Flower Species Recognition on a Smartphone

Saahil Shihaz

BSc Computer Science
The University of Bath

May 2022

Abstract

Abstract goes here.

Contents

List of Figures

Acknowledgements

I like to acknowledge ...

1 Introduction

This section will outline the overall plan for this dissertation, starting with an in-depth look at the problem and a brief look at the domain.

1.1 Problem Description

The technological era that we live in has introduced many ground breaking achievements that constantly push the barrier of what is possible as well as introduce many new challenges that require complex solutions. One such challenge is big data processing, specifically, recognising patterns in data and drawing conclusions. Unfortunately machines don't have the ability to understand data the way that humans do and humans don't have the processing capability of modern machines. Due to obvious ethical and biological barriers, we cannot make humans fill the role of computers that compute data on a large scale, therefore, we must explore the alternative, making computers as smart as humans. This is where machine learning steps in, with which we have made great advancements in. What this project will focus on in particular, is granting the advanced capabilities of machine learning to lower end hardware.

This project aims to investigate the application of machine learning techniques to recognize images of flowers species on mobile devices. I will look at implementing standard machine learning algorithms that are effective in image classification as well as alternative deep learning techniques which are more effective in carrying out the same task. It will be interesting to compare both types of implementations in terms of accuracy, speed and performance and then transferring them to a mobile hardware environment which is traditionally weaker than standard machines such as desktops and laptops. Ultimately, we want to understand the best way in making a mobile software solution that can make use of machine learning methods and still maintain a seamless user experience.

Mobile devices have the advantage of portability and flexibility compared to PCs at the expense of pure processing power, storage and battery life. Advancements in machine learning can boost the abilities of mobile devices by allowing them to make informed decisions to aid the user. Traditional algorithms can't make decisions like "What is this flower?" without being cumbersome and inaccurate, we need something that can make good decisions and evolve, similar to human thinking. Smart phones and tablets are packed with more advanced technology than they've ever had like high resolution camera, sensors, displays and mobile processors, each of these are resources that a well written machine learning algorithm can take advantage of, for example, in our case, a mobile phone that can provide high resolution photos of flowers. The more detailed data we can use to aid our machine learning process, the better.

Traditionally, mobile devices as well as similar devices with sensors do some light pre-processing of data, then they send it to the cloud which can handle actions that require intensive processing, this introduces some level of latency because of

the communication between device and the cloud (**olascoaga2021hardware**, p. 3). Latency, being a key issue, is important in some use cases such as autonomous vehicles, mobile gaming, activity tracking for vulnerable populations, etc (**olascoaga2021hardware**, pp. 3-4).

With some raw information, a (classical) machine learning process can identify features, these could be used by a classifier that can make predictions given a set of data it hasn't seen before (**lecun2015deep**). Features are sourced from the representation of an object, in turn the representation is defined by the data input. An example of a feature would be the presence or absence of thorns on the stem of a flower (**goodfellow2016deep**, p. 22). Traditional machine learning practices incorporated feature engineering that required designing custom algorithms for particular task which can be time consuming (**liu2020representation**). There is also difficulty in understanding what features should be extracted, for example, it may be hard to represent flower petal shapes properly from raw pixel values if there are shadows being cast on it (**goodfellow2016deep**, p. 23). Representation learning is a method that can fix such issues by providing mappings not from just the representation of data to the output, but from representation to representation (**goodfellow2016deep**, p. 24). There are however, still hurdles to overcome, these are described as "factors of variation" where external factors might affect the source data, such as, the age of a flower, which could affect the petal shape and the season which may affect a flower's appearance. Factors like this make it difficult to get representations in the first place (**goodfellow2016deep**, p. 24).

Deep learning is a part of machine learning that aims to overcome limitations of classical machine learning techniques by expanding upon representation learning. Deep learning can be split into two unique parts:

- **Distributed Representation:** These are used to represent objects within a more compact and dense manner, instead of having representations for each type of object, for example a collection of words in a sentence, we could store the frequency of each word like the bag-of-words problem (**liu2020representation**). This is a sparse representation and introduces problems with space and time complexity. Therefore distributed representations aim to tackle the sparsity problem as they are harder to model (**Brownlee2017**).
- **Deep Architecture:** The idea of layering to represent neurons in a human brain. You can imagine it as a map of nodes that takes an input, processes it through the different layers where at each step, a set of units calculate a weighted sum of their inputs from the previous layer and pass the result to the next layer until it gets to output units that generate a result. This is an example of a feedforward neural network (**lecun2015deep**).

Input into a deep learning algorithm starts at the visible layer, which contains our set of input pixels that we can directly observe. This data is passed into a network of hidden layers, each of these layers represent an abstract feature

that we can't normally observe by looking at the input data such as locations of edges and contours (**goodfellow2016deep**, p. 26).

By making use of TensorFlow (Lite) we can produce classification models using languages like Python, C++ or Java, then convert said models into small packages that an Android/IOS application can use to generate predictions based on an input. TensorFlow is developed by Google and provides an machine learning based suite of tools to design, test and deploy ML solutions. The Lite version that we will be using is designed specifically for mobile devices and IoT devices that may not have the support of powerful hardware. Using TensorFlow we can write models that use classical ML techniques as well as deep learning techniques like Convolutional Neural Networks (CNN) (**googleTF2**). There are examples that can be built specifically for flower classification within the API documentation which can serve as a starting point for the project.

1.2 Main Objectives

- Analyse existing mobile based image recognition software.
- Investigate the advantages and disadvantages of both classical ML and DL implementations and compare the two using analysis tools, this will be done on PC hardware.
- Design and implement an Android application that can recognize images using DL.
- Discuss the feasibility of DL on smartphones.
- Explore future improvements for the Android app.

2 Literature and Technology Review

Ground-breaking achievements in technology and specifically machine learning have given us the tools and capabilities to tackle the key problem of image recognition. What this project aims to demonstrate is the application of deep learning within a mobile application to recognise flowers. It will be interesting to see how deep learning performs on mobile hardware which is generally less powerful than desktop PCs and laptops. Additionally, we will get to explore what sort of optimisations need to take place in order to get a feasible mobile based solution. The limited computing resources that we have to work with when creating our solutions is what makes this project challenging. Mobile devices typically contain smaller mobile processors, limited storage and batteries. These restrictions are in place to make mobile phones more efficient and to ensure that they last longer when not connected to a power source. In addition to this, the subject of machine and deep learning is complex and many find the concepts challenging to understand. What this literature review aims to do is identify key sources of information to help breaking down the underlying subject and discuss the quality of the research available.

2.1 Mobile Machine Learning

The potential of smartphones has still not been fully realised. With advancements in machine learning we can take the capabilities of smartphones to the next level by leveraging the advanced hardware within them to carry out complex tasks. This section will identify the key milestones within smartphone technology and how it leads us to integrating machine learning to make full use of the hardware.

Figure 1: Samsung Galaxy S21 Ultra 5G: Boasting an impressive array of camera sensors on the back (**three2021**).

2.1.1 Evolution

Firstly, we want to look at the last 10 years of technological advancements within the smartphone space. With this information, we can hopefully gain some insight into the how much their capabilities have evolved. To do this, we will collect key aspects of specification data from the Samsung Galaxy flagship line of smartphones. Samsung currently hold the top spot in global market share at 20.8% as of Q3' 2021, this position is typically held by Apple or Samsung and can vary from a quarter to quarter basis (**odea2021**).

Phone (Year)	Processor	Storage (GB)	Memory (GB)	Cameras (MP)
S2 (2011)	Dual-core 1.2 GHz	32	1	8
S3 (2012)	Quad-core 1.4 GHz	64	1	8
S4 (2013)	Octa-Core (4x1.6 GHz, 4x1.2 GHz)	64	2	13
S5 (2014)	Quad-Core 2.5 GHz	32	2	16
S6 edge+ (2015)	Octa-Core (4x2.1 GHz, 4x 1.5 GHz)	64	4	16
S7 edge (2016)	Octa-Core (4x 2.3 GHz, 4x 1.6 GHz)	128	4	12
S8+ (2017)	Octa-Core (4x 2.35 GHz, 4x 1.9 GHz)	128	6	12
S9+ (2018)	Octa-Core (4x 2.8 GHz, 4x 1.7 GHz)	256	6	12/12
S10+ (2019)	Octa-Core (4x 2.84 GHz, 4x 1.78 GHz)	1024	12	12/12/16
S20 Ultra 5G (2020)	Octa-Core (1x 2.84 GHz, 3x 2.42 GHz, 4x 1.8 GHz)	512	16	0.3/12/48/108
S21 Ultra 5G (2021)	Octa-Core (1x 2.84 GHz, 3x 2.42 GHz, 4x 1.8 GHz)	512	16	10/10/12/108

Table 1: All data is sourced from **gsm**

We can see a clear increase in smartphone capability in multiple categories like the processor speed, core count, storage, memory and camera capabilities. There are of course may more different areas that are not listed like sensors, screen size and battery life which have also seen massive improvements over the last 10 years. We have only been seeing improvements in this space, therefore we can assume that we will continue to see improvements in the near future. What we must also consider is price and accessibility, flagship smartphones represent the top of the line offerings from each smartphone manufacturer and are of course priced as such. Low-end to mid-end smartphones are still more capable than their predecessors albeit their spec sheet may not be as impressive as high-end versions in the same generation. Therefore, we must ensure some level of scalability within our machine and deep learning processes. How can we make sure our processes can run efficiently on lower end hardware as well as high end hardware?

kulendran2014 (2014), highlights how improvements in smartphones have created a boom in the number of smartphone based application designed to aid surgeons and patients in multiple facets of the medical industry like plastic, orthopaedic and general surgery. They conduct an expansive review of different solutions and analyse how the evolution of smartphones got them to the point that make them extremely useful as a tool to aid us. Ultimately, what this project aims to do is provide a robust software solution to recognise flower species on a smartphone, but we cannot ignore the fact that smartphones have come a long way in the hardware and operating system space to allow us to even conceive of a system.

2.1.2 Where does this lead us to, today?

ML and AI has become such an important part of smartphones that manufacturers now have dedicated processors for ML and AI tasks. Google includes a Tenor Processing Unit (TPU) in their Pixel line of phones (**triggs2021**). Samsung, Qualcomm and Apple use their own solutions for machine learning processing by having their own bespoke processors. These processors are used to compute specific actions that require the decision making and accuracy capabilities of machine learning. Google Tensor in particular aids tasks such as speech recognition that is accurate but not taxing, therefore saving battery life. Tensor also applies to processing photographs and provides additional features to videos (**gupta2021**). With such a focus on smartphones, to the point that they get dedicated hardware for ML, we should be seeing a huge increase in applications that integrate ML in some way, as well as the entire process of designing and implementing such solutions being carried out more rapidly, as developers learn to leverage the hardware.

2.2 Computer Vision

Since we are working with analysing images, the area of computer vision plays a big part in our research. In order to identify flower species we must first discuss

techniques to analyse the incoming image data to make predictions using ML and DL.

2.2.1 History

SzeliskiRichard2011CV:A (2011) outlines significant occurrences in each decade starting from the 70s, thought to be the beginning of computer vision, all the way through to the 2000s. In the early 70s, researchers sought to emulate human intelligence in a machine by first solving the visual problem. It was hypothesised that if a computer could first recognize objects in the real world that it could then move onto the next step of using reasoning and problem solving at a high level. The first processes conducted to understand the 3D world were to extract edges to recognize 3D objects from 2D lines in an image.

The 80s were described to have a lot more focus on mathematical techniques for analysing scenes. Various algorithms and models were conceived as well as improvements in the contour and edge detection space. Researchers found that a lot of these algorithms could be thought of as "optimization problems" when they were described using the same mathematical framework.

We see more improvements in the field during the 90s including the production of 3D surfaces, tracking and image segmentation. However, what is probably more relevant to this project is statistical learning techniques that also started to appear during this decade. In 1991, we see a paper by **turk1991face** (1991) that described the concept of "eigenfaces". These are the product of converting images of faces into feature images. These feature images are essentially the training set. Recognition occurs "by projecting a new image into the sub-space spanned by the eigenfaces". The new face is then classified by comparing it's position relative to the known set of faces. Emphasis was placed on the limiting the scope of the allowed images, as such the system was trained and ready to accept profile straight-on images of the subject. In addition to that, they aimed to have the system compute a result in a reasonable time, which of course, is one of the goals of this project. The research hoped to improve on it's predecessors that used, at the time, traditional methods of recognising features such as eyes', noses and mouths and their relative position to each other. The work done with eigenfaces shows great similarities with the machine learning techniques we see today, by essentially creating feature vectors and comparing the distance of known vectors in the same space.

SzeliskiRichard2011CV:A (2011) continues with their insight into the 2000s where see the various improvements like more efficient algorithms and what finally dominates the latter half of the 2000s; applying machine learning techniques to computer vision to aid visual recognition research.

2.3 Machine Learning

This project will be using ML techniques to compare efficiency and accuracy to the more evolved deep learning. What we must first consider is how machine

learning works in the context of computer vision. **CamastraFrancescoMLfA** (2015) summarise this and broke down ML development around three primary research points:

- Task-Oriented Studies, improving performance of learning systems in a predetermined set of tasks.
- Cognitive Simulation, emulating the human brain and designing processes around the human thought process.
- Theoretical Analysis, "the theoretical investigation of possible learning methods and algorithms independently of application domain".

They also produce a taxonomy to represent the balance of two entities they describe: the "teacher" and the "learner". The teacher, being the programmer, the one that designs the learning process and the learner being the computer system. The idea of inference is also introduced where a system can derive knowledge from previous observations. The taxonomy breaks down the amount of work that both the "learner" and the "teacher" need to do into four categories: Rote Learning, Learning from instruction, Learning by analogy and Learning from examples.

What we are more interested in is learning from examples where the "learner" infers the most out of the other categories in the taxonomy. The idea of the "learning problem" is introduced where the system needs to find a "general rule that explains the data given only a sample of limited size". Learning techniques are broken down into four more categories: Supervised learning, Reinforcement learning, Unsupervised learning, Semi-supervised learning.

zhu2005semi (2005) highlights semi-supervised learning in their survey as the combination of supervised and unsupervised learning where we use both labelled and unlabelled data for training of the classifier. They point to the survey done by **seeger2000learning** (2000) in particular that provides more insight into the concept of semi-supervised learning. Their rationale for the concept in general was applying the ability for a system to make predictions based on knowledge it doesn't have. A supervised system has all labelled data to aid it's training therefore it's basis on making predictions is described as a "security belt" by Seeger. The model will basically make predictions within its limited scope, what we would call "overfitting" (**tom1995**). Unsupervised learning heavily relies on prior assumptions for their final result this is because it doesn't have a knowledge base to rely on. By using a balanced combination of both implementations we can "balance the impact of prior assumptions". Seeger also highlights the fact that labelling the data is a taxing process, fortunately for us, existing data sets already exist with labelled and unlabelled images for flower species which will be useful for training. Therefore, a semi-supervised approach is feasible for the ML approach of this project.

2.3.1 Feature Extraction

dishaa2021 (2021) provides an introductory guide to feature extraction. They describe feature extraction as one of the two ways to reduce dimensionality with the other being feature selection. Extraction produces new features which are described as "linear combination of the existing features". The process aims to use less features to encapsulate the same image information.

tian2013 (2013) conducts a review of image feature extraction techniques that are worth considering. They start with discussing extracting colour features such as histograms and colour "moments" from specific colour spaces such as RGB and HSV. The paper also compares different types of colour features, for example, histograms are simple to compute but are sensitive to noise. This feature will be important as flowers come in many different colours, but it cannot solely be relied on as different species can share similar colours. We can also extract information about the texture of an image, this is where we start thinking about analysing groups of pixels together. Texture in the context of images is a way to describe the perceived smoothness, roughness or bumpiness of an image though spatial variations in pixel intensity levels (**mathworks**). Lastly, the paper goes into depth about shape features and points to different sources that go into the subject with more depth, but to summarise, shape features are split into two broad categories of contour and region based. This is where the features are calculated from shape boundaries and image regions respectively. A simple example of shape feature is the circularity ratio where you measure how close a shape is to a circle by calculating the ratio of the area of a shape to the area of a circle with the same perimeter (**mingqiang2008survey**). Shape analysing will be very important in this project because flower shapes can differ greatly and can therefore serve as a way to easily differentiate between species.

2.3.2 Classification

brownlee2020 (2020) provides an easy-to-understand breakdown of classification within ML. They describe it as the process of assigning "a class label to example from the problem domain". In our case, that means classifying a flower as species A as opposed to B, C or D. They also go into detail about the different classification methods such as:

- Binary classification e.g. it's flower A or it's flower B.
- Multi-class classification, where we have more than two classes.
- Multi-label classification, this is where we have multiple predictions for classes based on a probability. This can be a path we could take if we wanted to produce multiple predictions for a flower species and then provide the likelihoods of each prediction to the user.

Next, we will look at classifiers, which help us carry out the classification stage. Fortunately, there is no shortage of types of classifiers in the ML space. **MohammedMohssen2017Ml:a** (2017) covers the most popular ones in good

detail such as Naïve Bayes, k-Nearest Neighbour and Support Vector Machines (SVM). Starting with Naïve Bayes, this is a supervised classifier based on probability that assumes all attributes are independent:

$$P(c|E) = \frac{P(E|c)P(C)}{P(E)} \tag{1}$$

Where E is classified as the class $C = +$ if and only if

$$BC(E) = \frac{P(C = +|E)}{P(C = -|E)} \geq 1$$

BC is our Bayesian classifier, + and – are two separate classes (**zhang2004optimality**).

zhang2004optimality (2004) states that Naïve Bayes is superb in classification and demonstrates the classifier based version of it in Equation 1. They explore the optimal conditions of Naïve Bayes and propose that it is most optimal when the dependencies among attributes cancel out since Naïve Bayes works best when each attribute is independent, this is relevant as we want to explore optimisation in the project to ensure the highest level of efficiency when making the flower predictions.

MohammedMohssen2017Ml:a (2017) states that K-Nearest Neighbours (KNN) is one of the "simplest" of all the ML algorithms. **rosebook2016** (2016) discusses how to implement an image classifier using KNN where we can convert an image into a set of feature vectors on a graph, any new points get classified based on the k number of nearest neighbouring points, there's no real learning in this process, just the calculation of where the nearest points are, based on (usually Euclidean) distance.

NobleWilliamS2006Wias (2006) describes SVM as a way to tackle binary classifications, which means in the context of flower classification it only really answers questions like "is it Flower A?". They state that you would need to train multiple "one-versus-all" classifiers. For this project, that may not be appropriate if we want to support a large number of flower species.

ML techniques are certainly not useless and can still provide results, however, the research in the space has evolved to a new level, aiming to improve upon these traditional ML techniques in all evaluation categories. This is where Deep Learning (DL) comes in.

2.4 Deep Learning

DL will of course be our alternative approach to recognizing flower species. Where ML is basically our baseline, our DL implementation should hopefully highlight how much better it is compared to the ML approach.

2.4.1 Neurons and Perceptrons

ScarpinoMatthew2018Tfd (2018) introduces the concept of Perceptrons in their book about using TensorFlow to implement DL. First, we must discuss neurons and how they relate to understanding the foundation of DL.

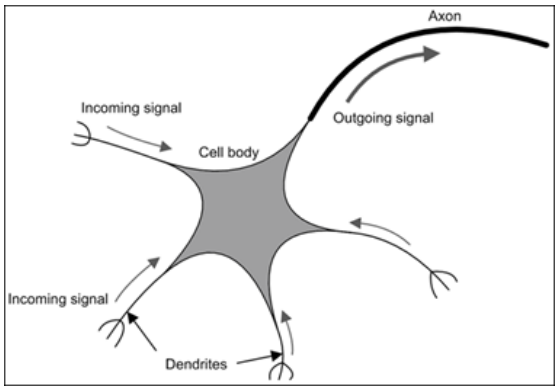

Figure 2: Simple diagram of a neuron (**ScarpinoMatthew2018Tfd**).

What they choose to highlight in particular are three points that describe a neurons functionality (and ultimately how it relates to perceptrons):

- A neuron receives one or more incoming signals and produces one outgoing signal.
- A neuron's output can serve as the input of another neuron.
- Every neuron has a threshold, and the neuron won't produce output until its electricity exceeds the threshold.

This page by **anonpercep** (n.d.) highlights a brief history of perceptrons, though it serves a starting point to learn more about the concept. Perceptrons were coined by Frank Rosenblatt in 1962 (**rosenblatt1961principles**), though his research is a bit outdated for our analysis, therefore here is a more modern version:

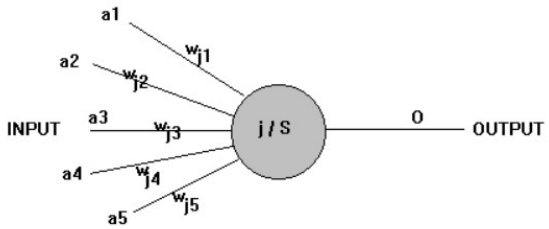

Figure 3: Diagram of a perceptron (**anonpercep**).

Each input on the left is weighted and the summed within the circle node. If the summation meets a certain threshold, the output will be 1, if it doesn't meet the threshold then a 0 is outputted (**ScarpinoMatthew2018Tfd**). Scarpino highlights some improvements to the model that was made including the weights that we discussed earlier as well as additional biases assigned with the incoming signals and an "activation function" that generates the output signal. Scarpino goes further by linking activation functions directly with in built TensorFlow functions that carry out the same task. Making it quite useful to understand the link between the TensorFlow API and the underlying DL context. Once we start linking perceptrons and arranging them into layers we get a neural network as shown here:

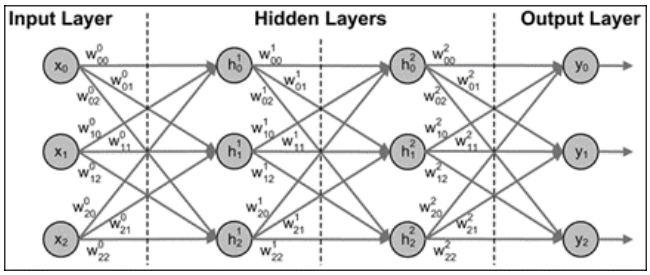

Figure 4: Diagram of a layered network of perceptrons (**ScarpinoMatthew2018Tfd**).

2.5 Convolutional Neural Networks

Bengio, Goodfellow and Courville (2015) and Scarpino (2018) go into detail about CNNs, Scarpino in particular is a useful source on how it works with image classification in TensorFlow. However, it's useful to have some sort of starting point for the subject. **saha2018** (2018) highlights the key features of a CNN and their purposes such as the individual layers: convolutional (kernel), pooling and classification (see Figure 5).

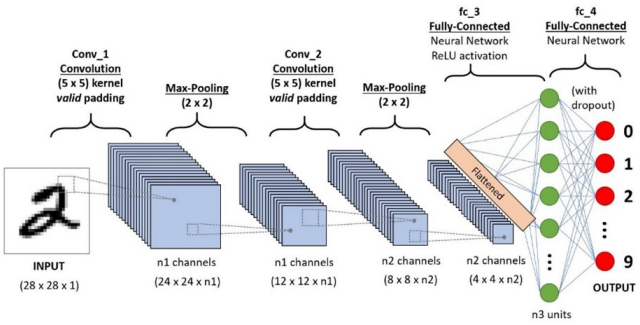

Figure 5: Example of a CNN process (**saha2018**).

They also highlight a feature of CNNs that make images easier to process, where it reduces the size of images "into a form that is easier to process, without losing features which are critical for getting a good prediction". This helps with our scalability approach when it comes to dataset sizes in particular. The ELI5 (Explain Like I'm 5) format is quite useful and allows us to highlight the key points of each layer (Saha, 2018):

- Convolution: Applying the kernel to extract high and low level features.
- Pooling: Reduces the spatial size of the output from the convolution. Decreases the "computational power" needed for data processing. Additionally, highlights features that are dominant.
- Classification: The pooling output is converted into column vectors and fed into a feed-forward neural network where the model is able to distinguish between features and classify them.

Saha (2018) also highlights that there are actually different implementations of CCNs, therefore they may not function in exactly the same format. Dive into Deep Learning (**diveintodeeplearning**) has a run down of multiple CNN types starting with LeNet-5 and more modern approaches like AlexNet, VGG, NiN, GoogLeNet, etc. We can also see how to implement them using TensorFlow which will prove useful when implementing our own solution for flowers.

goodfellow2016deep (2015) go into detail about Deep Learning from the concept of perceptrons to modern implementations. Something they talk about that is interesting is the increasing data set and model sizes over time, which is quite applicable to the project since we are working with modern mobile hardware. They discuss how the increasing capabilities of computer hardware have led to the development of larger models and that neural networks tend to double in size roughly every 2.4 years. They predict that the trend will continue further on in the future. What is also relevant is the dataset sizes. Storing datasets take up storage space, they state that a deep learning algorithm (as of 2015) is stated to perform at acceptable levels with "around 5,000 labelled examples

per category, and will match or exceed human performance when trained with a dataset containing at least 10,000,000 labelled examples. That is of course, extremely large and is most definitely going to take up a lot of storage. Therefore, the book does re-iterate the earlier point of making use of unlabelled data like with semi-supervised learning. **goodfellow2016deep** (2015) dig deep into the subject of DL and explain subjects from the applied maths to the modern practices of DL and the research in the field. This can prove useful in fully understanding the various processes in place including optimisations and ways to increase accuracy that we will need to consider when designing a DL model for the mobile application.

2.5.1 ML vs DL

DL is an obvious evolution from ML, but it is worth highlighting the key differences for clarity because ultimately this project will compare how ML and DL compete with each other. **kav2020** (2020) breaks down how DL is different from ML. They highlight that DL takes the initiative by automating feature extraction to lessen human intervention and that ML is more reliant on humans, where humans normally define the characteristics to look out for, as well as their priorities. DL is stated to "require more data points to improve its accuracy" compared to the ML counterpart.

8359287 (2018) goes into depth about the key differences when discussing approaches to ML and DL in the context of cybersecurity. However, the same reasoning can be applied in our situation. The key points they highlight are:

- Data dependencies: DL performs better with larger datasets like mentioned earlier as well as ML outperforming DL with smaller data sets.
- Hardware dependencies: DL requires a lot of matrix calculations and therefore a Graphical Processing Unit (GPU) can be used to optimise these processes. Note that mobile hardware do contain GPU hardware but they are not on the same scale as dedicated GPUs you find in PC hardware. Therefore, it will be interesting to see how DL fares against ML when we keep this hardware dependency in mind.
- Feature processing: Once again iterating on the point mentioned before, DL can extract features directly from the data and requires less human intervention.
- Execution time: DL algorithms take a lot longer to train compared to ML, this is dependant on the amount of data.
- Interpretability: Because of the complexity of DL it is hard to determine how a DL algorithm generated a result, whereas ML is more clearer.

I have summarised the key points, but they go into much more detail which could be helpful in the evaluation stage of comparing the two approaches of ML and DL.

2.6 Flower Classification

We will discuss further how flower classification is carried out including the use of ML and DL techniques, what features are extracted, the datasets that are used and the key challenges.

2.6.1 Existing Methods

Starting with what is known as the "Hello world" of ML, Iris flower classification serves as a simple and easy to understand project for developers to implement. The idea is to classify between three classes: Versicolor, Setosa and Virginica. There are many tutorials that can be followed online, this particular one by **DataFlairND** (n.d.) provides additional background information about ML as well as how it will apply to the Iris project which is useful for our understanding. The tutorial uses the features of sepal length/width and petal length/width to determine the class of a flower. By using those inputs, they use a SVM to predict the species of a flower with 96% accuracy. I mentioned earlier that it may not work well for a large number of species, but it certainly works well for a small number of classes.

Nilsback2008 (2008) demonstrate the effectiveness of a multiple kernel SVM on the oxford flowers 17 dataset. They manipulate the flower data to get key features such as the colour HSV values, the flower texture, shape and histogram of gradients (HOG) which "captures the more global spatial distribution of the flower" like the where the petals are arranged. They achieved an accuracy of around 88.3%. This is impressive considering the key challenges they highlight within flower classification. They state that flowers can share a lot of similarities between classes which can make it difficult to differentiate between species. Flowers are also "non-rigid objects" and therefore can appear in many different variations. Overall, they do a good job of explaining their reasoning for their dataset, citing the large variation of representations for each flower, and how they extract features from it, as well as how to build the classifier.

For deep learning, there is an extensive study that looks at using transfer learning, which is a technique of retraining TensorFlow models for different data sets. **Xia2017** (2017) use the Inception-v3 model to train a classifier for the Oxford-17 and Oxford-102 flower datasets. They go into detail about the steps that took place to carry out the transfer learning as well as how to reconfigure the last layer of the network to only have 17 and 102 outputs for each dataset as it defaults to 1000. They found that the model for the for Oxford-17 and Oxford-102 datasets produced 95% and 94% accuracy respectively. This is very impressive performance, and their breakdown will be helpful when it comes to my own implementation. The paper really outlines the simplicity and flexibility of the Google's TensorFlow, however, we will still need to investigate if these great results will translate to a mobile implementation as well.

2.6.2 Existing Apps

A large part of the project is to develop a fully functioning app that is developed to be more like a commercial product in addition to using deep learning techniques for the flower classification. This means developing features that will aid the user with using the app outside of main use case of recognising flowers. We will look at existing solutions that are already real products used by real people.

Pl@ntNet is a popular tool that has more than 10 million downloads on the Google Play Store alone (**googleplay**). It relies on volunteers to validate images and a search engine to identify them. **joly:hal-01182775** (2015) go into detail about the overall experience of the app as well as provide insight into how it works.

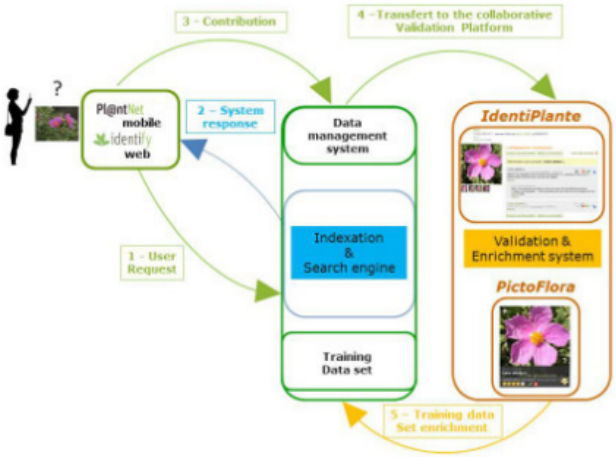

Figure 6: Diagram of Pl@ntnet user scenario (**joly:hal-01182775**).

Figure 6 shows clearly the type of system in place for the application. The user uses their device to query the search engine and get feedback, their image is also transferred the collaboration platform if they chose to. It can then be independently verified and added back to the training set. The search engine is then retrained on a nightly basis. The paper doesn't go into much more detail about the search engine itself apart from mentioning that progress in machine learning and computer vision should improve the performance of identification. Unfortunately, there aren't any new papers that provide a better look at the app, so it is hard to understand what changes have been made over the last several years as well as how what methods they use to build the search engine. There is however, a dataset now available for use that covers over a thousand plant species with over 300 thousand images (**camille˙garcin˙2021˙5645731**). This is of course out of the scope of the project as it doesn't strictly contain flowers, but it does go into detail about how to use the dataset as well as how

to load the data and build a model with it.

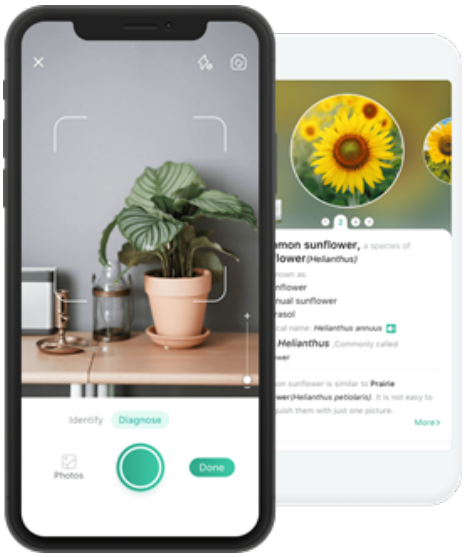

Figure 7: Screenshot of PictureThis app (**picturethis**).

PictureThis is another plant identification app (shown above) that doesn't go into detail about how it works but does note that it requires an internet connection to function properly. This suggests that it must communicate with a server in order to generate a prediction for images. The app is very streamlined and has an easy-to-use UI that also contains useful features such as how to care for the plant and important information about it. My approach will be different in the sense that any identification process will be carried out on the device, however, it is still important to consider alternative methods and how they perform, so that we can compare approaches.

2.7 Evaluation

One of the key points of the project is having to evaluate the ML and DL approach, what we haven't discussed yet however, is how do we go about doing this?

10.1145/1163593.1163596 (2006) identify a process named "k-fold cross validation" where the data set is "divided into k subsets". One of the subsets is used to test the classifier and the rest (k-1) subsets is used as the training set. They use three performance metrics to test their ML systems: accuracy, precision and recall. Accuracy being the percentage of correct decisions over the total number of test instances. Confusion matrices can help us with representing accuracy by providing a "summary of prediction results" where we use the count of accurate and inaccurate predictions per class to show which particular

classes the classifier may be struggling with (**Brownlee2020b**). Precision and recall are a bit more complex. Fortunately, **shung2018** (2018) demonstrates how these two differ to accuracy. Precision is the number of instances that are correctly determined over the total number of instances that are guessed, this is made up of correctly guessed instances as well as instances that are incorrectly guessed. Recall is the number of correctly predicted instances over the true number of instances in the class. In addition to these evaluation methods, **10.1145/1163593.1163596** (2006) outline measuring CPU and memory usage. Fortunately, TensorFlow (lite) contains benchmarking tools for us to measure: Initialization time, inference time of warmup state/steady state, memory usage during initialization time and overall memory usage (**googleTF**). I will also assess real world speed and accuracy by analysing the app's performance during development. The TensorFlow (lite) guide also contains tutorials on how to choose the best model for the task by comparing model size and accuracy of different models as well as the time it takes to make the prediction. The lite version is designed specifically for mobile and internet of things (IoT) hardware, so the additional tools will prove useful for the project later when we are at the evaluation stage.

Chockwanich (2019) use the same evaluation methods outlined when comparing different DL models implemented in TensorFlow. They also look at CPU usage percentages and processing time. They were able to make a clear conclusion of which model is better by evaluating all factors. A term called f1-score was also mentioned in their analysis. Shung (2018) also explains the relevancy of f1 score, it is essentially a way to determine a "balance between precision and recall". **kors2021** (2021) discusses the F1 score and it's purpose in providing a better accuracy statistic that accounts for "imabalanced data", this is when you don't have a good balance of data for each class and therefore the classifier makes inacurate predictions heavily skewed towards classes you have more data for. The F1 score is calculated by:

$$\frac{2 * Precision * Recall}{Precision + Recall}$$

2.8 Summary

The review has highlighted the progression ML and DL from the early concepts and how everything eventually fit together to form what we know today. The area will keep getting more exciting as we learn to optimise our current algorithms, come up with new ones and make use of advancing hardware capabilities. ML and DL is getting more and more accessible as manufacturers allow the use of their specialised hardware to developers who can make use of APIs built specifically for these tasks. Overall, we have built on the solid foundations of previous research and it's interesting to see how the field develops in the future. The project hopes to aid the field by investigating the ML and DL approaches in the mobile format as well as explore why we see certain results from the evaluation.

3 Investigation

The main body of this dissertation will be split into two sections: the first being an analysis of TensorFlow's deep learning python API and how it compares to traditional machine learning python libraries like scikit. The second being the design and development of the flower classifier app. We will start with the investigation portion of the project, walking through my initial predictions, methodology and findings. The idea is to approach this investigation from the perspective of a software developer that is analysing the best approach for method of flower classification to use in their product. This includes assessing the quality of the resources available and discussing the possible challenges.

3.1 Predictions

The main prediction is that implementing a convolutional neural network is more suitable than using classical machine learning methods for this task. Suitability will be judged based on the metrics described in section **??** Evaluation. Furthermore, the process of implementing both approaches and the challenges that were faced will be described. Additional predictions mainly align with what was found in subsection **??** when discussing the key differences between ML and DL.

3.2 Design of Experiments

In this section, the implementation of the ML and DL classifiers will be individually described. The results from each approach will then be compared to fully understand the advantages of DL over ML. Furthermore, the development process including the various challenges faced will be discussed.

3.2.1 Oxford Flowers 102 Dataset

This dataset consists of 102 different flower species that occur within the UK. Each class contains between 40 and 258 images each. The images are described to have large variations between scale, pose and lighting, even within the classes itself. First the dataset will be downloaded and managed by the TensorFlow Datasets module (TFDS) which will download the dataset to a generic directory and can directly manage the image and dataset data including filenames, class names and data splits. The data splits defined by TFDS consists of 6,149 images for the test set and 1,020 images for the train and validation sets each. This split is atypical as you would ideally have a larger number of images in the train set rather than the test set. The current split is thought to be a mistake on Google's side (**githubissue**). As a result, I will swap the train and test datasets and carry out the training process with the larger split which would be 75% of the dataset (**TFOX102**).

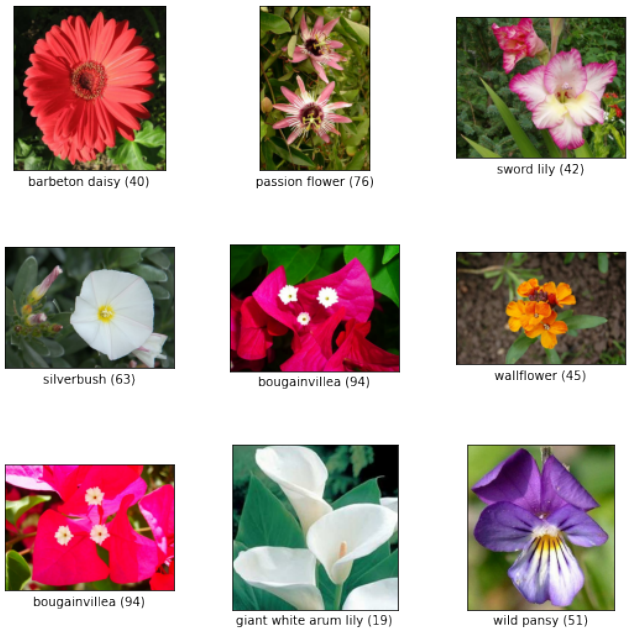

Figure 8: Example images from the dataset.

3.2.2 Classical Machine Learning

For this approach I decided to go with a Support Vector Machine (SVM) classifier that takes in features generated using "bag-of-words", HSV (Hue, Saturation, Value) colour values and histogram of gradient values. I went with SVM because good performance has already been achieved with this particular dataset before with **Nilsback2008** (2008) as stated in the literature review. A value of 98.5% accuracy has also been achieved be using a CNN to extract the features (**mete**). I will not use a CNN to extract the features as the purpose of this investigation is to evaluate both approaches separately instead of combining aspects from both. Bag of words will be used as it is simple to implement, but will also provide consistent information about key points found in the image, despite how the image is presented in terms of factors like rotation and scale (**mohan**). By using an additional library named cv2, we can use a K-Means trainer to produce feature clusters. The "words" will be produced by extracting key points using Scale Invariant Feature Transform (SIFT). HSV values are useful as they give us the relevant colour data as well as information about the luminance in the image (**chapelle1999support**). Histogram of gradients values provide information about the general shape of the object; this is useful in mapping the various shapes and sizes that flowers come in. The words will then be clustered by the trainer and make up the "vocabulary". The dataset images will be resized to have a height and width of 299 pixels to match the

input image conditions of the deep learning approach. All SVM parameters will be set to the default that is defined by the documentation.

3.2.3 Deep Learning

The CNN used in this approach will be Inception V3, a pre-trained network, specifically trained on the iNaturalist dataset, which contains 675,170 training and validation images from 5,089 categories (**paperswithcode**). This means that the model has been optimised for recognizing plants and animals which makes it the best candidate to be used for transfer learning to allow it to recognize flower species. It is also listed as the second-best model hosted on TensorFlow. Inception V4 exists and has better performance to V3 but there are no fine-tuneable V4 models available that will allow transfer learning to take place.

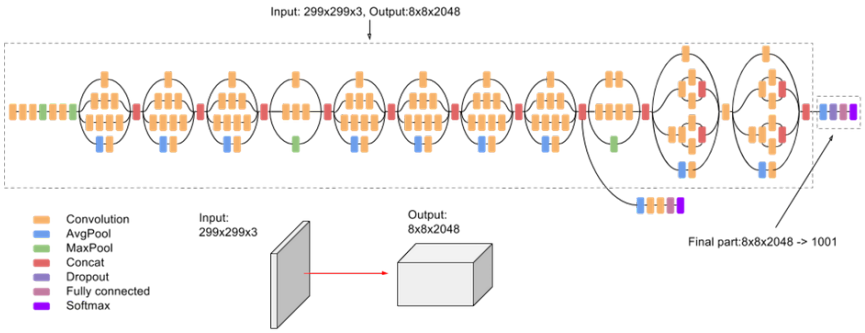

Figure 9: The Inception V3 model diagram (**GoogleCloud**).

Compared to the initial research of deep learning models presented in figure **??** within the literature review, figure **??** demonstrates how complex deep learning models can truly be. We discussed the terms convolution and pooling within section **??**, however there are additional layers here that were not discussed:

- Concatenate (Concat) takes a list of tensors and outputs a single combined tensor (**kerasconcat**).
- Fully connected layers map all inputs in the previous layer to every "activation unit" of the layer next to it (**singhsurya**).
- Softmax is where probabilities are assigned to each label based on the likelihood of the image belonging to that label. All probabilities add up to 1 (**googledevcnn**).
- Dropout mitigates overfitting of a dataset by randomly dropping out neurons on each pass of the network when training (**seb**).

The images will go through some data pre-processing such as resizing to have

a width and height of 299 as that is the requirement of the input tensor for Inception V3. They will also have to have their values rescaled from 0-255 to between 0-1. Random flips and crops will also be added as part of the pre-processing pipeline. Images will be batched into sets of 32 images, this means that 32 images will be trained per step per epoch. Hyper parameter tuning is also required to try and get the best performance possible. These are the hyper parameters will be adjusted and their supposed effect:

- **Optimiser** is used to improve the speed and performance the model by adjusting the parameters of the model during training to minimise loss and maximise accuracy. (**maithani**). The types of optimiser that will be tested are Adam, Stochastic Gradient Descent (SGD), AdaMax. There are a few more optimisers available that are not listed, as they are unsuitable for this dataset.
- **Learning rate** is the rate at which a model learns, a larger value means that the model learns faster at the expense of producing substandard weights for the model (**andreaperlato**). Values from 0.01-0.0001 will be tested, moving down a magnititude at each step.
- **Dropout** which was described earlier when discussing the Inception V3 model. Increasing this value will mean a larger percentage of nodes will get removed. Values within the range 0.2-0.4, in increments of 0.1 will be tested.

Using a TensorFlow module called TensorBoard, we can test the hyper parameter combinations efficiently and produce the metrics for each combination. TensorBoard allows the developer to view how the hyper parameters affected the results. In an effort to decrease overall training time and save time, I will conduct some preliminary testing to see which optimiser is more suitable with just baseline parameters. Once that is selected, I only need to test the different learning and dropout rates using a grid search. This is when every possible combination is tested. When testing with a large number of hyper parameters, one can use other methods like random search to decrease overall tuning time by randomly sampling hyper parameters from a range based on a statistical distribution, this means that more effective hyper parameters are tested to avoid spending time on hyper parameters that will not affect the overall performance that much (**sayak**).

3.2.4 Environment

Both approaches will be developed and ran on the same machine, a custom desktop PC that contains these main components:

- An AMD Ryzen 3600 4.2Ghz 6 Core/12 Thread CPU
- 16GB 3200Mhz DDR4 Memory

The PC ran on Ubuntu 20.04 LTS with the relevant python3 and TensorFlow

libraries needed to run the Jupyter Notebooks locally. VS Code was used as the Integrated Development Environment (IDE) with the Python and Jupyter extensions. It is possible to run model training on the GPU instead of the CPU, however, only Nvidia GPUs are directly supported. As a result of not having access to one, we will be mainly using the power of the CPU to carry out training.

3.2.5 Metrics

The key metrics I will analyse along with accuracy, precision, recall and F1 are:

- Loss, a measure of how bad the model's predictions are (**googletrainloss**). This metric only applies to the deep learning model as it is calculated during the training process as the model tries to minimise it.
- Area under the receiving operating characteristic curve (AOC). You get an ROC curve by plotting the true positive rate against the false positive rate. The area under this curve indicates how well the model is selecting the correct prediction against all other predictions (**googleroc**).

3.2.6 Results

Preliminary DL Findings

Another factor to consider when carrying out training is the number of epochs. An epoch is a full pass over the training set. Multiple passes are needed to minimise loss and fully train the model. Through preliminary testing of the model, I found that 5 epochs are more than sufficient as we reach the maximum validation accuracy shown in figure [x]. If we increase the number of epochs, we risk overfitting (**geeksforgeeks**). Optimisers were tested individually to determine how much they would affect the results under standard conditions. I found that SGD was unsuitable for this task, producing accuracy results at almost half of what Adam and AmaMax were producing. Overall, Adam produced the best results, albeit AdaMax was only a percentage point behind. Therefore, I decided to conduct any main hyper parameter tuning, purely using the Adam optimiser. All initial training is done to only the first epoch to reduce overall execution time.

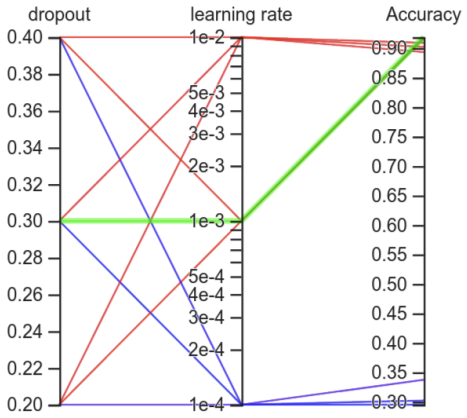

Figure 10: Graph generated in TensorBoard from HP training results.

Figure **??** above is a parallel co-ordinates view shown within TensorBoard that clearly show what accuracy results certain combinations of hyperparameters produce. Highlighted in green shows the joint first highest with a dropout of 0.3 and a learning rate of 0.001. A dropout of 0.2 with the same learning rate produces the same result. A learning rate of 1e-4 produces a significant reduction of first epoch performance, this is because we would need to increase the number of overall epochs to get optimal results. Ideally, we would test the number of epochs along with the other hyperparameters to produce fairer results, however, that would significantly increase training time with a grid search. If a developer has access to more specialised hardware, suited for machine learning tasks, this would not be too much of an issue.

Overall, the final model will use a learning rate of 0.001 and dropout of 0.3. This was decided after considering the results from the hyper parameter tuning as well as the proposed learning rate being the default one provided by the TensorFlow API.

Performance

Metric	SVM (%)	Inception V3 (%)
Accuracy	24.30	95.69
Loss	-	17.95
Precision	29.40	96.14
Recall	24.30	95.69
F1	26.60	95.91
ROC AUC	89.60	99.97

Table 2: Results output from the classifier after predicting against the testing dataset.

Precision, recall and ROC AUC values are weighted, meaning that they consider class balance as they calculate the metrics for each class and find the average weighted by the number of correct instances for each class (**scikitprec**). This was necessary to account for the slight imbalances we have with the number of images in each class. The deep learning approach clearly comes out on top with excellent results.

The Inception V3 test contains additional information about accuracy (figure **??**) and loss (Appendix **??**) for the training and validation datasets while the model is being trained. It tells us how the model improves per epoch.

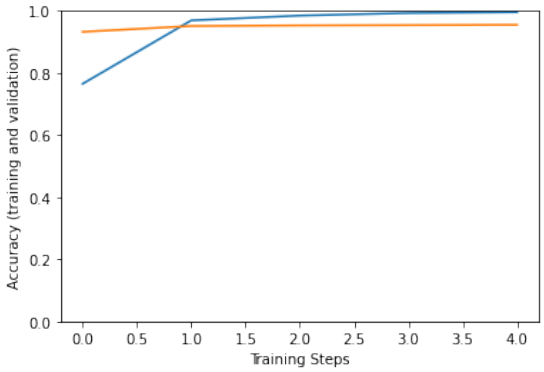

Figure 11: Graph of accuracy against the number of training steps.

The model needs to be converted to a TensorFlow lite file in order for it to be allowed to be used in the Android application. Once we convert it, we can reload the model and run inference on it to check if it is still effective. The result after doing that produces an accuracy of 100% if we just test it on the first batch of 32 images. Of course, his is not indicative of real-world performance, and we will need to do additional real-world performance profiling once the app is developed.

3.2.7 Analysis

Outcome

It is clear that the DL approach is vastly superior in classifying flowers than the SVM in all aspects. The Inception V3 approach has high accuracy meaning it can correctly identify the classes of most of the images in the test set. High precision indicates that a large portion of correct identification were genuinely correct. High recall demonstrates how well the classifier identified correct instances. You might notice that accuracy and recall are the same value for both classifiers, this is because they represent the same thing in non-binary classification. Recall is shown to be:

$$\frac{TP}{TP + FN} \tag{2}$$

Where TP is number of true positives and FP is the number of false negatives ().

Appendix

A Inception V3 Results Graphs

A.1 Loss

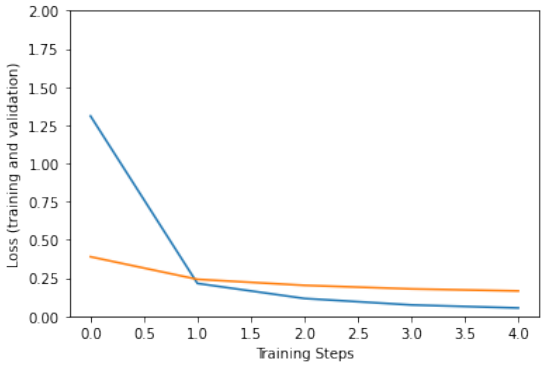

Figure 12: Graph of loss against the number of training steps.