# An AI-Driven, Self-Learning Image-Sharing Platform

by

Alex O Regan

This thesis has been submitted in partial fulfillment for the degree of Bachelor of Science in YOUR DEGREE

in the
Faculty of Engineering and Science
Department of Computer Science

May 2019

# Declaration of Authorship

I, Alex O Regan , declare that this thesis titled, 'An AI-Driven, Self-Learning Image-Sharing Platform' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for an undergraduate degree at Cork Institute of Technology.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at Cork Institute of Technology or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this project report is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: ______________________________

Date: ______________________________

CORK INSTITUTE OF TECHNOLOGY

## *Abstract*

Faculty of Engineering and Science
Department of Computer Science

Bachelor of Science

by Alex O Regan

In recent years, the modern consumer has seen exponential growth in available processing power, and reflecting this growth is an explosion of Artificial Intelligence-based technologies that are further-augmenting the abilities of our societies and citizens. One-such sub-domain of Artificial Intelligence is that of Image Recognition, which can be defined as the endeavour to give human, or sometimes super-human seeing capabilities to machines. Image Classification has a range of applications from self-driving automotive technology to disease-detection in CT scans. This paper aims to leverage contemporary techniques within the field of Image Recognition in order to construct an image-sharing platform that, given data uploaded by it's users, can continually improve it's own image recognition abilities. The paper will make use of modern frameworks such as Keras and Flask, and will also implement high-performance Artificial Intelligence models that take inspiration from the human brain.

# *Acknowledgements*

Dr. Donna O'Shea - My project supervisor, who continually provided me with the resources and books necessary to complete my work throughout the semester.
Dr. Ted Scully - A helping hand for the technical details regarding AI and it's modern areas of research.

# Contents

# List of Figures

# List of Tables

# Abbreviations

| | |
|---|---|
| **NASA** | **N**ational **A**eronautics and **S**pace **A**dministration |
| **AI** | **A**rtificial **I**ntelligence |
| **IBM** | **I**nternational **B**usiness **M**achines |
| **ML** | **M**achine **L**earning |
| **AWS** | **A**mazon **W**eb **S**ervices |
| **SEO** | **S**earch **E**ngine **O**ptimization |
| **SaaS** | **S**oftware **as** **a** **S**ervice |
| **ANN** | **A**rtificial **N**eural **N**etwork |
| **NN** | **N**erual **N**etwork |
| **CNN** | **C**onvolutional **N**eural **N**etwork |
| **BP** | **B**ack **P**ropagation |
| **API** | **A**pplication **P**rogramming **I**nterface |
| **CDN** | **C**ontent **D**elivery **N**etwork |
| **HTML** | **H**yper **T**ext **M**arkup **L**anguage |
| **DB** | **D**ata**B**ase |
| **SQL** | **S**tructured **Q**uery **L**anguage |
| **GPU** | **G**raphics **P**rocessing **U**nit |
| **XP** | E**x**treme **P**rogramming |
| **LTS** | **L**ong-**T**erm **S**table |
| **JSON** | **J**avaScript **O**bject **N**otation |

*For/Dedicated to/To my…*

# Chapter 1

# Introduction

## 1.1 Motivation

In the past few decades, the amount of processing power available to the average consumer has skyrocketed. A current-day IPhone, for example, has vastly more processing power than NASA's moon lander had in 1969. With this increase in technological ability comes software that previous computer scientists would have conceivably considered science-fiction - one-such example of this is software that leverages Artificial Intelligence (AI), which aims to give human-like intelligence to machines. The average consumer-level phone, among other things, is now capable of utilizing advanced AI for a variety of uses - keyboards predict your next word, photo applications automatically detect faces in the photos you take, and recommendation engines can now predict what a consumer wishes to buy, sometimes before the consumer them-self knows.

Among such uses for our modern-day AI capabilities is the field of Image Recognition - a sub field of AI that focuses on teaching machines to recognize things in images. Image Recognition has been shown to be hugely effective for a number of use-cases - it is utilized in Tesla self-driving cars that must monitor their surroundings, and Snapchat which uses image recognition to apply 'filters' to users' faces. One-such use of image recognition, which this project aims to implement, is to add meta-data to images using image-recognition. For example, if an image-classifier saw that an image contained a dog, it may add this information to the image's description. This is known as "image tagging". Tech giants such as Google, IBM, and Amazon currently make up a huge presence in this area with cloud-based AI ready for consumer use in the form of Google ML-Kit, IBM Watson, AWS Rekognition, and so on, each sporting their own forms of image analysis which, among other things, are capable of performing this type of tagging on images.

One important aspect of developing an image-recognition solution such as in the case of tagging images, is the sheer amount of data necessary when creating the software in question. A sample-set of images must be sourced that is used to effectively ”train” image-recognition software for it’s task. Not only must the sample-set be accurate and properly-shaped (for example, having images with the correct dimensions), there must also be a substantial amount of sample images, sometimes in the millions[5, 3.3, 4]. This project attempts to provide a novel method of sourcing the required sample data that, in tandem with optimizations to lower the required sample-data size, could potentially allow the software to be entirely self-sufficient in terms of it’s learning.

## 1.2 Executive Summary

The goal of this project is to utilize AI Image Recognition technologies and contemporary optimization techniques to build an intelligent image-sharing platform. This platform will allow users to upload and share their photos, and add descriptive tags to them, similar to a Twitter ”hash-tag”, but will also use image recognition to generate it’s own learned models for tagging images. By combining user-generated tags with user-feedback on the accuracy of the application’s tags, along with the accompanying photos, the image recognition model can be continually improved. This is a potential solution to the problem of sourcing data for the Machine Learning aspect of the application, and doubly functions as a space where photographers, artists, and others may store and share their images.

## 1.3 Contribution

This project contributes to the fields of cloud computing and image recognition. As said previously, modern optimization and image recognition techniques will be utilized for the project. One such technique is the use of ’checkpoints’ when training your AI - when the model you are training achieves a certain level of performance for a certain task, it is common for the developer to publish the configuration that allowed them to achieve that performance - certainly in this case, if a high level of performance is reached during my project, the configurations for it will be published on GitHub and left open-source for other developers to use. Furthermore, since this application aims to continually learn using new images that are added by users, it may be possible to train models that have not yet been developed by others, although this is not a guarantee, since the landscape of image recognition is quite vast.

On-top of contributing to the open-source AI and image recognition communities, this project will also contribute to photographers, artists, and any others that wish to share their images with the public online. One logical benefit of the system continually-improving it's descriptive capabilities is it's potential to improve it's own search-engine-optimization (SEO), which has the benefit of it's user's listings showing up more often in Google/Yahoo/Bing searches. Conversely, those who are looking for a specific image online, whether it be for personal or commercial use will also see these search listings more often, making their search easier and more effective.

## 1.4 Structure of This Document

This document aims to give a complete overview of the research and development undertaken for the project. The following is a complete breakdown of the chapters, and the ideas/concepts that each one addresses.

Chapter 1 gives a brief introduction to the thematic area and provides a motivation for the project. Section 1.2 attempts to give an elevator pitch of what the project will be, while section 1.3 outlines some of the areas that the project hopes to contribute to. Finally 1.4 describes the function of each chapter in the paper.

Chapter 2 aims to outline the background of the paper. The information highlighted in this chapter is a necessary prerequisite to building the system pitched in Chapter 1. It does this by first diving into the thematic area itself (section 2.1), giving an overview of the technology produced by the field and the history of the field since its conception. This section also aims to clarify some of the common terminology used in the field - phrases such as "Artificial Neural Network" and "Machine Learning" will be detailed in order to fully grasp their significance. Section 2.2 then identifies some of the major players in the AI SaaS market, and the features that each of them provide to the consumer, with section 2.3 doing a deep-dive into the inner workings of the field, explaining the current state of the art for the technology that this project will utilize.

Chapter 3 will tackle the problem itself - it begins by defining the problem as clearly as possible; a clear problem definition will lay the foundations for eliciting the proper objectives and requirements of the solution. Section 3.2 will then outline the project objectives - the final success of the project lays in, and will be measured by the completion of these objectives. Section 3.3 will look at the functional requirements of the project, and 3.4 will look at the non-functional requirements.

Chapter 4 will outline the approach taken to implement the system. Beginning with section 4.1, the high-level architecture of the system will be outlined, followed by a decomposition of each component within the system. Following this is an overview of the technologies that will be used to implement the subsystems - this includes programming languages and frameworks. Section 4.2 will then assess any risks involved in the implementation of the system, this can have huge value during the implementation of the system as it can help to mitigate development issues that may otherwise stunt progress. Section 4.3 outlines the methodology the will be adhered to during the implementation. Section 4.4 will highlight an implementation plan schedule that, while ambitious, must also remain feasible given the time-constraints of a single semester. Section 4.5 determines the metrics on-which the implementation will be evaluated for success - this goes further than simply asking if the system functions - it should determine a useful value that gives an intuitive understanding of the overall effectiveness of the process. Finally, section 4.6 will highlight any prototyping that has been done for the project thus-far.

Chapter 5 will lay out any conclusions made during the research phase of the project.

# Chapter 2

# Background

## 2.1 Thematic Area within Computer Science

This project's area of exploration is a blend of Artificial Neural Networks (ANN's), and Software as a Service (SaaS). ANN's are a specific implementation of a broader field known as Machine Learning(ML), which is in itself part of the broader field of Artificial Intelligence(AI). This project also has ties to the SaaS field, as it focuses not-only on constructing high-performance ANN's, but also hosting these ANN's on the cloud for use by third parties. The necessary prerequisites of AI, ML and ANN's will be looked at in this chapter, as-well as a review of the current landscape of SaaS/AI solutions. This will aid in identifying the niche that this project aims to provide for.

Put simply, the field of AI in computing aims to give machines the ability to mimic human-like intelligence. Put into practice, this means designing systems that can reason, learn, perceive and plan. As in humans, its desirable for these machines' behaviours to have some degree of flexibility when it comes to handling unforeseen circumstances - a doctor for example, will not always see the same set of symptoms when diagnosing the flu, one flu-patient may have a cough, and another a fever, but the doctor's underlying intelligence allows them to diagnose both patients accurately nonetheless. Similar behaviour is desirable in AI applications, and as-such they often draw from past experience and knowledge when performing tasks[2, pg.4].

While humanity has been fascinated with the concept of artificial, intelligent beings (e.g. The Golem) for centuries[6], many believe the 1956 Dartmouth Conference on Artificial Intelligence to be the birth-place of the modern field of digital artificial intelligence[7]. Since that time, there have been numerous initiatives with the aim of developing the field for use in industry, but disillusionment often followed such initiatives due to a lack of processing power[8]. To illustrate this lack of computational power, Hans Moravec hypothesized in 1975 that a PDP-10 time-sharing supercomputer, which could perform 340,000 instructions per second[9], fell short of mimicking human vision in real-time by a factor of between 100,000 and 1.6 million[10].

In contrast with the computational starvation the was prevalent in the 20th century, the processing power required to build and run specialized intelligent systems as proposed by Moravec is presently at the consumer level. Hardware such as Nvidia's 'Titan V', which can perform $9.8 * 10^{13}$ Floating-Point Operations per Second (98 TeraFLOPS) in certain scenarios[11], is readily available for purchase[12]. Reflecting this abundance of processing power, companies such as Tesla, Google and Space-X are utilizing AI for applications such as self-driving cars, rocket-guidance, and speech recognition.

Outside of the mainstream applications for AI, the research is also valuable to a number of other fields including medical diagnosis, and the stock market - where companies will often utilize predictive AI algorithms and classification techniques to augment their products and abilities[13, ch. 1.1].

Software as a Service, also known as SaaS, is a broad term used to describe any software or application that is delivered over the internet, and often refers to delivering that software on-demand[14]. In the case of AI, many tech-giants have a steak in the market for building SaaS tools that aid developers and other companies in designing and deploying AI. In the case of IBM Watson, for example, a full suite of image recognition training and deployment functionality is offered in an intuitive GUI-driven package right in the browser[15]. These solutions and their viability to AI-based projects will be looked at in greater detail later in the chapter.

Machine Learning is only a small subset of AI research, which focuses on intelligence derived from learning and experience. This is a key facet of intelligence in living things - a Developer, for example, is expected to be of a higher competence if they have experience in their industry of choice. The methods behind machine learning revolve around giving systems experience of problems as a means of solving them. That is to say, machine-learning software is designed to learn over time[13, ch. 1].

In ML, the data-structure that will receive training to complete a problem is often referred to as a model, and the application of algorithms to improve our model's performance is known as optimization. Digging deeper into the concept of learning, we can view a machine learning model as a function that we want to construct, that will give a desirable output when presented with the data we wish to process. A practical application of this would be an image-classification model that would take an image as input (depicting some object, say, a car), and output the class of object that the image depicts ("car"). We then implement an optimization algorithm of some sort on our model, with the intent of improving it's accuracy by tweaking the parameters of it's problem-solving function.

Learning can be achieved using many different techniques, as the field of ML contains a plethora of algorithms and data-structures. The use-cases for each technique aren't necessarily mutually exclusive either - since machine learning as a whole focuses on constructing a function that approaches 100% accuracy as knowledge is gained, often the same problem can be solved with more than one Machine Learning algorithm, and moreover, it is often wholly unclear as to which technique should be used. This is a side-effect of what is known as the No Free Lunch theorem, which, put loosely, states that there is no one perfect algorithm for solving every problem. [16]

Artificial Neural Networks (ANN's) are a type of ML model that learn by simulating the behaviour of neurons in a brain (hence, the term "neural"). Optimization is performed on ANN's by adjusting both the frequency and strength at-which neurons pass data to one-another. Much like the inner workings of our own brains, the eventual inner workings of an ANN are almost entirely incomprehensible, since each layer of neurons in an ANN represents a new layer of abstraction that make up the model's 'understanding' of the problem at hand. For this reason, an ANN is said to be a 'black box'. While there is ongoing research into identifying means of extracting knowledge from ANN's, their inner nature is still considered a deterrent to developers who may often need to debug such models[17, p.1].

One of the most popular examples of a complex problem that can be solved by neural networks is the classification of handwritten digits. While a human brain has no problem reading a handwritten number and parsing a value from it, boiling this action down to a highly accurate algorithm is incredibly difficult - primarily due to the fact that no two handwritten numbers will ever be exactly alike.

One popular approach to this problem is to construct a basic neural network, and train it to recognize the digits. Just like in human brains, the network's performance on the task is prone to some margin of error, and it will be essentially useless when starting from scratch, but with each iteration of training, it's neurons will be adjusted to perform better. This is done using an algorithm known as "back-propagation", and when applying this algorithm to recognize handwritten digits, it's possible to train a network to about 99% accuracy [18].

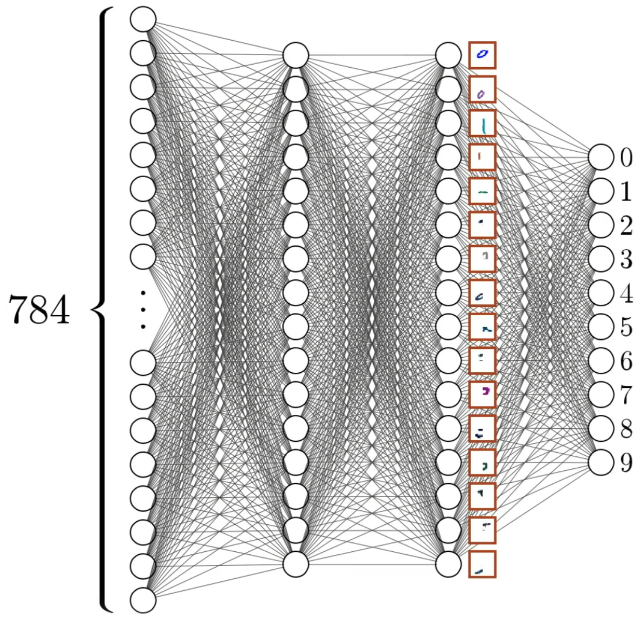


Figure 2.1: [1] A graphical representation of a simple digit-classifying neural network. This network accepts 784 pixels on the left, and produces an answer represented by a neuron on the right. The middle layers are the neural network's 'reasoning', where it creates abstractions of the data that were found to be effective.

Despite the above drawbacks, this approach to ML has shown huge potential in dealing with problems where this is no clear algorithmic solution available - problems that otherwise would require impractical or impossible amounts of processing power and code.

## 2.2 A Review of AI Solutions

There is a multitude of SaaS companies currently operating within the AI market, each targeting similar use-cases that enterprise-level customers looking for AI-solutions would face. AWS SageMaker, for example, provides servers that are pre-loaded with high-performance hardware and Machine Learning frameworks so that it's customers can build their own AI solutions without the need to purchase their own equipment[19]. On the other hand, IBM Watson skips the need to build AI solutions altogether by offering pre-trained models to the customer. These models are mainly in the field of speech and text analysis, and aim to automate many communication-based tasks in the hospitality industry, customer care industry, and so on[15], but also provide a high-performance image recognition model that can be extended and fine-tuned by the user. Finally, Google AI sits somewhere between SageMaker and Watson, providing both ML tools such as TensorFlow, and also pre-trained services such as text-to-speech and speech-to-text. The below table highlights some of the key focuses and differences between AI solutions currently offered by tech giants.

| | **Pre-trained ML Services** | **ML Frameworks (for building custom models)** | **Primary Areas of Focus** | **Secondary Areas of Focus** |
|---|---|---|---|---|
| **IBM** | Watson | Watson Studio | Speech analysis, Natural Language Synthesis | Optical classification, Custom ML model creation |
| **Google** | ML Kit | TensorFlow, TensorFlow Lite for Android | Optical classification, Custom ML model creation | Text analysis (smart reply) |
| **AWS** | Polly, Rekognition, Deeplens, etc. | SageMaker, Lex, TensorFlow on AWS, Apache MXNet on AWS, and more. | Optical classification, Speech synthesis, Custom ML Model Creation | |

Table 2.1: Comparisons between AI solutions by tech giants. "Primary Areas of Focus" are the areas that the company offers the most features for - for example, most of ML Kit's features revolve around optical classification, while only one relates to text synthesis (smart reply).

From the table above, it is clear that there is a huge presence occupying this market. Each of the organizations shown above offer their solutions as part of suites, where customers get a range of features when signing up for a single product. In the case of IBM and AWS, their suites appear to be designed to cater to a collection of use-cases that enterprises may face when developing products for a specific market. This of-course, is incredibly lucrative when selling to high-paying enterprise customers, but free-tier price plans are also offered for SageMaker and Watson that offer some level of free use for small periods of time. On Google's end, there are no price plans or subscriptions required for using ML Kit, anyone is free to download the ML Kit SDK, however, these services are designed specifically for Android and iOS applications, which won't fit every use case for those in need of a pre-trained ML model on other platforms.

For smaller companies in need of AI solutions, the general consensus is that the organization in question should design their own tailored AI solution, often with the advice of outsourcing the actual programming to a dedicated AI-design agency[20]. It may also be feasible to use one of the above products such as SageMaker, if the company in question has the skill-set and resources. If the company in question is interested in popular applications of machine learning such as image classification, text-to-speech, etc, then they would benefit greatly from a package such as Watson, since it provides both of these models, as-well as many others, pre-trained and ready for use over the cloud. This is highly relevant to the area that this project explores, since the project could conceivably utilize IBM Watson, AWS Rekognition, or another image recognition service for it's tagging capabilities. That being said, since services such as these behave like a "black box", so to speak, it would be impossible to publish the configurations that made our models effective, furthermore, all image-recognition capabilities would then be vendor-locked to the service, which can restrict an application depending on it's requirements.

## 2.3 Current State of the Art

Since this project aims to provide it's AI functionality from the cloud, it seems clear that there will be a need for an API that will be used to access these functions - there should be no real difference in design compared to any other API in existence today, however, there are quite a lot of considerations to make when determining how-best to design ANN's and ML models that will be hosted on an API, such as the time taken to execute, the amount of data that must be exchanged, and so on. In this section, we will look at some approaches to implementing ANN's and ML Models for the topics of Character Recognition and Facial Detection, and then dive into general object recognition and how it can be optimized.

### 2.3.1 Handwritten Digit Recognition

Handwritten Digit Recognition has been an area of study for many as early as the 80's[18]. As illustrated by Leon Bottou and colleagues, it is possible to achieve acceptable digit classification performance through the use of many different ML models [21], but since this paper is primarily interested in the application of this problem within an API hosted on the internet, we will choose an implementation that is both fast at classification, and computationally inexpensive to operate. With these constraints in mind, we will look at a 'fully-connected, multi-layer neural network', since it shows acceptable accuracy, but also incredibly low memory requirements and training time, comparatively speaking [21, fg.2, fg.3,fg.4].

A fully-connected, multi-layer neural network, despite it's name is an incredibly simple type of ANN. As with all ANN's, a neuron in this ANN is nothing more than a computation involving a set of weighted inputs and a variably-biased output which is normalized to a value between 0 and 1 (sometimes -1 and 1). This output is sometimes referred to as an 'activation' value. In such a neural network, neurons that comprise the network are arranged into 'layers' - inner layers that are neither the immediate input or output are known as 'hidden layers', or 'convolutions'. For a neuron in the $n^{th}$ hidden layer $H_n$, input will be received from neurons in $H_{n-1}$, and output it's own signal to neurons in $H_{n+1}$. The term 'fully-connected', refers to the manner of connecting different layers of neurons - a fully-connected layer is one where every single neuron in layer $H_n$ is connected to every single neuron in $H_{n+1}$. With this neural structure in mind, we can define the output/activation of any one neuron as the following:

$$\alpha_j = \sigma(\sum_{i=1}^{n}(x_i * w_{ij}) + b_j)$$

FIGURE 2.2: [2, ch.5.15] Equation showing the Activation value for neuron $j$, which is equal to the sigmoid (normalizes values to values between 0 and 1) of the sum of all of neuron $j$'s weighted inputs, plus a bias value $b$. Formula from Rob Callan's book 'Artificial Intelligence', adapted to include activation biases

The aforementioned Leon Bottou specified a fully-connected ANN with two hidden layers, of 300 units(neurons) each for a decimal digit classifier, but further experimentation can be done with other amounts of neurons and character sets too, if time spent training the model is acceptably low.

### 2.3.2 Increasing Sample-sets with Data Augmentation

As mentioned previously in this document, one of the major requirements of training neural networks is that an effective amount of data is used, which can at times mean sourcing as many as 1.3million images[5, 3.3, 4]. At times when this sheer amount of data is not available, one can opt to artificially increase the size of their dataset by making small distortions and transformations to their existing images, as exemplified [22, ch.2] by Simard et al. By skewing, rotating, and translating the image subtly, an effectively new image can be created. The key to this technique is not to make too drastic a change - the image must still resemble it's label, for example in the case of handwritten digits, an image of a '2' must still resemble a '2' after such transformations have occurred. It follows then that applying this technique once to every image in a dataset would effectively double the size of that dataset.

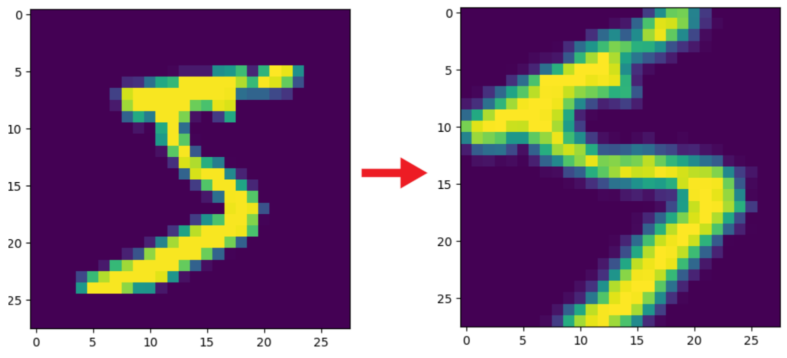


FIGURE 2.3: Two images depicting the same data-point of a handwritten '5' digit, before and after data augmentation procedures have been applied. The alterations applied were for rotation, zoom, and shearing. The second image, while still similar in features to the original digit, occupies vastly different coordinates, meaning it can effectively serve as a second, unique sample.

### 2.3.3 Training ANN's with Gradient Descent and Backpropagation

With the structure and nature of how a neural network operates now explored, we must look at the school of thought necessary to improve their accuracy with training. We must attempt to train our model by correcting it when it get's a prediction or classification wrong. To do this we must first adapt a specific view of our problem - if we are to use the popular method of 'gradient descent' to train our model, we must view our neural network as a function whose margin of error is mapped to a geometric gradient[3, ch.1]. On this hypothetical gradient, the lowest point represents the best solution our model can output, that is to say, it's error, also-known-as "cost", is minimized at this minimum point. This plane is then navigated by altering the weights of the neural network in such a manner as to travel gradually down this gradient.

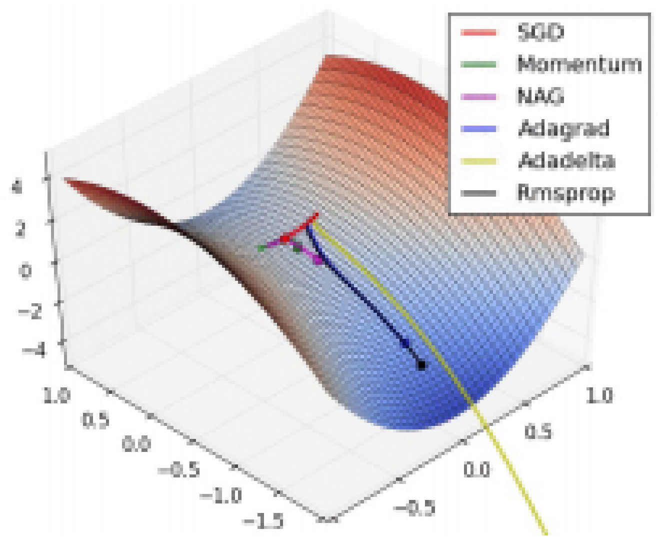


FIGURE 2.4: [3, ch.4.9] A visual representation of the paths taken by various optimization algorithms down the gradient of a problem-space, towards the lowest-error state

With this visualization in mind, we may now implement an algorithm known as "backpropagation" to tune the weights of our ANN's neurons in such a way as to ensure our model's accuracy continually travels down the gradient of the problem space, thereby minimizing the cost of the function. In order to minimize to cost of an ANN, each neuron must be iterated over, starting at the output layer, and have it's weights altered according to the slope of the gradient. We are able to find the slope of the gradient by deriving the consequential change in the cost of the model over the change in a certain neuron's weight, using the Chain Rule[Leibniz, 1684] to determine the exact relationship between the two variables[23, ch.5.5].

$$m_c = (\delta c)/(\delta w_n)$$

FIGURE 2.5: Where $m_c$ denotes the slope of the cost function, $\delta c$ is a change in the cost, and $\delta w_n$ is a small change in the weight of a neuron

This formula is applied to an output neuron, and we then traverse backwards through the network, applying the same formula to each connection. Once this is done, we take an average of the changes in weights for each neuron and apply the changes to them, there-by descending the cost gradient.

### 2.3.4 Neural Network Fine-Tuning / Transfer Learning

The final area of neural networks that must be explored in this document, and arguably the most important concept for the purposes of this work is that of transfer learning. Transfer learning is yet another technique aimed at reducing the work necessary to construct a viable ANN[24, ch.1], and it does so by attempting to re-use certain sections of other pre-trained neural networks that perform similar tasks. L. Y. Pratt, 1993[24] puts forth a scenario where a neural network has been trained to recognize English speech in American accents, and a new neural network for recognizing the British accents must be built. In such a scenario, it has been shown viable[25] to recycle many network layers and their pre-trained weights, and simply retrain only small, dissimilar layers of the network. This is a highly valuable technique to leverage, as the high number of open-source pre-trained neural networks currently in existence is quite large, making the training portion of any similar networks far easier, if said-weights were to be cloned.

#### 2.3.4.1 Transfer-Learning with Image Classification ANN's

While the above explores the effectiveness of transfer learning for the purposes of speech recognition, a similar level of performance has been achieved when using transfer learning for image recognition ANN's[4]. It has been shown that the inner convolutional layers of a deep neural network can be largely left untouched when one wishes to alter the classes that the system is capable of tagging. This occurs due to convolutional layers being trained as highly-generalized feature-extraction mechanisms that work on a very broad range of images[26, ch.4.2]. These layers can be cloned to new classifiers with low loss of performance as-long-as the types of images involved in training are vaguely similar, that is to say, a neural network trained for recognizing CT-scan images may not transfer well to a network designed to classify dogs and cats, however in the case of training a classifier with similar classes but from a different dataset, the results are quite successful[4, ch.4].

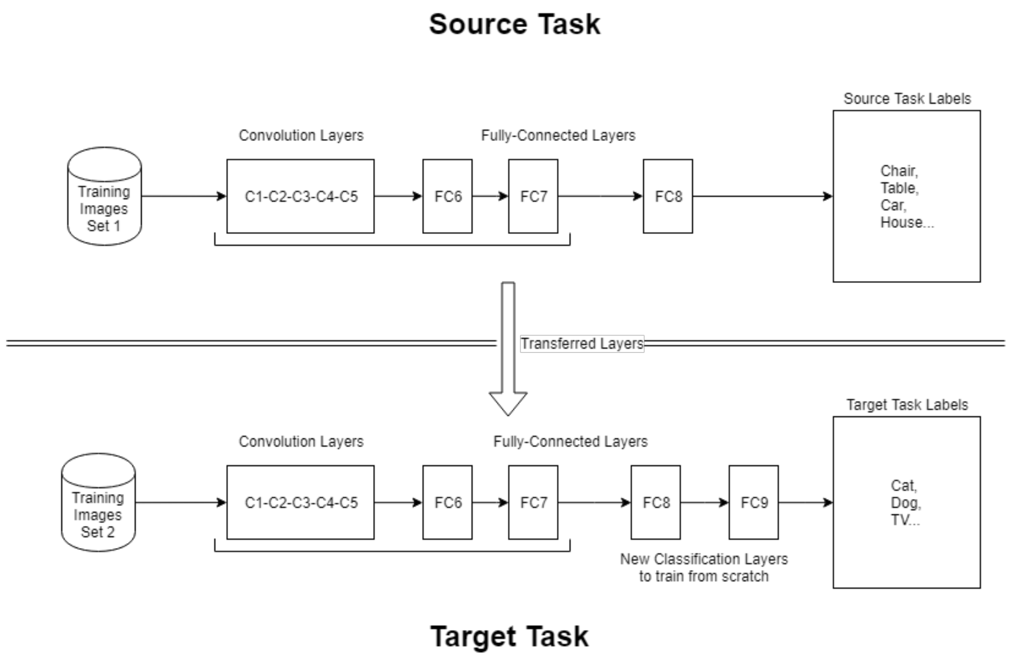


FIGURE 2.6: [4] A pre-trained image classifier with feature-extraction and classification layers labelled "source task", and new ANN to be trained on a different dataset contained a clone of the original ANN's feature-extractors.

In the above figure, a pre-trained image classifier is shown under 'Source Task', which features five convolutional layers and two fully-connected layers that function as feature-extractors, with a single fully-connected layer acting as the network's final classification. Below it is the target ANN we wish to train on a different dataset, and contains a clone of the feature-extraction layers, along with two fully-connected layers to adapt the features and classify the new images. This effectively reduces the number of layers that need training from scratch to just two, which is of-course a huge boost in efficiency compared to training a full network.

# Chapter 3

# An Intelligent, Self-Learning Image Sharing Platform

## 3.1 Problem Definition

The field of artificial neural networks and especially neural networks that are used for the purposes of image recognition require vast amounts of data in order to function effectively. In the case of the VGG16 convolutional network, whose construction is fully documented in K. Simonyan and A. Zisserman's paper 'Very Deep Convolutional Neural Networks For Large-Scale Image Recognition', 1.3 million sample images were used over the course of the network's training, and the entire training session took between 2 and 3 weeks to finish[5, 3.3, 4]. On-top of this, a further 50 thousand images were used for validation purposes, and a further 100 thousand for testing purposes[5]. Clearly, a huge data-source is required for such an endeavour to be even remotely feasible, and on-top of having the images themselves, each image must be labelled appropriately so that the system can determine when it is correct or incorrect during training and testing. In the case of the VGG16, the data-source used was ImageNet - a large collection of labelled and logically-ordered images that can be used by anyone, given that they possess the resources to download and process such astronomical amounts of data (The aforementioned researchers used 4 parallel Nvidia Titan Blacks[5], the MSRP of each of these cards reaching over $1000[27].). Based off the information found in this paper, the problem becomes quite clear:

- **Utilizing large datasets:** Primarily, the scale of the data needed is far too large for many uses, as even if the data can be downloaded effectively, it takes a herculean amount of computational power to process. It follows that any optimizations that would lower the required dataset size would hugely benefit the training and testing of any neural network.

- **Sourcing datasets:** Secondly, given the scenario where a data-source such as ImageNet does not contain the images desired by a particular user, other avenues must be looked at for sourcing these categories of images. With this in mind, there is good reason to attempt to expand the number of categories that we can train networks on, and to explore other means of acquiring the desired data.

- **Benefits of image tagging:** Finally, as a secondary problem, ineffective meta-tagging can lead to difficulty in finding the information that one seeks. The effective meta-tagging of document can be of huge benefit to those who wish to find specific categories of information, and when improperly tagged, can be of huge detriment - this holds true for images, and since this project focuses on the tagging of images, the act of doing so will inherently aid in search-engine optimization, which will in-turn aid the process of searching for data when using a search engine such as Google.

## 3.2 Objectives

Given the above problems that must be solved, the objectives of this project, which will be used to measure the overall success of the implementation are as follows:

1. Host uploaded images on the cloud, and serve them on the web.
2. Generate meta-tags for images based on their contents.
3. Collect user-feedback on meta-tag accuracy for use in performance metrics.
4. Collect images with commonly-used auto-generated tags and fine-tune the image-tagging system with them.
5. Save "checkpoints" when the automatic tagging performs better after fine-tuning.

## 3.3 Functional Requirements

1. The web-application will allow a user to upload a photo.
2. The web-application will provide the user with suggested meta-tags for their image.
3. The user can select which generated tags are appropriate, and also add their own.

4. The user can view, delete, and edit a list of their uploaded images.
5. An Administrator can view metrics regarding the application, such as the most (and least) successful auto-generated tags.
6. An Administrator can view metrics based on the most popular user-generated tags.
7. An Administrator can trigger the fine-tuning of the tagging mechanism and view it's testing performance.
8. An Administrator and decide whether or not to deploy a tagging mechanism.

## 3.4 Non-Functional Requirements

1. The application will use a GUI-based web front-end.
2. The application will authenticate users using a name/email and password.
3. The application must use persistent storage to accommodate images and associated tags.
4. The recognition system must not take too long to generate tags.
5. The recognition system must have high accuracy.

# Chapter 4

# Implementation Approach

## 4.1 Architecture

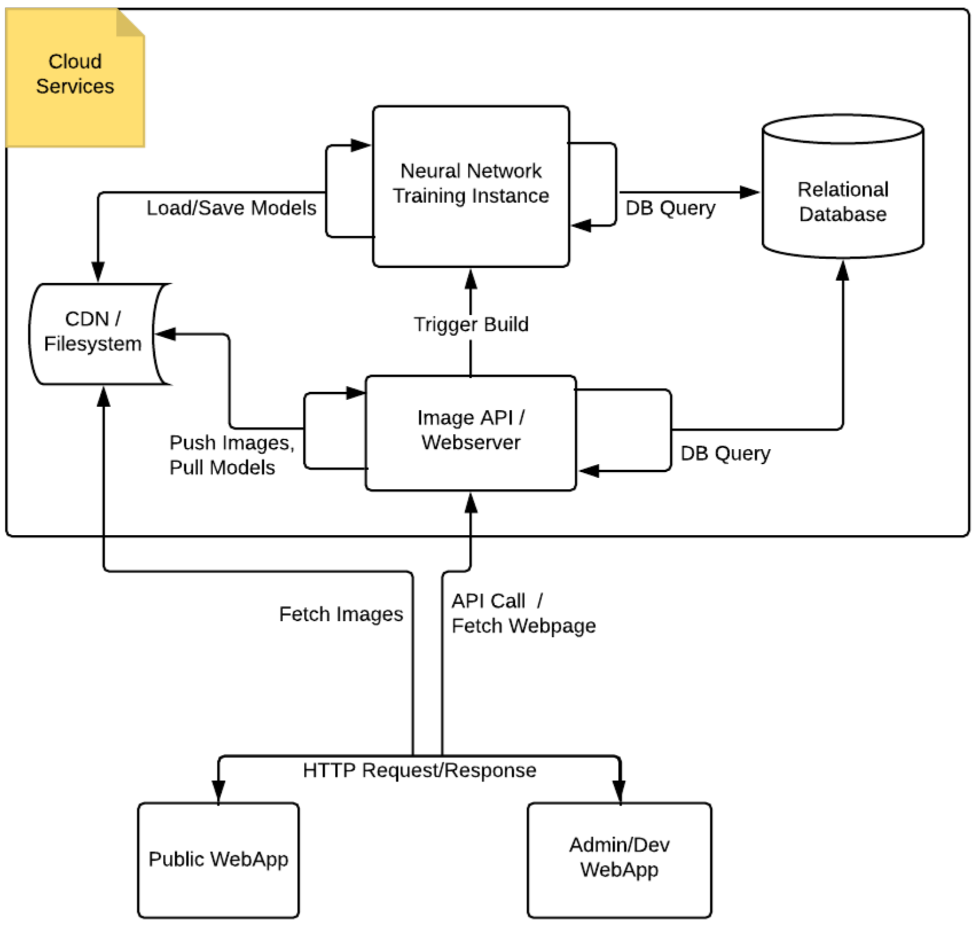


FIGURE 4.1: A high-level overview of the proposed system.

In this section, the underlying architecture of the system will be looked at. We will first identify the top-level applications and services necessary for the implementation of the proposed system, before decomposing each one into their respective constituents. Following this we will dive into the frameworks and libraries that will be used as the bedrock of the system, this will include possible web services that could be utilized for deployment purposes.

### 4.1.1 Image API / Webserver

It may become apparent from the above diagram that the image API is the core of this project, as it will be responsible for all critical application functionality - this functionality ranges from the basic classification of user-images, to spawning new training instances that will attempt to fine-tune the core neural networks in use.

The API is written in Python3.6, which is an interpreted language that has many compatible machine-learning libraries available. For this application, the Keras machine-learning library will be used. One of the main reasons for choosing this library is that it comes packaged with many high-performance, pre-trained neural networks, one of-which we will adapt using the transfer-learning technique[28]. Keras also provides utility functions for saving and loading trained neural networks[29], which will be used to keep track of the system's latest classification configurations. Python also has a number of web-hosting libraries available to it such as Flask, which will be used to host API endpoints. Finally the mysql-connector package for Python will be used so that the API can communicate with a database for persisting data.

Upon being spawned, the image API must perform setup operations. Primarily, it must establish a database connection so that it may manage users and their images. The relational mapping of users to their images is imperative to the correct operation of the web-app. Next, it must fetch the latest neural network configuration file from the CDN/File-system (depending on whether the application has been deployed to the cloud or run locally), which will be used for any inbound tagging requests. Since the Keras package is used, the configuration file will be in the format 'HDF5'. Once the trained neural network has been loaded, the API can begin accepting inbound requests for image tagging, whether it be for generating new tags for an image, getting previously-generated tags for an image, or for spawning a new training instance to fine-tune a new classifier. These endpoints are mapped using Flask, and all inbound requests will be managed according to these mappings. As-well-as providing functionality to the user, the API also serves web-pages. Flask comes bundled with a HTML templating engine known as Jinja2, which can be used to inject information dynamically into web-pages before serving them. This can be incredibly useful for creating dynamic web-applications where a user has personalized web-pages.

### 4.1.2 Front-end

The front-end in the case of this system represents the customer-facing, or client-side interface that will be used to actuate the system's functions. Since this project presents Administrator-only features such as viewing metrics, there must essentially be two separate classes of web-page: public web-pages that anyone can sign up to view, and admin web-pages, where the user in question must have escalated privileges. Both front-end solutions will be quite simple, with the public front-end largely consisting of three web-pages - one to view all uploaded images, one to view specific images and get shareable links, and one for uploading and tagging new images. For the admin-page, there will be a page to view details regarding past classification models, and another to view the specific details of each tag within the classifier.

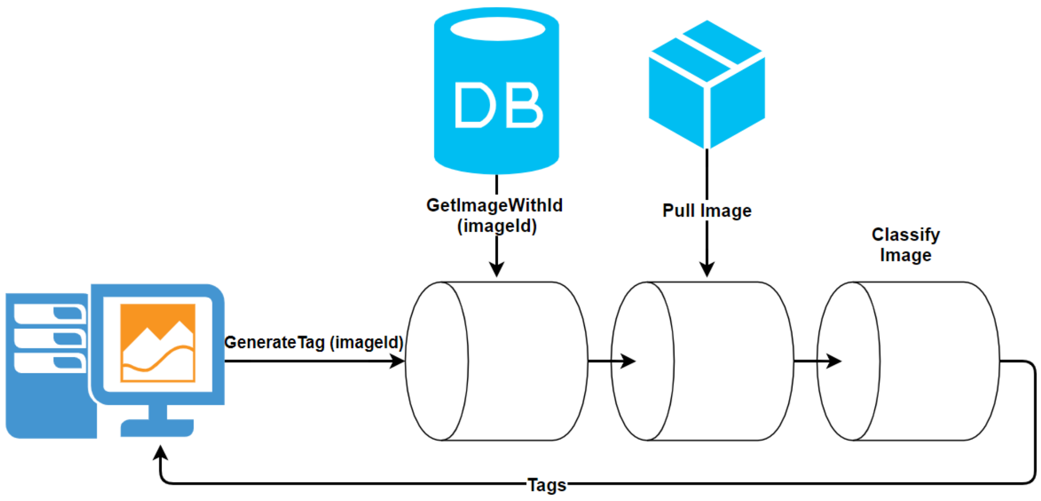


FIGURE 4.2: A high-level overview of the route taken to tag an image.

### 4.1.3 Training Instances

A crucial part of this application is the subsystem that will programmatically attempt to train new, better versions of the classification neural network. This process demands the most computational resources out of any of the subsystems, and therefor should be run in it's own environment, free of any starvation of resources that may occur due to it sharing them with, say, the web-server. Similarly to the API, Python3.6 will be used with Keras in order to train the ANN.

The training instance will utilize metrics collected from users in order to determine the baseline effectiveness of the current classifier. If no metrics exist for certain tags, the initial testing scores from the previous training session will be used. From there, the training instance will source a dataset to train and test on - it will first search for user-generated images for each tag it can classify - an equal amount of images for each tag must be sourced so that the ANN doesn't develop a bias to more popular tags. If there is a sufficient amount of images for each tag, the training instance will fetch the latest classifier configuration file, and begin fine-tuning it against the user-generated images. However, if there is an insufficient amount of images for each tag, the training instance will attempt to incorporate as many user-generated images as possible into the original batch of images used in the last training session, and will attempt to train the network from scratch. The batch of images will then be split into a training dataset, and a smaller testing dataset.

Once training has been completed, the training instance will test itself against the testing set it made previously. If the performance is better than the currently-used classifier, it will save the model and upload it to the CDN/file-system and tag itself as the latest version. For this reason, the API should have a private endpoint for the training instance to call that will trigger a reload of the API.

Since the re-training of the neural network is a continuous process of learning, a mechanism must exist that will trigger a new training instance. A potential solution to this is to trigger it on a specific interval, however, the API may also keep count of how many images have been uploaded since the last training session, and may trigger it after a certain threshold is reached.

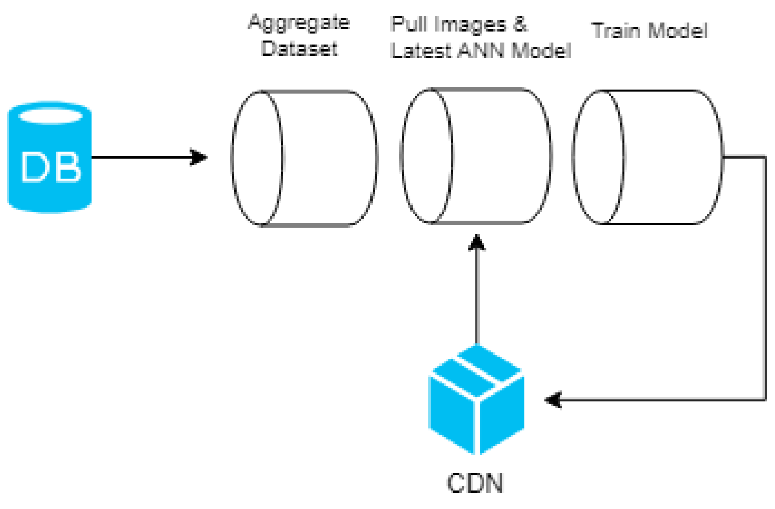


FIGURE 4.3: The high-level flow of data through the training instance.

### 4.1.4 VGG16 Image Recognition Network

In order to utilize the power of neural networks with transfer learning, a pre-trained, high-performance network must be used that has been trained to recognize natural images, i.e. images that were taken of real-life objects. One such neural network is offered as part of the Keras package. The package in question is known as the VGG16[30], and has won awards for it's ultra-high performance when classifying images from the ImageNet database. With this ANN loaded, the top 3 layers will be taken off, and a custom-built classification ANN will be trained on-top of it.

### 4.1.5 Relational Database

The relational database is an important component of this project as it maintains a mapping of users, their images, and the tags for each image. When a new image is uploaded by a user, the API first inserts an image record into the database with the date it was created, the user who uploaded it, and the name it was given by the user. The image will then be added to the CDN with a unique file-name that corresponds to it's database entry. Once the upload is complete, the API will attempt to tag the image, and will return any tags that are somewhat likely to be accurate. Once the user chooses the final tags to add to the image, they will be added to the image's database entry, along with the predicted tags that the user didn't choose - this data can then be leveraged by training instances to determine the classifier's performance. For the purposes of this project, Oracle MySQL Database will be used, however there are many free database solutions available that all offer very similar functionality.

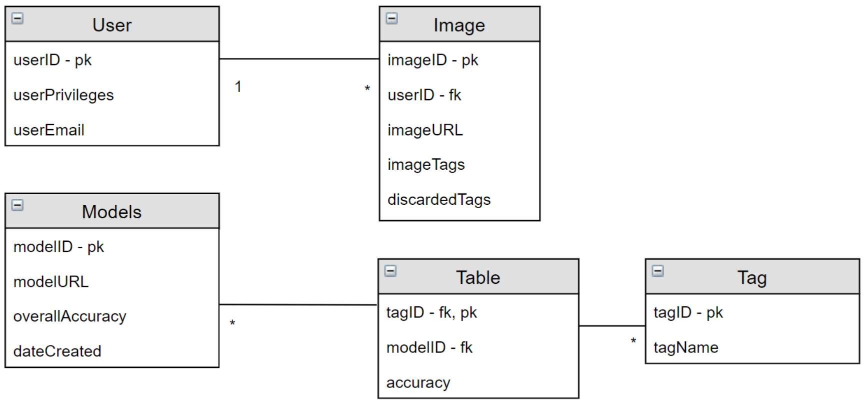


FIGURE 4.4: The structure of the relational database that will map the data collected regarding tags in each classifier, as-well as users, their images, and the image's tags.

Table 4.1: Initial risk matrix

| **Frequency/ Consequence** | **1-Rare** | **2-Remote** | **3-Occasional** | **4-Probable** | **5-Frequent** |
|---|---|---|---|---|---|
| **4-Fatal** | 🟨 4.2.1 | 🟥 | 🟥 | 🟥 | 🟥 |
| **3-Critical** | 🟩 | 🟨 4.2.2 | 🟨 | 🟥 | 🟥 |
| **2-Major** | 🟩 | 🟩 | 🟨 | 🟨 4.3.3 | 🟥 |
| **1-Minor** | 🟩 | 🟩 | 🟩 | 🟨 4.3.4 | 🟨 |

## 4.2 Risk Assessment

While the above architecture helps immensely in detailing how a system will eventually work, it is almost impossible to completely eradicate potentially fatal risks from occurring during such a large undertaking in software development. The more thought that is put into risk assessment in the early stages of development, the less costly it will be to mitigate any potentials issues. A paper by Maurice Dawson et al. illustrates a similar trend with respect to software issues[31], where it is shown that the cost of fixing a bug during implementation is 6.5 times as costly as fixing it during the design phase, 15 times as costly during the testing phase, and 100 times as costly during the maintenance phase. Below is an exploration of any conceived risks and their possible mitigations:

### 4.2.1 Loss of Crucial Project Data

Lack of redundancy with data brings an inherent risk; persistent storage in a PC can fail at any moment and without warning, and such a loss would be fatal at any point in the development of the system, with the cost of the failure growing proportionally to the amount of work invested in it. For this reason, redundancy must be kept for all files that are essential to the operation of the system. In the case all code-based files, such web-pages, the API, the training instance files, and any SQL files, a remote code repository will be kept on GitHub which can be cloned at any time. In the case of assets that would be normally stored on a CDN / File-system, such as user images, neural network saved models, and so on, will be backed up regularly to cloud storage, such as OneDrive or Google Drive. Moreover, the git-flow[32] version-control paradigm will be used within the development repository in order to ensure that the code-base is never left in a broken state due to parallel work-flows.

### 4.2.2 Failure to Train Adequate Classifiers Within Time Constraints

The training of high-performance neural-network-based classifiers can often be an arduous endeavour, and represents a high-dimensional problem that consumes huge amounts of computational resources. Currently a Nvidia 1070ti GPU is at the disposal of this development which is on the higher end of consumer graphics cards, however these cards still pale in comparison to the multi-GPU parallel rigs that are provided to many researchers studying the field, therefor training is expected to take longer than shown in most research papers. While this is an acceptable trade-off under normal circumstances, the time-sensitive environment in-which this system will be built means that there is a limit to the number of attempts at training that are possible before development must cease completely. For this reason, the initial proof-of-concept classifier will serve as a benchmark of the headroom available in terms of potential training time. Based on the results of this benchmark, higher-class classifiers will be trained accordingly.

### 4.2.3 Failure to Classify Images Correctly / Bad Training Data-sets

This threat relates strongly to the previously-explored one - in such a scenario where anyone is permitted to upload an image, there is a strong chance that the image will be of a different nature than those used to train the initial classifiers. This issue is to be expected, primarily in the early stages of self-training, when only controlled, high-quality images have been used for training, before the pool of available images has been diversified. In a sense, this risk mill mitigate itself over time, however it isn't certain how long it would take, or how much processing power would be necessary before the issue is minimized. If such an issue does arise, it may be manually remedied via the training of normally-frozen feature-extraction layers operating within the network - this is a costly remedy but may prove necessary. Another issue inherent in this system is the possibility of "griefing", where-by users intentionally tag photos incorrectly with the intent of undermining the learning algorithms at work. However, this system will not be considered for public release at least until after it's review, therefor this secondary risk is largely a non-issue.

### 4.2.4 Failure to Classify Image in an Effective Time-Frame

One of the metrics on-which the success of this system will be measured is the ability to tag images in a length of time that doesn't leave the client waiting. This is considered a bad usability heuristic and can lead the user to believe the application is broken. Many routes could be taken to optimize this issue in various stages of classification, however given the lack of resources available for this project, a suitable solution would be to enforce a maximum image resolution in order to constrain maximum load time when uploading a new image for classification.

## 4.3 Methodology

A considerable amount of time was spent considering various approaches to the development methodology that would be implemented for the project. While modern development teams often jump to using Agile, or sometimes Waterfall development as the best means of building and maintaining a project, it is not quite so intuitive an approach to take for a single-person team. Furthermore, in the case of Agile, since it focuses primarily on reacting to changes in elicited requirements from stakeholders, it seems wholly inappropriate given the more rigid, linear nature of the project. A more important concept arguably, is to ensure that the code written matches the pre-determined specifications as closely as possible, while avoiding the possibility of the final application deviating heavily from what was initially conceived.

For the purposes of small or one-manned teams, a common suggestion is to use Extreme Programming(XP). Extreme Programming improves a project in five essential ways[33]:

1. Communication: team members constantly communicate with the customer and other team-members. While this aspect may seem counter-intuitive in the current scenario, it is not entirely without merit; this concept will be realized by constantly referring to work done in terms of the goals that were laid out (representing the customer), and by periodical meetings with the project supervisor who is, in essence a team-member for the duration of the project.

2. Simplicity: design and implementation of the software is kept simple and clean. It behooves the developer - all the more in a one-manned team - to maintain as-simple a code-base as possible, especially in such circumstances where time is a strong factor. This will aid in maintaining good pace throughout the course of the development process.

3. Feedback: Through the testing of the software, feedback is found in how it performs, if it contains bugs, how quickly it runs, etc. Feedback will also be received, in this case, from the project supervisor who will provide guidance regarding the progress of the project.

4. Respect: While this aspect is slightly irrelevant in terms of respect for team-mates, it will certainly instill a sense of accomplishment with each new feature or feature-set that is implemented, and will power a drive to maintain constant delivery of features.
5. Courage: Through the above foundations of XP, the developer will attain a sense of courage to respond and adapt to any changes that may occur throughout the development life-cycle.

## 4.4 Implementation Plan Schedule

Planning will be an essential component in completing this project in an efficient manner. The goal for this work is to develop a minimum viable product as early as possible that is highly extensible. This will serve as the bedrock for the remainder of the work to be done. This will entail developing the core functionality of the program, i.e. the API and it's recognition capabilities as soon as possible. Subsequently, the act of hosting this functionality as an API endpoint will be explored, and from there, different endpoints will be added. The following schedule is constructed under the assumption that 2hours is spent each day developing the application. The expected sequence of work is as follows:

1. (Week 1) Install All Necessary Software: all necessary packages and libraries must be installed. This includes MySQL Database Community Edition from Oracle, the correct version of Python, Keras machine learning package, mysql-connector database package for python, Flask for web mapping. This will be installed on a Linux Ubuntu 18.04 LTS installation running dual-booted alongside Windows, since Linux offers more flexibility in general, not to mention Python was originally designed on Linux rather than Windows. Installing on Linux also provides support for the Docker containerization service, which would potentially allow for any application built to be deployed to the cloud while remaining compatible.
2. (Week 1-2) Implement a minimum-viable-product of the training instance - this will simply train a neural network based on stock images, likely sourced from the ImageNet library. To ensure that the training instance is programmed with good foundational code, it will be designed to accept configuration options via command-line arguments. This will streamline the process of spawning it from another python program i.e. the API. Once finished, it will save the trained classifier to a location specified via a command-line argument.
3. (Week 3) Implement the core functionality of the API - this means loading in a pre-trained model which has been saved by a training instance - the location to

look for the save-file can also be specified via command-line arguments. The API will be programmed to unpack and use this model, initially using a dummy set of images. This functionality will likely be encapsulated in a single file which may be initialized by other Python modules.

4. (Week 4) Build the Database model and a script to initialize it - once this is done, both the training instance module and the API module will be able to load the mysql-connector package and establish connections to the database. Initialization scripts for the Database will be provided with a ‘data.sql‘ file that will populate the tables with dummy content that will allow for more development on future API endpoints.

5. (Week 5) Implement an authentication/sign-up mechanism that will allow users to sign in and utilize a userID - this is crucial to all web-application functionality that provides personalized content to the user. This includes endpoints for viewing all a user’s images, uploading images, assigning public/admin privileges, etc. This means that any authentication done must integrate with the database.

6. (Week 6-7) Implement all remaining API endpoints. This is likely to be the most intensive part of the project, due to the sheer number of functions that must be written. The application known as Postman will be used to test these endpoints, or highly-primitive web-pages where appropriate.

7. (Week 8) Implement the front-end mapping on the server-side that will serve web-pages to the client. Through the use of the Jinja2 templating engine, these web-pages will be compiled in order to personalize them for each user.

8. (Week 8) Implement the view of all front-end web-pages, and devise a simple global style-sheet

9. (Week 9) Implement any front-end JavaScript to aid in the functionality of the web-app.

10. (Week 9-10) Extend the training-instance to analyze tag accuracy and construct new training/testing datasets.

11. (Week 10-11) Evaluate each module on the guidelines laid out in the following ’Evaluation’ section of the paper.

## 4.5 Evaluation

The evaluation of the systems is not only a means of determining the quality and usability of it's features, but also represents the overall effectiveness of deploying integrated neural networks into web applications. It is important that some form of scoring is applied to each and every important aspect of the system's functionality, which will range from it's overall accuracy, down to the intuitiveness and feasibility of the interface. In order to quantitatively measure each of these aspects, the system will be logically decomposed into it's major constituents, i.e. The front-end, the back-end (API), and the training instance.

For each of the constituents, the following questions must be asked:

- Does it work?
- Is the solution simple and easy to understand?
- Could it function better / be optimized?
- Are there any criticisms of the solution?
- Does it accomplish the objective that it was originally designed to address?

Of course, with each of these questions, there is room for interpretation, and the questions must be adapted to each system appropriately. In the case of the question "does it work?", for a classifier this may refer to how accurate the model is or how much it has/hasn't improved over it's predecessor. In terms of the front-end, we may ask how easy the interface is to use, and if it violates any major usability heuristics. With these questions explored, a score out of ten will be assigned for each question and for each subsystem, and will be averaged to find a final score for the subsystem. All subsystem scores will then be averaged to find the overall score for the application implementation.

## 4.6 Prototype

While no true implementation has been done specifically for the purposes of the project, some proof-of-concept work has been done pertaining to the mapping of API endpoints within Flask, as-well as some rudimentary work with classification pertaining to optical character recognition which, while not directly relevant to the project, greatly aids in the learning curve of working with high-dimensional data and ML pipelines.

Shown in Appendix figures A.1 and A.2 are proof-of-concept request mappings written with Flask, that listen for requests and return a date and time that has been marshalled to the JavaScript Object Notation (JSON) format. This is the basis for all endpoint mappings, with of-course the difference being that in the final application, the response JSON will be populated with tags relating to a classified image, or some other meaningful data related to the endpoint.

Shown in Appendix figure A.3 is sample code from a digit-classification project that was originally finished before the beginning of the college semester, and modified during the semester to use a custom-written neural-network saving/loading module. This work provided a fundamental understanding of the nature of neural networks, especially those pertaining to image classification. Figure A.4 shows the output of the training program, which was able to train the neural network to be 98% accurate in the domain of classifying handwritten digits. Not-only did this side-project provide invaluable learning regarding neural networks, but also regarding the processing and sourcing of the data that would be used for the code. Images must be converted from 2-dimensional arrays of pixels to a flat array of 784 pixel values, and have their values normalized to between 0-1 instead of 0-255. This provided me with a simple understanding of TensorFlow, a machine learning library used by Keras, and how to configure it to utilize the GPU for highly-parallel computation.

Some basic wireframing has been done to get a sense of how to public and admin front-ends will look in the final application. The goal is to utilize a material-design[34] library in order to construct an intuitive and beautiful front-end. Appendix figure B.1 depicts the first page the public user will be greeted with after login. Material-design is designed by Google and as-such, this application looks incredibly similar to web-applications such as Google Drive, Google Docs, etc. The header of the page will be common across all pages and contains a search-bar for searching for the user's images via their tags, as-well as the user's icon which is used for account functions such as editing their information, logging out, etc. The main section of the page depicts a grid of all the user's uploaded photos. When hovered over, a list of tags for the photo and an options button appears. When the image is clicked, it will navigate the user to a page for viewing the photo on a larger scale. If the options button is clicked, the user will be shown an options menu where they can choose to view, edit, delete or share their photo.

Appendix figure B.2 shows the main admin front-end page, which they will be greeted with when entering the admin-only area of the website. Following the material-design specification[35], this page uses a common layout and header to all other pages, with the exception of the main pane, which now shows a list of classifier deployments rather than image thumbnails. Each entry in this list is an accordion-menu that can expand-collapse, as depicted by the angular arrows on their right-side. For each list item, the ID, date of creation, and accuracy are shown. After expanding a list item, a further breakdown of the tags and each tags accuracy will be shown for that specific classifier. The admin will also be given the option to re-deploy whichever classifier they choose, or can also delete records of the classifier. Finally, the large '+' icon on the bottom-right now serves as a new training instance that the admin can trigger at any moment. Once the training instance is finished, instead of automatically deploying the classifier, it will simply add it to the list, where the admin can choose to deploy or delete it.

# Chapter 5

# Implementation

Implementation of the application was documented continuously over the course of its development. As is the case in many development environments, the implementation phase of this project yielded a large variety of challenges and learning curves. This section explores these events and aims to highlight their effects on the end-result.

## 5.1 Difficulties Encountered

Various difficulties were encountered during implementation that necessitated additional problem-solving and decision-making. These difficulties varied in terms of their impact on the overarching goals of the project, and are documented as-such.

### 5.1.1 Medium: Database integration with a Flask web-server.

One major downside to the application's chosen approach to Flask servers[36] is that it does not easily allow for the sharing of objects between components, which ultimately inhibited the application's ability to share a database connection across all necessary files.

1. In order to resolve this issue, a global "settings" script was written that contained important objects such as the database connection. This allowed all relevant modules to access the database. While global variables can often cause issues of their own, in this case it was one of the few viable workarounds.
2. While this issue did not represent a flaw in the core concept of the project, it still posed a serious threat to development, mainly due to how far along in the

development cycle it arose. Had the aforementioned solution failed, it may have been necessary to refactor the entire application.

3. Thankfully, this problem did not have a cascading effect on other components of the project. The methodology towards development remained the same and continued as normal after the issue was resolved.

4. Although this issue took a considerable amount of time to overcome, it did not drastically alter the project's implementation schedule. Many attempts were made to solve the issue before the above solution was implemented - these attempts made up the majority of the time consumed.

5. No changes to the evaluation plan were needed following the mitigation of this issue.

6. As mentioned previously, in order to identify the most viable solution, possible approaches were explored by making minimal prototypes. The first attempt involved encapsulating all server components in objects, which could be passed a database connection and stored as a class variable.

### 5.1.2 Easy: A race condition in the training instance.

The training instance was designed such that it would begin building a new model when triggered by a HTTP request. Training is executed on a new process thread, allowing the training session to persist even if the HTTP request is killed (e.g. the requesting user navigates away from the web-page). This design choice introduced to possibility of two simultaneous requests triggering two independent builds on the same GPU, which led to the builds corrupting each-other and eventually failing.

1. There are many possible solutions to this problem that would allow for parallel training instances: the installation of a second GPU, or the construction of a "parallel model"[37]. Due to the time and monetary constraints however, a simpler solution was required. With this in mind, the solution chosen was to implement a flag indicating whether or not a model is currently being trained, and denying all new training requests until the flag shows no active instances.

2. This issue, while intimidating to encounter, did not pose any threat to the project, but rather necessitated that a design choice be made regarding the application's parallel capabilities.

3. The asynchronous events that occurred to produce this issue are intrinsic to web-based applications. Therefor it's reasonable to assume that similar issues can arise

elsewhere when this fact is not considered. Following the identification of this issue, all design choices made within the scope of the project were prefaced with the question "Will this function in a concurrent/asynchronous environment?".

4. This issue did not consume any considerable amount of time due to the simplistic solution that was implemented, therefor it left the implementation schedule unaffected.
5. This issue somewhat altered the evaluation plan of the assignment as it denies any possibility of training multiple models at once. With this in mind, any metrics relating to parallel training are now out of scope.
6. The decision-making process for this issue was short-lived, due to the infeasibility of the alternatives. Out of all hypothesized solutions, the chosen-implementation stood alone in-that it wouldn't consume an impractical amount of time or investment.

## 5.2 Actual Solution Approach

Although the aforementioned difficulties did not drastically impede implementation, the application still underwent a number of minor iterations as experience was gained.

### 5.2.1 Architecture

The broader architecture of the application remained largely unchanged over the course of its implementation. The constituent-components that make up the system are still dependent on the originally-planned technologies (in all but one case), and interact with one-another in the same manner.

#### 5.2.1.1 Implementation: Image API / Webserver

The Image API / Webserver is a Python Flask server providing API endpoints for use by both public users and administrators. Endpoints and functionality are logically grouped according to their function. In practice, this means creating a unique Python file for administrator-only endpoints named `admin.py`, as-well the file `image.py` which is responsible for all image-oriented endpoints. Finally, the `views.py` file is responsible for serving all web-pages. This multi-file approach to the server was ultimately a success, despite the aforementioned difficulty regarding the sharing of a common database connection across files.

When started, the API loads a pre-trained image classification model into memory and uses this model to classify incoming images. In tandem with a database connection, all incoming images are sanitized, classified, and inserted into the database. Since the API also serves web-pages, the Jinja2 templating engine is used to inject all necessary data into web-pages on-demand. The API *would* also be responsible for all user authentication, however, as will be detailed later in this chapter, this feature was not implemented in the final application. For this reason, all requests currently received by the API are treated as having administrator privileges.

#### 5.2.1.2 Implementation: Front-end

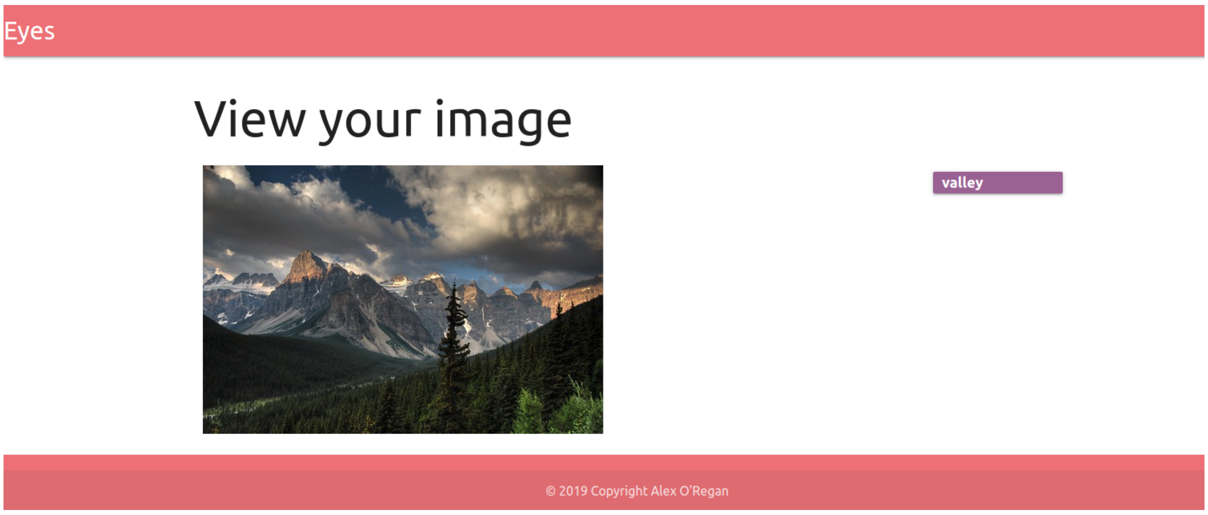


FIGURE 5.1: The front-end view of a tagged-image, depicting any image tags on the right-hand side, with the image itself on the left.

The front-end was originally planned with the intent of showing two separate types of web-page: administrator-only web-pages that would allow for the creation of new training instances, and public web-pages where a public user could sign in and view/upload their images. In the project's current state, this separation of web-pages has no materialized, primarily due to the time-constraints experienced during implementation. Currently, a primitive skeleton-website has been developed that allows for the uploading and classification of images. A rudimentary admin web-page has also been completed which allows for the starting and viewing of training instances. It's clear that there is still a huge amount of potential polishing to be done in this area of the application, however with that said, the core functionality has been built, with the exception of authenticated, user-specific functionality.

#### 5.2.1.3 Implementation: Training Instances

During the planning phase of the project, the training instance was to be implemented in the form of a VGG16 neural network which would be built using the transfer learning technique. During the implementation phase, however, additional research was made into the various neural networks offered by the Keras framework, and it was found that there were better DNN architectures available for the purposes of this project. In particular, the DNN known as DenseNet[38] was documented as having comparable (if not superior) performance to the VGG16, while consuming only a fraction of the resources. Upon further inspection within the Keras framework, it was found that the full DenseNet-based classifier, trained to classify 1000 classes had a maximum of 20,242,984 parameters, while the full VGG16-based classifier had a maximum of 138,357,544 (see Figure A.5 of Appendix A). For this reason, the DenseNet201 model was chosen over the VGG16, with the expectation that it would allow for savings in memory resources, which could then be utilized in other areas of the application.

The remainder of the training instance was implemented according to the original specification. When the application is triggered, it will begin training a new classifier. Database queries are made in order to locate images containing the appropriate classes, and a table is constructed representing a training dataset. Transfer learning is then implemented on the DenseNet model and a new classifier is trained and saved, with its details logged in a database. In order to trigger the training instance, the image API makes a gRPC-based call to the training instance module, which listens indefinitely for said-trigger. This call contains all information required by the training instance, such as the classes that the new model must classify. The call also allows for an optional checkpoint argument, which instructs the training instance to pick up where a previous model left off in training.

#### 5.2.1.4 Implementation: Relational Database

The database built for this application is crucial to the proper function of both the training instance and image API. The schema for this database outlined previously has since undergone important iterations to ensure it optimally represents the data it's given. In the original design, an image (represented by the 'Image' table) contained a column that would hold all tags associated with it. Since this column was not logically tied to the actual 'tagName' column in the 'Tag' table, it would be likely that duplication of tags could occur. This was noticed during the construction of the database, and the schema was revised to address the issue.

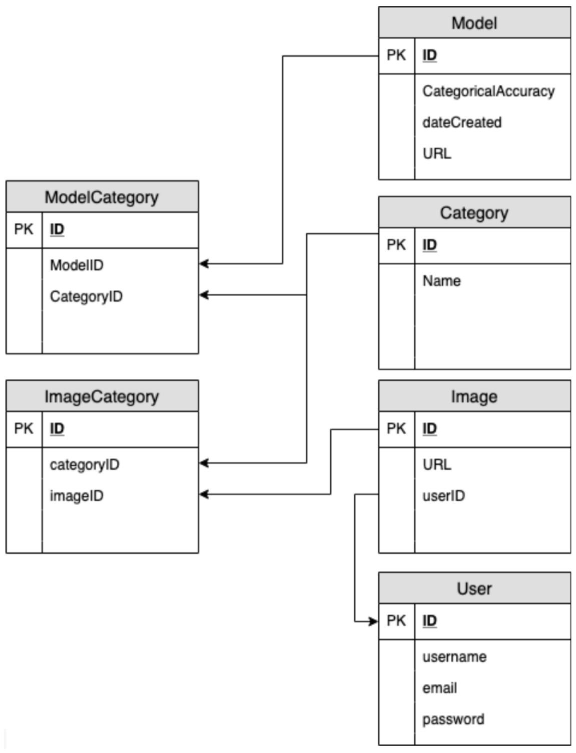


FIGURE 5.2: The revised database schema used by both the image API and training instance. Compared to the original design, this database has an additional table 'ImageCategory' that logically links images to tags, ensuring that an image's tags are accurate.

### 5.2.2 Risk Assessment

The risks originally planned for in the planning phase of this project, while reasonable, have now evolved drastically as new experience was gained from the implementation of the system.

#### 5.2.2.1 Failure to Train Adequate Classifiers Within Time Constraints

While some small-scale work had been attempted on the topic of transfer learning (training a 2-class classifier with a few-hundred images), it remained unclear whether or not training would remain feasible as the number of classes and images scaled. This question remained unanswered until roughly 10,000 images were gathered over 18 classes of image (600 images per class). At this scale, a model was trained to 79.9% accuracy over the course of about 13 minutes. This metric seems adequate given the number of photos that get uploaded to the internet on a day-to-day basis. Additionally, given that some classes have over 1000 images available on ImageNet, in contrast with the 600 per-class used for the aforementioned model, it would stand to reason that even better results are possible. With this information in mind, this risk is considered mitigated.

#### 5.2.2.2 Loss of Crucial Project Data

While it remains true that the loss of certain portions of data within this project would be catastrophic, the risk of such an event was handled appropriately over the course of the implementation. From the very beginning of development, all project files were tracked using Git and also stored remotely in a GitHub repository. Included in these files was a database dump file which would allow for the reconstruction of the database, as-well-as all the original data contained within it. In terms of files are *are not* tracked via Git, for example, the often-large models that are frequently trained during testing, data loss would not represent a large risk to the project, given that, as mentioned previously, decently-performing models can be trained in under 15 minutes.

#### 5.2.2.3 Failure to Classify Images Correctly / Bad Training Data-sets

The failure to classify images effectively is an ongoing risk which must be considered in all instances of training. The previously-noted concept of "griefing" is still a prominent risk to any system that learns from public data. This effect is partially mitigated by the simple fact that a poorly-performing model (trained on bad data) would likely never reach deployment, however additional measures could potentially be taken in the future that would further reduce the risk associated with this phenomenon. Outside of bad data, trained models can reach up to 90% in certain cases, therefor the risk of such a model consistently performing poorly would be rare.

#### 5.2.2.4 Failure to Classify Image in an Effective Time-Frame

The risk of a classification request consuming an unreasonably long duration of time remained prominent for a large portion of the implementation phase, due to the fact that the Image API was not fully implemented until after the training instance was functional (this order of development was necessary, since a classification request could not be made until a model existed to complete the request). However, once a functional API endpoint for predicting images had been built, it became quickly apparent that HTTP requests for the purposes of classification adequately met the time requirements. This, despite the Image API conducting all DNN computation on the CPU (the GPU being reserved for training instances).

### 5.2.3 Methodology

The methodology outlined in the planning phase was followed for the majority of the implementation phase. Weekly deliverables were presented to the project supervisor in the form of small reports, outlining the work that had been done. On-top of this, a weekly meeting was scheduled where the completed work could be discussed, and future work could be planned. In terms of Extreme Programming, communication was maintained for the majority if the implementation phase, with slight deviations from the weekly meetings when time became scarce. Simplicity was maintained in the codebase, primarily through the use of the Git "submodule" feature, which allowed for the logical separation of the project into it's constituent pieces. The use of this feature ensured that each module in the project was always entirely separated from the rest, preventing any blurring of the roles of each piece. Feedback on the work was received both from the project supervisor, as-well-as from metrics received from the application itself. The feedback received via analysis of model accuracy was vital to the application's success - on one occasion, it became clear that the training instance was not functioning correctly. This was determined via the observing of model validation accuracy, which was incredibly low. This led to the realization that data was not being adequately-shuffled prior to beginning training.

In the last few weeks of development, the weekly meeting process that would normally allow for communication with the project supervisor did not occur regularly, largely due to the increasing shortage of time at this point in the implementation.

### 5.2.4 Implementation Plan Schedule

The implementation schedule was an aspect of this project that largely fell by the wayside. It became clear very early in the development if the application that the amount of work required by certain parts of the system were underestimated, while other pieces were overestimated. The schedule quickly evolved from focusing equally on all components of the application, to focusing primarily on the machine-learning aspects of the application, as it represented the most crucial piece of the architecture. The work per week unfolded as follows:

- (Week 1) Week 1 proceeded as originally-expected: all necessary prerequisites were installed. A huge portion of this work had already been completed due to personal projects that used similar technologies, and the rest of the week was then spent constructing a primitive environment where pre-trained DNN's could be experimented with.

- (Week 2) During this time, a testing dataset was also constructed with images from ImageNet, which allowed for the first attempts at transfer learning and performance evaluation. It was also at this point, after successfully loading the VGG16 model from a Keras package, that the question of other, better-suited DNN's arose. However, this question was left unanswered for the remainder of the week as work was allocated to manipulating Keras models for the project's purposes. Overall, this week proceeded according to the original plan, however there was still more necessary work to complete regarding transfer learning, such as the saving of trained-models.

- (Week 3) It was at this point in the implementation schedule that development deviated from what was planned. Rather than beginning work on the Image API, Week 3 was predominantly spent researching other architectures of DNN that would outperform the VGG16, with the end-result being the decision to swap out the VGG16 with DenseNet201 - this architecture is more lightweight in terms of resource requirements. It was also at this point that transfer learning was successfully implemented, however the training set was too small to draw any meaningful conclusions from it's performance.

- (Week 4) This week was originally designated as the point where a database would be developed for both the Image API and training instance. While there was a conscious effort to begin work on this, a more appropriately-sized dataset was instead constructed that would allow trained DNN's to perform adequately. Database development was prioritized for the following week instead. Work on the Image API designated for week 3 was also deferred once again.

- (Week 5) This week was originally designated as the point where user-authentication would begin development, however, it was instead spent developing the database schema that was originally planned for the previous week. It was also at this point in development that the training instance achieved the ability to train an accurate model, and save it to persistent storage. The model implemented could classify 10 classes with 92% accuracy. It was also at this point that the first prototype of the Image API was built. The API was given a single endpoint that would trigger the training instance to begin building. With regards to user-authentication, it became clear that this feature was low-priority compared to other modules in the system, hence it was deferred to the final stages of development. This allowed for more work to be allocated to orchestrating the primary pieces of the application, and how they would communicate.

- (Week 6 - 7) The original plan for this time-frame was to implement all remaining API endpoints that would allow for a function web application. However, there was still work left unfinished with regards to the database. While an initialization script had been written for the database, it had not been properly populated yet. By the end of this week, the training instance was capable of pulling all data regarding datasets from the database. The remaining amount of time left in this week was spent refining the communication between the Image API and the training instance, and developing a new Image API endpoint that would allow for the uploading and classification of new images.

- (Week 8) Week 8, while originally designated for developing front-end, was instead allocated primarily to migrating all existing image data into the database, and adding additional endpoints to the Image API. Another feature of the training instance was also added - the ability to automatically chose a dataset, and fully manage its own training. This feature was originally allocated for week 9/10, but considering the large amount of work left to do in other areas of the application, it seemed prudent to fully-complete the training instance in-case it was not possible to finish it later. A small amount of work was also allocated to writing the Image API functionality that would serve web-pages, however no web-pages were actually designed at this stage, bar a bare-bones form for submitting new images for classification.

- (Week 9 onward) It was at this point in the implementation that time pressure began stunting the weekly-deliverables for the project. Development from this point onward did not adhere to the original schedule. Development was done on the application's front-end, which allowed for the core mechanics of image-tagging and uploading, as-well as triggering builds. User-specific functionality did not come to fruition.

### 5.2.5 Evaluation Plan

The plan to evaluate the application remains exactly the same as it originally did - despite all difficulties encountered during the development of the system, all proposed evaluation methods and metrics remain the feasible. Measurements taken will include HTTP request time for a classification, developed-model categorical accuracy on unseen data, the heuristic quality of the front-end, number of features successfully implemented, and more.

### 5.2.6 Prototype

Where no proof-of-concept had been developed following the research phase, the implementation phase began with the goal of producing prototypes of the required functionality as quickly as possible. The results of the prototypes built over the course of development were included in the weekly reports, and as the implementation phase comes to a close, a near-complete product can be fully-initialized with a single command.

# Chapter 6

# Testing and Evaluation

## 6.1 Fulfillment of Functional Requirements

The functional requirements of the system represent the core functionality that must be implemented before an application can be considered complete. The initial functional requirements laid out during the research phase were as follows:

1. The web-application will allow a user to upload a photo.
2. The web-application will provide the user with suggested meta-tags for their image.
3. The user can select which generated tags are appropriate, and also add their own.
4. The user can view, delete, and edit a list of their uploaded images.
5. An Administrator can view metrics regarding the application, such as the most (and least) successful auto-generated tags.
6. An Administrator can view metrics based on the most popular user-generated tags.
7. An Administrator can trigger the fine-tuning of the tagging mechanism and view it's testing performance.
8. An Administrator and decide whether or not to deploy a tagging mechanism.

Of the four functional requirements revolving around public users, many have been only partially-fulfilled. This is mainly due to a key feature of the application not being implemented: user-authentication. While the application does allow for the uploading of images, these images will not be tied to the individual that uploads them. The second requirement can be considered complete at the system is capable of generating tags for any image uploaded. Once again with reference to requirement three, while it is possible within the application to update the data linked to an image (such as its tags), this act is not restricted to the owner of an image, again due to lack of user authentication. This same sentiment applies to requirement four, where an image can be viewed, deleted, or edited, but not at the user level.

As with the public user-oriented requirements, the lack of authentication extends also to administrator-level features. Currently, while not restricted to the public, functionality exists to allow administrators to view tags with the highest number of corresponding images, however there is no differentiation between user-generated tags and auto-generated tags. In terms of the final two functional requirements, the administrator dashboard allows for the training of new models as-well as the activation of any existing models.

From the above evaluation, it is clear that the application would greatly benefit from an extended development period. While the system does serve as a decent proof-of-concept, it is not currently ready for deployment to the public.

## 6.2 Metrics

In order to truly quantify the success of this project, a variety of scores must be produced based on each important aspect of the application. For this reason, the fulfillment of both functional and non-functional requirements will be investigated, based on analysis of a range of metrics. Additionally, the level of completion of certain components such as the GUI front-end will be looked at, and what features are missing.

### 6.2.1 Fulfillment of Non-functional Requirements

Non-functional requirements are a useful gauge of the operational quality of an application. During the planning phase of this project, the following non-functional requirements were put forward:

1. The application will use a GUI-based web front-end.
2. The application will authenticate users using a name/email and password.

3. The application must use persistent storage to accommodate images and associated tags.
4. The recognition system must not take too long to generate tags.
5. The recognition system must have high accuracy.

In terms of fulfillment of each of the proposed by criteria, almost all have been satisfied, with the primary exception of-course being user-authentication which remains unimplemented. The Image-API is fully capable of serving web-pages which provide a GUI to the client. Moreover, it is also fully capable of persisting all image data uploaded. In relation to the final two non-functional requirements, a numerical value must be produced before the application's effectiveness in the area can be determined. In short, the results found for both test cases was positive and can be determined fulfilled.

## 6.3 System Testing and Results

To identify the speed at-which the application can classify an image, a number of factors must be considered. Since this application offers image classification via a HTTP network request, it would not be accurate to simply measure the interval of time between form submission and server response. This is due to the fact that such a method overlooks the possibility of a low-bandwidth environment, line-noise which would lead to dropped packets, or the uploading of unreasonably-large files; all of-which would drastically increase request duration. With these factors in mind, testing was conducted such that they were disregarded - the units measured represent the interval of time between the completion of the client request, and the starting of the server response. This value serves as the minimum possible classification time, which would then be increased by varying amounts determined by the aforementioned-factors.

The same image was submitted ten times for classification at four different resolutions: `3840x2160, 1920x1080, 960x540, 480x270`. From the repeated measurement of the interval between classification request and server response, an average classification time of 1.185 seconds was recorded. This was done by logging the time at the beginning of the request and the very end of the request within the Image API. It is worth noting that, despite accounting for image size in the HTTP Request, the size of the image still had a proportional effect on classification time (see Figure 6.1). This is most likely due to the simple fact that more raw pixels required processing. It can be concluded with this information in mind that for the vast majority of images, the application will not take an unreasonable duration of time to classify images.

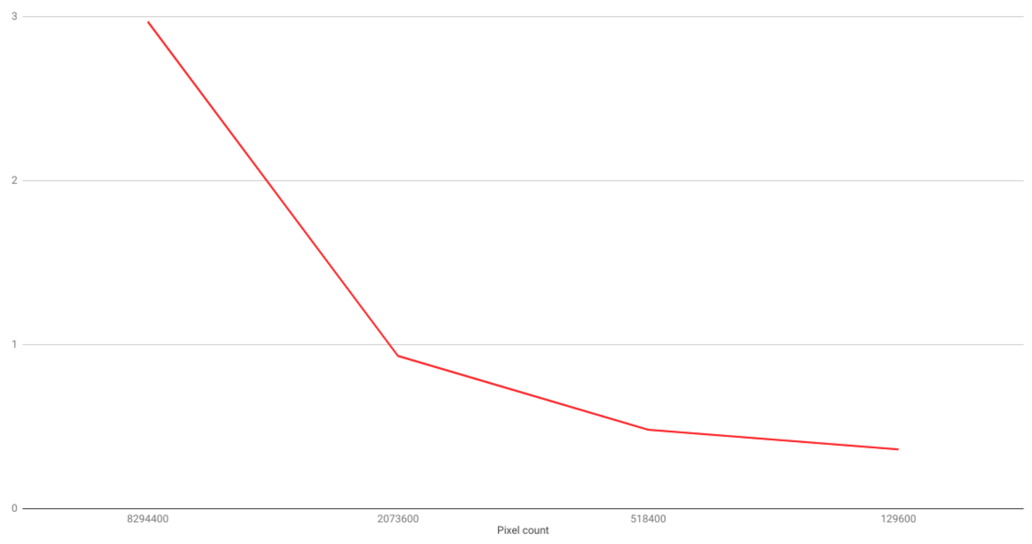


FIGURE 6.1: A graph depicting the relationship between an image's pixel count, and the average time taken to classify the image

In terms of the accuracy of the recognition model itself, a model was trained to classify 18 classes of image, with 600 images per class used in training. The model finished training with 79.9% categorical accuracy, when tested on images that it had never been exposed to previously. For such a relatively small data-set, this result is reassuring. For popular classes of images, this number of samples should be more than achievable for a reasonably popular image sharing platform. Figure 6.2 shows the entire history of the model's categorical accuracy over the course of the training session.

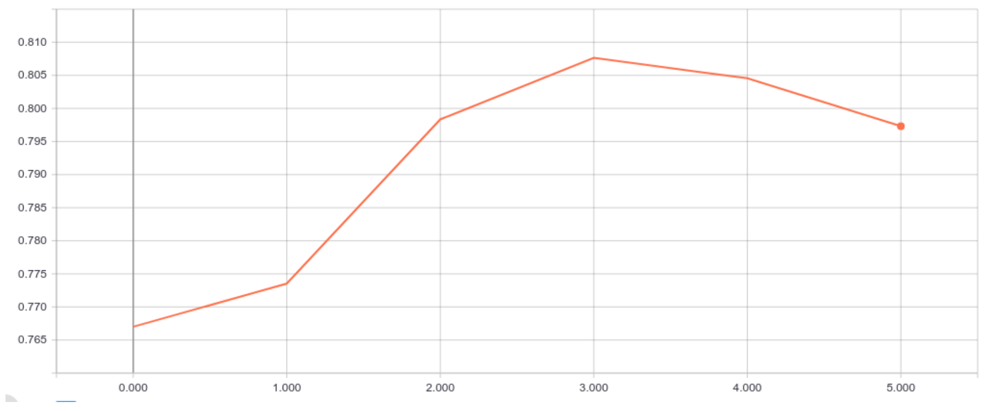


FIGURE 6.2: A graph depicting the relationship between an image's pixel count, and the average time taken to classify the image

# Chapter 7

# Discussion and Conclusions

## 7.1 Solution Review

With an application such as this, it can be difficult to quantify to what degree it will solve the original problem in the real world, since the application would ideally require deployment and use by the public before an adequate measurement could be made. Training sessions on the testing data-set, such as that depicted in the evaluation section show huge potential for the transfer-learning technique in terms of it's ability to adapt models to the required task. It would be reasonable to assume that, given a decent user-base, this application's effectiveness would persist in a publicly-available environment, with the assumption of-course, that the remaining risks within the system can be managed adequately. such as the sanitation of images, etc.

Outside of this speculation, I believe the application is almost fully capable of solving the original problem, with the exception of user-authentication, which represents the final piece of the puzzle. Despite this missing feature, the most important parts of the system have been completed: the construction of transfer-learned Deep Neural Networks, the construction of training datasets, and the hosting of an AI-driven API for public use, have all been fully implemented. If it's assumed that the user-specific functionality will be implemented later, this application will be more than capable of solving the original problem.

## 7.2 Project Review

Overall, I'm incredibly satisfied with the outcomes and progress of the project as a whole. Both the research and implementation phases were unquantifiably lucrative in terms of the experience gained. State-of-the-art techniques in the field of Deep Neural Networks were successfully implemented which allowed for a novel solution to the real-world problem of sourcing diverse and accurate data-sets. If it were to be re-implemented, more focus would be placed on creating a fully-functional front-end in order to fully actuate the image-sharing process. That being said, the most important systems within the application also cannot be neglected (e.g. the transfer learning), and since these aspects of the project were indeed successfully built, this project should be considered a huge success.

## 7.3 Future Work

Given the amount of work that had to be done in such a restricted time-frame, many areas of the application were written in less-than-optimal ways. With this in mind, there is a multitude of optimizations that could be made in order to improve the performance and quality of the application. The most obvious area of improvement is the front-end which should be fully-polished, with all aspects of the website adhering to a consistent style, allowing for optimal ease-of-use. Moreover, all missing user-specific features on the website, such as basic authentication should be built.

On the machine-learning side of the system, a key-feature that would greatly augment the feature-set of the application is the possibility for parallel training instances, where multiple models can be trained at once. This would allow the application to scale in accordance with the hardware it has available - if it were deployed to a multi-GPU environment, parallel training instances could fully-utilize the hardware.

Finally, a strategy that this application would greatly benefit from would be the implementation of other AI-based models in tandem with the current classifier architecture. For example, through the use of an image localization model, all irrelevant areas in an image could be discarded during classification, leaving only the piece of the image that the classifier should concern itself with. This could greatly aid in the performance of the classifier. Another example of this would be the use of an image-hashing algorithm in data-set sanitation - this could be used to detect duplicate images in a dataset, reducing model over-fitting.

## 7.4 Conclusion

To conclude I feel this project is a legitimate and novel solution the a real-world issue, with the only remaining uncertainty revolving around if the public would choose to adopt such an application, hence allowing it to thrive. Overall I feel this project was a success, both in terms of the problem it solves, and the experience and knowledge gained from it.

# Bibliography

[1] G. Sanderson, "But what *is* a neural network? — deep learning, chapter 1," 2017, image created by Grant Sanderson for his video on neural networks on YouTube. [Online]. Available: https://www.youtube.com/watch?v=aircAruvnKk

[2] R. Callan, *Artificial Intelligence.* Palgrave MacMillan, 2003.

[3] S. Ruder, "An overview of gradient descent optimization algorithms," *arXiv preprint arXiv:1609.04747*, 2016.

[4] M. Oquab, L. Bottou, I. Laptev, and J. Sivic, "Learning and transferring mid-level image representations using convolutional neural networks," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[5] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014. [Online]. Available: http://arxiv.org/abs/1409.1556

[6] J. Trachtenberg, *Jewish Magic and Superstition: A Study in Folk Religion.* University of Pennsylvania Press, United States (2004), 1939. [Online]. Available: https://books.google.ie/books?id=Sy3po8cI0FYC&printsec=frontcover&source=gbs_ge_summary_r&cad=0#v=onepage&q&f=false

[7] R. Kline, "Cybernetics, automata studies, and the dartmouth conference on artificial intelligence," *IEEE Annals of the History of Computing*, vol. 33, no. 4, April 2011. [Online]. Available: https://ieeexplore-ieee-org.cit.idm.oclc.org/document/5477410

[8] B. G. Buchanan, "A (very) brief history of artificial intelligence," *AIMagazine*, 2005. [Online]. Available: https://web.archive.org/web/20070926023314/http://www.aaai.org/AITopics/assets/PDF/AIMag26-04-016.pdf

[9] S. H. Fuller, "Price/performance comparison of c.mmp and the pdp-10," in *Proceedings of the 3rd Annual Symposium on Computer Architecture*,

ser. ISCA '76. New York, NY, USA: ACM, 1976. [Online]. Available: http://doi.acm.org.cit.idm.oclc.org/10.1145/800110.803580

[10] H. Moravec, "The role of raw power in intelligence," 1976. [Online]. Available: http://www.frc.ri.cmu.edu/users/hpm/project.archive/general.articles/1975/Raw.Power.html

[11] T. PowerUp, "Nvidia titan v specs," 2018. [Online]. Available: https://www.techpowerup.com/gpu-specs/titan-v.c3051

[12] N. Corporation. [Online]. Available: https://www.nvidia.com/en-us/titan/titan-v/

[13] A. T. Mehryar Mohri, Afshin Rostamizadeh, *Foundations of Machine Learning.* The MIT Press, 2012.

[14] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia, "A view of cloud computing," *Commun. ACM*, vol. 53, no. 4, pp. 50–58, Apr. 2010. [Online]. Available: http://doi.acm.org/10.1145/1721654.1721672

[15] "Ibm watson website." [Online]. Available: https://www.ibm.com/watson/ai-assistant/

[16] W. Hume. [Online]. Available: http://www.no-free-lunch.org/

[17] J. M. Benítez, J. L. Castro, and I. Requena, "Are artificial neural networks black boxes?" *IEEE Transactions on neural networks*, vol. 8, no. 5, pp. 1156–1164, 1997. [Online]. Available: https://www.researchgate.net/profile/Juan_Castro15/publication/5595919_Are_Artifi_cial_Neural_Networks_Black_Boxes/links/09e4150d40ff685694000000/Are-Artifi-cial-Neural-Networks-Black-Boxes.pdf

[18] J. D. D. H. R. H. W. H. L. J. Y LeCun, B Boser, "Handwritten digit recognition with a back-propagation network," AT&T Bell Laboratories, Tech. Rep., 1990. [Online]. Available: http://yann.lecun.com/exdb/publis/pdf/lecun-90c.pdf

[19] "Aws sagemaker website." [Online]. Available: https://aws.amazon.com/sagemaker/features/

[20] D. Rajnoch, "Why most startups should outsource their machine learning work." 2017. [Online]. Available: https://medium.freecodecamp.org/why-most-startups-should-outsource-their-machine-learning-work-d98d89144223

[21] L. Bottou, C. Cortes, J. S. Denker, H. Drucker, I. Guyon, L. D. Jackel, Y. LeCun, U. A. Muller, E. Sackinger, P. Simard, and V. Vapnik, "Comparison of classifier methods: a case study in handwritten digit recognition," vol. 2. IEEE, 1994, pp. 77–82 vol.2.

[22] P. Y. Simard, D. Steinkraus, and J. C. Platt, "Best practices for convolutional neural networks applied to visual document analysis," in *null*. IEEE, 2003, p. 958.

[23] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85 – 117, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0893608014002135

[24] L. Y. Pratt, "Discriminability-based transfer between neural networks," in *Advances in neural information processing systems*, 1993, pp. 204–211.

[25] L. Y. Pratt, J. Mostow, C. A. Kamm, and A. A. Kamm, "Direct transfer of learned information among neural networks." in *AAAI*, vol. 91, 1991, pp. 584–589.

[26] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*. Springer, 2014, pp. 818–833.

[27] "Pc mag - nvidia titan black." [Online]. Available: https://uk.pcmag.com/graphic-cards/32560/evga-gtx-titan-black

[28] F. Chollet *et al.*, "Keras applications," https://keras.io/applications/#vgg16, 2015.

[29] ——, "Saving/loading whole models," https://keras.io/getting-started/faq/#savingloading-whole-models-architecture-weights-optimizer-state, 2015.

[30] ——, "Vgg16-ilsvrc-winner," http://www.robots.ox.ac.uk/~vgg/research/very_deep/, 2014.

[31] M. Dawson, D. N. Burrell, E. Rahim, and S. Brewster, "Integrating software assurance into the software development life cycle (sdlc)," *Journal of Information Systems Technology & Planning*, vol. 3, no. 6, pp. 49–53, 2010.

[32] "Introducing gitflow." [Online]. Available: https://datasift.github.io/gitflow/IntroducingGitFlow.html

[33] "A gentle introduction to extreme programming." [Online]. Available: http://www.extremeprogramming.org/

[34] "Material design guidelines." [Online]. Available: https://material.io/design/guidelines-overview/

[35] "Material design." [Online]. Available: https://material.io/design/

[36] "Flask documentation." [Online]. Available: http://flask.pocoo.org/docs/0.12/blueprints/#blueprints

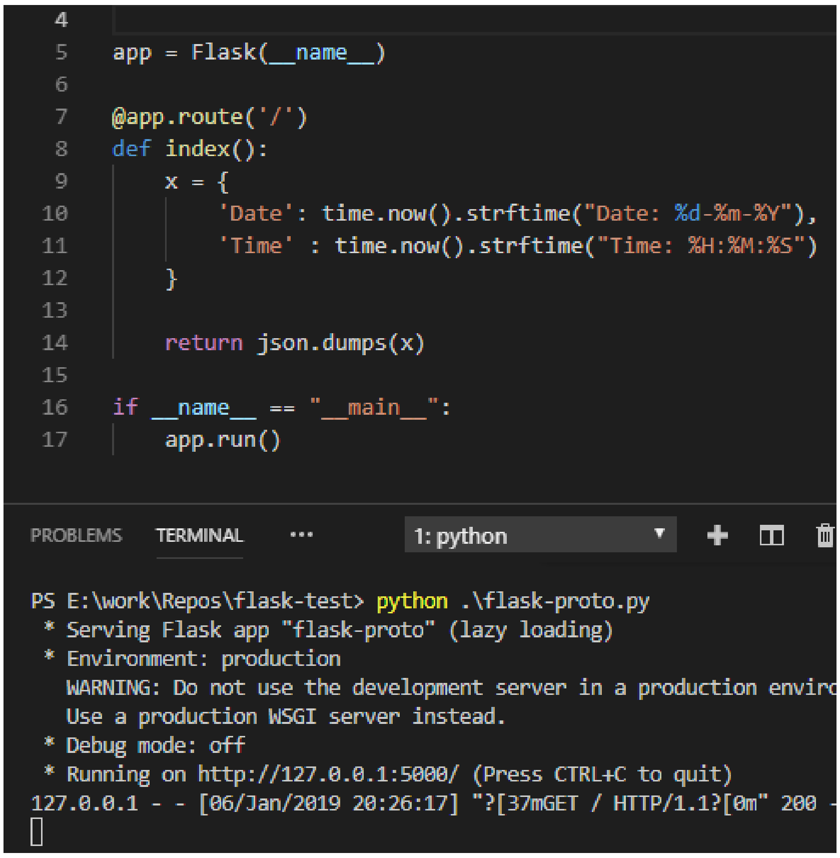
[37] "Parallel model hypothesis." [Online]. Available: https://github.com/keras-team/keras/issues/3287#issuecomment-235565515

[38] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, “Densely connected convolutional networks,” 2016. [Online]. Available: https://arxiv.org/pdf/1608.06993.pdf

# Appendix A

# Code Snippets

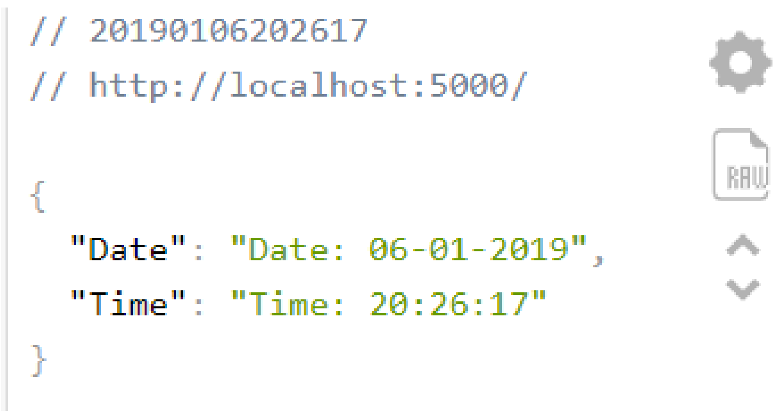
## A.1 Flask Proof-of-Concept Endpoint



```python

app = Flask(__name__)

@app.route('/')
def index():
    x = {
        'Date': time.now().strftime("Date: %d-%m-%Y"),
        'Time' : time.now().strftime("Time: %H:%M:%S")
    }

    return json.dumps(x)

if __name__ == "__main__":
    app.run()
```

```
PS E:\work\Repos\flask-test> python .\flask-proto.py
 * Serving Flask app "flask-proto" (lazy loading)
 * Environment: production
   WARNING: Do not use the development server in a production enviro
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [06/Jan/2019 20:26:17] "?[37mGET / HTTP/1.1?[0m" 200 -
```

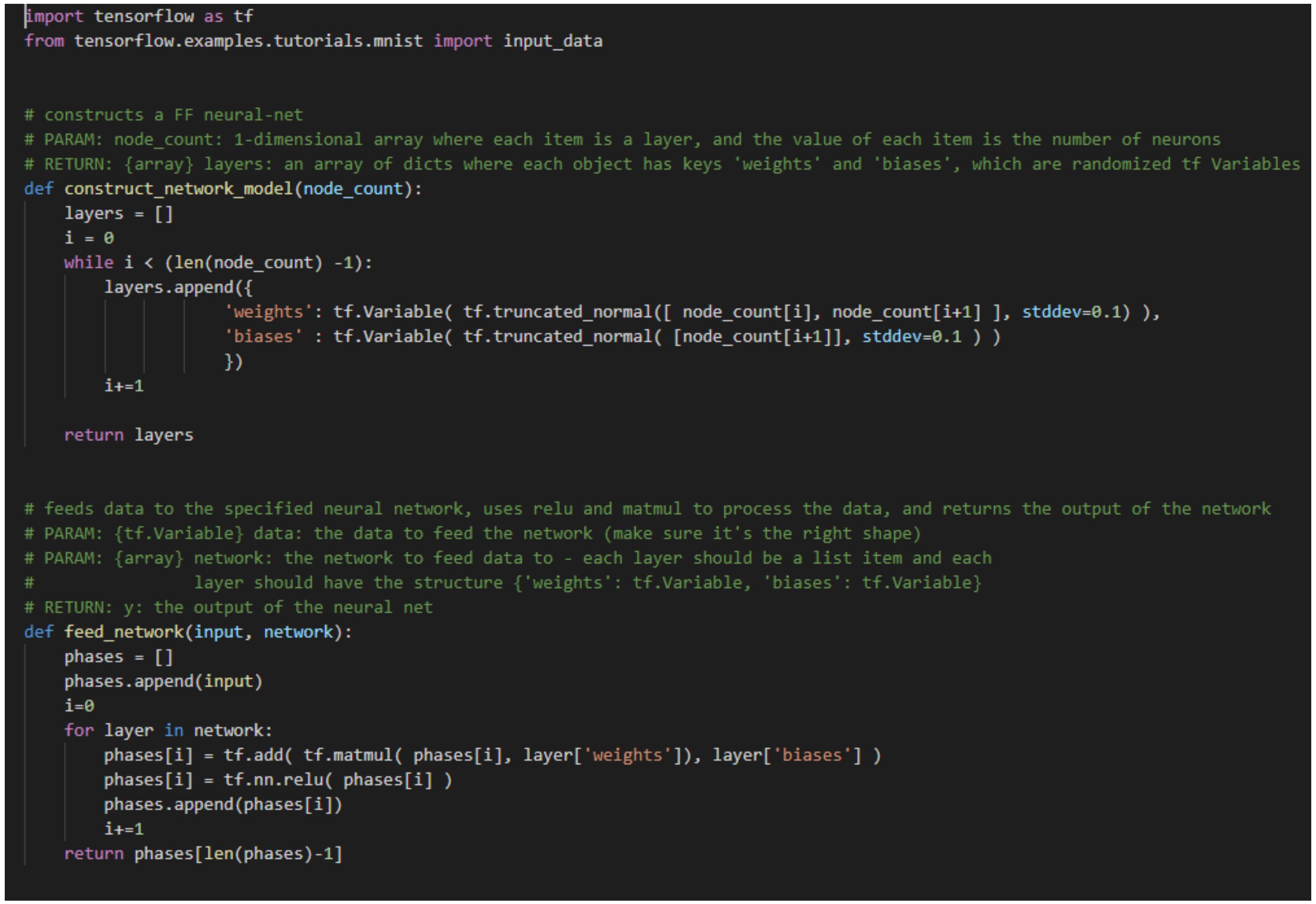
Figure A.1: A screenshot of a simple Flask route that listens for requests on port 5000 and returns the date and time in JSON format.



```
// 20190106202617
// http://localhost:5000/

{
  "Date": "Date: 06-01-2019",
  "Time": "Time: 20:26:17"
}
```

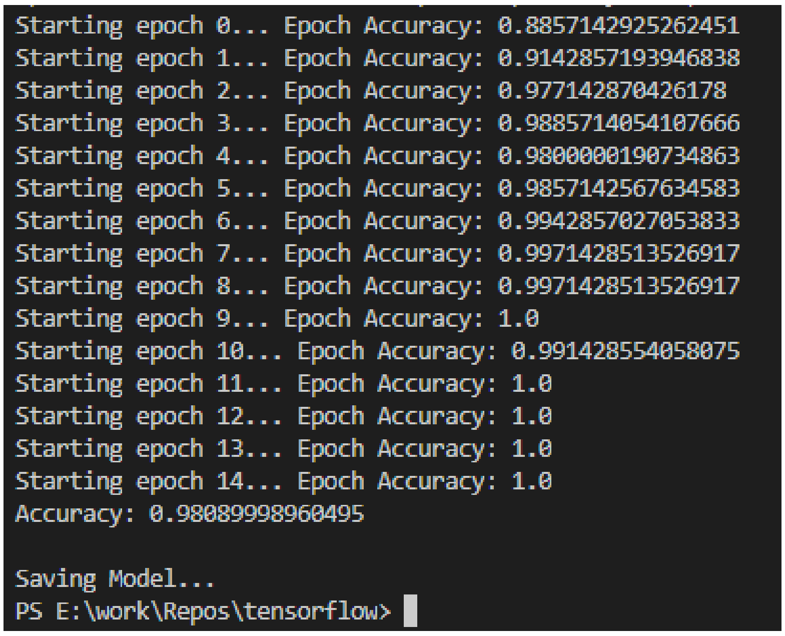
FIGURE A.2: A screenshot of the response data sent from the code shown in .

## A.2 Prior work on classification ANN's



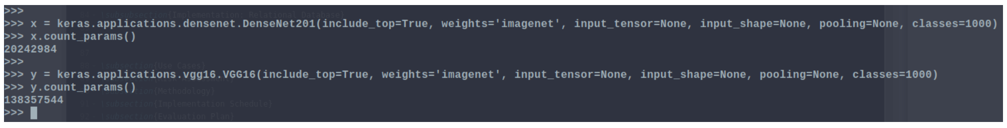
```python
import tensorflow as tf
from tensorflow.examples.tutorials.mnist import input_data


# constructs a FF neural-net
# PARAM: node_count: 1-dimensional array where each item is a layer, and the value of each item is the number of neurons
# RETURN: {array} layers: an array of dicts where each object has keys 'weights' and 'biases', which are randomized tf Variables
def construct_network_model(node_count):
    layers = []
    i = 0
    while i < (len(node_count) -1):
        layers.append({
                'weights': tf.Variable( tf.truncated_normal([ node_count[i], node_count[i+1] ], stddev=0.1) ),
                'biases' : tf.Variable( tf.truncated_normal( [node_count[i+1]], stddev=0.1 ) )
                })
        i+=1

    return layers


# feeds data to the specified neural network, uses relu and matmul to process the data, and returns the output of the network
# PARAM: {tf.Variable} data: the data to feed the network (make sure it's the right shape)
# PARAM: {array} network: the network to feed data to - each layer should be a list item and each
#                 layer should have the structure {'weights': tf.Variable, 'biases': tf.Variable}
# RETURN: y: the output of the neural net
def feed_network(input, network):
    phases = []
    phases.append(input)
    i=0
    for layer in network:
        phases[i] = tf.add( tf.matmul( phases[i], layer['weights']), layer['biases'] )
        phases[i] = tf.nn.relu( phases[i] )
        phases.append(phases[i])
        i+=1
    return phases[len(phases)-1]
```

FIGURE A.3: Construction of a feed-forward neural network. Each layer is represented by a Python dictionary with keys "weights" and "biases" which, through matrix multiplication and addition, represent connections between neurons. Input is fed into the first layer and travels through the network.



```
Starting epoch 0... Epoch Accuracy: 0.8857142925262451
Starting epoch 1... Epoch Accuracy: 0.9142857193946838
Starting epoch 2... Epoch Accuracy: 0.977142870426178
Starting epoch 3... Epoch Accuracy: 0.9885714054107666
Starting epoch 4... Epoch Accuracy: 0.9800000190734863
Starting epoch 5... Epoch Accuracy: 0.9857142567634583
Starting epoch 6... Epoch Accuracy: 0.9942857027053833
Starting epoch 7... Epoch Accuracy: 0.9971428513526917
Starting epoch 8... Epoch Accuracy: 0.9971428513526917
Starting epoch 9... Epoch Accuracy: 1.0
Starting epoch 10... Epoch Accuracy: 0.991428554058075
Starting epoch 11... Epoch Accuracy: 1.0
Starting epoch 12... Epoch Accuracy: 1.0
Starting epoch 13... Epoch Accuracy: 1.0
Starting epoch 14... Epoch Accuracy: 1.0
Accuracy: 0.98089998960495

Saving Model...
PS E:\work\Repos\tensorflow>
```

FIGURE A.4: Console output of the training script that continually evaluates the effectiveness of the network on the training data and tweaking it between batches of training images. Once training is complete, the model is saved, which is equivalent to storing a checkpoint in the training cycle.



```
>>>
>>> x = keras.applications.densenet.DenseNet201(include_top=True, weights='imagenet', input_tensor=None, input_shape=None, pooling=None, classes=1000)
>>> x.count_params()
20242984
>>>
>>> y = keras.applications.vgg16.VGG16(include_top=True, weights='imagenet', input_tensor=None, input_shape=None, pooling=None, classes=1000)
>>> y.count_params()
138357544
>>>
```

FIGURE A.5: Python output depicting the number of parameters present in the full DenseNet model (top half) compared to the VGG16 (bottom half)

# Appendix B

# Wireframe Models

## B.1 Web-App Wireframes

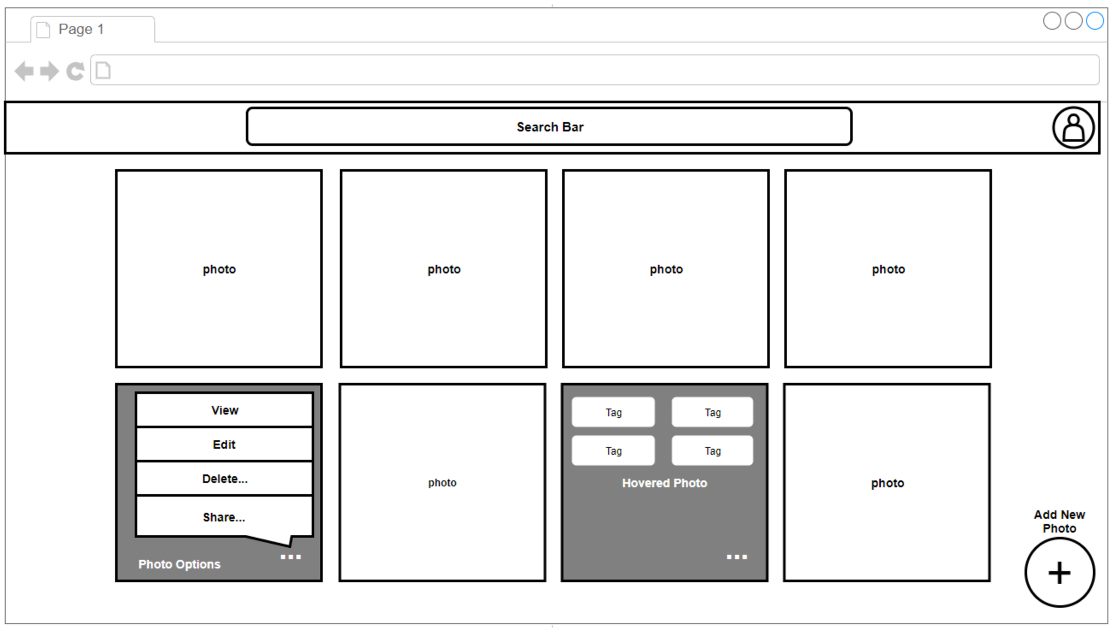


FIGURE B.1: A simple wireframe of the desired look of the main overview page for public users of the application.

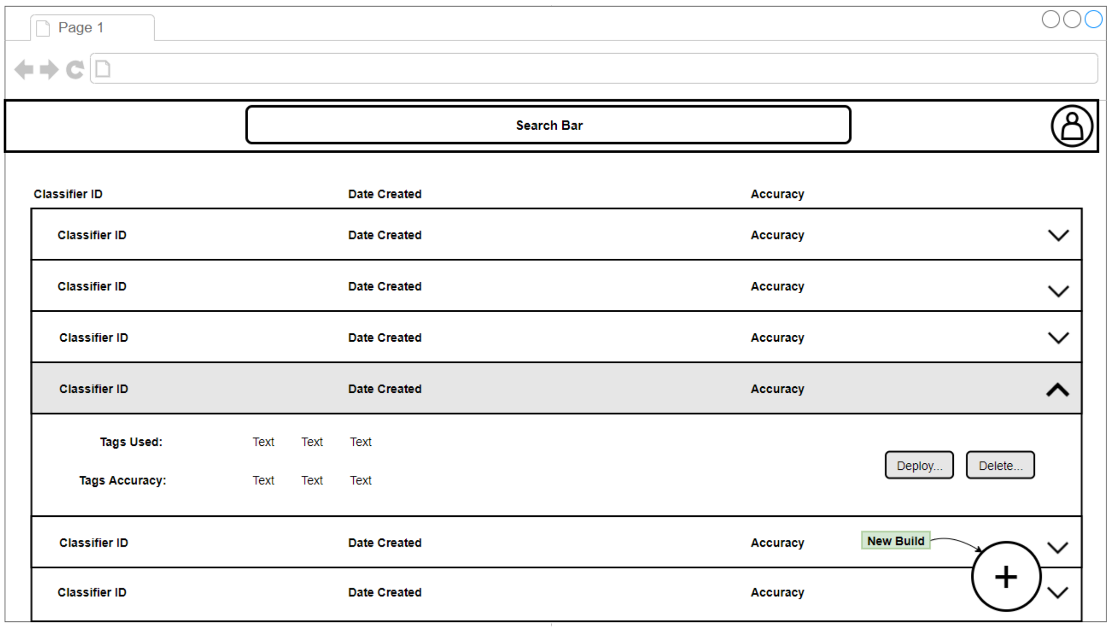


FIGURE B.2: A wireframe of the main admin-page after an admin logs in.