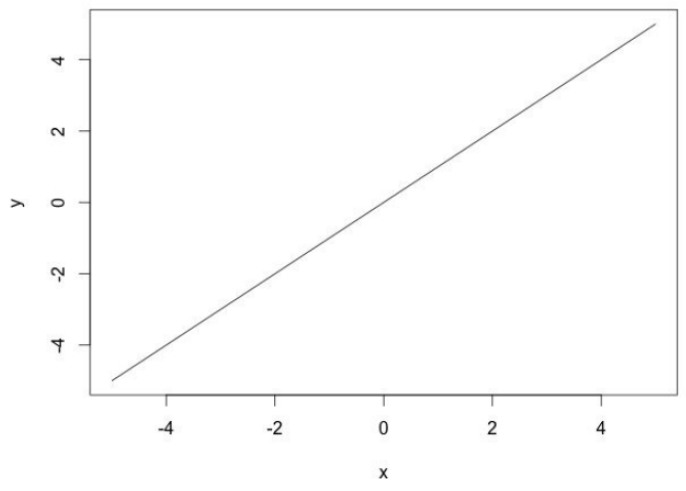


Figure 0-3 Linear Activation Function

Sigmoid Activation Function

The sigmoid or logistic activation function is a typical choice for feed-forward neural networks that only output positive numbers. It is used to ensure that the output values remain within a small pre-defined range (figure 2-7).[10]

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

Figure 0-4 Sigmoid Activation Function Equation

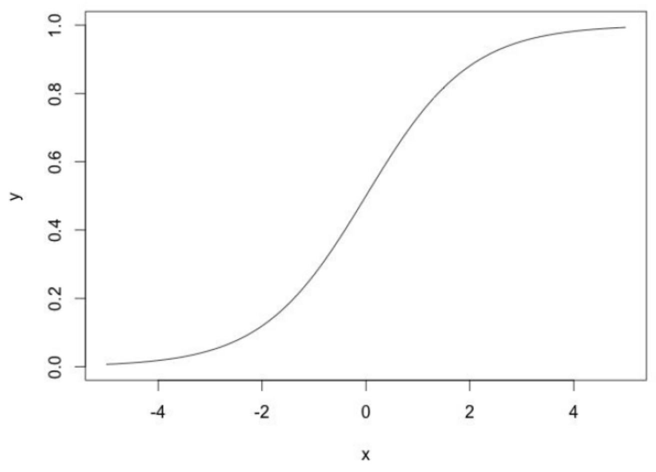


Figure 0-5 Sigmoid Activation Function Graph

SoftMax Activation Function

The SoftMax activation function is typically used on a classification neural network. In such neural networks, there are pre-defined classes that each of the inputs should be categorized to. The neuron with the highest numerical value claims the input within its class. [10]

$$\boxed{\phi_i = \frac{e^{z_i}}{\sum_{j \in group} e^{z_j}}}$$

Figure 0-6 The SoftMax Activation Function Equation

I will use the SoftMax activation function later in chapter 4 to classify hand written digits to their pre-defined classes from 0 to 9.

Rectified Linear Units (ReLU) Activation Function

Most neural networks utilize the ReLU activation function on its hidden layers while either SoftMax or linear functions on the output layer. The ReLU function basically removes the negative part of the function. [10]

The ReLU function has the following equation and graph.

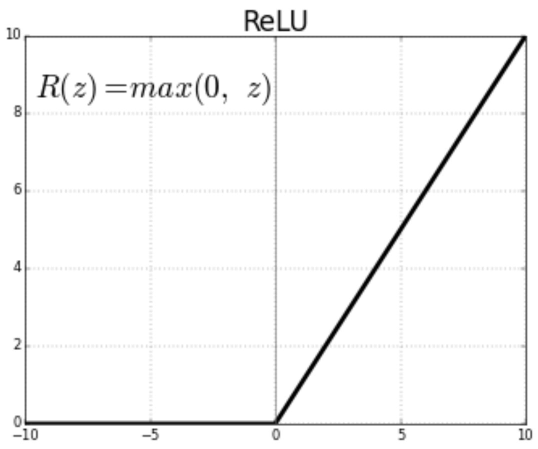


Figure 0-7 - ReLU Activation Function

Deep Learning Foundational Algorithms

Gradient Decent

This algorithm is particularly important for determining how the weights of the NN will get updated. Although, it’s being used extensively in Deep Learning, it is also used in many different machine learning models. The main concept behind Gradient Decent is to find the minimum of a function. In order for that to be achieved, several steps proportional to the negative of the gradient of the function at the current point are taken.

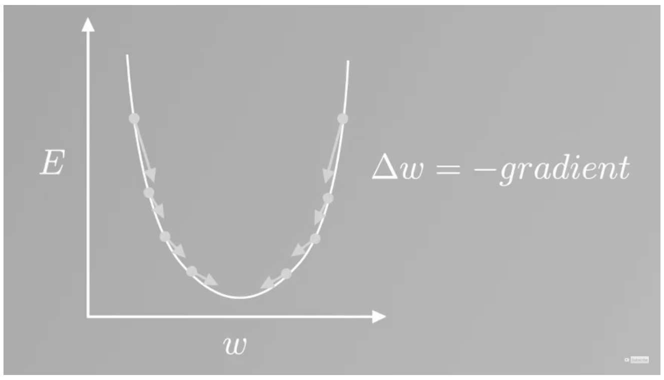


Figure 0-8 - Gradient Decent

Backpropagation

Backpropagation is an essential concept in neural networks. It was introduced by Rumelhart, Hinton, & Williams (1986) and is still very widely used today. Backpropagation is a type of gradient decent, which is the computation of a gradient on each weight in a neural network for each training element. As neural networks are not expected to output the expected values from the first feed forward run, the gradient of each weight will give an indication about how much the weights need to be adjusted to achieve the expected output. In a case when the output is exactly what was expected then the gradient of the weight would be 0, meaning that there is no change needed to adjust the weights.

The gradient is the derivative of the error function at the weight's current value. The error function is present in the algorithm to measure how far are the current output values from the expected ones according to the already labelled data. As the neural network trains, optimization takes place and error is minimized.

To sum up, when a neural network is initialized, random weights are assigned to inputs then fed to the neurons. Then, those values are summed and enter the activation function then sent to the next layer. The outermost layer's output is compared to the expected output using an error function. This error function backpropagates the adjustments needed to the weights, reaching a final optimized minimum error (figure 2-9). [9]

Deep Learning Models

Feedforward Neural Network

Unlike a Recurrent Neural Network (mentioned below), a Feedforward Neural Network's nodes do not form a cycle and are rather all forwardly connected. This form of ANNs were the first and simplest to conceptualize. The information moves only forward – from input layers to the hidden layers then to output layers. Hence, there is no loops or feedback within the network.

Convolutional Neural Networks

Convolutional neural networks or CNNs or ConvNets are a kind of feed-forward Artificial Neural Network in which the connection between its neurons is inspired by the form found in the animal visual cortex. Individual cortical neurons respond to incentives in a region known as the receptive field. Those receptive fields of various neurons overlap such that they tile the visual field. The response of an individual neuron to stimuli within its receptive field can be approximated mathematically by a convolution operation.
A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels)[12].

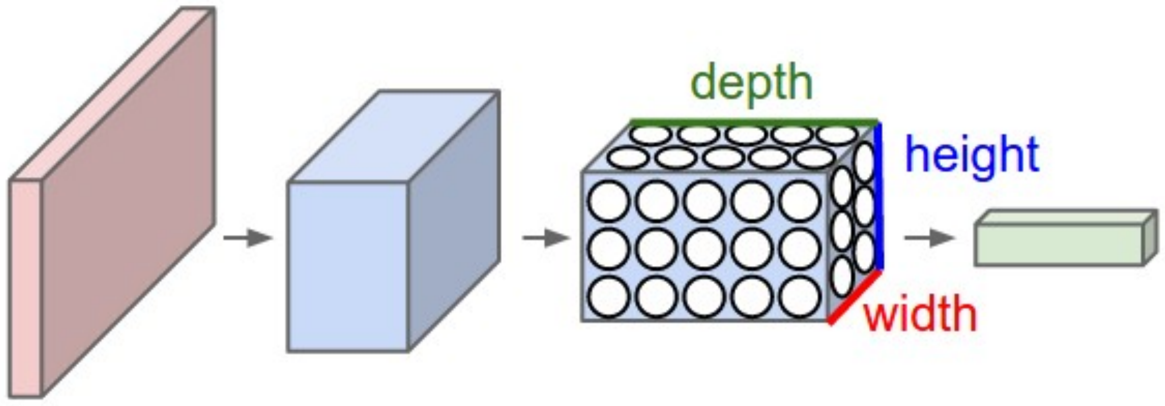
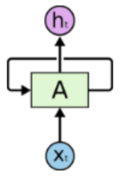


Figure 0-9 CNN for Image Recognition. Adopted from http://deeplearning.ir

Recurrent Neural Network (RNNs)

Unlike Feedforward Neural Networks, RNNs have loops within their design. This means that the network doesn't not only take into consideration the input at hand, but also takes the bigger picture into its scope. This makes RNNs functioning more like a human brain which takes, for example, all the sentences of a paragraph into consideration rather than focusing on one sentence or a couple of words to form meaning.



Recurrent Neural Networks have loops.

Restricted Boltzmann Machines

A Restricted Boltzmann Machines (RBM) is an unsupervised learning model that approximates the probability of sample data. The "restricted" part of the name comes from the fact that there are no connections between units in the same layer. The parameters of the model are learned by maximizing the probability function of the samples. Since it is used to approximate a probability density function it is often referred to in the literature as a generative model. Generative learning involves making guesses about the probability distribution of the original input in order to reconstruct it.

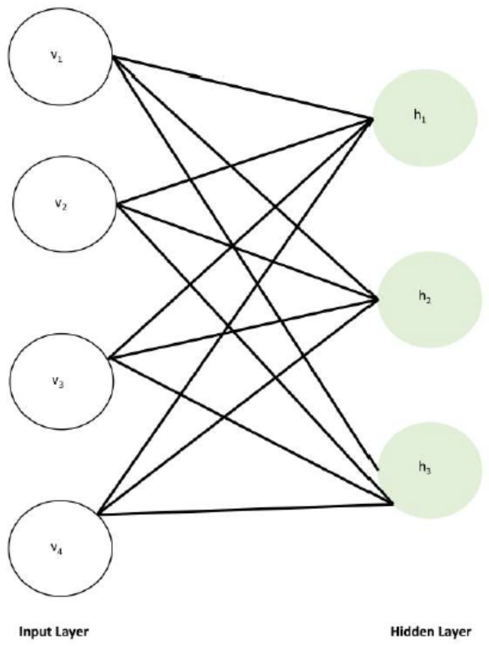


Figure 0-10 - RBM -

Figure 2-13 shoes a graphical presentation of an RBM, where it consists of 2 layers and there are a number of units with inner layer connections. Connections between layers are symmetric and bidirectional, allowing information transfer in both directions. It has one visible layer containing four nodes and one hidden layer containing three nodes. The visible nodes are related to the input attributes so that for each unit in the visible layer, the corresponding attribute value is observable[9].

Autoencoders

The autoencoder is an unsupervised three-layer feature learning feedforward neural network, shown in figure, it is very similar to the multilayer perceptron; it includes an input layer, a hidden layer and an output layer. The number of neurons in the input layer is equal to the number of neurons in the output layer. The number of hidden neurons is less (or more) than the number of input neurons. It is different from an MLP because the output layer has as many nodes as the input layer, and instead of training it to predict some target value y given inputs x, an autoencoder is trained to reconstruct its own inputs x.

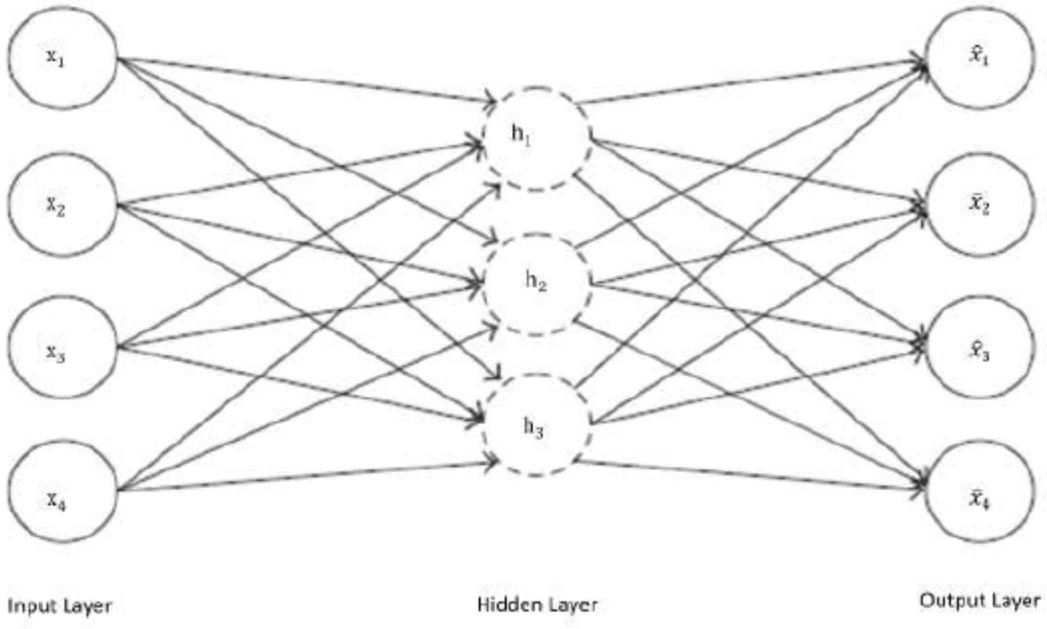


Figure 0-11 - Autoencoder

The autoencoder consists of an encoder and a decoder. The mapping of the input layer to the hidden layer is called encoding and the mapping of the hidden layer to the output layer is called decoding. The encoder takes the vector of input attributes and transforms them typically through sigmoid activation functions in the hidden layers into new features. The decoder then converts these features back to the original input attributes.

Deep Learning Tools

Deep Learning Libraries

TensorFlow

TensorFlow is a Machine Learning software library that was developed by the Google Brain team for internal use within Google. It was released under the Apache 2.0 open-source license on November 9, 2015. It is one of the most widely used Machine Learning libraries. Google uses TensorFlow for most of its almost all of its Machine Learning and Deep Learning applications. From Gmail and Google Photos to Google search YouTube services, pretty much most of Google's systems now involve some kind of ML or DL.
TensorFlow is considered a low-level library which requires deep understanding of the underlying sciences of mathematics, statistics as well as the ML/DL science itself in order to get started with TensorFlow. However, it provides state of the art methods and classes that revolutionized the field's pace of evolution greatly.
I will use TensorFlow in the practical part of this thesis to demonstrate how it can be use it for a simple image classification problem.

Keras

Keras is an opensource neural network library written in Python. It is a wrapper on top of several prominent lower level libraries such as TensorFlow, Theano, PlaidML as well as Microsoft Cognitive Toolkit. Keras is designed with one main goal in mind and that is to let its users harness the power of neural networks and go from concept to a working prototype as effectively and as efficiently as possible, which enabled swift experimentation for users and that made Keras extremely popular. It is also well-known for being user-friendly, modular and extensible.
I will demonstrate Keras's ease of use and powerfulness in the practical part of this thesis as I use it to apply deep learning on my own web scrapped data set.

Cloud Computing Solutions

The Cloud Computing Market

Cloud computing has solely changed the way technology works. The ability to host a website on the cloud, manage files and perform actions that were impossible in the past are all results of the presence of cloud computing. Although many new users tend to only look at the cost

when it comes to choosing a cloud platform, several business needs need to be taken into account to make a final decision. The three major cloud platforms are compared in this research using several factors such as computing power, storage, databases, location and documentation[13].
The main idea and power of cloud computing comes from the fact that it enables companies and giants such as Spotify and Netflix to not worry about infrastructure and the technicalities that come with maintaining a cloud. The three main cloud platforms in the market are Google Cloud Platform (GCP), Amazon Web Services (AWS), Microsoft Azure. Cheap cloud computing prices are nowadays existent as a result of the fierce competition, which helps big businesses capitalize on growth and small businesses grow easily[13].

The Three Big Players

The first major player in the market is **Google Cloud Platform (GCP)**. The initial introduction in the market was done by Google to power their own services: Google and YouTube. Enterprise services were later on built enabling anyone to host in the cloud. The second major player is **Amazon Web Services**, referred to as AWS. AWS is one of the longest operating cloud platforms in the market. The have extensive computing services and essential cloud aspects such as mobile networking and deployment are covered. The third and final cloud platform being compared is **Microsoft Azure**. Being almost as old as GCP in the market, Azure also provides a complete set of cloud services.

Comparing the 3 big players

Computing & Processing Power

The processing power that cloud services offer is referred to as compute. The more compute power, the better. All three major players offer compute power at various levels.
Amazon Web Services (AWS) offer EC2(Elastic Compute Cloud) which is capable of handling all compute services. EC2 works by managing virtual machines that come with preconfigured settings to facilitate use or can be configured by the owner. On the other hand, Google Cloud Platform (GCP) offers Google Compute Engine (GCE) to perform the same essential operation. In addition, Microsoft Azure offers Virtual Machines and Virtual Machine Scale Sets. All three players support containers and are easy to manage and are portable. Users can add more stats or move them to new locations easily.

Although all three players are similar when it comes to compute power, the real difference lies in comparing price and user experience, which is what users would take into account when choosing a cloud platform according to compute power[13].

Storage

Storage is an essential parameter when it comes to understanding and analyzing how a cloud platform operates. The way cloud storage operates is entirely different than the normal SSD on our day-to-day computers as multiple issues need to be solved and it also needs to ensure that no data is lost during transfers.
Amazon Web Services (AWS) offers S3 (Simple Storage Service), which is extensive in community support, documentation, etc. and is considered a proper storage solution with plenty of resources. On the other hand, Google Cloud Platform also offers Google Cloud Storage and Azure offers Microsoft Azure Storage which are considered reliable services. Although all three players offer similar storage solutions, Amazon is considered to have the better storage solution but are more costly[13].

Locations

Location is essential when it comes to deploying the application. Making sure the application performs at its peak by having the lowest possible route to the intended customer base is crucial. All three major players offer great coverage worldwide when it comes to location. However, Amazon leads as it has 42 availability zones. Google Cloud Platform has a presence across 33 countries, and Microsoft Azure covers 32 regions. New regions are added on a regular basis by all three players[13].

Databases

Software terms and conditions and how each cloud platform service manages data on their cloud is quite essential and should be taken into account when investing in any of the platforms.
Given that database images from different vendors enables a user to start more easily, Google seems to be slacking when it comes to providing this option to the end user. On the other hand, Amazon Web Services and Microsoft Azure both offer more several options for the end user to work with. Amazon’s RDS (Relational Database Service) provides support for Oracle and other major databases, as their services manages everything from updating to patching to offering solutions to common database issues. Similarly, Cloud SQL offers SQL database

handling features for Google Cloud Platform, and Azure SQL offers the same for Microsoft Azure. Both also offer high-performance database choices[13].

Documentation

Documentation is the last parameter used for comparing the three major cloud platform players in this research. Documentation is essential when it comes to choosing the appropriate cloud platform for a user as ease of use is a very valid factor that should be taken into account. Amazon Web Services offers the best documentation, followed by Microsoft Azure and Google Cloud Platform. AWS's strong documentation features comes from its age in the market and contribution by various people over the course of a decade[13].

Applying deep learning on Image Classification – Practical Part

This is the practical part of the thesis. I will compare how easy it is to get started with Deep Learning with a dataset available online to the challenges faced when working on a specific custom-made problem. The easiness of the first option comes from the fact that there is a great abundance of datasets online that are already pre-processed and ready out of the box for deep learning applications. On the other hand, working on my own dataset proved to be very challenging from the moment I define the problem I want to solve, to data collection, data labelling, data cleansing and preprocessing. As I will showcase, all those stages are skipped if I work with a dataset available online.

Applying Deep Learning on a dataset available online

With the rise and popularity of DL, came the great abundance of datasets available online for anyone to experiment with. Dataset sharing platforms like Kaggle have emerged with the sole purpose of sharing datasets for experimentation as well as posting competitions where data scientists compete towards solving the data analysis problem at hand as efficiently and as effectively as possible. Beside the abundance of datasets, tutorials working on those popular datasets became accessible on websites like YouTube and Medium. It is worth to mention that there are very popular datasets that became the standard for some DL problems that the libraries decided to embed them within their frameworks. One of those libraries is the MINST dataset of handwritten images. I will explore this dataset as well as applying DL on it in the section: ***TensorFlow on the MINST Set.***

Applying Deep Learning on my own dataset

As I first introduced myself with to DL, I found myself in a matter of days already having my first DL model up, running and accurate. This swift progress misled me to think that DL is indeed an easy field to work with. Of course, I soon came to the realization that this is far from the truth. I soon learnt that the reasons why my progress was that fast was due to the fact that the dataset was already collected, cleansed and preprocessed for the DL model. Also, the dataset had a handful of tutorials on YouTube and Medium that guided me on how to load the dataset on to the model, visualize it, determine DL parameters as well as running the model and evaluating it.

As I will showcase in the upcoming sections how all the steps mentioned above that made my DL journey seemingly easy in the beginning, naturally were up to me to figure out. Starting with data collection (next section) I will explore the challenges that I faced and how I overcame them.

Problem Statement – Image Classification of a dataset of shoes

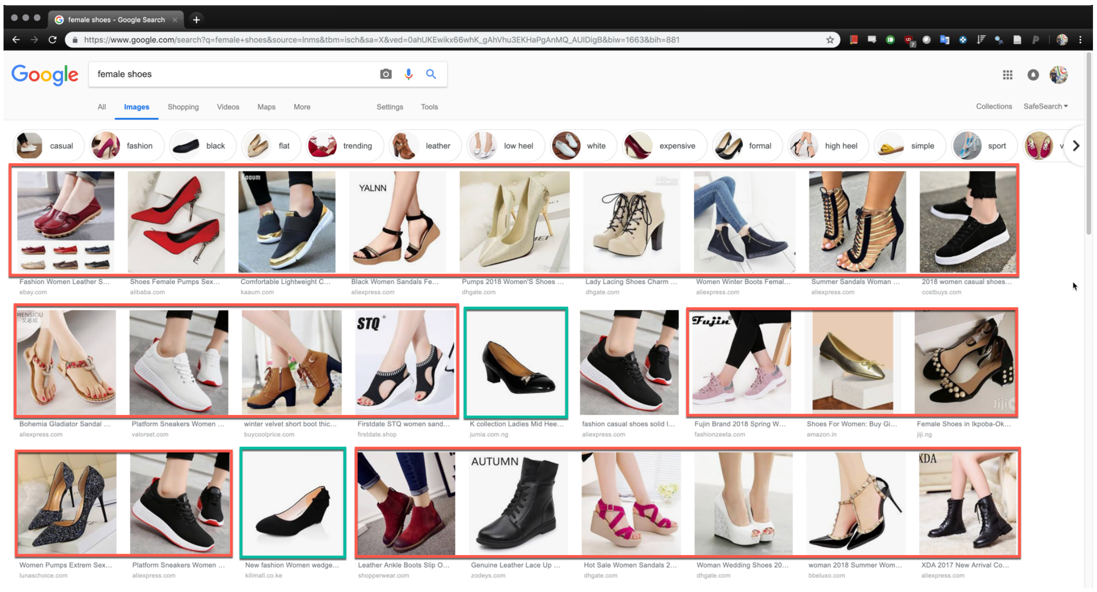
The problem that I decided to work with is a classification problem where I would like to feed the DL model various images of female shoes upon which the model shall train. Then, when I give the model new sets of images it is expected to correctly classify them to their correct classes.

Data collection

The first challenge I faced when I attempted to apply DL on my own classification problem with my own dataset was actually collecting the data. Data in general and images in my case are available everywhere on the web with a huge abundance. They are available for anyone to grab, but unfortunately, they are extremely unstructured. For that reason, I had to think and research thoroughly in order to find a good source for my images then build the program to scrap it. But first let me describe the aspects of a good image for DL applications.

Aspects of a good image for training a Deep Learning model

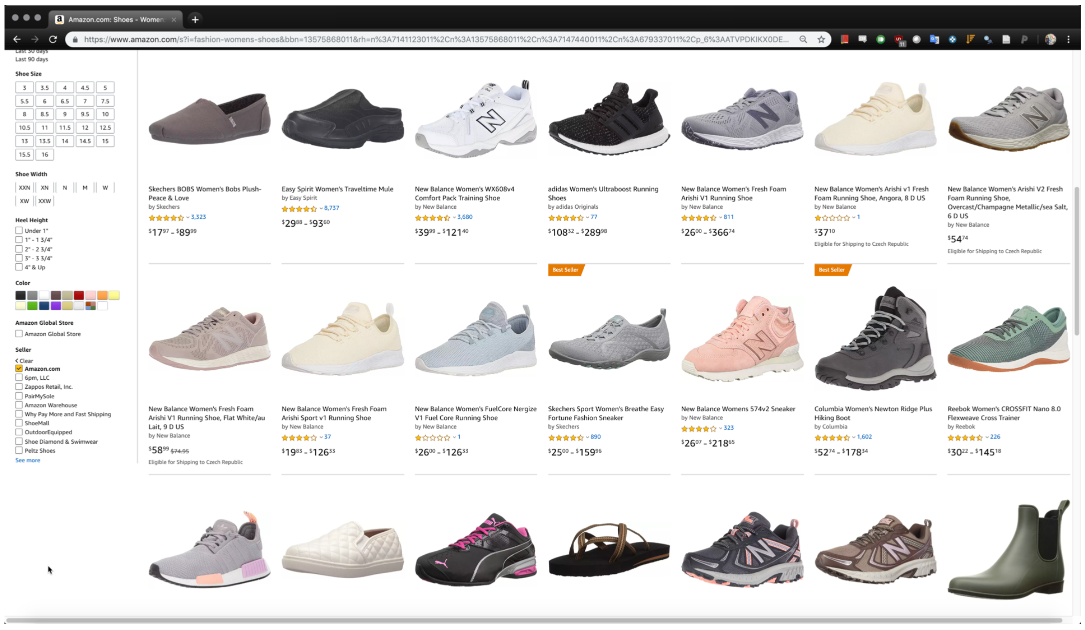
If I go ahead and Google "female shoes" this is the result that I get.



From a sample of 27 results (above), I can quickly opt out just 2 pictures are good material for DL (framed in green). Just 7% of the small sample are considered good quality images for my purposes. This demonstrates how scarce good quality images are on the web. It is also worth mentioning that practically I would need a dataset of about 12000 images. So, I would need to pragmatically choose the good quality images from the bad which is quite impossible.
The aspects of a good image quality include but are not limited to: **position of subject in image, one subject per image, resolution of image and the size of the image.** The reason why those 2 images are considered good quality is because in those 2 images, there is one subject in the image and that is the shoes that we want our model to train on. The second reason is that the background is white which makes the subject standout with minimal distortion. As we can see the rest of the pictures, though the level of quality varies between them, I can consider them all as bad quality and that is because all of them have lots of noise in the image. Either a human being wearing the shoes, or the shoes being placed in a room. Whatever it is, this is considered as noise which will distort the DL model from learning effectively.
In addition, the 2 images have the subject exactly in the center , have good resolution and are more or less the same size. All those aspects are worth mentioning when it comes to the quality of the image. As we will see later, those images are nothing to a computer but an array of numerical arrays. For example, a dataset with various resolutions would mean various array dimensions and that would be very hard to work with.

Amazon.com images

I selected Amazon.com as website from which I will collect my dataset of images.



At first glance, we can see the huge difference of quality between the images that google returned from all over the web and the images that Amazon returns. The images above have one subject per with zero distortion, all of the subjects are relatively in the same position within the image. The images all have white background and have the same size.
For this reason, Amazon seemed like the perfect place for collecting my dataset. So, in the following section I will introduce the topic of WebScraping, which is task of the programmatic collection of data from the web. And then I will showcase how I used it to collect my dataset of images.

Introduction to WebScraping

After we discussed the importance of data in today's world. It's time to deep dive into one of the most prominent methods of attaining it. The Internet contains massive amounts of data. Most of this data is free for anyone to access. However, most of the time, this data is embedded within the structure of webpage which makes it hard to attain. Also, there is the manual side of attaining this data. Say you want to attain a data set of 1000 images for training your AI model. You can right-click-save-as every and each image, but how much time and effort would that take. For those two reasons, we need Web Scraping to programmatically extract the data we need as we program them. [14]
On the other hand, in a perfect world, people wouldn't actually need to use **Web Scarping** to extract data from the web as webpages -in a perfect world- provide APIs to share their data with anyone who needs them in an organized manner. In the real world, in fact, many

websites do provide APIs to access their data, but they also put restrictions of quotas on the frequency of requests as well as the amount of data being accessed. So this means that we cannot always rely on APIs to get our needed data, and that is why we use Web Scraping. [14]
So, web scraping is the practice of collecting data by any means other than a program that interacts with an API (or, obviously, a human using a web browser). This is most commonly done by writing an automated program that queries a web server, requests data (usually in the form of HTML and other files that comprise web pages), and then scans the data to extract the necessary information.

Note on the Legality of Web Scraping

Web Scraping is still a new field that is taking its first steps. And so, laws and rules are still vague and unclear. However, the rule of thumb would be that if the data you are scraping will be used for personal use, then it is okay. On the contrary, if the data is going to be republished, then the scraper needs to be caution with the original publisher's rules. All scrapers, though, need to scrape the web politely. This is done, to name a few, by defining a user agent to identify the person scraping and also by sending download requests at a reasonable rate. [14]

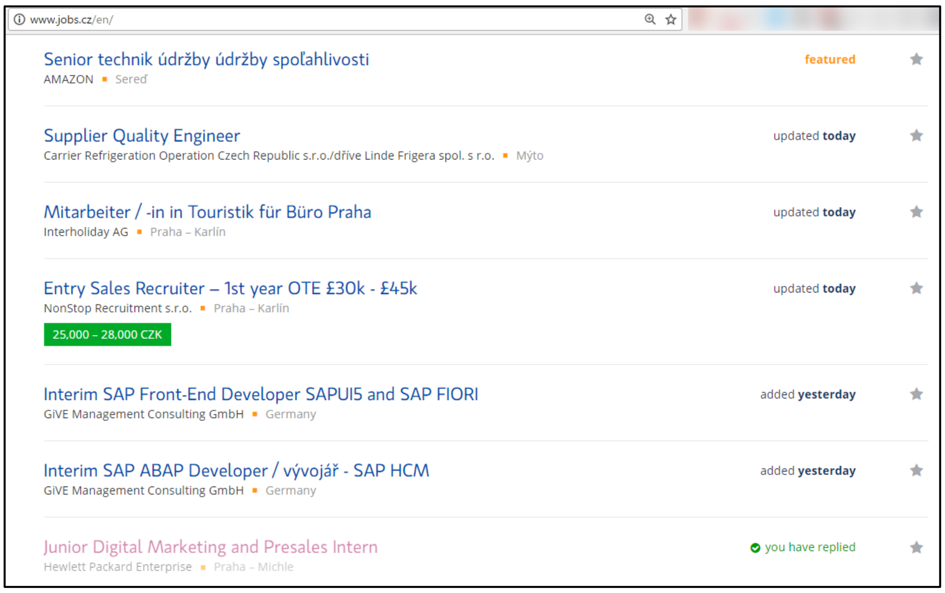
WebScraping with Python

Python is a high-level, general purpose, object-oriented language. High-level means that the programming language is further from the computer's machine language (like Assembly) and closer to the human communication languages[15]. Other examples of high-level languages would be C++ and Java. High-level languages, especially Python, are highly readable, shorter and take less time to write. Object-oriented means that the programming language follows the logic of how we think as humans about the program that we are writing [16]. In addition, Python has a massive community as well as a huge variety of well written libraries and frameworks. Those libraries and frameworks do most of the heavy lifting which makes the programmer focusing on the task at hand without worrying too much about what is happening under the hood.
That is why I chose Python for Web Scraping and also for the deep learning later.

BS4

BeautifulSoup is a Python library that parses a web page's HTML then lets the programmer easily navigate within. BeautifulSoup is considered the easiest method of Web Scraping. It will lay the basis that will ease the understanding of the main Web Scraping platform that I will use; Scrapy. [14]

I will jump directly into an example then analyze it. The example below will scrap the titles of vacancy postings from Jobs.cz. The webpage looks as follows:



www.jobs.cz/en/

Senior technik údržby údržby spoľahlivosti
AMAZON ▪ Sereď
featured

Supplier Quality Engineer
Carrier Refrigeration Operation Czech Republic s.r.o./dříve Linde Frigera spol. s r.o. ▪ Mýto
updated **today**

Mitarbeiter / -in in Touristik für Büro Praha
Interholiday AG ▪ Praha – Karlín
updated **today**

Entry Sales Recruiter – 1st year OTE £30k - £45k
NonStop Recruitment s.r.o. ▪ Praha – Karlín
25,000 – 28,000 CZK
updated **today**

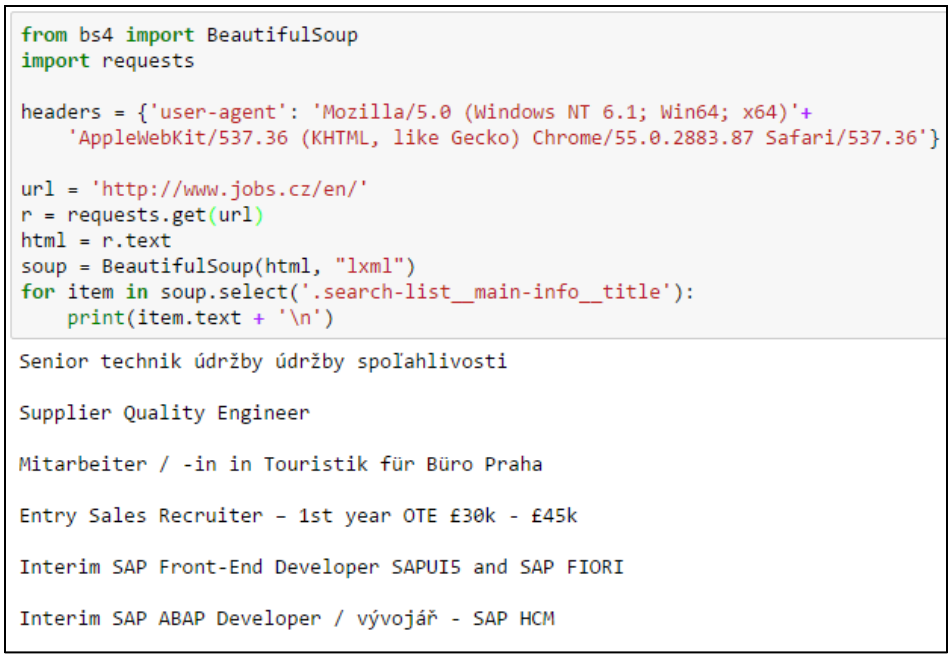
Interim SAP Front-End Developer SAPUI5 and SAP FIORI
GIVE Management Consulting GmbH ▪ Germany
added **yesterday**

Interim SAP ABAP Developer / vývojář - SAP HCM
GIVE Management Consulting GmbH ▪ Germany
added **yesterday**

Junior Digital Marketing and Presales Intern
Hewlett Packard Enterprise ▪ Praha – Michle
you have replied

Figure 0-1- Jobs.cz to be scraped

Our code looks as follows:



```
from bs4 import BeautifulSoup
import requests

headers = {'user-agent': 'Mozilla/5.0 (Windows NT 6.1; Win64; x64)'+
    'AppleWebKit/537.36 (KHTML, like Gecko) Chrome/55.0.2883.87 Safari/537.36'}

url = 'http://www.jobs.cz/en/'
r = requests.get(url)
html = r.text
soup = BeautifulSoup(html, "lxml")
for item in soup.select('.search-list__main-info__title'):
    print(item.text + '\n')
```

```
Senior technik údržby údržby spoľahlivosti

Supplier Quality Engineer

Mitarbeiter / -in in Touristik für Büro Praha

Entry Sales Recruiter - 1st year OTE £30k - £45k

Interim SAP Front-End Developer SAPUI5 and SAP FIORI

Interim SAP ABAP Developer / vývojář - SAP HCM
```

Figure 0-2- Web Scraping Example

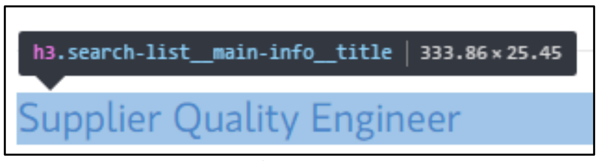
Let me explain each line of the code that we used:

1. I imported the 2 libraries that we are going to use; *requests* and *BeautifulSoup.* Requests is the library used to make a request to `get` a web page.
2. I assigned declared the variable *header* and I assigned the `user-agent in` it.
    - User agents tells the server or the website owner details about you so that you are less likely to be considered a malicious bot and be blocked.
3. I assigned the URL to be scraped in the variable `url`.
4. I used requests library's `get` method to make a request to the URL.
5. I used request library's `text` method to return the HTML document as text and store it in the variable *html.*
6. I created a BeautifulSoup object to store the parsed the HTML document returned by requests in the variable `html` into the new object variable: `soup`.
7. Now that the variable *soup* has the parsed HTML, we can navigate the DOM easily using BeautifulSoup's methods. Here we created a `For` loop iterated by the `find_all` method that finds all matches for `".search-list__main-info__title" and` for each match the for **loop prints the outcome of the finding**.

So, what is *"*`.search-list__main-info__title`*"* and how did I know that it contains the titles of the job postings? This is why I introduced CSS and HTML earlier. I will explain in the upcoming section.

Selecting elements in an HTML document

There is a huge amount of ways to select and find elements within an HTML webpage. Depending on the scraping task itself, the selection may be based on a very complex expression or as simple as the one used above. The one that I used above, simply selects any HTML element within the document that has a class attribute equal to *"*`search-list__main-info__title`*"*. The *"*`.`*"* used in the beginning in our code is just to say that the following is a class. We can use `"#"` if we are selecting elements based on ID. Let's see how to know the class or ID of the item you want to scrap.[14]
Normally, people would view a webpages source and manually find the elements. But that is hectic and inefficient. That is why we have the Google Chrome Developer tools and Firebug, if you are using Firefox. So, I by right clicking the item that I need to scrape and choose *Inspect* (on *Chrome).* I get the below:


h3.search-list__main-info__title | 333.86 × 25.45

Supplier Quality Engineer

Figure 0-3- Finding an element's class in HTML

This basically means that the title you are interested in is wrapped inside <h3> tags and has a class of “`search-list__main-info__title`”.

Scrapy

Scrapy a Python web framework for scraping data from the web. Scrapy was designed to scrape huge amounts of data while handling most of the manual work that libraries like BeautifulSoup do not. It features also a robust system of storing data both locally and remotely in databases as well as in all of the major data formats available today (AWS Storage Services). For example, if the user wants to scrape product listings from website that contains 16 different web pages, Scrapy will allow them to perform those requests all at once simultaneously (multithreading). If scarping a page takes a normally takes one second, then you’d need 16 seconds to perform that task. But scrapy handles them all in one second. If each page has 100 listings, then 1600 requests are needed to be made. Moreover, if the user is storing the data into the cloud, which for example would take 3 seconds, that is 4800 write requests all performed in parallel. [17]

Furthermore, scrapy has a huge community with a very clear and concise knowledge base, which is a major advantage when starting to work with a new framework. Also, Scrapy handles and understands broken HTML. I provided examples how BeautifulSoup selects elements using selectors from web pages. But Scrapy has a higher level interface called XPath which I will explore in the upcoming section.[17]

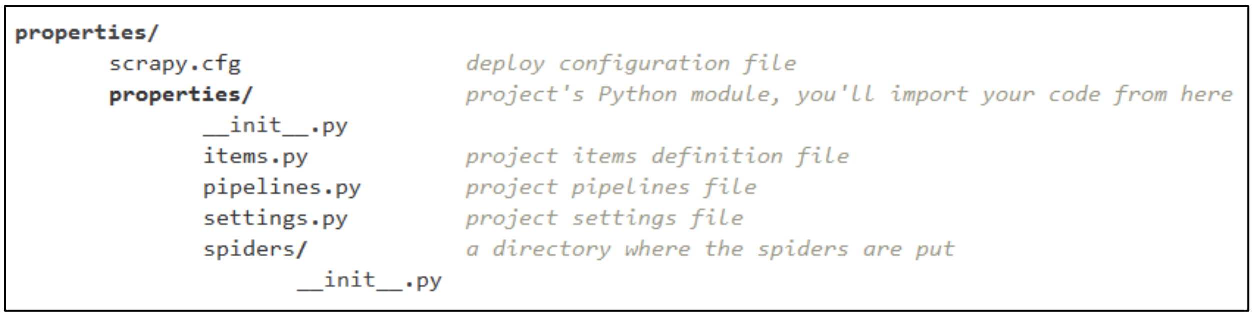
Collecting the Dataset using Scrapy

To start a new Scrapy project, the following is typed in the command line or terminal with *properties* being the project’s name:

```
$ scrapy startproject properties
```

Figure 0-4 - Scrapy Project Initiation

The file tree structure goes as follow:



```
properties/
       scrapy.cfg               deploy configuration file
       properties/              project's Python module, you'll import your code from here
              __init__.py
              items.py          project items definition file
              pipelines.py      project pipelines file
              settings.py       project settings file
              spiders/          a directory where the spiders are put
                     __init__.py
```

Figure 0-5 - Scrapy Project File Structure

I will now demonstrate Scrapy with a practical example that I have built to scrape Amazon.com for images to be used for deep learning later.

Scraping Amazon.com for Images

I have used Scrapy to scrape 16000 images off images, I will explain the procedure in this section.

Exploring settings.py

Settings.py gives users a lot of functionality. Using pre-defined functions already embedded with Scrapy, the user can just adjust those settings to tweak their scrape. The table below shows the most important settings and their description.

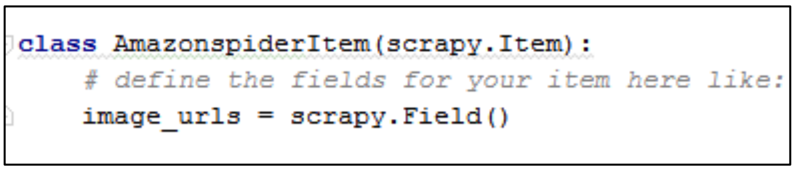
ROBOTSTXT_OBEY	If enabled, Scrapy will respect robots.txt policies. It is a way of web scraping politely.
DOWNLOAD_DELAY	Adjust the time between 2 subsequent downloads. Again, to protect the servers.
AUTOTHROTTLE_ENABLED	This option, if enabled, would let Scrapy handle the speed of download traffic according to the Scrapy server and the website being crawled. So, mitigating a lot of problems for the user and the website server.
USER-AGENT	A part of being polite when scraping a webpage. A variable that defines the scraper, their email and the project they're working on.

Table 1- Scrapy's Settings.py

Items.py

This file has a single class that contains any item that is being considered for scraping. It could by an image, a text listing, a video, etc.

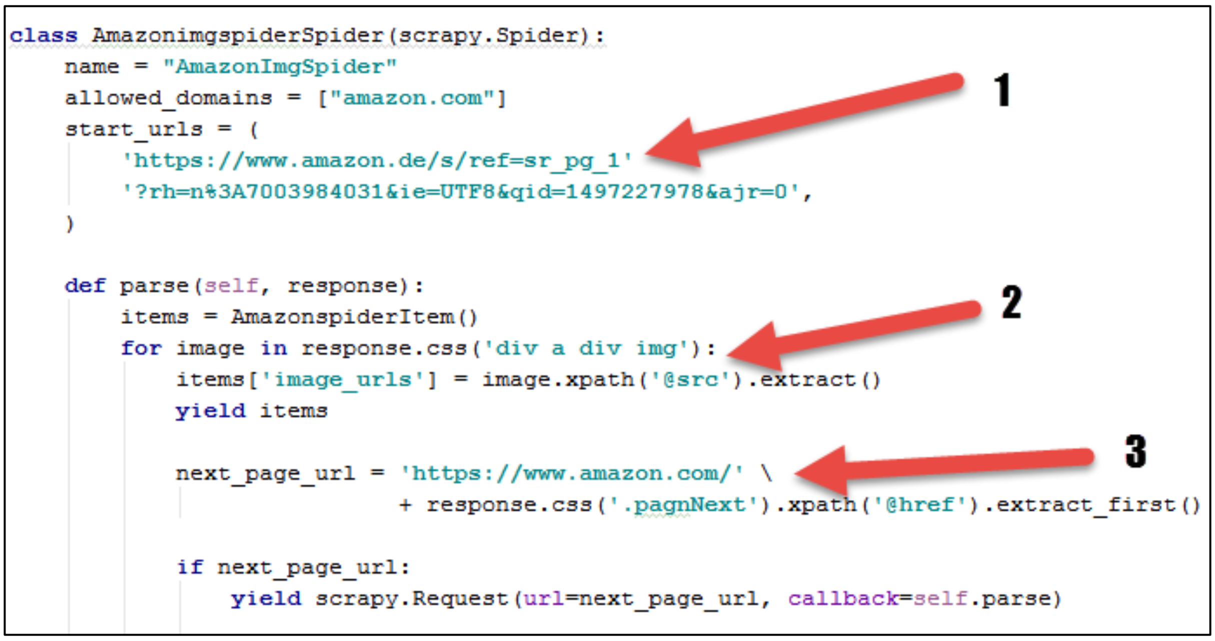
Here the field is of the URL of the images of shoes to be downloaded.



```
class AmazonspiderItem(scrapy.Item):
    # define the fields for your item here like:
    image_urls = scrapy.Field()
```

The Spider

Spiders is just another name for web crawlers. And web crawling is the act of swiftly moving through the web in a programmatic manner, while scraping occurs when there is data handling involved.



```
class AmazonimgspiderSpider(scrapy.Spider):
    name = "AmazonImgSpider"
    allowed_domains = ["amazon.com"]
    start_urls = (
        'https://www.amazon.de/s/ref=sr_pg_1'
        '?rh=n%3A7003984031&ie=UTF8&qid=1497227978&ajr=0',
    )

    def parse(self, response):
        items = AmazonspiderItem()
        for image in response.css('div a div img'):
            items['image_urls'] = image.xpath('@src').extract()
            yield items

            next_page_url = 'https://www.amazon.com/' \
                            + response.css('.pagnNext').xpath('@href').extract_first()

            if next_page_url:
                yield scrapy.Request(url=next_page_url, callback=self.parse)
```

Figure 0-6- Amazon Spider

As simple as it seems, the whole spider that allowed me to download 16000 images consists (as shown in figure 3-7) of a class that has 3 small parts.

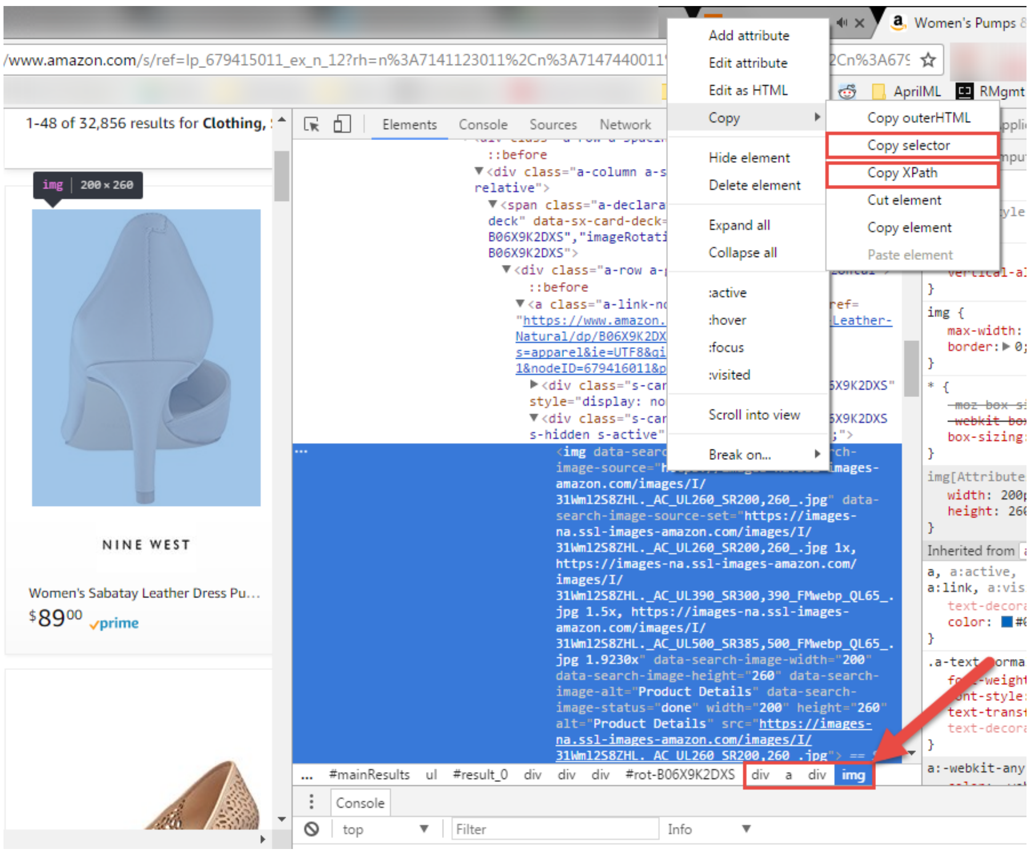
1. start_url:

This is just the starting URL where the crawl needs to be started. In my case, it is the URL of the first women shoes product listing.

2. for loop, downloading images:

As explained before in the BeautifulSoup section, here we use CSS selectors to select the image elements. This for loop basically creates a list of all the <***img***> elements that are inside a <***div***> element that are inside an <***a***> element that are inside a <***div***> element. To put that simply, it is just as written in the code:

“ `div > a > div > img` ”. The procedure for obtaining that path is shown in the figure below.



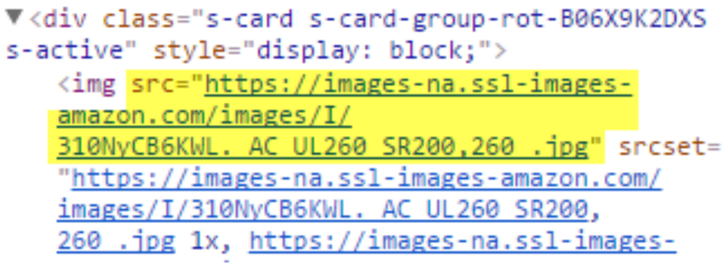
```
  ::before
▼<div class="a-column a-s
relative">
  ▼<span class="a-declara
  deck" data-sx-card-deck=
  B06X9K2DXS","imageRotati
  B06X9K2DXS">
    ▼<div class="a-row a-
      ::before
      ▼<a class="a-link-no
      "https://www.amazon.
      Natural/dp/B06X9K2DX
      s=apparel&ie=UTF8&qi
      1&nodeID=679416011&p
        ▶<div class="s-car
        style="display: no
        ▼<div class="s-car
        s-hidden s-active"
          <img data-searc
          image-source="h
          amazon.com/images/I/
          31Wml2S8ZHL._AC_UL260_SR200,260_.jpg" data-
          search-image-source-set="https://images-
          na.ssl-images-amazon.com/images/I/
          31Wml2S8ZHL._AC_UL260_SR200,260_.jpg 1x,
          https://images-na.ssl-images-amazon.com/
          images/I/
          31Wml2S8ZHL._AC_UL390_SR300,390_FMwebp_QL65_.
          jpg 1.5x, https://images-na.ssl-images-
          amazon.com/images/I/
          31Wml2S8ZHL._AC_UL500_SR385,500_FMwebp_QL65_.
          jpg 1.9230x" data-search-image-width="200"
          data-search-image-height="260" data-search-
          image-alt="Product Details" data-search-
          image-status="done" width="200" height="260"
          alt="Product Details" src="https://images-
          na.ssl-images-amazon.com/images/I/
          31Wml2S8ZHL._AC_UL260_SR200,260_.jpg"> ==
```

Figure 0-7 - How to obtain a CSS Selector

A list has been created of all the image elements, but the images has not been downloaded to this point. The previously mentioned items.py had a field called ***image_url***. It is time to use this field to cast the image URLs into. Here I used the XPath Selector (the feature that Scrapy is very popular for having) method to obtain the URL.

```
        items['image_urls'] = image.xpath('@src').extract()
```

This line means: for each "image" (the for loop index) in the list of <***img***> elements, extract using XPath, the ***@src*** of the currently selected element which returns a URL. This URL for each element is appended into items['image_urls'].

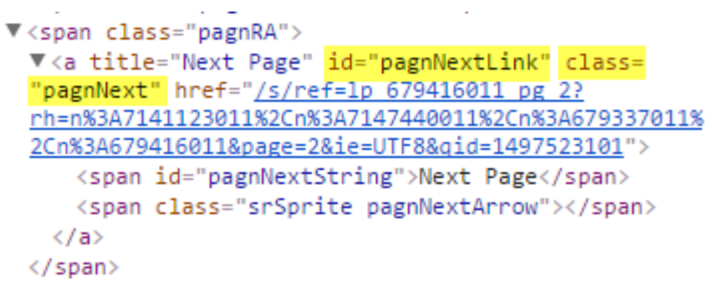


```
▼<div class="s-card s-card-group-rot-B06X9K2DXS
s-active" style="display: block;">
    <img src="https://images-na.ssl-images-
    amazon.com/images/I/
    310NyCB6KWL._AC_UL260_SR200,260_.jpg" srcset=
    "https://images-na.ssl-images-amazon.com/
    images/I/310NyCB6KWL._AC_UL260_SR200,
    260_.jpg 1x, https://images-na.ssl-images-
```

Figure 0-8 - img src

3. for loop, next_page_url:

After obtaining all image URL the crawler should then open the next page.



```
▼<span class="pagnRA">
  ▼<a title="Next Page" id="pagnNextLink" class=
  "pagnNext" href="/s/ref=lp_679416011_pg_2?
  rh=n%3A7141123011%2Cn%3A7147440011%2Cn%3A679337011%
  2Cn%3A679416011&page=2&ie=UTF8&qid=1497523101">
      <span id="pagnNextString">Next Page</span>
      <span class="srSprite pagnNextArrow"></span>
    </a>
  </span>
```

Figure 0-9 - grabbing next page's URL

Luckily, Amazon has a specific class and ID attributes for the next page's link. I utilized that below.

```
        next_page_url = 'https://www.amazon.com/'
                + response.css('.pagnNext').xpath('@href').extract_first()
```

This line of code utilizes the 2 methods of selection. First the element having the class attribute= **pagnNext** is grabbed using CSS selectors, then within that element the **'@href'** is grabbed using XPath.

The Resulting Data Set

The above spider scraped16,000 images off Amazon.com and storing them locally. This data set will be utilized later for deep learning.

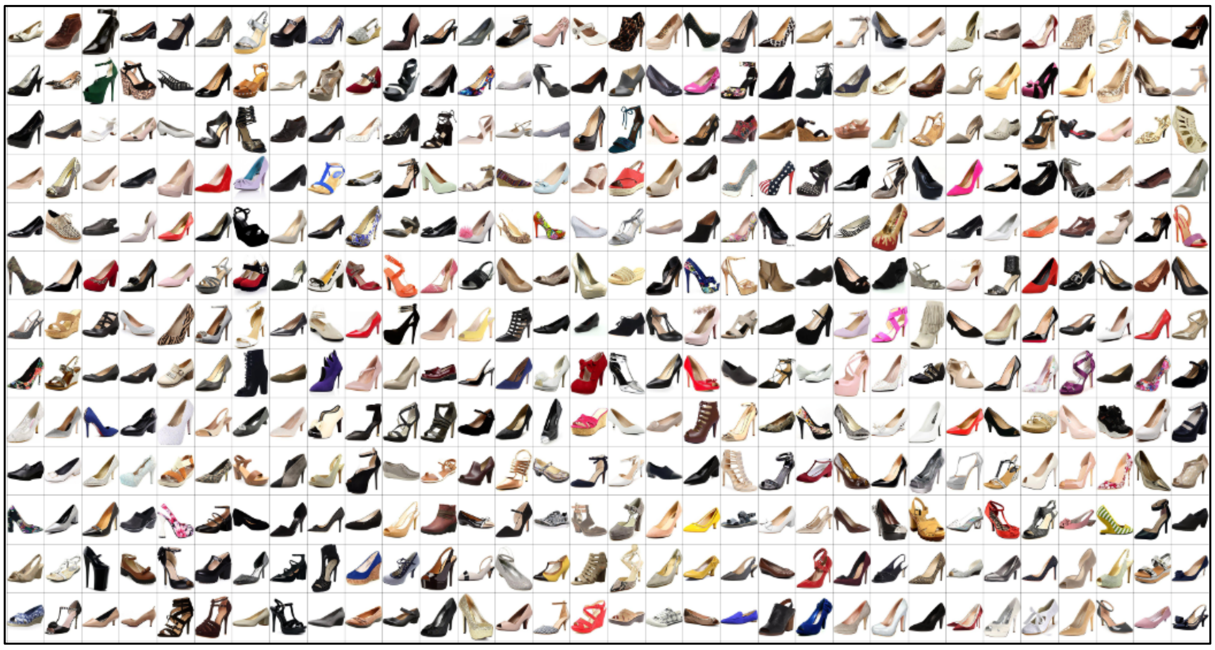


Figure 0-10 - Scraped Images

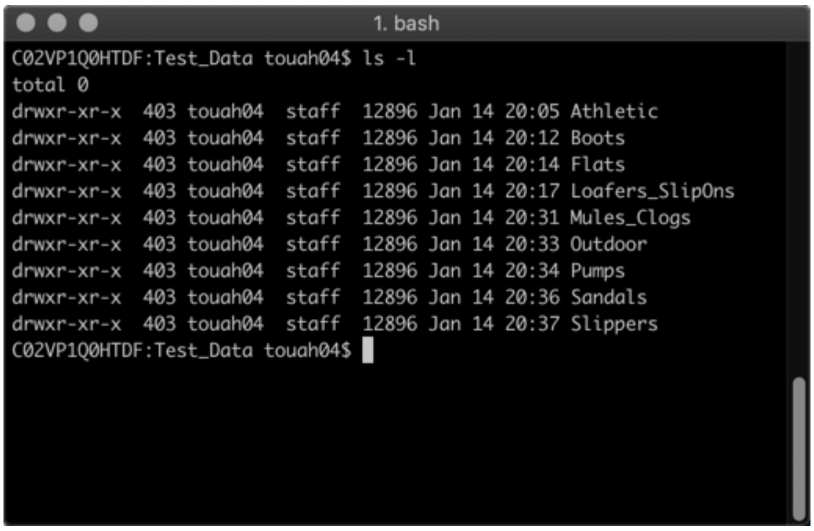
Although the dataset was quite good from a Deep Learning perspective, I soon learnt that my task of data collection was not over. The reason for will be mentioned in the upcoming section.

The problem of unlabeled data (Supervised, Unsupervised learning revisited)

The collected dataset is 16000 images that are of good quality yes, but they are all unlabeled. This means that the dataset will restrict me to using only unsupervised learning techniques when I apply DL on my dataset, unless I manually label the data. Unfortunately, when compared to supervised learning techniques, I found out that unsupervised learning was a much less ripe field . Also, the majority of the techniques were still theoretical and have minimal applications in the DL libraries like TensorFlow and Keras. I also realized that I would extremely advance statistical knowledge in order to proceed with my task.
I realized that in order to use Supervised Learning techniques, I would need to manually label the dataset, which proved impossible since I would need to at least label 80% of the 16000 images.
For the reasons mentioned above, I decided to take a step back and try to tweak my Python/Scrapy script in order to download the images already labelled.

Refactoring the Scrapy code instead of manually data labelling

I realized that while the script is crawling the website and downloading the images, there must be a class or a field within the DOM that mentions what type of shoes that is. Once, I found this field, it was just a matter of tweaking the script so that it stores that label in a variable and then downloading the image into a directory named after its label instead of downloading all of them in a single directory. The code is included in the Appendix.
Here are the resulting directories with the images inside already labelled.



```
C02VP1Q0HTDF:Test_Data touah04$ ls -l
total 0
drwxr-xr-x  403 touah04  staff  12896 Jan 14 20:05 Athletic
drwxr-xr-x  403 touah04  staff  12896 Jan 14 20:12 Boots
drwxr-xr-x  403 touah04  staff  12896 Jan 14 20:14 Flats
drwxr-xr-x  403 touah04  staff  12896 Jan 14 20:17 Loafers_SlipOns
drwxr-xr-x  403 touah04  staff  12896 Jan 14 20:31 Mules_Clogs
drwxr-xr-x  403 touah04  staff  12896 Jan 14 20:33 Outdoor
drwxr-xr-x  403 touah04  staff  12896 Jan 14 20:34 Pumps
drwxr-xr-x  403 touah04  staff  12896 Jan 14 20:36 Sandals
drwxr-xr-x  403 touah04  staff  12896 Jan 14 20:37 Slippers
C02VP1Q0HTDF:Test_Data touah04$
```

Having all my dataset labelled meant that I can proceed with applying supervised learning techniques instead of unsurprised learning ones.

Preparing the dataset for Deep learning

After having all the images nicely labelled into directories, the next step would be to analyze the images and prepare them for the DL model. There are some aspects that need to be considered when to this preparation.

The Numerical representation of an image to a computer

Since computers do not *see* images the way we see it. It's important to understand how a computer understands an image. I will take the image below as an example.

This picture is a sample of the images collected, it belongs to the "Athletic" class. The picture is colored and not greyscale. This image's resolution is 200 x 260, when I load this image in a Python array. The array's shape is (200,260,3). This 3 means that the array is 3 dimensional and that is because the computer actually sees this image as 3 images. One red, one blue and one green. This is the RGB system of representation of colored images. Each pixel in each of the 3 images, has a value that is when combined with the other 2 values gives the colored pixel that we see on our screens.
Now that I explained this array representation, I will dive into the analysis and preparation of the images.

Color

The first thing to think about after understanding how images are interpreted by a computer, is how to reduce the amount of data. Think of it as a sort of dimensionality reduction. Is there a way to reduce the dimension of my arrays to less than 3? Yes, by considering the following questions.
Wil color affect the model's accuracy?
Will color affect the model's efficiency of learning?
The answer to the first question is no. Color can be essential in other types of problems. But in my case, we only care about the edges and general shape of the shoe. Color will not make the classification better or worse. Hence, I can change all the images in my dataset to greyscale.
As for the second, the answer is definitely yes! And that is because changing the images from color to greyscale as I explained above, will change the shape of the arrays of each image from (200,260,3) to (200,260,1), which is just (200,260). making all the arrays 1 dimension sized instead of 3 will contribute greatly to the efficiency of learning of my model.

As I will include in the appendix, I wrote a Python script that changes the images from color to greyscale.

Size

Generally speaking and from a theoretical standpoint, size should not matter much. However, in my case in particular, I came across pre-trained models in Keras which I can harness their power. Those pre-trained models use Transfer Learning, the concept that I explained earlier in the theoretical part of this thesis.
The pre-trained model that I used could only accept images (arrays) of the size: (224, 224). For that reason, I had to write a python scrip (included in appendix) that cropped the picture from the top and added white pixels (padding) to the sides so that each image gets transformed from (200,260) to (224, 224).

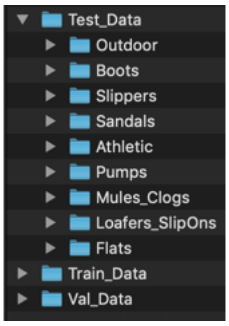
Directories

Another thing that I had to adapt for in order to use the pretrained Keras model, is the structure of directories in which my images are located.
Instead of having all classes in one director, I had to divide my data into 3 directories:

- Train

- Test
- Validate

I also wrote a Python script (included in appendix) that divided the images into the directories required by the pretrained model (show below).



```
▼ Test_Data
    ▶ Outdoor
    ▶ Boots
    ▶ Slippers
    ▶ Sandals
    ▶ Athletic
    ▶ Pumps
    ▶ Mules_Clogs
    ▶ Loafers_SlipOns
    ▶ Flats
▶ Train_Data
▶ Val_Data
```

Noise

Another very crucial observation that I have had about images, is the great effect that noise can have on a neural network's predictions.

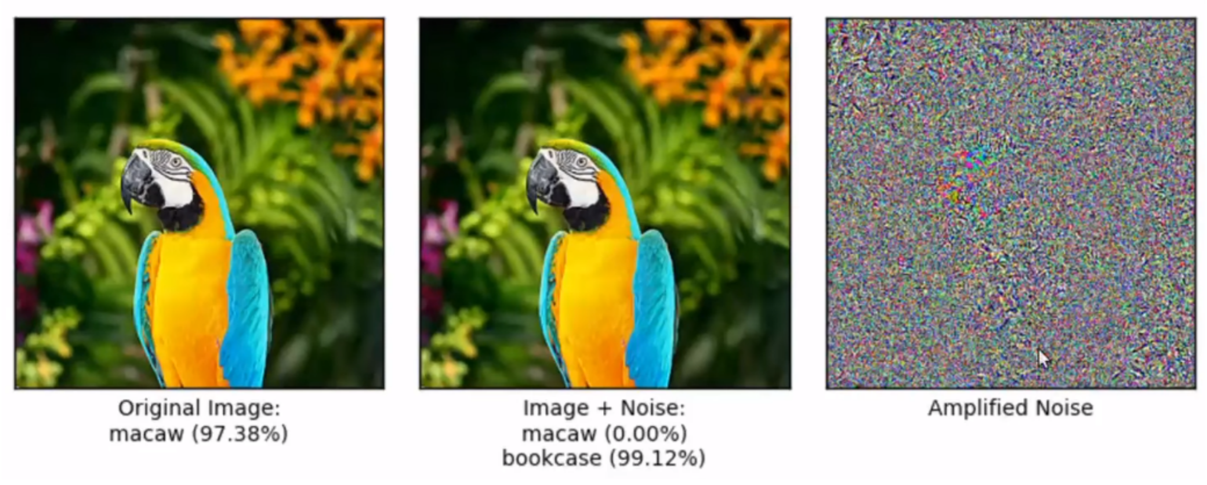


Figure 0-11 - Effect of Noise on ORI

In figure 5-1, noise has been added of only a 3.0 limit per pixel. This is just a mere change of 1.2% per pixel value. In this example, the noise has been added to the Inception Model – A model that has been developed and pre-trained by TensorFlow to recognize and predict the content of images. The user can use the model for predicting images or use it for transfer learning on their own project.

Noise has been added in a wat that will alter the prediction of the image to a bookcase. After adding the noise, ***which is invisible to the human eye***, the true prediction (Macaw) decreased from 97% to zero.

I believe that this is a very crucial observation as humans are starting to depend on computer vision for use in self-driving cars, for example. This case shows that an invisible noise to the human eye can alter the vision of the machine entirely, which could lead to drastic life threatening consequences in the future, if not addressed properly.
I checked for noise within the dataset that I have collected and fortunately did not find any.

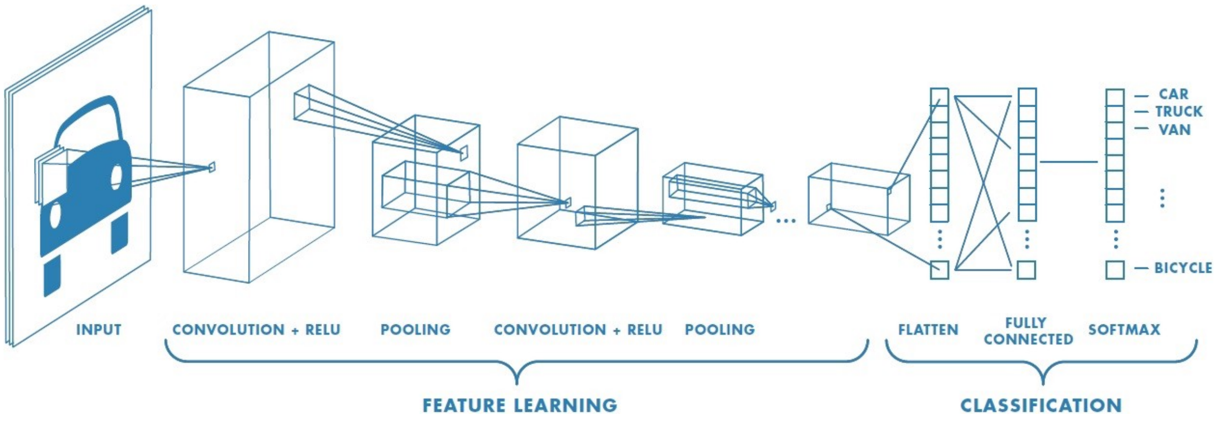
Applying Deep Learning on my dataset

The dataset is finally ready for DL. In this section I will apply deep learning on the dataset I collected using the Keras library and also on the famous MINST dataset using TensorFlow.

Choosing the right Deep Learning model for Image Classification

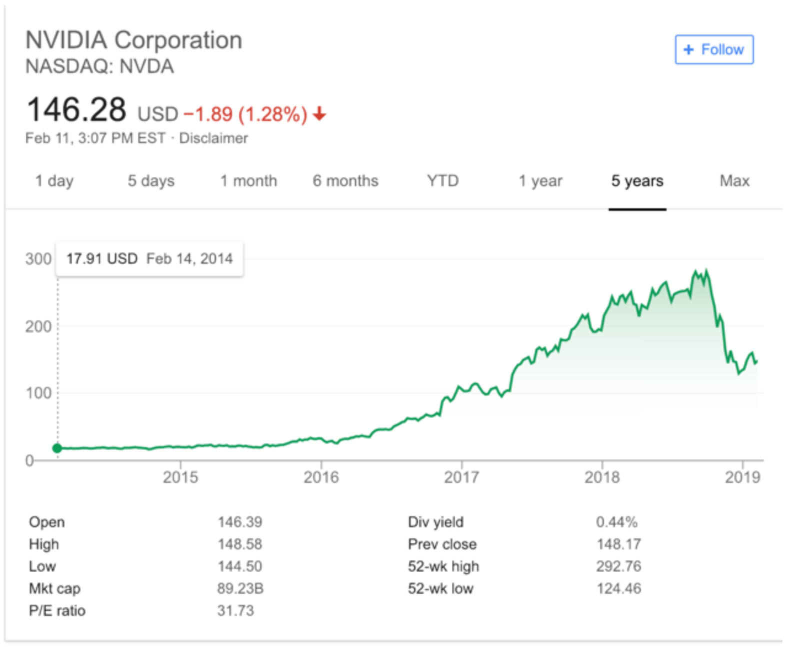
convolutional neural networks (CNNs or ConvNets)

I explained the theory behind CNNs in the theoretical section. They are the best model to use when it comes to Image classification.
Convolutional neural networks are used to find patterns in an image. They do that by convoluting over an image and looking for patterns. In the first few layers of CNNs the network can identify lines and corners, but we can then pass these patterns down through our neural net and start recognizing more complex features as we get deeper. This property makes CNNs really good at identifying objects in images.



Local Vs Cloud

DL is a very computationally expensive process. And as bigger the dataset gets, the more and more expensive it gets to train a DL model. Luckily for Nvidia, the Computer gaming company, producing GPUs became the best business decision they have ever made.


NVIDIA Corporation
NASDAQ: NVDA
146.28 USD −1.89 (1.28%)
Feb 11, 3:07 PM EST · Disclaimer

Open	146.39	Div yield	0.44%
High	148.58	Prev close	148.17
Low	144.50	52-wk high	292.76
Mkt cap	89.23B	52-wk low	124.46
P/E ratio	31.73		

As ML and DL became more and more popular, the sales of GPUs skyrocketed and with it, Nvidia’s shares.
As it turns out, GPUs are so fast because they are so efficient for matrix multiplication and convolution. The reason for this is memory bandwidth and parallelism. CPUs are latency optimized while GPUs are bandwidth optimized. A good analogy would be a Ferrari (CPU) and a truck (GPU). A Ferrari can go so fast but has very limited capacity, while a truck is the opposite. So, it depends on the application at hand, in my analogy of the truck and Ferrari, DL needs to deliver huge loads of volume not necessarily in fast way.
Powerful GPUs are not very accessible to everyone, that is why the big Internet Services companies like I discussed before, started to make a business out of renting GPUs for machine and deep learning. I, for example, ran my model locally on a Mac. Macs has AMD GPUs. Unfortunately for me, AMD GPUs are not supported by Keras library.
In later sections, I will run the model using those big internet companies and also locally, I will show the results and compare.

Trying out the 3 big players [AWS, Google & MS Azure]

I compared those 3 services earlier in the theoretical section. Here, I will compare them from my own practical experience.
All 3 providers offer the one service that I would be using for my thesis and that is Virtual Machine (VM) custom made for Deep Learning. They run on GPUs and come out of the box:

- Ubuntu OS
- Python

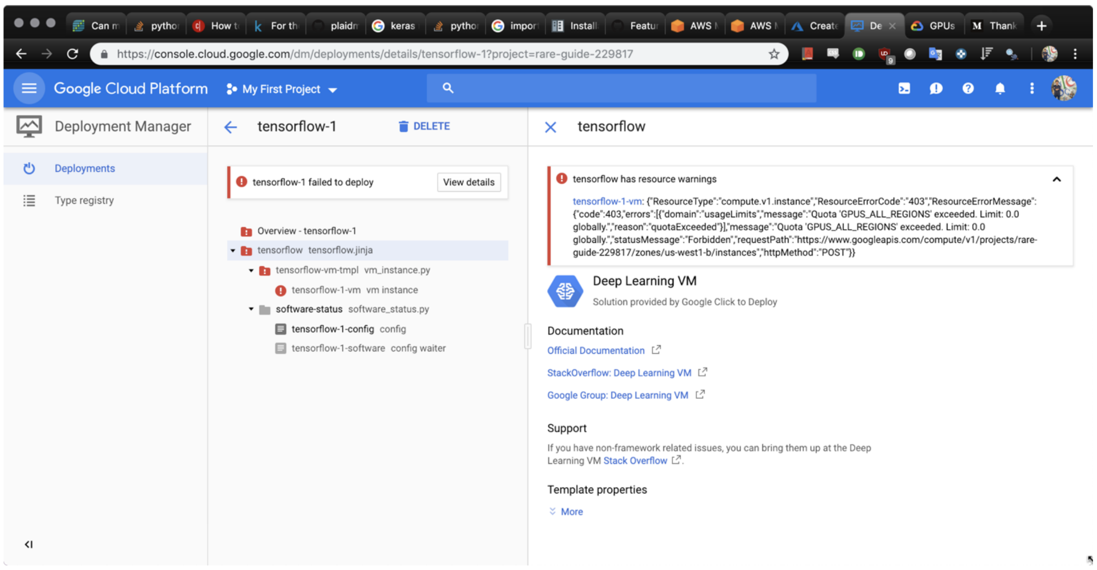
- Nvidia GPU drivers optimized for deep learning
- TensorFlow, Keras and all the necessary libraries for ML/DL.

What varies from one provider to the other are:

- the level of technical support that they offer
- how big is the community of developers that use this particular provider
- student credit offered for students for free
- ease of use (User interface)
- documentation
- ability to SSH into the created VM, push and pull files
- Support for Jupyter Notebooks (Data science industry standard nowadays).

Google

Google checked all the boxes above. They even gave me 300$ worth of credit as a student. They only had one problem that took me a long time to figure out. When I ran the VM I got an error that was not clear.



tensorflow-1 failed to deploy

Overview - tensorflow-1
tensorflow tensorflow.jinja
tensorflow-vm-tmpl vm_instance.py
tensorflow-1-vm vm instance
software-status software_status.py
tensorflow-1-config config
tensorflow-1-software config waiter

tensorflow has resource warnings

```
tensorflow-1-vm: {"ResourceType":"compute.v1.instance","ResourceErrorCode":"403","ResourceErrorMessage":
{"code":403,"errors":[{"domain":"usageLimits","message":"Quota 'GPUS_ALL_REGIONS' exceeded. Limit: 0.0
globally.","reason":"quotaExceeded"}],"message":"Quota 'GPUS_ALL_REGIONS' exceeded. Limit: 0.0
globally.","statusMessage":"Forbidden","requestPath":"https://www.googleapis.com/compute/v1/projects/rare-
guide-229817/zones/us-west1-b/instances","httpMethod":"POST"}}
```

Deep Learning VM
Solution provided by Google Click to Deploy

Documentation
Official Documentation
StackOverflow: Deep Learning VM
Google Group: Deep Learning VM

Support
If you have non-framework related issues, you can bring them up at the Deep Learning VM Stack Overflow.

Template properties
More

I tried to look online for help but could not find anything. Finally, and after a long time, tech support replied to my email saying that there is a GPU quota that I need to increase from 0 to whatever number of GPUs I need. Their response was too late that I it did not give me a chance to try their services due to the timeframe of this thesis.

MS Azure

Azure gave me 100$ as a student. It is worth to mention that registration was the easiest since CVUT is already registered with MS services.
I could deploy the VM easily and SSH into the VM from my physical machine. The hardest part was that the VM was not ready with Jupyter Notebooks out of the box. Which was a bit tricky to set up.

Amazon Web Services AWS

AWS gave me 45$ of credit as a student, but that was one year ago when I first tried exploring Deep Learning. Unfortunately, I could not have more credit to experiment with AWS for this thesis. This was a shame, because AWS is the market leader now. It has the biggest community of developers and they're famous for their outstanding customer support.

Conclusion – Selecting one cloud services provider

I had to go with MS Azure due to the lack of credits problems provided by AWS and the slowness of Google's tech support.

Using TensorFlow on the MINST Set – locally on my physical machine

I will be using in this example the MNIST data set. This data set consists of a training set of 60,000 labelled hand written digits and another 10,000 as test set of hand written digits.

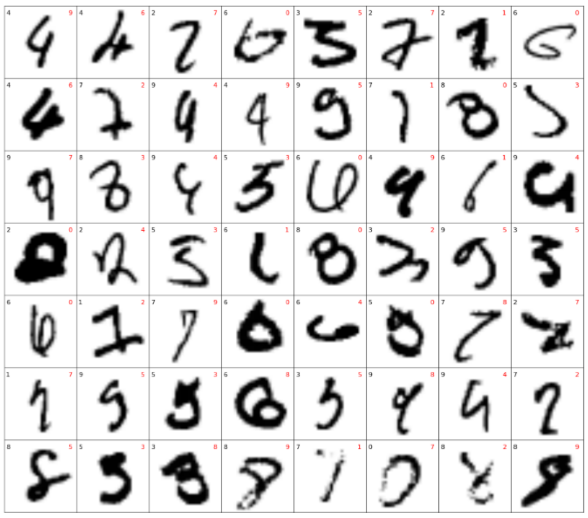


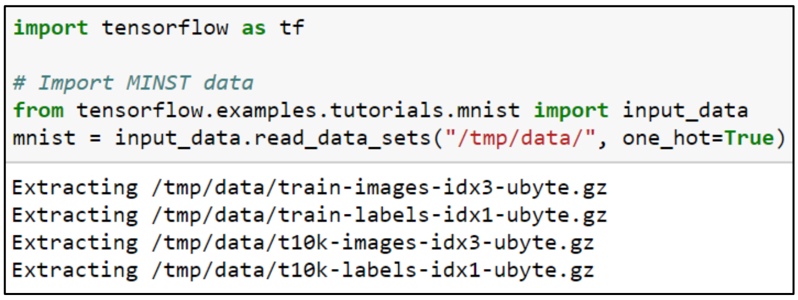
Figure 0-12 - MNIST Data Set
The images are black and white images of size 28 x 28 pixels, or 784 pixels total. Our features will be the pixel values for each pixel. Either the pixel is "white" (blank with a 0), or there is some pixel value.

I will try to correctly predict what number is written down based solely on the image data in the form of an array.

Working with the MNIST data set in Deep Learning is the equivalent of printing "Hello World" when starting to learn a new programming language.

Importing TensorFlow & MNIST

This data set is so popular to the extent that it is embedded within TensorFlow.



```
import tensorflow as tf

# Import MINST data
from tensorflow.examples.tutorials.mnist import input_data
mnist = input_data.read_data_sets("/tmp/data/", one_hot=True)
```

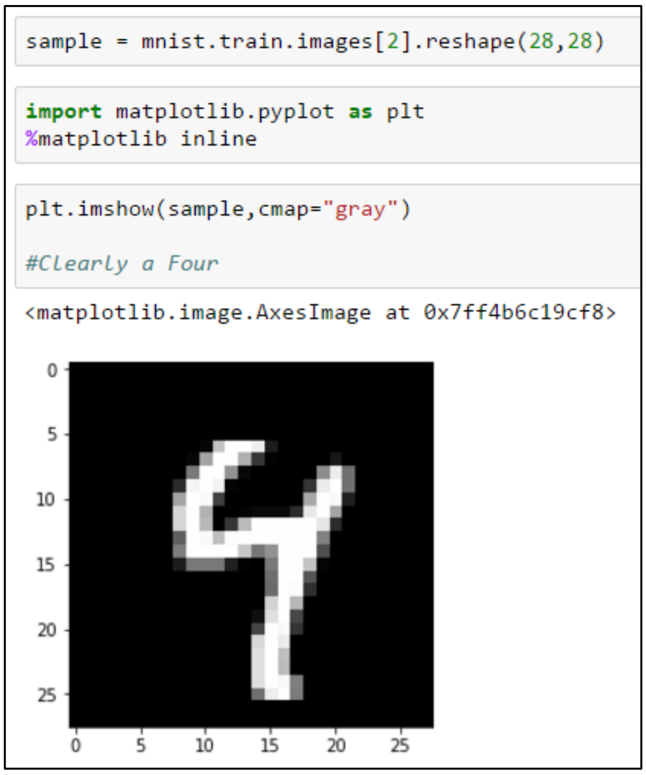
```
Extracting /tmp/data/train-images-idx3-ubyte.gz
Extracting /tmp/data/train-labels-idx1-ubyte.gz
Extracting /tmp/data/t10k-images-idx3-ubyte.gz
Extracting /tmp/data/t10k-labels-idx1-ubyte.gz
```

Figure 0-13 - Importing MNIST

Visualizing a Sample

First, I reshaped the vector into a 28*28 matrix. As mentioned before, the data set consists of flattened 28*28 = 784 vectors. The reshape is made to visualize the image in 2D.

The Matplotlib library (imported below) is essential to visualizing data in Python, but it is beyond the scope of this paper.



```
sample = mnist.train.images[2].reshape(28,28)
```

```
import matplotlib.pyplot as plt
%matplotlib inline
```

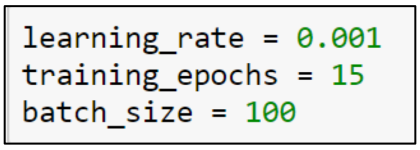
```
plt.imshow(sample,cmap="gray")

#Clearly a Four
```

```
<matplotlib.image.AxesImage at 0x7ff4b6c19cf8>
```

Determining the deep learning parameters

As mentioned, this data set is very popular and so, the optimum parameters for initializing a deep learning using this data set are already known. However, I will tweak those parameters after the first trial and analyze how it affects the accuracy of the model's predictions.
These parameters are:

- Learning Rate - How quickly to adjust the cost function.
- Training Epochs - How many training cycles to go through
- Batch Size - Size of the 'batches' of training data



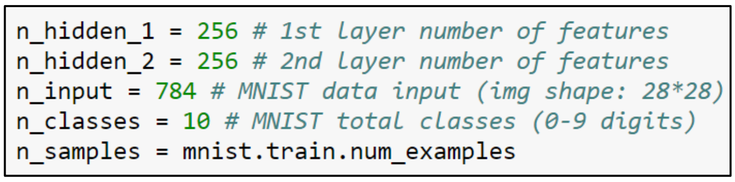
```
learning_rate = 0.001
training_epochs = 15
batch_size = 100
```

Figure 0-14 - Learning Parameters

Network Parameters

These parameters define our neural network. They vary according to how the data looks like.

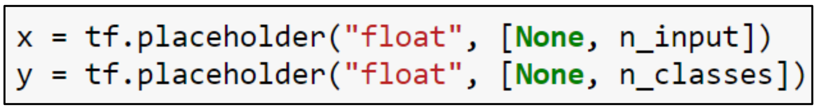
4. *n_hidden_1*: Number of features in the first layer
5. *n_hidden_2*: Number of features in the second layer
6. *n_input*: Data shape (i.e. L * W pixels)
7. *n_classes*: Data total number of classes
8. *n_samples*: Training samples



```
n_hidden_1 = 256 # 1st layer number of features
n_hidden_2 = 256 # 2nd layer number of features
n_input = 784 # MNIST data input (img shape: 28*28)
n_classes = 10 # MNIST total classes (0-9 digits)
n_samples = mnist.train.num_examples
```

Figure 0-15 - Network Parameters

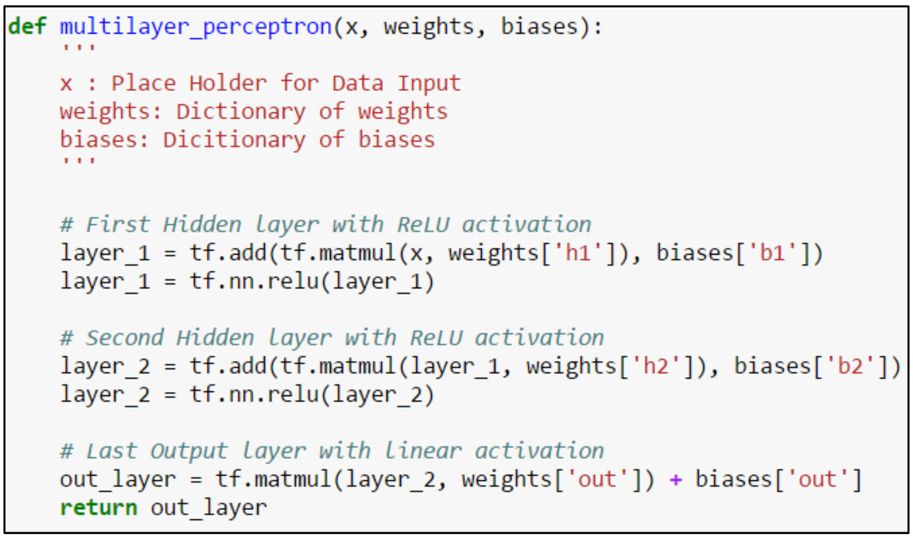
TensorFlow Graph Input



```
x = tf.placeholder("float", [None, n_input])
y = tf.placeholder("float", [None, n_classes])
```

- Any TensorFlow computation is represented as a dataflow graph.
- A placeholder is simply a variable that we will assign data to later.

Multi-Layer Model



```
def multilayer_perceptron(x, weights, biases):
    '''
    x : Place Holder for Data Input
    weights: Dictionary of weights
    biases: Dicitionary of biases
    '''

    # First Hidden layer with ReLU activation
    layer_1 = tf.add(tf.matmul(x, weights['h1']), biases['b1'])
    layer_1 = tf.nn.relu(layer_1)

    # Second Hidden layer with ReLU activation
    layer_2 = tf.add(tf.matmul(layer_1, weights['h2']), biases['b2'])
    layer_2 = tf.nn.relu(layer_2)

    # Last Output layer with linear activation
    out_layer = tf.matmul(layer_2, weights['out']) + biases['out']
    return out_layer
```

Figure 0-16 - Defining the MLP

Firstly, the input data array is received then is sent to the first hidden layer. Then weights (randomly only initially) will be assigned to the data a and then sent to a node to undergo an activation function along with the Bias. Then it will continue on to the next hidden layer, and so on until the final output layer. In this example, there are just 2 hidden layers. The more the hidden layers the higher the possibility for a more accurate model. However, it is a trade-off between run time and accuracy.

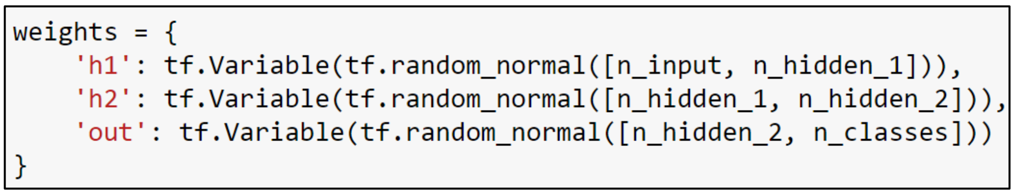
Once the transformed data reaches the output later it is evaluated using the error function that determines how far off is the output from the expected results. In this case, how many classes are correct.

Then, an optimization function is applied to minimize that error and adjust the weights accordingly.

Application of this optimization can be adjusted by altering the learning rate parameter. The lower the rate the higher the possibility of higher accuracy, but it is again a trade-off between accuracy and run time. At some point, lowering the learning rate will not increase accuracy and will reach a reach terminal value.

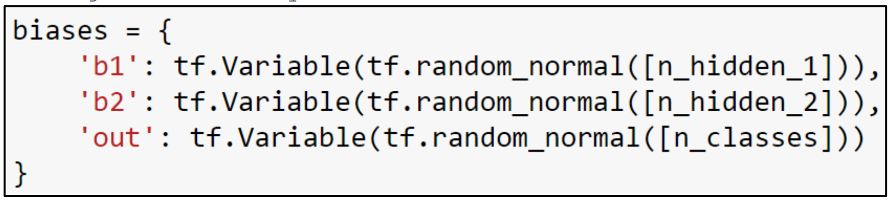
Weights and Bias

In order for our tensorflow model to work, two dictionaries containing our weight and bias objects for the model need to be created. The **tf.variable** object is used in this example. This is different from a constant because TensorFlow's Graph Object becomes aware of the states of all the variables. A Variable is a modifiable tensor that lives in TensorFlow's graph of interacting operations. It can be used and even modified by the computation.



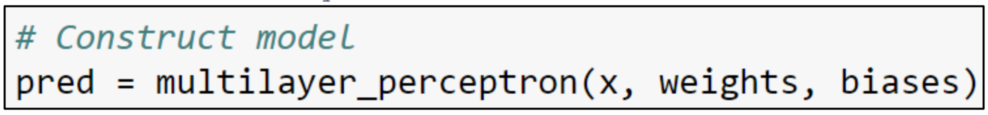
```
weights = {
    'h1': tf.Variable(tf.random_normal([n_input, n_hidden_1])),
    'h2': tf.Variable(tf.random_normal([n_hidden_1, n_hidden_2])),
    'out': tf.Variable(tf.random_normal([n_hidden_2, n_classes]))
}
```

Figure 0-17 - Weights Dictionary



```
biases = {
    'b1': tf.Variable(tf.random_normal([n_hidden_1])),
    'b2': tf.Variable(tf.random_normal([n_hidden_2])),
    'out': tf.Variable(tf.random_normal([n_classes]))
}
```

Figure 0-18 - Biases Dictionary

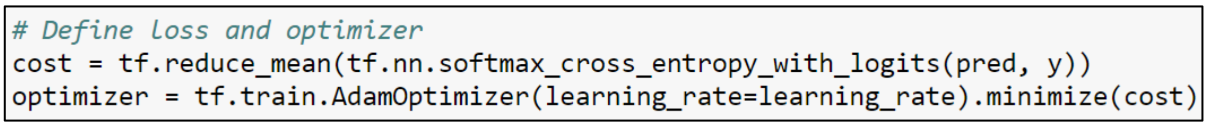


```
# Construct model
pred = multilayer_perceptron(x, weights, biases)
```

Figure 0-19 - Model Construction

Cost and Optimization Functions

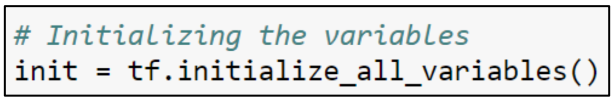
Here TensorFlow's built-in functions will be used.



```
# Define loss and optimizer
cost = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(pred, y))
optimizer = tf.train.AdamOptimizer(learning_rate=learning_rate).minimize(cost)
```

Figure 0-20 - Cost and Optimization Functions

Initialization of Variables



```
# Initializing the variables
init = tf.initialize_all_variables()
```

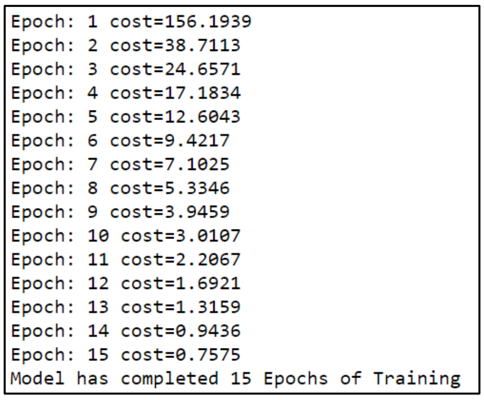
Figure 0-21 - Variable Initialization

Running the Session

The session has 2 loops. The outer loop which runs the epochs, and the inner loop which runs the batches for each epoch of training

```python
# Launch the session
sess = tf.InteractiveSession()

# Intialize all the variables
sess.run(init)

# Training Epochs
# Essentially the max amount of loops possible before we stop
# May stop earlier if cost/loss limit was set
for epoch in range(training_epochs):

    # Start with cost = 0.0
    avg_cost = 0.0

    # Convert total number of batches to integer
    total_batch = int(n_samples/batch_size)
```

```python
    # Loop over all batches
    for i in range(total_batch):

        # Grab the next batch of training data and labels
        batch_x, batch_y = mnist.train.next_batch(batch_size)

        # Feed dictionary for optimization and loss value
        # Returns a tuple, but we only need 'c' the cost
        # So we set an underscore as a "throwaway"
        _, c = sess.run([optimizer, cost], feed_dict={x: batch_x, y: batch_y})

        # Compute average loss
        avg_cost += c / total_batch

    print("Epoch: {} cost={:.4f}".format(epoch+1,avg_cost))

print("Model has completed {} Epochs of Training".format(training_epochs))
```



```
Epoch:  1 cost=156.1939
Epoch:  2 cost=38.7113
Epoch:  3 cost=24.6571
Epoch:  4 cost=17.1834
Epoch:  5 cost=12.6043
Epoch:  6 cost=9.4217
Epoch:  7 cost=7.1025
Epoch:  8 cost=5.3346
Epoch:  9 cost=3.9459
Epoch:  10 cost=3.0107
Epoch:  11 cost=2.2067
Epoch:  12 cost=1.6921
Epoch:  13 cost=1.3159
Epoch:  14 cost=0.9436
Epoch:  15 cost=0.7575
Model has completed 15 Epochs of Training
```

Figure 0-22 – Session Output

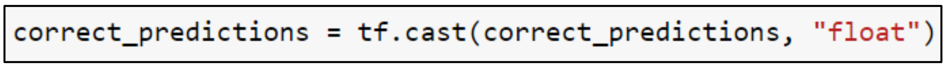
Model Evaluations

Tensorflow comes with some built-in functions to ease model evaluate, including ***tf.equal*** and ***tf.cast*** with ***tf.reduce_mean***.

```python
# Test model
correct_predictions = tf.equal(tf.argmax(pred, 1), tf.argmax(y, 1))
```

Figure 0-23 - Model Evaluation

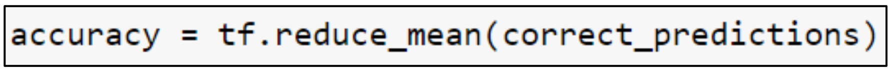
In order to get a numerical value for the predictions, ***tf.cast*** will be used to cast the Tensor of Booleans back into a Tensor of Floating point values in order to take the mean of it.



```
correct_predictions = tf.cast(correct_predictions, "float")
```

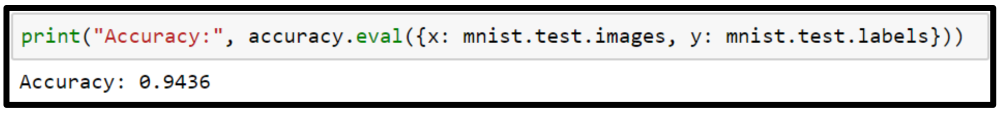
Figure 0-24 - Getting Numerical Values

Now the ***tf.reduce_mean*** function will be used in order to grab the mean of the elements across the tensor.



```
accuracy = tf.reduce_mean(correct_predictions)
```

Now all what is left is to pass in the actual test data (labelled images) to evaluate accuracy. The ***eval()*** method lets the user directly evaluate this tensor in a Session without needing to call ***tf.sess()***.



```
print("Accuracy:", accuracy.eval({x: mnist.test.images, y: mnist.test.labels}))
Accuracy: 0.9436
```

Figure 0-25 - MLP's Accuracy

An accuracy of 94% has been achieved. This accuracy can still be increased if the learned rate got decreased and more hidden layers were added to the model.

Using Keras on my own web scrapped dataset

As mentioned before, Keras abstracts all the hard work so that data scientists can experiment and prototype their work as fast as possible. I demonstrated working TensorFlow and it required a deep solid understanding of the different parameters in order to get started. Keras, however, is much simpler to work with. It is built on top of TensorFlow (as well as libraries). I will showcase in the upcoming section how easy it is to get a model running using Keras.

Keras pre-trained applications – Transfer Learning

Keras offers applications that are basically deep learning models that are made available alongside pre-trained weights. These models can be used for prediction, feature extraction, and fine-tuning. They use Transfer Learning (described before), so the models are extensively pretrained on different objects. It is a very beneficial concept as I will demonstrate.
I will be using the **VGG16** model, with weights pre-trained on ImageNet. ImageNet is an image dataset organized according to the WordNet hierarchy. Each meaningful concept in WordNet, possibly described by multiple words or word phrases, is called a "synonym set" or "synset". There are more than 100,000 synsets in WordNet, majority of them are nouns (80,000+). In ImageNet, we aim to provide on average 1000 images to illustrate each synset. Images of each concept are quality-controlled and human-annotated. In its completion, we hope ImageNet will offer tens of millions of cleanly sorted images for most of the concepts in the WordNet hierarchy.

Keras VGG16 Application

All code is included in the appendix of this thesis.
Though I needed to adapt my dataset to the requirements of VGG16, after that everything was very smooth. However, VGG16 turned out to have lots of hidden layers that was very expensive to run locally.

Keras Sequential Model – creating my own neural network

This model is offered by Keras where I had the ability to create my own neural network by defining layers. This turned out to be a very tough and complex process. Though much simpler than TensorFlow, it still required a deep understanding of everything, since I realized that adding layers without knowing what is happening, eventually accuracy comes to a halt and eventually gets reduced.

Comparison and Takeaways

These are the results of my experimentation.

	Keras Sequential		**Keras VGG16**	
	Azure	Locally	Azure	Locally
Time (mins)	15	150	12	96
Accuracy	15%	16%	29%	27%

It is clear that there is a huge difference in accuracy between running DL on my own dataset versus running DL on the most popular dataset online. I found out that DL is an iterative process. In order to achieve such high accuracy as the one I achieved when running the TensorFlow model on the MNIST dataset (94%), I would need to:

1. Research the topic further in order to reach a more solid and deep understanding of every step in the process so that I have a concrete ground upon which I can base my selection of the different parameters involved.
2. run and rerun the model while tweaking and changing the parameters and types of added hidden layers till an adequate accuracy get achieved.
3. More time for training the model as my time was limited due to the timeframe of this thesis.

Conclusion - Reviewing Deep Learning for Image Classification

This thesis about was extremely complex for me since I had no knowledge whatsoever of computer science or AI or Deep Learning. I taught myself everything that was mentioned in this thesis (and much more that was not worth mentioning but was necessary to learn to get to this humble level) and with absolutely zero external help and while being just a bachelor's thesis, I consider it an achievement that I am proud of.

Deep Learning is changing the world as we know it. It is an extremely powerful tool that has an endless amount of potential. It is indeed a very complex and tough field to study and research. It requires a concrete understanding and knowledge of statistics, linear algebra and computer science.

As a solution of the classification of images, I found that DL was enormously effective. However, there are a lot of problems that I faced when I tried to apply DL on my own real world problem. But in the same time, it is quite achievable. The field is still young and evolving exponentially. According to arxiv.org, 60 papers are published per day on Machine Learning (which is the parent field of Deep Learning). This shows the limitless potential that those fields currently have.

Challenges faced

This journey seemed like a constant, yet enjoyable, struggle with a vast amount of problems faced on a daily basis. Thanks to the vast Python community and rising popularity of DL, I managed to solve all the challenges either through trial and error, digging through library documentation or asking on forums on online. Examples of challenges that I faced (all mentioned in the thesis and how I overcame them) are but not limited to:

- Collecting Data
- Labelling Data
- Preparing Data
- Using Pretrained models vs. training my own model

I learned how to overcome the problem of scarcity of structured data by planning adequately before scraping the web. I managed to obtain a very high quality data set. After that, I found out that the data is not labeled which turned out to be a major obstacle. So, I came up with the idea of tweaking my web scraping script to label the dataset for me. I then realized that the structured labelled dataset is still not ready for DL. I wrote several other scripts to resize,

discolor the images of my dataset and finally divide the images into 3 different categories: training, testing and validating.
Lastly, my conclusions on DL for image classification is that it is extremely effective and efficient. Though, my results were not as good as I anticipated, there is definitely room for improvement. This improvement lies within studying deep learning further as well as tweaking the model and implementing trial and error mind set to optimize the model as much as possible.

References

[1] B. Marr, ‘Why Artificial Intelligence Will Change Our World And Why It Needs To Be Purposeful’, *Forbes*. [Online]. Available: https://www.forbes.com/sites/bernardmarr/2016/05/25/why-artificial-intelligence-will-change-our-world-and-why-it-needs-to-be-purposeful/. [Accessed: 28-May-2017].

[2] ‘Industrial Revolution - Facts & Summary’, *HISTORY.com*. [Online]. Available: http://www.history.com/topics/industrial-revolution. [Accessed: 28-May-2017].

[3] ‘1926: Field Effect Semiconductor Device Concepts Patented | The Silicon Engine | Computer History Museum’. [Online]. Available: http://www.computerhistory.org/siliconengine/field-effect-semiconductor-device-concepts-patented/. [Accessed: 28-May-2017].

[4] ‘Vint Cerf and Bob Kahn, co-inventors of TCP/IP protocol | FierceTelecom’. [Online]. Available: http://www.fiercetelecom.com/special-report/vint-cerf-and-bob-kahn-co-inventors-tcp-ip-protocol. [Accessed: 28-May-2017].

[5] ‘John McCarthy: Computer scientist known as the father of AI | The Independent’. [Online]. Available: http://www.independent.co.uk/news/obituaries/john-mccarthy-computer-scientist-known-as-the-father-of-ai-6255307.html. [Accessed: 28-May-2017].

[6] ‘A new dawn’, *artificial-intelligence*. [Online]. Available: https://www.ubs.com/microsites/artificial-intelligence/en/new-dawn.html. [Accessed: 13-Jun-2017].

[7] Chris Smith, ‘The History of Artificial Intelligence’.

[8] ‘Data, data everywhere’, *The Economist*, 25-Feb-2010.

[9] N. D. Lewis, *Deep learning made easy with R: a gentle introduction for data science*. Place of publication not identified: AusCov, 2016.

[10] J. Heaton and J. Heaton, *Deep learning and neural networks*. St. Louis, MO: Heaton Research, Inc, 2013.

[11] ‘Multilayer Perceptron — DeepLearning 0.1 documentation’. [Online]. Available: http://deeplearning.net/tutorial/mlp.html. [Accessed: 15-Jun-2017].

[12] ‘CS231n Convolutional Neural Networks for Visual Recognition’. [Online]. Available: http://cs231n.github.io/convolutional-networks/. [Accessed: 18-Jun-2017].

[13] P. on F. 20, 2017 in Connected Devices, Industrial, S. Cities, and Transport, ‘Who wins the three-way cloud battle? Google vs. Azure vs. AWS’, *ReadWrite*, 20-Feb-2017.

[Online]. Available: https://readwrite.com/2017/02/20/wins-three-way-cloud-battle-google-vs-azure-vs-aws-dl1/. [Accessed: 18-Jun-2017].

[14] R. Lawson, *Web Scraping with Python: scrape data from any website with the power of Python*. 2015.

[15] ‘What is High-Level Language? Webopedia Definition’. [Online]. Available: http://www.webopedia.com/TERM/H/high_level_language.html. [Accessed: 04-Jun-2017].

[16] ‘Introduction to object-oriented languages’, *Lynda.com - from LinkedIn*. [Online]. Available: https://www.lynda.com/Programming-Foundations-tutorials/Introduction-object-oriented-languages/83603/90484-4.html. [Accessed: 04-Jun-2017].

[17] D. Kouzis-Loukas, *Learning Scrapy: learn the art of efficient web scraping and crawling with Python.* 2016.

[18] R. Mitchell, *Web scraping with Python: collecting data from the modern web*. 2015.

[19] ‘HTML basics’, *Mozilla Developer Network*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/HTML_basics. [Accessed: 03-Jun-2017].

[20] ‘class’, *Mozilla Developer Network*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTML/Global_attributes/class. [Accessed: 03-Jun-2017].

[21] ‘CSS basics’, *Mozilla Developer Network*. [Online]. Available: https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/CSS_basics. [Accessed: 03-Jun-2017].

[22] ‘Organizing data with dictionaries’, *Lynda.com - from LinkedIn*. [Online]. Available: https://www.lynda.com/Python-tutorials/Organizing-data-dictionaries/62226/71001-4.html. [Accessed: 04-Jun-2017].

Appendix

Resizing and greyscale script

```
def resize_and_greyscale():
    path =
'/Users/touah04/Desktop/final_push/Thesis/pycharm/deeplearning/resize_trial_all/Dat
a/'
    parent_path_resized =
'/Users/touah04/Desktop/final_push/Thesis/pycharm/deeplearning/resize_trial_all/res
ized_Data'
    final_size = 224
    # loop for items in Data: ['Test_Data', 'Train_Data', 'Val_Data']
    parent_dirs = os.listdir(path)
    for parent_dir in parent_dirs:
        if parent_dir == '.DS_Store':
            continue
        else:
            # loop for items within each Data category
            data_category_dirs = os.listdir(os.path.join(path, parent_dir))
            for data_category_dir in data_category_dirs:
                if data_category_dir == '.DS_Store':
                    continue
                else:
                    shoe_imgs = os.listdir(os.path.join(path, parent_dir,
data_category_dir))
                    for shoe_img in shoe_imgs:
                        path_to_img = os.path.join(path, parent_dir,
data_category_dir, shoe_img)
                        if shoe_img == '.DS_Store':
                            continue
                        else:
                            im = Image.open(path_to_img)
                            size = im.size
                            ratio = float(final_size) / max(size)
                            new_image_size = tuple([int(x * ratio) for x in size])
                            img_name = os.path.splitext(shoe_img)[0]
                            im = im.resize(new_image_size, Image.ANTIALIAS)
                            new_im = Image.new("L", (final_size, final_size),
color=255)
                            new_im.paste(im, (
                            (final_size - new_image_size[0]) // 2, (final_size -
new_image_size[1]) // 2))
                            path_to_resized_img = os.path.join(parent_path_resized,
parent_dir, data_category_dir)
                            if not os.path.exists(path_to_resized_img):
                                os.makedirs(path_to_resized_img)
                            new_im.save(path_to_resized_img + '/' + img_name +
'_resized.jpg', 'JPEG', quality=90)
```

Keras VGG16

```
import os, random
import numpy as np
import pandas as pd
import PIL
import keras
import itertools
from PIL import Image

from keras import backend as K
from keras.optimizers import Adam
from keras.models import Sequential
from keras.layers import Dense
from keras.preprocessing.image import ImageDataGenerator
```

```
import zipfile
import time

def read_data_from_dir():
    paths_of_classes_req = os.listdir(os.path.join(os.getcwd(), 'Data/Test_Data'))
    classes_required = []
    for path in paths_of_classes_req:
        if path != ".DS_Store":
            classes_required.append(path)


def keras_model(classes_required):

    '''Converting Data Format according to the backend used by Keras
    '''

    root_path = os.getcwd()

    batch_size_train = 16
    batch_size_val = 16
    batch_size_test = 16

    datagen =
keras.preprocessing.image.ImageDataGenerator(data_format=K.image_data_format())

    '''Input the Training Data
    '''
    train_path = os.path.join(root_path, 'Data/Train_Data')
    train_batches = ImageDataGenerator().flow_from_directory(train_path,
target_size=(224, 224),

classes=classes_required, batch_size=batch_size_train)

    '''Input the Validation Data
    '''
    val_path = os.path.join(root_path, 'Data/Val_Data/')
    val_batches = ImageDataGenerator().flow_from_directory(val_path,
target_size=(224, 224), classes=classes_required,

batch_size=batch_size_val)

    '''Input the Test Data
    '''
    test_path = os.path.join(root_path, 'Data/Test_Data/')
    test_batches = ImageDataGenerator().flow_from_directory(test_path,
target_size=(224, 224), classes=classes_required,

batch_size=batch_size_test)

    test_imgs, test_labels = next(test_batches)

    y_test = [np.where(r == 1)[0][0] for r in test_labels]

    vgg16_model = keras.applications.vgg16.VGG16(include_top=False,
weights='imagenet', input_tensor=None,
                                                 input_shape=(224, 224, 3),
pooling=None, classes=9)
    vgg16_model.summary()


    model = Sequential()  # Iterate over the functional layers and add it as a
stack
    model.add(Dense(9, input_shape=(224, 224, 3)))
    for layer in vgg16_model.layers:
        model.add(layer)

    model.layers.pop()
    model.summary()

    for layer in model.layers:  # Since the model is already trained with certain
weights, we dont want to change it. Let it be the same
        layer.trainable = False

    model.add(Dense(9, activation='sigmoid'))  # Add the last layer
    model.summary()

    # Compile the model
    model.compile(Adam(lr=.00015), loss='categorical_crossentropy',
metrics=['accuracy'])

    # Train the model
    model.fit_generator(train_batches, steps_per_epoch=5,
                        validation_data=val_batches, validation_steps=5, epochs=1,
verbose=1)


if __name__ == '__main__':
    start_time = time.time()
    keras_model(read_data_from_dir())
    print("--- %s seconds ---" % (time.time() - start_time))
```

Keras Sequential

```
from keras.applications.vgg16 import VGG16
from keras.preprocessing import image
from keras.applications.vgg16 import preprocess_input
import numpy as np

import os, random
import numpy as np
import pandas as pd
import PIL
import keras
import itertools
from PIL import Image

from keras.layers.convolutional import Convolution2D, MaxPooling2D, ZeroPadding2D
from keras import backend as K
from keras.models import Sequential
from keras.layers import Input, Dropout, Flatten, Conv2D, MaxPooling2D, Dense,
Activation
from keras.optimizers import RMSprop
from keras.callbacks import ModelCheckpoint, Callback, EarlyStopping
from keras.optimizers import Adam
from keras.models import Model
from keras.utils import np_utils
from keras.models import Sequential
from keras.layers import Input, Dropout, Flatten, Conv2D, MaxPooling2D, Dense,
Activation
from keras.optimizers import RMSprop
from keras.callbacks import ModelCheckpoint, Callback, EarlyStopping
from keras.utils import np_utils
from keras.preprocessing.image import ImageDataGenerator, array_to_img,
img_to_array, load_img

import zipfile
import time
def read_data_from_dir():
    paths_of_classes_req = os.listdir(os.path.join(os.getcwd(), 'Data/Test_Data'))
    classes_required = []
    for path in paths_of_classes_req:
        if path != ".DS_Store":
            classes_required.append(path)

def keras_model(classes_required):
    '''Converting Data Format according to the backend used by Keras
        '''

    root_path = os.getcwd()

    batch_size_train = 16
    batch_size_val = 16
    batch_size_test = 16
    num_classes = 9
    intereseted_folder = 'Documents'
    STANDARD_SIZE = (224, 224)

    '''Converting Data Format according to the backend used by Keras
    '''
    datagen =
keras.preprocessing.image.ImageDataGenerator(data_format=K.image_data_format())

    '''Input the Training Data
    '''
    train_path = os.path.join(root_path, 'Data/Train_Data')
    train_batches = ImageDataGenerator().flow_from_directory(train_path,
color_mode='grayscale', target_size=(224, 224),

classes=classes_required, batch_size=batch_size_train)

    '''Input the Validation Data
    '''
    val_path = os.path.join(root_path, 'Data/Val_Data/')
    val_batches = ImageDataGenerator().flow_from_directory(val_path,
color_mode='grayscale', target_size=(224, 224),

classes=classes_required,

batch_size=batch_size_val)

    '''Input the Test Data
    '''
    test_path = os.path.join(root_path, 'Data/Test_Data/')
    test_batches = ImageDataGenerator().flow_from_directory(test_path,
color_mode='grayscale', target_size=(224, 224),

classes=classes_required,

batch_size=batch_size_test)

    test_imgs, test_labels = next(test_batches)

    y_test = [np.where(r == 1)[0][0] for r in test_labels]


    model = Sequential()
    # model.add(ZeroPadding2D((1, 1), input_shape=(224, 224, 1)))
    model.add(Dense(9, input_shape=(224, 224, 1)))

    model.add(Convolution2D(64, 3, 3, activation='relu'))
    model.add(ZeroPadding2D((1, 1)))
    model.add(Convolution2D(64, 3, 3, activation='relu'))
    model.add(MaxPooling2D((2, 2), strides=(2, 2)))

    model.add(ZeroPadding2D((1, 1)))
    model.add(Convolution2D(128, 3, 3, activation='relu'))
    model.add(ZeroPadding2D((1, 1)))
    model.add(Convolution2D(128, 3, 3, activation='relu'))
    model.add(MaxPooling2D((2, 2), strides=(2, 2)))
    model.add(ZeroPadding2D((1, 1)))
    model.add(Convolution2D(256, 3, 3, activation='relu'))
    model.add(ZeroPadding2D((1, 1)))
    model.add(Convolution2D(256, 3, 3, activation='relu'))
    model.add(ZeroPadding2D((1, 1)))
    model.add(Convolution2D(256, 3, 3, activation='relu'))
    model.add(MaxPooling2D((2, 2), strides=(2, 2)))

    model.add(ZeroPadding2D((1, 1)))
    model.add(Convolution2D(512, 3, 3, activation='relu'))
    model.add(ZeroPadding2D((1, 1)))
    model.add(Convolution2D(512, 3, 3, activation='relu'))
    model.add(ZeroPadding2D((1, 1)))
    model.add(Convolution2D(512, 3, 3, activation='relu'))
    model.add(MaxPooling2D((2, 2), strides=(2, 2)))

    model.add(ZeroPadding2D((1, 1)))
    model.add(Convolution2D(512, 3, 3, activation='relu'))
    model.add(ZeroPadding2D((1, 1)))
    model.add(Convolution2D(512, 3, 3, activation='relu'))
    model.add(ZeroPadding2D((1, 1)))
    model.add(Convolution2D(512, 3, 3, activation='relu'))
    model.add(MaxPooling2D((2, 2), strides=(2, 2)))

    model.add(Flatten())
    model.add(Dense(9, activation='softmax'))  # Add the last layer
    model.summary()

    # Compile the model
    model.compile(Adam(lr=.00015), loss='categorical_crossentropy',
metrics=['accuracy'])

    # Train the model
    model.fit_generator(train_batches, steps_per_epoch=5,
                        validation_data=val_batches, validation_steps=5, epochs=1,
verbose=1)


if __name__ == '__main__':
    start_time = time.time()
    keras_model(read_data_from_zip())
    print("--- %s seconds ---" % (time.time() - start_time))
```

Understanding the Web

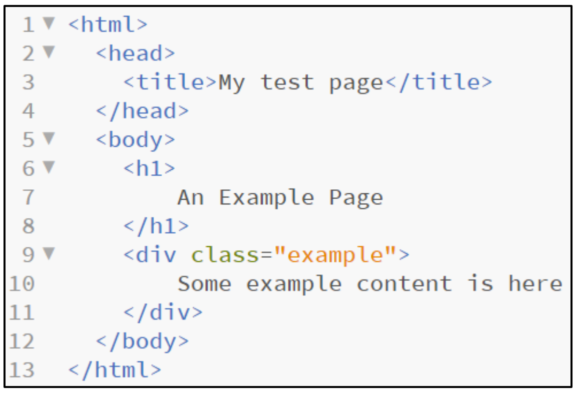
As you dig deeper in the world of Web Scraping, you will be impressed by those web browsers that we take for granted and the code that lies underneath those webpages. In fact, the websites that you open on a daily basis and absolutely love, would look hostile and unpleasant without: 1- **JavaScript** which, makes a webpage interactive. 2- **CSS**, that is responsible for formatting the webpage and make it looking nice and welcoming. 3- **HTML**, that describes and defines the content of the webpage.[18]

HTML & CSS

In order to fully understand the fundamentals of Web Scraping, one will to understand the mechanics of code behind a webpage as that is how we are going to access and select exactly the data that we need.

HTML

Some people think that Hypertext Mark-up Language (HTML) is a programming language. In fact, HTML is a mark-up language that outlines the content of the webpage by a series of elements. Each of those elements dictate how the content wrapped inside it is to be rendered in the browser on the user's screen. All the HTML code should be surrounded between <html> and </html> tags. Also, tags like <head> & <body> are essential parts of a webpage's HTML. Let's explore one of the abovementioned elements: the paragraph element, which is defined by the tag <p>. This element defines the content wrapped inside as a **paragraph**. Now that we understand what an HTML tag is, let's look at a simple HTML code to analyse it.[19]



```html
<html>
  <head>
    <title>My test page</title>
  </head>
  <body>
    <h1>
        An Example Page
    </h1>
    <div class="example">
        Some example content is here
    </div>
  </body>
</html>
```

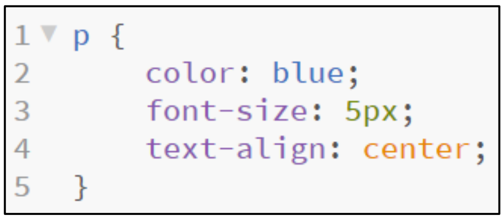
As mentioned before, everything should be enclosed between a <html> and a </html>. The <head> tag contains content that describes things that are not rendered on the screen. It has keywords and information like a page's description that will be useful for search engines to find the webpage.

The <body> element wraps all the viewable content of the webpage inside it. It can contain tags such as <p> (paragraph) <img> (images) <li> (lists) and <a> (links). The <body> element above has an <h1> tag and a <div> tag. The <div> tag simple defines a division or a section within the document while the <h1> tag is simple a heading. Headings in HTML is marked up from <h1> to <h6>[18].

There is something very important to notice, though. There is an attribute given to the abovementioned <div> element called *class*. There are 2 types of attributes that can be given to any kind of HTML element: class or ID. These 2 methods are used to select elements when writing your CSS (To apply style to this particular element) or JavaScript (to trigger that element programmatically)[20].

CSS

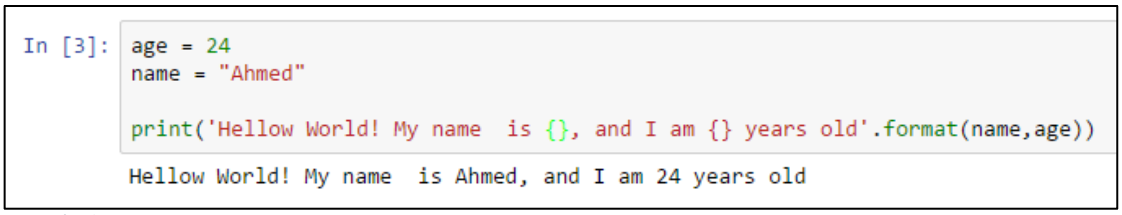
As much as HTML is not a programming language, Cascading Style Sheets (CSS) also isn't. However, it is not a mark-up language either. From its name, we can say CSS is a styling language that lets you style the elements included in your HTML documents[21]. Put very simply, if you want to apply some styling to all the paragraph elements in your HTML, you put the following code in your CSS file:



```css
p {
    color: blue;
    font-size: 5px;
    text-align: center;
}
```

We can see how CSS defines things like colour, position and background of different elements on a webpage.
For knowing more about HTML & CSS, I suggest opening the tutorials of W3Schools as well as right clicking anywhere in a webpage and navigate *page source*.

Python Crash Course

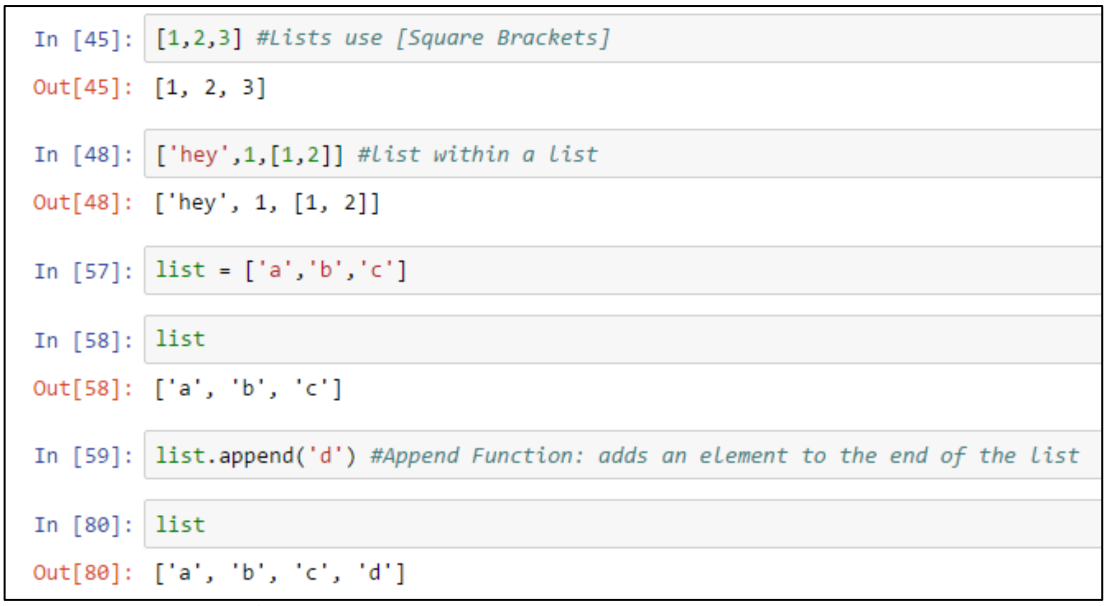


```
In [3]: age = 24
        name = "Ahmed"

        print('Hellow World! My name  is {}, and I am {} years old'.format(name,age))

        Hellow World! My name  is Ahmed, and I am 24 years old
```

Figure 0-1 - Hello World!

From the example above, we can see how Python is highly readable and very logical to write.

Lists

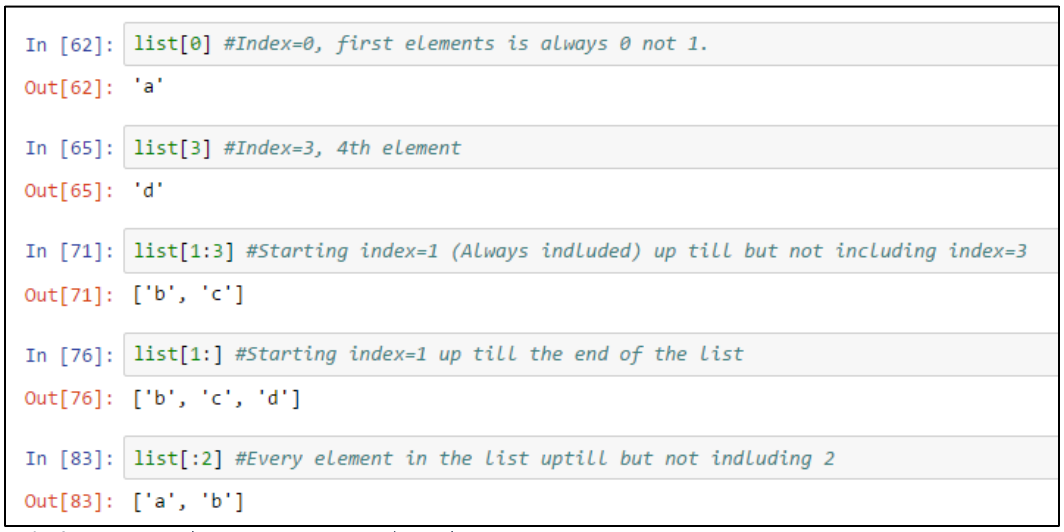


```
In [45]: [1,2,3] #Lists use [Square Brackets]

Out[45]: [1, 2, 3]

In [48]: ['hey',1,[1,2]] #list within a list

Out[48]: ['hey', 1, [1, 2]]

In [57]: list = ['a','b','c']

In [58]: list

Out[58]: ['a', 'b', 'c']

In [59]: list.append('d') #Append Function: adds an element to the end of the list

In [80]: list

Out[80]: ['a', 'b', 'c', 'd']
```

Figure 0-2 - Python Lists

The Python list object is the most universal classification provided by the language. Lists are ordered collections of randomly typed objects, and they have no fixed size. They are also mutable—unlike strings, lists can be altered.

Indexing and Selection

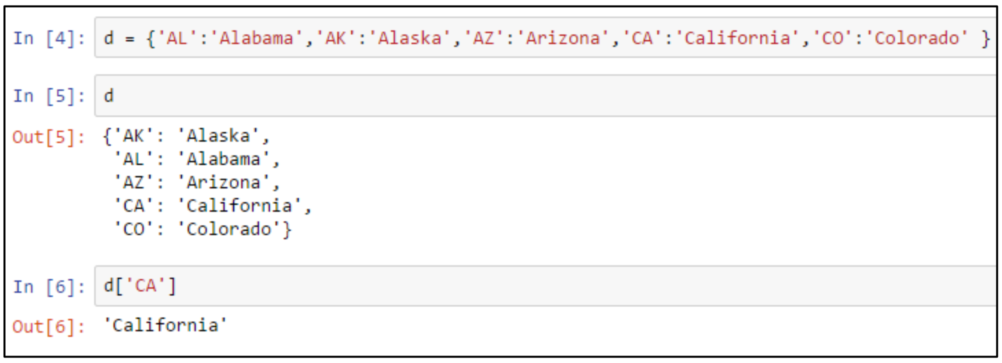


```
In [62]: list[0] #Index=0, first elements is always 0 not 1.
Out[62]: 'a'

In [65]: list[3] #Index=3, 4th element
Out[65]: 'd'

In [71]: list[1:3] #Starting index=1 (Always indluded) up till but not including index=3
Out[71]: ['b', 'c']

In [76]: list[1:] #Starting index=1 up till the end of the list
Out[76]: ['b', 'c', 'd']

In [83]: list[:2] #Every element in the list uptill but not indluding 2
Out[83]: ['a', 'b']
```

Figure 0-3 - Indexing and Selection in Python

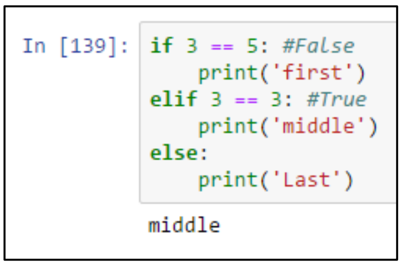
Dictionaries

A dictionary in Python is a list but with a custom index for each element called “key”. The key can have any value and it is how you call or select the element.[22]



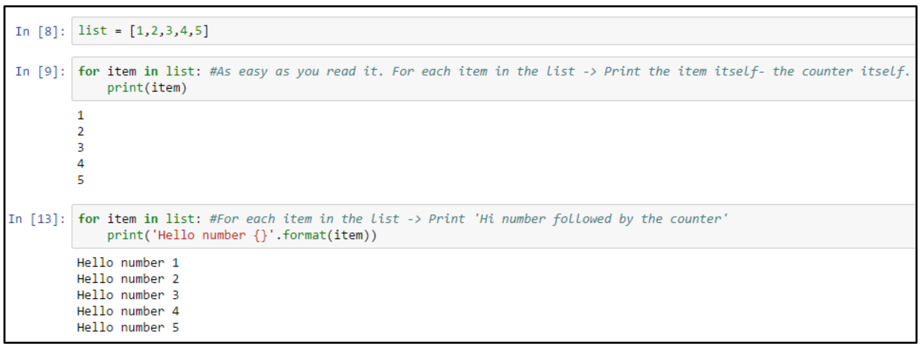
```
In [4]: d = {'AL':'Alabama','AK':'Alaska','AZ':'Arizona','CA':'California','CO':'Colorado' }

In [5]: d
Out[5]: {'AK': 'Alaska',
          'AL': 'Alabama',
          'AZ': 'Arizona',
          'CA': 'California',
          'CO': 'Colorado'}

In [6]: d['CA']
Out[6]: 'California'
```

If, elseif and else statements



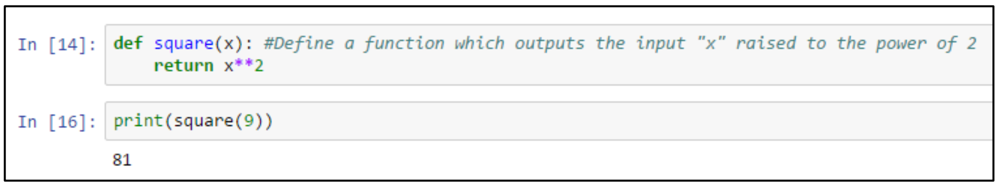
```
In [139]: if 3 == 5: #False
              print('first')
          elif 3 == 3: #True
              print('middle')
          else:
              print('Last')

          middle
```

Loops



```
In [8]: list = [1,2,3,4,5]

In [9]: for item in list: #As easy as you read it. For each item in the list -> Print the item itself- the counter itself.
            print(item)

        1
        2
        3
        4
        5

In [13]: for item in list: #For each item in the list -> Print 'Hi number followed by the counter'
             print('Hello number {}'.format(item))

         Hello number 1
         Hello number 2
         Hello number 3
         Hello number 4
         Hello number 5
```

Functions



```
In [14]: def square(x): #Define a function which outputs the input "x" raised to the power of 2
             return x**2

In [16]: print(square(9))

         81
```