Brand Logo Detection and Recognition

B.Sc. (HONS) Computing with Mobile App Development

Supervisor: Robert Sheehy

Second Reader: Helen Fitzgerald

Student Number: T00178529

Student: Csaba Bangó



Institute of Technology Tralee, Dromthacker, Tralee, Co. Kerry 2018

# Abstract

Artificial Neural Networks (ANN) are computational models based on mathematics, biology and computer science. Due to the large amount of unstructured data, Deep Learning algorithms are developed every day. Computer scientists are training computers to teach themselves, enabling machines to mimic the way human experience is acquired. ANNs have been applied to a variety of tasks including image processing, where experience-based algorithms brought great success. Transfer Learning – a method of Machine Learning – is the approach when a model is trained for a specific task (video prediction, natural language processing, object detection etc.) which is partially recycled and repurposed to a second target, a custom object. The release of Tensorflow Object Detection API allowed lightweight computer vision models to be utilised as a base of various computer vision tasks.

The aim of this thesis is to evaluate how specific objects - brand logos - can be detected and recognised using this API and the Tensorflow Machine Learning framework. The data used was pre-processed and stored in TFRecord files that were originated from Flickr’s logo dataset in training and testing sets. Once the data was formatted efficiently it was fed into a Single Shot MultiBox Detection, MobileNet model and configured to replace the last few layers of the neural network. When the learning was satisfactory the inference graph was restored to the API. The new model was then ready to detect in real-time.

As image processing is a well-studied area, Google (parent of Tensorflow) provides detection models work with high accuracy. Further improvements could have been made with additional image augmentation on the input data.

Table of Contents

[Abstract 1](#_Toc513500354)

[1 Introduction 5](#_Toc513500355)

[2 Computer Vision 6](#_Toc513500356)

[2.1 Introduction 6](#_Toc513500357)

[2.2 Computer Vision 6](#_Toc513500358)

[2.3 Logo Detection 8](#_Toc513500359)

[3 Neural Networks 10](#_Toc513500360)

[3.1 Introduction 10](#_Toc513500361)

[3.2 Artificial Intelligence 10](#_Toc513500362)

[3.3 Machine Learning 10](#_Toc513500363)

[3.4 Deep Learning 11](#_Toc513500364)

[3.5 Artificial Neural Networks (ANN) 12](#_Toc513500365)

[3.5.1 Key Fundamentals 12](#_Toc513500366)

[3.5.2 Activation functions 13](#_Toc513500367)

[3.5.3 Nodes 14](#_Toc513500368)

[3.5.4 Input layer 14](#_Toc513500369)

[3.5.5 Hidden Layer 15](#_Toc513500370)

[3.5.6 Output layer 15](#_Toc513500371)

[3.5.7 Feedforward 15](#_Toc513500372)

[3.5.8 Backpropagation 15](#_Toc513500373)

[3.6 Convolutional Neural Network (CNN) 16](#_Toc513500374)

[3.7 Faster R-CNN 16](#_Toc513500375)

[3.8 Transfer Learning 16](#_Toc513500376)

[4 Technologies 18](#_Toc513500377)

[4.1 Machine Learning Libraries 18](#_Toc513500378)

[4.1.1 Theano 18](#_Toc513500379)

[4.1.2 Scikit-learn 18](#_Toc513500380)

[4.1.3 Tensorflow 18](#_Toc513500381)

[4.1.4 Tensorflow Lite 18](#_Toc513500382)

[4.2 Evaluation of Machine Learning Libraries 19](#_Toc513500383)

[4.3 Tensorflow Object Detection API 20](#_Toc513500384)

[4.4 Programming languages 20](#_Toc513500385)

[4.5 Datasets 20](#_Toc513500386)

[5 Methodology 21](#_Toc513500387)

[5.1 Research undertaken 21](#_Toc513500388)

[5.2 Research Question 21](#_Toc513500389)

[5.3 Proposed Project Implementation 21](#_Toc513500390)

[5.3.1 Functional specifications 22](#_Toc513500391)

[5.4 System design 23](#_Toc513500392)

[5.4.1 Architectural Design 23](#_Toc513500393)

[5.5 Prototype 24](#_Toc513500394)

[6 Implementation 25](#_Toc513500395)

[6.1 Sprints 25](#_Toc513500396)

[6.1.1 Sprint 1 – Data preparation 26](#_Toc513500397)

[6.1.2 Sprint 2 – Converting data to TFRecord 30](#_Toc513500398)

[6.1.3 Sprint 3 – Setup Configuration file 33](#_Toc513500399)

[6.1.4 Sprint 4 – Train the model (graph) 35](#_Toc513500400)

[6.1.5 Sprint 5 – Export trained model 38](#_Toc513500401)

[6.1.6 Sprint 6 – Apply the model to classify logos 40](#_Toc513500402)

[6.1.7 Sprint 7 – Train the model on the Cloud 42](#_Toc513500403)

[7 Findings & Conclusions 44](#_Toc513500404)

[7.1 Computational Performance 44](#_Toc513500405)

[7.2 Training Observations 45](#_Toc513500406)

[7.3 Research Question Answered 49](#_Toc513500407)

[8 References 50](#_Toc513500408)

# Introduction

To implement a Logo Recognising Application, the field of Object Detection, a subset of Computer Vision must be investigated. This is a widely researched area, where the most advanced Machine Learning (ML) algorithms are utilised. Thanks to these Deep Learning methods, this technology is currently one of the most matured parts of Artificial Intelligence. The increasing number of models developed are perfecting the performance of these applications rapidly. This thesis discusses several of these algorithms as well as the role of Artificial Neural Networks (ANN), ML and its use in the field of Computer Vision. It will also explain how the logo recognising application can apply some of these techniques while benefiting from Transfer Learning.

# Computer Vision

## Introduction

Computer Vision (CV) is an area of science that enables computers to observe the way humans can see. This ability allows for considerable improvements in various fields of computer science. This includes enabling the reconstructing of 3D objects, increases security in the monitoring system, improves copyright protection, facial and object recognition, avoiding obstacles in autonomous driving, pharmaceutical computer systems, etc. Furthermore, understanding brands on images and in videos means new opportunities in social media marketing strategies (Yue Gao, 2014) by developing statistical data on the company’s online performance (Tie Liu, 2008).

The image data can take many forms including videos, images, depth video sequences and the view from multiple cameras. Some of these computational processes are considered highly expensive. Consequently, effective automation of computer vision tasks, the involvement of Artificial Intelligence (AI) and Machine Learning (ML) techniques have become inevitable.

## Computer Vision

The first occurrences of computer vision are from 1966 when Marvin Minsky (1927-2016, cognitive scientist and co-founder of the Massachusetts Institute of Technology’s AI laboratory) introduced the problem of computer vision. For many years scientists searched for the correct strategies to tackle this problem but the success ratio remained low. Partial success was achieved in this research area in 1973 with what was called The Constellation Model (M.A. Fischler, 1973). It attempts to detect certain points in the image and map them under mutual geometric constraints.

Numerous other experiments were introduced such as edge detection techniques (Lowe, 1985) however the speed of progress for the experiments was minimal.

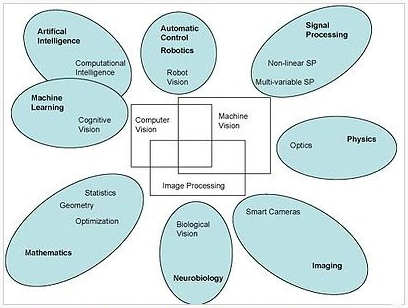


Figure 2.1 – Relationship between Computer Vision and other relevant fields (Chung, 2013)

The first real breakthrough came in 1989 (Y. LeCun, 1989) when the backpropagation algorithm was applied to the MNIST handwritten digit dataset. This method is still used in artificial neural networks to calculate the error contribution on the neurons based on the adjustment of their weight and bias. LeCun and Co. also proved that the convolution of filters was one of the main reason why the recognition worked. (The techniques applied here are typically present in brand logo recognition.)

Some other notable milestones:

1998 Successful Face Recognition. (H.A. Rowley, 1998)

1999 SIFT (Scale Invariant Feature Transform) algorithm (Lowe, 1999) where Lowe introduced a technique to not slide windows instead have better features in a way that every pixel has a gradient and an orientation.

## Logo Detection

As computer vision technologies have developed they have become more embedded into software applications. In 2001 the first object detection framework was developed by Paul Viola and Michael Jones that could detect objects in real-time. (Paul Viola, 2001). Even though initially it aimed to solve the face recognition problem it soon became understood that this algorithm may be used to detect a variety of other objects. This technique is called the Haar Cascade Detection and is implemented in many image processing libraries including OpenCV and MATLAB. The core concept of this technique is that the cascade function is trained using positive and negative images and later matched with these elements in other images where the same object is to be detected. For four years this algorithm was considered the most efficient Machine Learning technique applied in real-time detection until in 2005, Dalal’s Histograms of Oriented Gradients (HOG) took over. “After reviewing existing edge and gradient based descriptors, we show experimentally that grids of histograms of oriented gradient descriptors significantly outperform existing feature sets.” (N. Dalal, 2005) HOG shared some similar features to its predecessor Haar. A multi-scaling sliding window was used with both approaches significantly slowing down the process. However, it was the first time when boundaries properly worked. In 2012 Machine Learning arrived at the next milestone with the new algorithm introduced by Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. At the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) their system successfully detected objects with the implementation of a deep convolutional neural network (DCNN). They trained a large DCNN to classify 1.2 million images into 1000 different classes and achieved an average of 85% accuracy (Alex Krizhevsky, 2012).

Logo detection is a special case of image recognition that aims to recognise the logo name of the input image or video. It is utilised in a wide range of applications for intelligent transportation (Shengmei Lin, 2016), commercial application for social media analysis as well for the protection of intellectual properties in web-based platforms (Steven C.H. Hoi, 2015). Before any brand logo datasets were published firstly the classification problem needed to be solved. In logo detecting applications numerous methods stand. The earlier approaches took advantage of the boosting-based cascaded classifier, which adopted low-level feature representations. However, the detection accuracy here remained low (Qiting Ye, 2017). [Pourghassem](http://ieeexplore.ieee.org/search/searchresult.jsp?searchWithin=%22Authors%22:.QT.Hossein%20Pourghassem.QT.&newsearch=true) proposed a system based on a two-stage sequential segmentation and classification strategy. He used a multilayer perceptron and k-nearest neighbour classification techniques to distinguish between logos. This resulted in high accuracy and could detect a limited number of manually proposed brands (Pourghassem, 2012). Most recent researchers state that logo detection applications are best coupled with forms of Convolutional Neural Networks (CNNs). Forrest’s DCNN based application reached a considerably high accuracy but the cost remained high(Forrest N. Iandola, 2015). In 2016 Boia proposed a system to use homographic class graphs for brand logo localization and recognition respectively (Raluca Boia, 2016) (Raluca Boia, 2015). In a 2017 research paper produced by Bombonato, a method called Single Shot MultiBox Detector (SSD) was implemented that experimented with “pretrained weights and the impact of warp transformations in the input images” (Leonardo Bombonato, 2017). This approach uses a single deep neural network that outperformed the arguably most successful model ever, Deep Convolutional Neural Network (DCNN) by the combination of the two.

In conclusion, many approaches exist regarding logo detection. Based on this research, when applying a machine learning technique several important criteria must be set. What kind of dataset is being used (include classified images, the amount and quality of classes involved), how important the speed is, what accuracy is acceptable, what computing power is necessary and given etc. These issues will further be elaborated in Chapter 4.

# Neural Networks

## Introduction

This chapter reviews the areas of Artificial Intelligence, Machine Learning and Deep Learning. Computer Vision (CV) is a subset of Deep Learning. Computer vision is a subset of artificial intelligence that is also part of the general computer science itself. To understand how CV works the developer must learn both conceptual and practical aspects of machine learning and artificial intelligence.

## Artificial Intelligence

Artificial Intelligence is a branch of computer science, a neuroscience-inspired system that utilises the aspects of human cognition such as perception, communication, problem-solving and most importantly, learning. (Dominiek Sandra, 2009) In earlier days, this concept was explained differently but the intentions were clear for decades: “The term artificial intelligence denotes behaviour of a machine which, if a human behaves in the same way, is considered intelligent.” (Asa B. Simmons, 1988)

## Machine Learning

Machine Learning is a subset of artificial intelligence. It involves numerous techniques that allow machines to learn from experience and does not require direct programming as it is learning by self.

AppliedAICourse has put together a list to outline the key concepts required to understand ML (AppliedAICourse, 2017).

1. Probability and statistics
2. Linear Algebra (geometry)
3. Calculus and Numerical optimisation
4. Classification and regression techniques
5. Clustering techniques
6. Matrix and Factorisation
7. Neural networks and Deep learning

Neural networks and Deep learning are to be discussed in the further sections. Points 1-3 were mentioned as more general purpose mathematical calculations but essential with NNs, as well as 4-6 that are very specific to ML techniques. Ultimately, all the above is critical when implementing Artificial Neural Networks.

## Deep Learning

Deep Learning is the subset of Machine Learning inspired by the natural neural network. It is a collection of algorithms that utilises multi-layered neural networks to carry out the computations based on the vast amount of data. This means that these processes are not task-specific, the network system trains itself. There are three types differentiated:

* Supervised 🡪 The majority of practical ANNs are supervised. This means that a network has an input layer, an output layer and at least one hidden layer. The algorithms are used to learn the path in between input and output layer. It is supervised because the learning happens based on a training dataset (has labelled data). Some common problems that can occur with this type of learning are related to classification and regression.
* Unsupervised 🡪 When a network has an input layer but no output layer the model must learn the structure before it would start processing the data. Unlike in supervised learning, there is no data to compare the results to, meaning that the correct answer is unknown. The most occurring related problem is clustering.
* Semi-supervised 🡪 In this scenario the input layer is given but on the output layer only some data is labelled. In this case, both supervised and unsupervised computation is applied. (Brownlee, 2016)

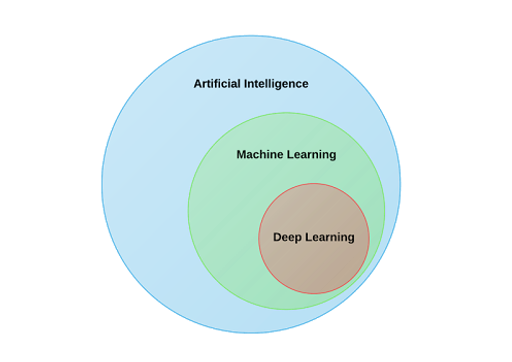


Figure 3.1 - (Amaratunga, 2017)

## Artificial Neural Networks (ANN)

“An Artificial Neural Network (ANN) is a computational model inspired by networks of biological neurons, wherein the neurons compute output values from inputs. It learns from its experience and errors. The neuron is basic calculating entities which compute from a number of inputs and deliver one output comparing with a threshold value and turned on (fired).” (Munish Puri, 2016)

### Key Fundamentals

Neural Networks were designed to have high parallelity. The concept applies to natural neurons as thousands of cells die yearly in the human central nervous system as well yet the functionality of the brain is not affected by it. With sequential computing, the simulation of the system would not be possible.

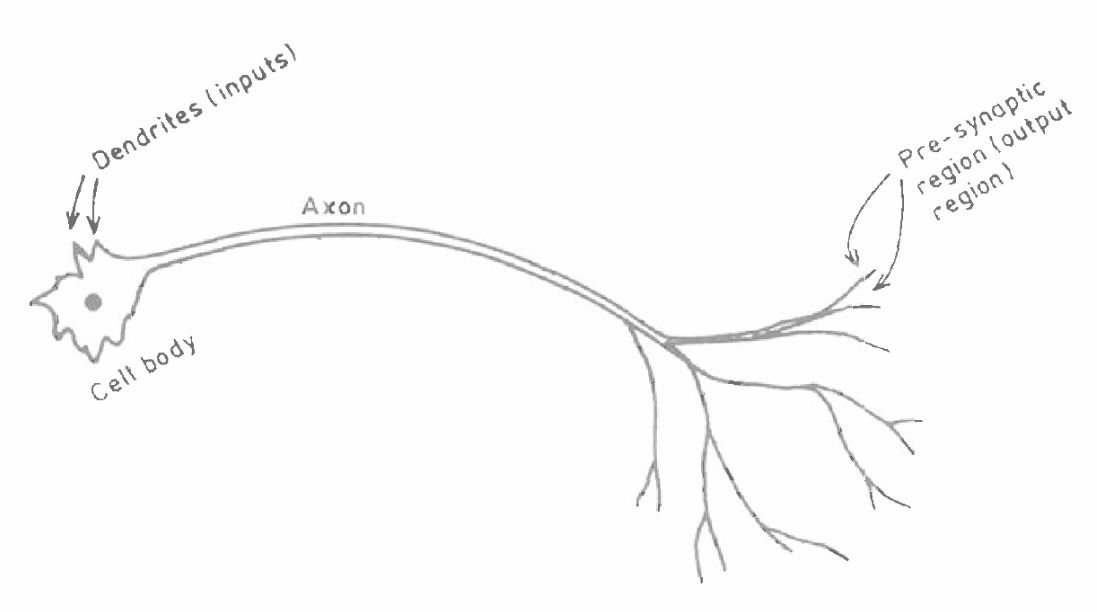


Figure 3.2 – A biological neuron (Graupe, 2014)

The main part of a neuron is the cell body. This is where the computation happens. By electrical triggers, messages (neural signals) are fired from there through the axon reaching other neurons. All neurons are interconnected but not all can communicate with each other. This happens because some transmitters have blocking functionalities. Furthermore, all connections are weighted giving different priorities to the messages. (Graupe, 2014)

The computer science of artificial neural networks were first designed in the form of Perceptron. (Rosenblatt, 1958) These models lack many capabilities of the ANNs of today but its key principles (input/output structure, activation functions etc.) still apply.

### Activation functions

The activation function (denoted as f) is a mathematical equation that serves to keep the outputs within certain boundaries as is the case in the biological environment. (Graupe, 2014) Also used for the measurement of classification performance of different dataset. Figure 3.3 below indicates some of the activation functions. The features are very similar, each producing convincing results. There are some differences that can be noted. Though both Tanh (or hyperbolic tangent function) and Sigmoid are nonlinear, the gradient with Tanh is stronger than it is with Sigmoid. It is noticeable, however, that as these two’s line expand, the Y values are less responsive to changes in X and the gradient disappears. If the activation reaches a straight curve, the learning is irrelevant. ReLU (Rectified Linear Unit) solves the gradient vanishing problem, making it the most recommended activation function (Gangming Zhao, 2017).

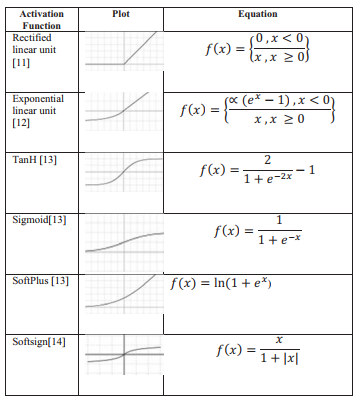


Figure 3.3 – Activation functions (Fatih Ertam, 2017)

### Nodes

Artificial neural networks are composed of multiple interconnected nodes. These nodes aim to mimic the biological neurons. Nodes perform different tasks depending on what layer they are on.

### Input layer

ANNs are organised in layers. Both single and multiple layered neural networks exist, however, multiple layered NNs are more typical due to its higher efficiency and accurate predictions. (More learning means better results.) The input layer is the first layer where patterns (like pixels of an image) are presented to the network. The nodes at this level take numerically expressed information and perform feeding forward.

### Hidden Layer

Hidden layers are located between the output layer and the input layer. It is called hidden because the results cannot be seen. The main disadvantage of ANNs is their “black box” nature. From its developer’s point of view, it is still hard to diagnose how it reaches its conclusions. (Fisher, 2007) Most the computations happen on this layer. Every single node on this layer can take inputs and perform some computational operations on the inputs that are to be passed to the next node(s). Data forward propagate and later backpropagate through these layers. These two are the main operations of the Artificial Neural Networks.

### Output layer

Output layer is a collection of output nodes. It is responsible for transferring data from the inside of the network to the outside world.

### Feedforward

Feedforward propagation refers to the method when the information only moves in one direction (forward) starting at the input layer travelling through the hidden and arriving at the output layer. There are no adjustments made on the weights of connections.

### Backpropagation

Backpropagation is the architecture of ANNs that allows the network to learn complicated, multidimensional mappings. (Hecht-Nielsen, 1989) Backpropagation consists of two major parts. Feedforwards propagation and backpropagation. When the information arrives at the output layer after the first forward propagation completed the output is compared with the desired result. Using the data learned the backpropagation starts adjusting the weights of connections to bring the next result closer to the desired result. When the difference is acceptably negligible the training is complete. The mathematical calculations involved in this process are highly complex, but most machine learning libraries offer off-the-shelf methods with the cost of losing significant control.

## Convolutional Neural Network (CNN)

This is a deep, feed-forward neural network. It utilises multi-layer perceptron to ensure minimal pre-processing. As a result, CNNs are vastly used in image processing but are not suitable for other systems requiring learning such as speech recognition system.

## Faster R-CNN

Tensorflow Object Detection API is an open sourced API by Google that is likely to be implemented in the proposed project. The API is based on this model.

In 2014 Region based Convolutional Neural Networks (RCNNs) were introduced and showed a highly successful approach. However computationally they are expensive.

## Transfer Learning

Transfer Learning, like other machine learning techniques, is based on the human observation. Humans can learn in inherent fashion and apply knowledge from previously studied tasks to new ones. Based on this concept, there are several approaches to mimic this phenomenon in computer science. Specifically, with object recognition, there are millions of parameters processed and classified by a CPU/GPU, that requires a significant amount of computing power to train a model from scratch. Transfer Learning is the technique in which pre-trained models are used as starting points and by leveraging the knowledge of the source, a new model is created. The 2018 Stanford Course, Convolutional Neural Networks for Visual Recognition, documents that early layers in CNNs, record more generic data, and only close to the exit stage are becoming more specific to the input data (Karpathy, 2018). Therefore, Transfer Learning is a commonly used approach to image recognition tasks and natural language processing problems. Obviously, this technique is far less time consuming, but there are other benefits as well, especially in these two categories. Higher start means that the initial skill on the source code is higher. (This is before any finetuning.) A higher gradient shows a better rate of improvements and the asymptote illustrates the converged skill that is better than it would otherwise be (Lisa Torrey, 2009). (see Figure 3.4)

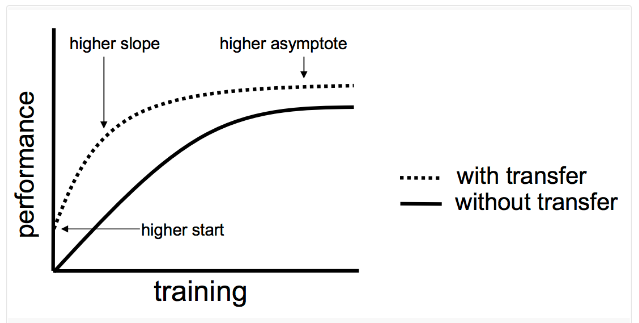


Figure 3.4 – Comparison of Classical approaches and Transfer learning (Lisa Torrey, 2009)

The nature of Transfer Learning is its ability to map tasks automatically when recognising correspondences. Both the source and target data must be in the same format for the transmission to operate correctly. When the two parties do not speak the same language, the additional mapping is required.

# Technologies

## Machine Learning Libraries

### Theano

Before Tensorflow appeared on the market Theano was considered the most favoured library. It is mainly used to compute mathematical expressions, especially with multidimensional arrays. Many features are shared with Tensorflow including great visualisation tools and speed performance. However, its developing community drastically dropped when Tensorflow arrived and gained popularity. The core language is Python. (KD, 2016)

### Scikit-learn

This is built on top of long existing Python packages. It is the most high-level library in the list discussed here. There are no real mathematical computations required by the developer, however, it does not allow for a deeper insight into any stage of the Deep Learning process. Tensorflow also outperforms Scikit-learn by having better GPU optimisation.

### Tensorflow

Tensorflow is open-sourced by Google and was developed for numerical computation. It provides a platform for expressing ML algorithms and the application where these algorithms get executed. Its unique structure uses data flow graphs instead of regular multidimensional arrays. The movements between layers are called flows. It supports Python as its core programming language, but significant implementations are done for C++ as well.

### Tensorflow Lite

“TensorFlow Lite provides an interface to leverage hardware acceleration, if available on the device. It does so via the Android Neural Networks library, released as part of Android O-MR1.” (Tensorflow, 2017)

## Evaluation of Machine Learning Libraries

There is a considerable amount of technologies that are available to achieve the given task. In 2016, a comprehensive investigation was carried out by the Research and Technology Center, Robert Bosch LLC. The study was performed on several ML frameworks including Theano, Tensorflow, Torch and others, based on the following criteria: Extensibility, hardware utilisation and speed. At the time of writing the research, Tensorflow was only recently released but showed great potential employing homogenious devices for its graph. The best performance, on both CPU and GPU trainings was achieved by Theano and Torch. By 2018 however, Tensorflow’s popularity rose so significantly that it left Theano and others behind.

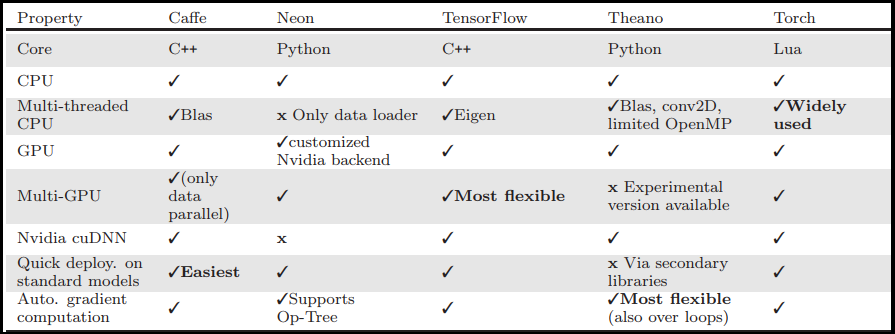


Figure 4.1 – Properties as of 02/08/2016 (Soheil Bahrampour, 2016)

* Machine Learning Libraries:
  + TensorFlow – Uses data flow graphs (chosen)
  + Scikit-learn – Considerable due to its large developer community
  + Theano – Primarily No.1 (2016) now second most popular ML library (Endsley, 2017)
  + PyTorch – Based on scripting language Lua
    - Keras – Neural Network library capable of running on top of Theano or Tensorflow

## Tensorflow Object Detection API

In June 2017 Google announced the release of the Object Detection API. It is a machine learning open-source framework built on top of Tensorflow, designed for image processing tasks. The research of computer vision requires massive amounts of data and computing power. The API was trained on the COCO dataset, which consists of 300 thousand images and 90 of the most common objects. The purpose of the release was to open up to broader research communities. Commercial use of the system is not possible. The framework includes a few trainable models including Single Shot Multibox Detector (SSD), SSD with Inception V2, Region-Based Fully Convolutional Network (R-FCN) with Resnet, Faster RCNN and Faster RCNN with Inception Resnet V2. Differences between models provide a tradeoff between speed and accuracy.

## Programming languages

* Python - General-purpose programming language (chosen)
* R - Statistical programming language

## Datasets

Numerous reliable resources are available for brand logos.

* Flickr27 – this dataset is used in this project
* Flickr32 – (available on request only)
* Flickr47 – made available in 2017

These datasets are publicly available and consist images from Flickr. In the 27 version, every brand has 30 variants. The package includes a relatively small number of 801 training images and 278 test images.

The 32 version differs in having one additional class (33rd class) that consist of images with no logos. This allows for better training of the network and higher accuracy of predictions. Also, an individual brand has no more than 18 images associated.

# Methodology

## Research undertaken

The research has indicated that creating an artificial neural network is a highly complicated process. The mathematical evaluations applied to ANNs are the results of decades of research and experimentation in the field of neuroscience and computer science respectively. This overview presented the reader with the fundamentals of artificial intelligence and the various models that may be considered for implementation. It investigated some of the most popular machine learning libraries and their usefulness, elaborating the reason why these are the preferred technologies in the professional environment.

## Research Question

An evaluation of how brand logo detection and recognition can be achieved with the implementation of an Artificial Neural Network. An investigation into computer vision and image processing.

## Proposed Project Implementation

The aim of this project is to build and train a network with the assistance of machine learning libraries. The proposed project is an application that aims to detect and recognise brand logos by using the artificial neural network. The initial version of the application is a desktop app that is using the device’s webcam to capture video frames. Once the images are recorded the system will make use of the trained (possibly pre-trained) network that learned from the injected dataset. Once the logos are recognised, the accuracy will be visualised to understand and inspect the learning phase. An advanced version of the application should be implemented on Android mobile platform to carry out the above-mentioned functionalities by taking advantage of the phone’s camera. In this scenario, further investigation is necessary into the variations of the chosen machine learning library such as TensorFlow For Mobile or the recently released TensorFlow Lite.

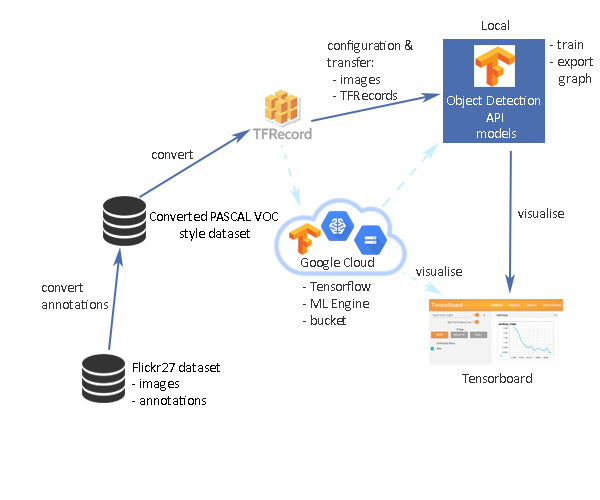
### Functional specifications

Functional specifications are described by prioritising deliverables using the MosCoW method.

|  |
| --- |
| MUST Have |
| * Apply a model provided by Tensorflow * Incorporate training data into the Tensorflow Object Detection API |
| SHOULD Have |
| * Be trained on the on the local machine * Export trained model back to local machine/mobile device |
| COULD Have |
| * Be trained on the Cloud * Make calls to the Cloud from a mobile device to apply real-time detection from a native Android application |
| WON’T Have |
| * Experiment with unsupervised learning to achieve dynamic learning approach |

## System design

### Architectural Design



## Prototype

|  |  |  |
| --- | --- | --- |
| Prototype | Start Date | Finish Date |
| 1 | 15/11/2017 | 22/11/2017 |

|  |  |  |
| --- | --- | --- |
| Task Number | Details | Status |
| 1 | Setup Tensorflow CPU version | Complete |
| 2 | Tutorials on Tensorflow Object Detection API  Tutorial available in Appendix A | Complete  More required as project progresses |
| 3 | Import necessary libraries for object detection   * Matplotlib * Jupyper * Pillow * Lxml * Cv2 (for the video streaming at a later stage) | Complete |
| 4 | Protobuf compilation | Complete |
| 5 | Import models directory from Tensorflow <optional> | Complete |
| 6 | Setup Tensorflow Object Detection API in jupyter. <optional> | Complete |
| 7 | Continue with loading in webcam images using tutorial.  Display webcam output with detected objects. | Complete |

# Implementation

## Sprints

The implementation chapter is built on the basic development of the model discussed in the prototype chapter. The initial plan involves six key phases in the development process. Every state includes further divisions (evaluated in the corresponding sprint sections). These phases are as follows:

1. Pre-processing - Prepare labelled images. Split dataset into training and test sample files.
2. Generate TFRecord formatted data from training and test sets.
3. Setup configuration file
4. Train the model (graph)
5. Export trained graph (freeze it)
6. Apply the “frozen” graph to classify logos.

### Sprint 1 – Data preparation

|  |  |  |  |
| --- | --- | --- | --- |
| Sprint Number | Sprint Name | Start Date | Finish Date |
| 1 | Input data preparation | 20/01/2018 | 15/02/2018 |

|  |  |  |
| --- | --- | --- |
| Task Number | Details | Status |
| 1 | Examine the dataset – Flickr27 -used for the application. Differentiate between the provided data and Tensorflow’s feature requirements. | Complete |
| 2 | Convert both test and training files to csv format. | Complete |
| 3 | Add image sizes to existing data. | Complete |
| 4 | Remove unnecessary column from initial data file. | Complete |
| 5 | Create newly structured csv files for conversion. | Complete |

Based on the detailed research undertaken the decision was made to use Tensorflow Object Detection API. This approach is called Transfer Learning. This technique allows skipping working with the millions of parameters that could take weeks to train. Instead, use a model that already has its convolutional neural network weights learned on a pre-defined object recognition tasks. (Jeff Donahue, 2013)

**Task 1**

The official documentation recommends that all data fed into this model must be in TFRecord format. TFRecord is a record-oriented binary format that most Tensorflow applications use. It processed by Tensorflow in a sequential order step by step. Reading data into a Tensorflow program may be achieved with typically three different approaches.

* Feeding – Feeding Python code directly into the Tensor which means directly into the graph
* Preloaded data – Small datasets can be fed as a constant or variable directly in the graph
* Reading from files – An input pipeline reads data at the beginning of the graph. This approach is being demonstrated in the project.

The Flickr dataset’s structure composed of the following elements:

Flickr\_logos\_27\_dataset\_images.tar 🡪 all images in .jpg format, not separated in training and test folders

Flickr\_logos\_27\_dataset\_query\_set\_annotation.txt 🡪 test directory containing: filename and corresponding class name

Flickr\_logos\_27\_dataset\_training\_set\_annotation.txt 🡪 training directory containing: filename, class name, a subclass of the class, xmin, ymin, xmax, ymax (location coordinates)

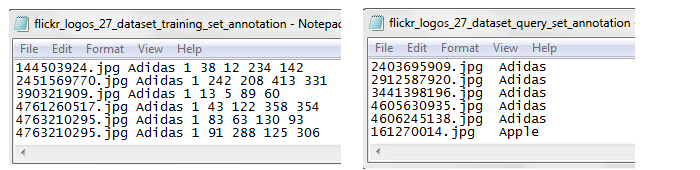


Figure 6.1 – Annotations of the initial dataset

Machine learning tasks such as image processing may be solved either supervised or unsupervised. This project applies the supervised learning approach (see explanation under section 3.4). The technique requires a set of data that is ideally split up into training and testing sets. Flickr’s logo dataset is coming with these pre-organised sections.

**Task 2.**

A simple conversion from txt to csv file formats was required. Reason: this is Tensorflow’s recommended approach. Regardless the level of difficulty some issues already arose at early stages. Discussed further in the Problems section.

**Task 3-5.**

Figure 6.2 (below) illustrates, the process of adding the necessary sizes of the images to the training and testing data. The initial training set had to be truncated first, then the new measurements were added. Also had to make sure, that the file did not contain any replications as it was in the original version. These tasks were achieved by writing several helper csv files until the data reached its final desired format; im\_filename, width, height, classname, xmin, ymin, xmax, ymax

After the merge - between the full set of measurements and the test/training sets - were complete, the missing coordinates in the test file were added manually. Alternatively, third-party software can be used to crop the images, which allows for automated data generation, but given the small size of the testing set, this approach was not necessary.

**Problems**

1. One of the issues encountered was when converting txt files to csv., even though initially the two text files (see Figure 6.1) were to be converted into csv with the same code base, their outcome was slightly different. The test file’s values were not comma separated. This problem was not apparent until later stages when TFRecord was generated and the .record file turned out to be empty. Naturally, this file halted progression with the training stage as well.

The solution to this issue was to use a different approach with the test file. While the training is following a more traditional method, the test text was converted by adding the additional commas to the features individually, closing rows with the last character removal.

1. Another issue was the confusing documentation on TFRecord. This, however, was not an apparent problem until later stages when training was again held up. Conversion into this protobuf was achieved several times yet was forced to return to this stage on multiple occasion as the training was not able to further progress. More on this issue will be covered in Sprint 2.

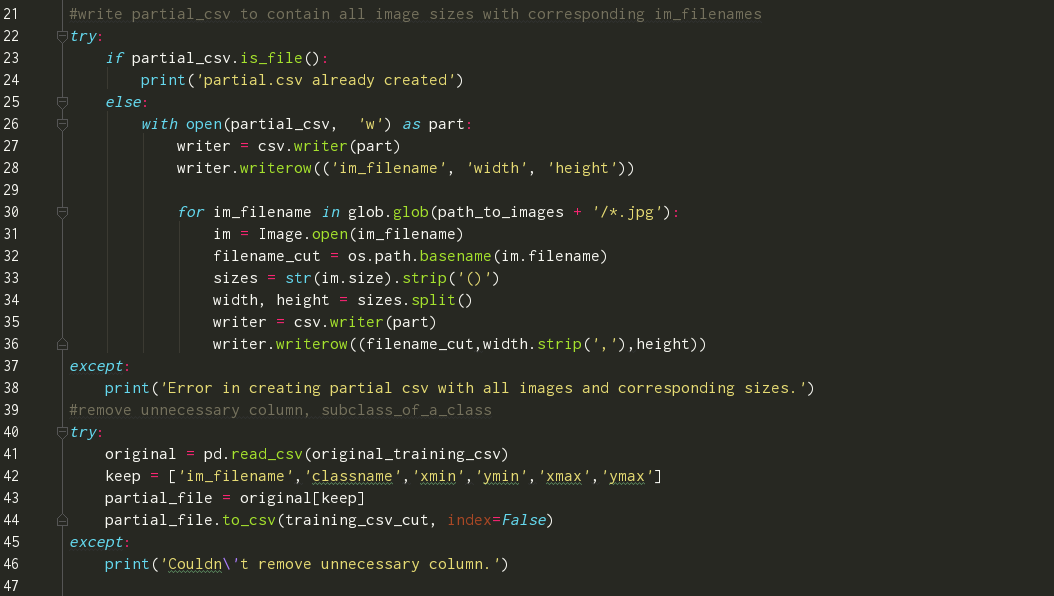


Figure 6.2 – The full process of creating a csv input data file.

### Sprint 2 – Converting data to TFRecord

|  |  |  |  |
| --- | --- | --- | --- |
| Sprint Number | Sprint Name | Start Date | Finish Date |
| 2 | Converting data into TFRecord | 15/02/2018 | 22/03/2018 |

|  |  |  |
| --- | --- | --- |
| Task Number | Details | Status |
| 1 | Tensorflow provides templates for the most common type of dataset conversions.  Analyse the templates. | Complete |
| 2 | Choose a template and tailor it to the flickr dataset. | Complete |
| 3 | Create label map. | Complete |
| 4 | Convert testing and training csv sets into TFRecord. | Complete |

**Task 1**

As indicated in the previous section, Tensorflow documentations in some areas may be a little deceptive. Tensorflow states that a data file, that is to be converted, must be in the following format; *feature 1, feature 2, …, feature n*

This approach is correct of course but written in general terms applying to all(!) the training models that may be created in Tensorflow. This means that the initial data (training set for instance contains: filename, class name, subclass of the class, xmin, ymin, xmax, ymax) should satisfy the requirements. The models from the Object Detection API however, require a more stringent architecture. The first attempts to convert the csv files were based on the original data structure, providing that the test set included no bounding box coordinates, and the training set included an extra, unnecessary column (as indicated earlier). To some extent, the conversion was successful and RECORD file could be generated. The issue again only became apparent at later stages when the API components were unable to work together with the provided files. The training set was perfectly converted given that Tensorflow simply ignored the extra column. The test RECORD, however, was damaged due to the missing elements. All of this has led to aligning the dataset exactly to the expected input requirements.

**Task 2-3**

Once the correct content was set, the code to generate the TFRecord was needed. There are several methods to create these files. The model provides two pre-made scripts for the most typical data structures, a PASCAL VOC data type conversion script and another for the Oxford Pet type of datasets. If none of these is the right fit, a custom code must be written. What is common in all three options are the following;

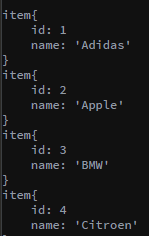
* Input Images must be in either JPEG or PNG format

Figure 2 – Snippet from the label map created

* Input data must include the bounding box coordinates for the location of the desired objects within the image
* Name of the class in the bounding box (encoded utf8)
* A label map must be associated with the dataset. It is a simple JSON-like list, providing a class and an ID for the class. This file must be saved in a protobuf text format (pbtxt)

For this project, a cloned script was used where only a few modifications were needed. It is largely based on a helper code provided by Tensorflow, where the required input data is more generic and better suited for this project than the other two options.

**Task 4**

After running this code (generate\_tfrecord.py) several issues were encountered. This was mainly due to the issues explained in section Sprint 2.1, where the input data was falsely defined. When these problems were eventually corrected the development could proceed to the next stage.

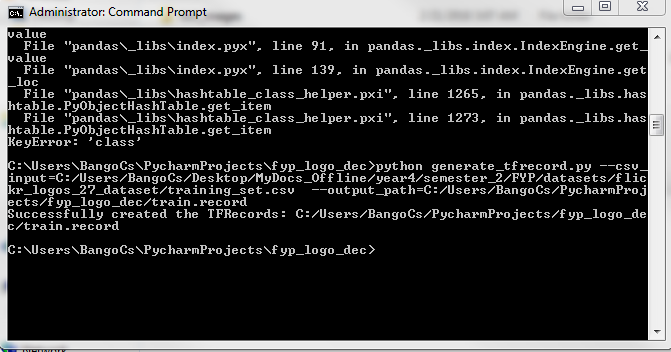


Figure 3 - TFRecord file is generated the first time.

### Sprint 3 – Setup Configuration file

|  |  |  |  |
| --- | --- | --- | --- |
| Sprint Number | Sprint Name | Start Date | Finish Date |
| 3 | Setup configuration file | 22/02/2018 | 01/03/2018 |

|  |  |  |
| --- | --- | --- |
| Task Number | Details | Status |
| 1 | Examine pre-trained models provided by the API | Complete |
| 2 | Define inputs | Complete |
| 3 | Configure the trainer with the right inputs. | Complete |
| 4 | Navigate all files in to the API’s object\_detection folder. | Complete |

**Task 1**

Tensorflow provides several models pre-trained on large datasets. Since the aim of the project is to detect logos on real-time camera footage, a fast model had to be selected. After evaluating the collection of models, the decision was made to use Google’s SSD Mobilenet V1. This model is built on the Single Shot MultiBox Detector (SSD) method. Mobilenet is a feature extension head, that is a series of convolutional blocks from other models such as VGG, Inception etc. It is a neural network, that is specifically designed to run on mobile devices. Its main characteristics are as follows:

* Works efficiently with PASCAL formatted datasets
* Smaller than models such as CNN or the faster RCNN
* Fast – speed: 30 ms

**Task 2-3**

These configuration files support the Transfer Learning objectives. The inputs to be defined are: fine tune checkpoint 🡪 must point to the default model checkpoint, label map path 🡪 points to the label map defined in Sprint 2.3, number of classes 🡪 number corresponding to the logo dataset, input path for both TFRecord files.

**Task 4**

The final step before the training may begin, is to move all the files created into the API’s cloned directory. The training of Tensorflow is initialised by establishing the configurations in the transferred files and the connection between the frozen model and the new input data.

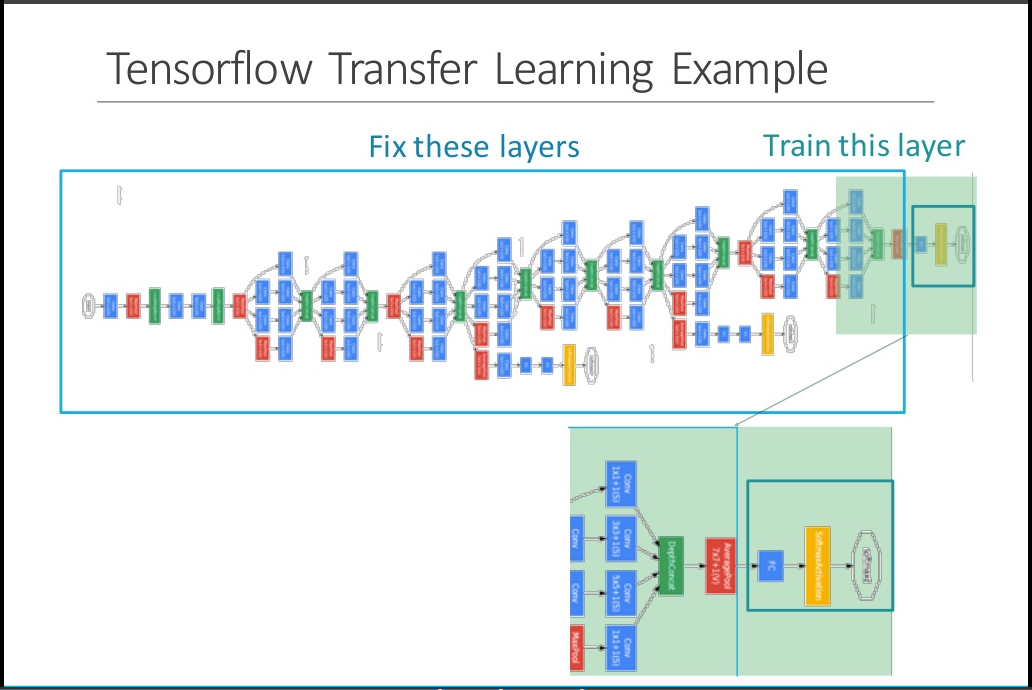


Figure 6.1 – An illustration of Transfer Learning with Tensorflow (Chang, 2016). It is also an accurate visualisation on what was achieved with the configuration file that allowed the training to begin.

### Sprint 4 – Train the model (graph)

|  |  |  |  |
| --- | --- | --- | --- |
| Sprint Number | Sprint Name | Start Date | Finish Date |
| 4 | Training the model (locally) | 01/03/2018 | 15/03/2018 |

|  |  |  |
| --- | --- | --- |
| Task Number | Details | Status |
| 1 | Once the model is chosen, all the configuration jobs are done and the data is in the right format, the training may begin. Usually, with CPU-only Tensorflow setup, this training takes hours.  When Tensorflow starts its training, the following tasks are executed:   * Read in provided files * Create new empty config files where the process gets saved step by step * Iterate through the data until the model is not overfitted. * Produce visualisation graphs on TensorBoard | Complete |

**Task 1**

By summarising the previous steps, the following prerequisites must be fulfilled:

* Training and testing data must be in the right format and are converted into record files
* The configuration file is set
* Label map is created
* All the above along with a folder of images are injected in the API

To train the model the API’s train.py script must be called initialising the training directory and the pipeline config path. The training directory contains the config file and the labelmap. Once the training has begun, all ongoing information (checkpoints, Tensorboard data etc.) are directed here. The pipeline config file points to the config file that has been modified in Sprint 3.

**Problems**:

There are a number of issues that may prevent the training process from being executed. The Object Detection API uses Protobufs to configure the training parameters. „Protocol buffers are Google's language-neutral, platform-neutral, extensible mechanism for serializing structured data” (Google Developers Official, 2018). A very typical issue (with Windows OS) occurs at the compilation of these proto files. According to Tensorflow’s documentation, the following command should execute this process;

protoc object\_detection/protos/\*.proto --python\_out=.

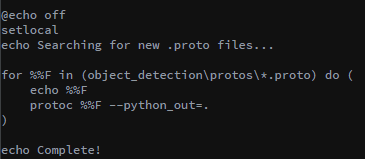
However, Windows has a problem passing in multiple files into proto execution. Solve this problem by adding an a .bat script (Figure 7.1.) (Anon., 2017). Once this file is added the slim directories can be appended to PYTHONPATH (See Figure 7.2).

Figure 7.1

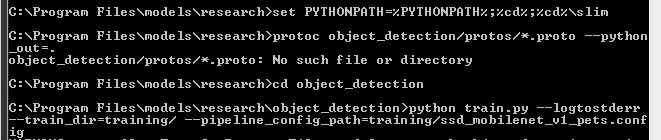
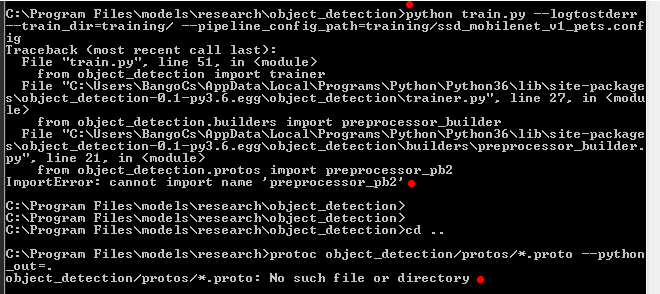


Figure 7.2

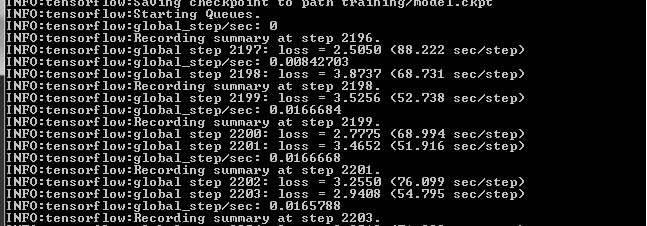


Figure 7.3

### Sprint 5 – Export trained model

|  |  |  |  |
| --- | --- | --- | --- |
| Sprint Number | Sprint Name | Start Date | Finish Date |
| 5 | Export inference graph | 22/03/2018 | 15/03/2018 |

|  |  |  |
| --- | --- | --- |
| Task Number | Details | Status |
| 1 | The model is trained and it’s training steps are saved along with it. However, this model is not reusable just yet. ‘Freezing’ the model is essential for the next phases. | Complete |

Figure 7.3 illustrates the process of the training. All these steps along with the periodically saved checkpoints are saved to a pre-defined path. As for many general tasks, Tensorflow provides a general script as support. export\_inference\_graph.py can be considered as another configuration file that must be tailored to the specific task. Input type, config file path, the last trained checkpoint and a name for the output directory must be specified as arguments. The aim is to ‘freeze’ the inference graph.

The necessary information can be provided when calling the execution;

python export\_inference\_graph.py

--input\_type image\_tensor

--pipeline\_config\_path training/ssd\_mobilenet\_v1\_pets.config

--trained\_checkpoint\_prefix training/model.ckpt-2615

--output\_directory logo2615\_graph

The first input to modify is the pipeline\_config\_file. This is the config file that was initialised and described in Sprint 3. When specifying the trained\_checkpoint\_prefix, the following requirements must be fulfilled: The last recorded step must include a .meta, a .index, and a .data-00000-of-00001 file. (The system saves the checkpoints every fifth step. If the training is stopped in between, often the checkpoints would not include all the necessary files. In this scenario, the developer must specify an earlier stage.) Above, the last saved checkpoint was at 2615 steps. The output\_directory is the name of the frozen graph that is to be exported. In this folder, along with the checkpoints, a new .pb file gets generated called the frozen\_inference\_graph.pb. Once the folder with the necessary files is in place, the model can be used in production. This also means that the exported data becomes considerably lighter, as the additional metadata is no longer stored.

### Sprint 6 – Apply the model to classify logos

|  |  |  |  |
| --- | --- | --- | --- |
| Sprint Number | Sprint Name | Start Date | Finish Date |
| 6 | Apply model to classify logos | 20/03/2018 | 05/04/2018 |

|  |  |  |
| --- | --- | --- |
| Task Number | Details | Status |
| 1 | Test the frozen inference graph on the default Tensorflow object detection tutorial. | Complete |
| 2 | Real-time video frames are read and fed into the model as input. | Complete |

There are multiple options to test whether the training was successful. Tensorflow offers a platform -Tensorboard- to monitor the training progress as well as to help understand, debug and/or optimise the learning. It is often monitored during runtime but is also available for evaluating how well the training performed after the process is finished. There are a number of different graphs set to audit a variety of aspects such as learning rate, loss, batch, global steps, queue etc. Based on these the developer can be assured that accurate results will be achieved at the end of the training. Yet, to test the model in action, the frozen inference graph must be integrated into the an application. For this task, several solutions can be found in the API. Object\_detection\_tutorial.py was specifically designed for this purpose and served the demonstration goals perfectly. Similarly to the previous steps, certain input points had to be injected. These include the path to the checkpoints (these are automatically carried over to the folder of the graph what was specified when exported the graph (Sprint 5)), to the inference graph .pb file, the labelmap (maps indeces to the specific logo classes), defining number of classes and the path to all the images. By default, this script reads in a number of test images (must also be specified) that is analysed using the model. Initially, the plan was to detect logos on real-time video footage therefore the static image loop had to be removed. Reading webcam footage was achiveved with the usage of cv2 and some numpy functions. However, at later stages, the decision was made to re-introduce the original design because it produced better predictions. (To be evaluated in Chapter 6.) Nevertheless, both implementations are able to produce predictions and were made accessable in the project folder.

### Sprint 7 – Train the model on the Cloud

|  |  |  |  |
| --- | --- | --- | --- |
| Sprint Number | Sprint Name | Start Date | Finish Date |
| 7 | Train the model on the Cloud | 05/04/2018 | 19/04/2018 |

|  |  |  |
| --- | --- | --- |
| Task Number | Details | Status |
| 1 | Setup Google Cloud | Complete |
| 2 | Setup ML Engine | Complete |
| 3 | Create a bucket, set up the data structure | Complete |
| 4 | Start a job | Started, Incomplete |

As predicted, running a training job locally on a CPU requires much more time than what a machine with GPU does. The tested processor is an Intel(R) Core(TM) i5-5200U CPU @ 2.20Ghz with installed memory: 8.00GB. To reach ~2100 steps took four days running constantly on 98-100% memory usage. Training the model on the Cloud would expectedly shorten the amount of time to just a few hours. For this project the obvious choice cloud platform was Google Cloud. Since Tensorflow was developed by Google, the platform provides numerous tools to assist machine learning tasks such as distributed training on the ML Engine.

**Task 1-3**

The prerequisites of the training are the follows:

* The input data must be in the correct format
* A valid Object Detection pipeline is configured
* The data must be placed in a ’bucket’ on the Google Cloud Storage
* Tensorflow and the API is installed locally

The model and the structure was validated by successfully running the training on the local machine. This is recommended as cloud services can be costly. The necessary tools for this project are Machine Learning Engine and Cloud Storage. Having the data stored in the ’bucket’ required minimal configuration. An important step was to modify the configuration file restructuring the corresponding paths to the Storage bucket. Since the local training had over 2000 steps in its checkpoint, the initial goal was to introduce this file to the cloud-based training.

**Task 4**

Once a newly cloned models directory is in place, and the data is configured on the cloud the learning may begin. Regardless of the tremendous effort to debug the various errors, this part of the project remained incomplete.

# Findings & Conclusions

Tensorflow Object Detection API has multiple benefits and is therefore an exceptional tool to introduce developers to Machine Learning. The framework provides an abundance of documentations and vast amounts of helpful tools and scripts. This chapter discusses how a successful implementation of the application was achieved with these tools.

## Computational Performance

Figure 8.2 (below) illustrates the total loss during training after running the job for ~2600 steps with the batch size of 24. To achieve this height the tested CPU was running for several days. Naturally, the time spent on learning also depends on the amount of classes. A model with a single class, using an average powered machine, can be trained up to the same level in just a few hours. It is important to note that highly accurate models are trained with steps between 10-25 thousand.

Tensorflow is a framework that performs very efficient computations in the field of Machine Learning, and it does even better when interacting with the Graphics Processor Unit (GPU). Unlike any other framework, Tensorflow is built on a concept of graphs and tensors. Their execution is unique in a sense that computations must be specified by creating these graphs which take multidimensional matrices (tensors) and do the work on them. Therefore, a CPU will perform much slower than if the model was trained on GPU or a serious of GPUs (Cloud). Fig. 8.1 illustrates how a CPU struggled to process the same execution.

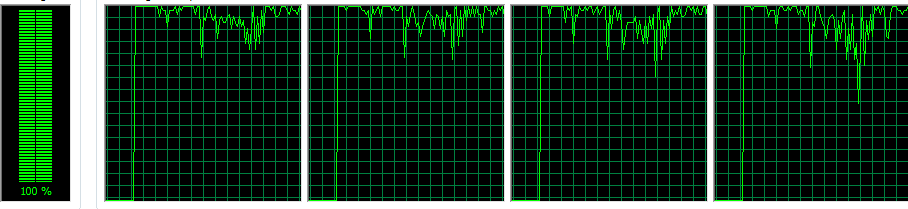


Figure 8.1

## Training Observations

Every time the training stopped, the graph of the checkpoints was exported and tested in an application. With the support of two applications, accuracy was tested with real-time video footage and individually fed in pictures. With the first method, images of logos displayed on a mobile phone were presented to the web camera. While the app could produce some recognition, the accuracy of the system remained low. Once the camera was shown real objects, the system performed better detection. This allows for the first observation, that high definition images have key importance while producing accurate predictions. The same tendency could be seen with the other method as well.

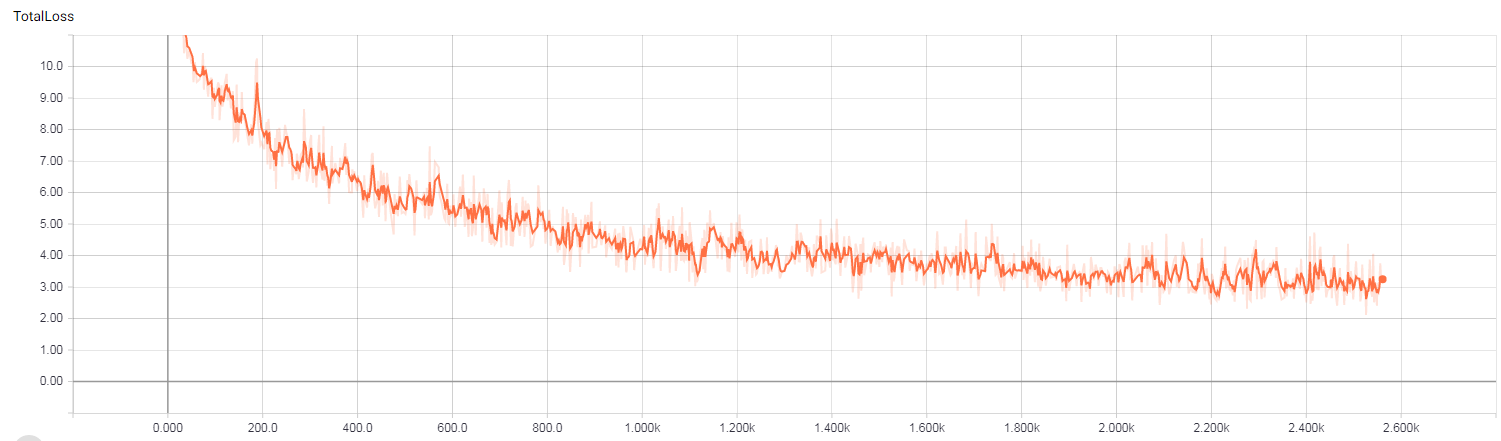


Figure 8.2 – The curve of Total Loss calculations

As mentioned earlier, the learning was accomplished with an SSD model, which utilises the Multibox method. Multibox’s loss function consists of two components, confidence loss and location loss. In the simplest form, it can be calculated with the following equation;

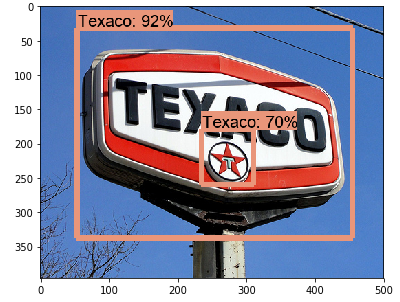
The helps balancing the contribution of the location loss. Calculating the total loss indicates how badly the training performed (Forson, 2017). In other words, the expression for the loss is a measure of error. Therefore, the lower the loss, the more accurate the model.

As expected, the value of loss at the start of the training is very high, but it is rapidly decreasing in the first few iterations. To support this statement, Table 1 presents the details.

Table 1- The improvement of loss value during training

|  |  |
| --- | --- |
| Steps | Total Loss in average |
| 45 | 10.5 |
| 250 | 8.2 |
| 1000 | 4.1 |
| 2000 | 3.5 |
| 3000 | 2.7 |

Ideally, training is considered complete, if this value is kept closest to 1.

The **Mean Average Precision** (illustrated on Figure 8.4) or mAP score is the accepted way to evaluate object detection tasks. This value is caused by the object needing to detect and measure multiple points, such as the accuracy of logo identification on an image, whether to determine the location of this object, and to determine the number of misclassified targets. “Precision measures the “false positive rate” or the ratio of true object detections to the total number of objects that the classifier predicted.” (Arlen, 2017) The mAP score is ideal when it is close to 1.0. In the light of these facts it can be deducted, that after ~3400 steps, Sprite had the highest average precision at 0.781 and Fedex had the lowest at 0.027. Therefore, the overall average is 0.3516. It is important to note, that these results were produced after a small number of training steps, thus they may be misleading. A simple example can prove this concept. During the testing phase of the model logos were evaluated differently at different stages. One of the best performing recognition was with Texaco, yet its mAP score after 3400 steps was as low as 0.158. Although the reason for performing so well, was because it was studied well at an early stage reaching a 0.5 mAP value after just 2600 steps. The fluctuation is constant. Correct precision rate can only be deducted after the normalisation of the graph is complete (consisting of tens of thousands of steps).

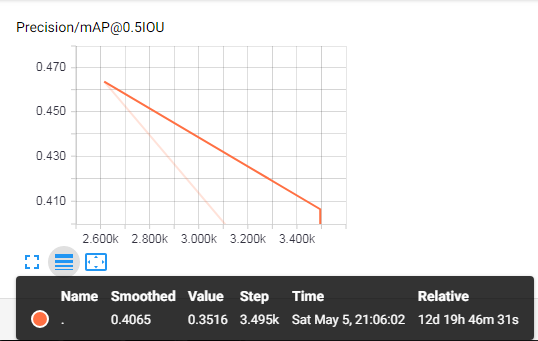


Figure 8.4

Figure 8.3

The performance by category scalars indicate the tendency of these changes. Figure 8.5 (below) belongs to Fedex, the least efficient class to date. At 2600 steps, its precision was as low as 9.6. By the time it reached 0.02 (still very low), the propagations had to run over 3400 turns. This diagram shows that although the rates are improving, it does so very slowly. Figure 8.6 indicates the performance of the Ferrari logo. It is a relatively steady observation ratio. With the same steps, the values between 0.375 and 0.5 show a decreasing tendency.

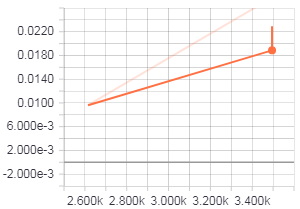
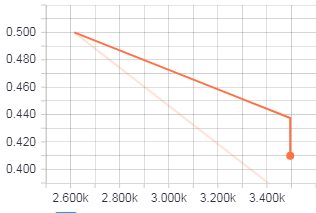


Figure 8.6

Figure 8.5

There are several reasons why the learning has led to limited success. Although the number of epochs is very low, other aspects have had impact on the performance. The dataset used was Flickr27 that contains 27 different logos, each with 30 images per class. The size of this dataset is considered very small, since an acceptable performance is usually achieved with around 5000 labelled examples per category (Ian Goodfellow, 2016). However, this problem has been solved by the Transfer Learning method chosen. On the other hand, the sizes of the images are extremely imbalanced having 313 different sizes, ranging between 22x90 and 500x500. It can be stated that although the implemented SSD Mobilenet does a fair amount of image augmentation, it does not prove to be overly satisfactory.

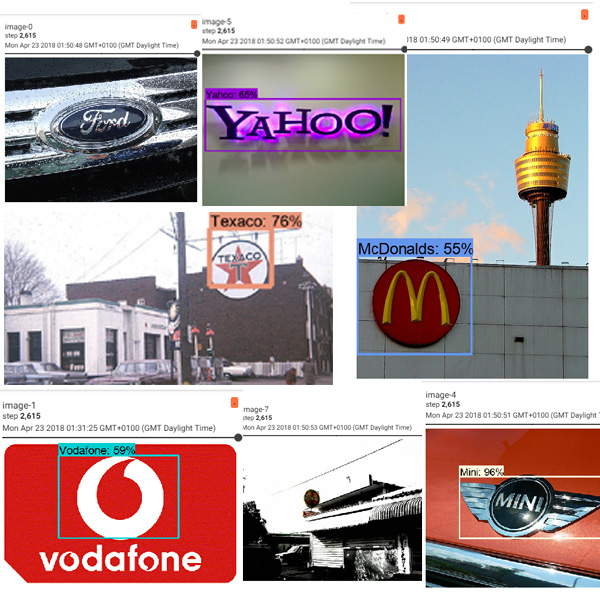


Figure 8.7

Overall, it can be stated, that working with Tensorflow despite all the documentations and tutorials, the framework is far from being a tool that is easy to adopt. It requires a lot of problem solving. The API is open-source. It is frequently updated, often causing breaking changes.

## Research Question Answered

The Tensorflow Object Detection API was successfully implemented for logo detection and recognition purposes. The training was carried out with a CPU, and therefore, the overall accuracy can be considered a limited success. However, on the basis of the progress observed, it can be concluded that achieving the right level of efficiency is much a question of how strong the machine is that the system is running on. Additional improvements could also be made by further image augmentation techniques, and / or larger, more sophisticated datasets. Finally, for experimental purposes, other, more accurate but high overhead models could as well result in more subtle outcome.

# References

Alex Krizhevsky, I. S. G. E. H., 2012. ImageNet Classification with Deep Convolutional Neural Networks. *NIPS'12 Proceedings of the 25th International Conference on Neural Information Processing Systems,* Volume 1, pp. 1097-1105.

Amaratunga, T., 2017. *Build Deeper.* First Edition ed. s.l.:s.n.

Anon., 2017. *Github.* [Online]   
Available at: https://github.com/tensorflow/models/issues/1934  
[Accessed 22 April 2018].

AppliedAICourse, 2017. *Youtube.com.* [Online]   
Available at: https://www.youtube.com/watch?v=PYKfXkd3t7c  
[Accessed 16 November 2017].

Arlen, T. C., 2017. *Medium.* [Online]   
Available at: https://medium.com/@timothycarlen/understanding-the-map-evaluation-metric-for-object-detection-a07fe6962cf3  
[Accessed 4 May 2018].

Asa B. Simmons, S. G. C., 1988. Artificial Intelligence-Definition and Practice. *IEEE journal of Oceanic Engineering,* 13(IEEE), p. 29.

Brownlee, J., 2016. *machine learning mastery.* [Online]   
Available at: https://machinelearningmastery.com/supervised-and-unsupervised-machine-learning-algorithms/  
[Accessed 12 December 2017].

Castle, N., 2017. *Supervised vs. Unsupervised Machine Learning.* [Online]   
Available at: https://www.datascience.com/blog/supervised-and-unsupervised-machine-learning-algorithms

Chang, M., 2016. *Applied Deep Learning 11/03 Convolutional Neural Networks,* s.l.: National Taiwan University.

Chung, B., 2013. *Familug.org.* [Online]   
Available at: http://www.familug.org/2013/09/xu-ly-anh-va-nhung-cau-chuyen-xung-quanh.html  
[Accessed 9 April 2018].

Dominiek Sandra, J.-O. Ö. a. J. V., 2009. *Cognition and Pragmatics.* s.l.:John Benjamins Publishing Company.

Endsley, R., 2017. *Opensource.com.* [Online]   
Available at: https://opensource.com/article/17/2/3-top-machine-learning-libraries-python  
[Accessed 23 November 2017].

Fatih Ertam, G. A., 2017. *Data classification with deep learning using Tensorflow.* Antalya, Turkey, IEEE.

Fisher, A. F. T. a. A. C., 2007. *Outcome Prediction in Cancer.* s.l.:Elsevier B.V.

Forrest N. Iandola, A. S. P. G. K. K., 2015. *DeepLogo: Hitting logo recognition with the deep neural network hammer.* s.l., arXiv.

Forson, E., 2017. *Towards Data Science.* [Online]   
Available at: https://towardsdatascience.com/understanding-ssd-multibox-real-time-object-detection-in-deep-learning-495ef744fab  
[Accessed 4 May 2018].

Gangming Zhao, Z. Z. J. W. H. G., 2017. Training Better CNNs Requires to Rethink ReLU. *CoRR,* Volume abs/1709.06247.

Google Developers Official, 2018. *Developers.Google.* [Online]   
Available at: https://developers.google.com/protocol-buffers/  
[Accessed 22 April 2018].

Graupe, D., 2014. *Principles of Artificial Neural Networks.* 3 ed. s.l.:World Scientific Publishing Co Pte Ltd.

H.A. Rowley, S. B. T. K., 1996. *Neural Network-Based Face Detection.* San Francisco, CA, USA, IEEE.

Hecht-Nielsen, 1989. *Theory of the backpropagation neural network.* Washington, DC, USA, IEEE.

Ian Goodfellow, Y. B. A. C., 2016. *Deep Learning.* s.l.:MIT Press.

Jeff Donahue, Y. J. O. V. J. H. N. Z. E. T. T. D., 2013. *DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition,* Berkeley, CA, USA: arXiv.

K. Susheel Kumar, S. P. P. K. S., 2010. *Multiple Cameras Using Real Time Object Tracking for Surveillance and Security System.* Goa, India, IEEE.

Kabalan Chaccour, G. B., 2016. *Computer vision guidance system for indoor navigation of visually impaired people.* Sofia, Bulgaria, IEEE.

Karpathy, A., 2018. *Github.* [Online]   
Available at: http://cs231n.github.io/transfer-learning/  
[Accessed 10 April 2018].

KD, 2016. *KD Nuggets.* [Online]   
Available at: https://www.kdnuggets.com/2016/04/top-15-frameworks-machine-learning-experts.html  
[Accessed 10 December 2017].

Leonardo Bombonato, G. C.-C. P. S., 2017. *Real-Time Single-Shot Brand Logo Recognition.* Niteroi, Brazil, IEEE.

Lisa Torrey, J. S., 2009. Transfer Learning. In: *Handbook Of Research On Machine Learning Applications and Trends: Algorithms, Methods and Techniques .* Madison WI, USA: IGI Global, p. 834.

Logograb, 2016. *Heineken Spectre case study,* Dublin: Logograb.

Lowe, D., 1999. *Object Recognition from Local Scale-Invariant Features.* Kerkyra, Greece, IEEE.

Lowe, D. G., 1985. *Perceptual Organization and Visual Recognition.* 1 ed. s.l.:Kluwe Academic Publishers Norwell.

M.A. Fischler, R. E., 1973. The Representation and Matching of Pictorial Structures. *IEEE Transactions on Computers ,* C-22(1), pp. 67-92.

Munish Puri, A. S. P. M. T. A. M. P., 2016. Chapter 1 Introduction to Artificial Neural Network (ANN) as a Predictive Tool for Drug Design, Discovery, Delivery, and Disposition Basic Concepts and Modeling. In: *Artificial Neural Network for Drug Design, Delivery and Disposition.* s.l.:Elsevier.

N. Dalal, B. T., 2005. *Histograms of oriented gradients for human detection.* San Diego, CA, USA, IEEE.

Paul Viola, M. J., 2001. Robust Real-time Object Detection. *International Journal of Computer Vision,* 57(2), pp. 137-154.

Pourghassem, H., 2012. *A Hierarchical Logo Detection and Recognition Algorithm Using Two-Stage Segmentation and Multiple Classifiers.* Mathura, India, IEEE.

Qiting Ye, Z. L. X. X. S. G., 2017. *GeLoGo: Detecting TV Logos from Web-Scale Videos.* Laguna Hills, CA, USA, IEEE.

Quoc-Bao Truong, B.-R. L., 2008. *New lane detection algorithm for autonomous vehicles using computer vision.* Seoul, South Korea, IEEE.

Raluca Boia, C. F. L. F., 2015. *Elliptical ASIFT Agglomeration in Class Prototype for Logo Detection.* s.l., BMVA Press.

Raluca Boia, C. F. L. F. R. D., 2016. . Logo localization and recognition in natural images using homographic class graphs. Machine Vision and Applications. *Machine Vision and Applications,* 27(2), pp. 287-301 .

Rosenblatt, F., 1958. The Perceptron: A probabilistic model for information storage and organisation in the brain. *Psychological Review,* pp. 386-408.

Savas Ozkan, E. E. G. B. A., 2014. *Performance analysis of local indexing methods for video copy detection.* Trabzon, Turkey, IEEE.

Shengmei Lin, C. Z. X. Q., 2016. *Comparative analysis of several feature extraction methods in vehicle brand recognition.* Nanjing, China, IEEE.

Soheil Bahrampour, N. R. L. S. M. S., 2016. *Comparative Study of Deep Learning Software Frameworks,* s.l.: ArXiv.

Stefan Romberg, L. G. P. R. L. R. v. Z., 2013. *Multimedia-computing.de.* [Online]   
Available at: http://www.multimedia-computing.de/flickrlogos/  
[Accessed 23 November 2017].

Steven C.H. Hoi, X. W. H. L. Y. W. H. W. H. X. Q. W., 2015. *LOGO-Net: Large-scale Deep Logo Detection and Brand Recognition with Deep Region-based Convolutional Networks,* s.l.: arXiv.

Tensorflow, 2017. *Tensorflow.* [Online]   
Available at: https://www.tensorflow.org/mobile/tflite/  
[Accessed 23 November 2017].

Wikipedia, 2012. *Wikipedia.* [Online]   
Available at: https://en.wikipedia.org/wiki/File:CVoverview2.svg  
[Accessed 25 October 2017].

Y. LeCun, B. B. J. S. D. D. H. R. E. H. W. H. L. D. J., 1989. Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation,* 1(4), pp. 541-551 .

Yue Gao, F. W. H. L. T.-S. C., 2014. *Brand Data Gathering From Live Social Media Streams.* Glasgow, United Kingdom, ACM.