Computing Project:
Flower finder and classifier

THE UNIVERSITY OF
MELBOURNE

School of Computing and Information Systems
COMP90055/COMP90019 Computing & Distributed Computing Project
Semester 1 – 2019

Group Members with student ID:

- Andi Hartarto Wardana Samijono : 979056
- Karan Katnani: 920130
- Min Xue : 897082

Supervisor: Professor Richard Sinnott
Total Credit Points: 25
Project Type: Software Development

Abstract

With advancements in technology, especially in the field of machine learning and AI, powerful technology such as object detection and classification, contribute to raising living standards in society as well as yield further benefits.

The central theme of our project is in the field of Computer Vision specifically Object Detection. The envisioned target of this project is to create a mobile app that can detect flowers from an unlabelled image containing flowers with reasonable speed and accuracy. To first obtain necessary data, we perform web scraping and retrieve data from the Open Images dataset. Then 5 models were employed in which 4 were variations of SSD using different meta-architectures and feature extractors such as Feature Pyramid Networks, Mobilenet, and Resnet and the last one is the YOLO v3 model.

Upon investigation of our results, we conclude that SSD FPN Mobilenet is the most suitable model in terms of latency, memory size, mean average precision(mAP@.50 IOU at 0.577) and recall for our mobile application. An IOS mobile app has been developed which employs 2 of our Mobilenet models to showcase differences in speed and performance. The app is able to take a picture, upload the image to a server from which detections were performed, and results were displayed in the app. The results would indicate the bounding boxes in the image, the number of flowers detected and the corresponding confidence values for each flower detected.

Keywords: Deep Learning, SSD, YOLO, Mobilenet, Resnet FPN, Tensorflow, Flowers detection, Mobile App

Declaration

We certify that:

- This thesis does not incorporate without acknowledgment any material previously submitted for a degree or diploma in any university, and that to the best of our knowledge and belief it does not contain any material previously published or written by another person where due reference is not made in the text.
- Where necessary we have received clearance for this research from the University's Ethics Committee and have submitted all required data to the Department.
- The thesis is 8615 words in length (excluding text in images, table, bibliographies, and appendices).

Acknowledgement

We extend our gratitude to Professor Richard Sinnott for granting us with the opportunity to work on this project. From time to time, we have received his advice on roadblocks or innovative ideas which we hadn't considered.

In addition, we would like to thank contributors working on Tensorflow especially in the field of Object Detection which was critical to the success of this project.

With gratitude, we thank Google for providing free GCP credits and allowing Open Image Dataset to be open source. GCP Cloud Machine Learning Engine had provided us with TPU processors able to train our models with relative ease and fast speed whereas the Open Images Dataset provided us with tens of thousands of flower images with bounding boxes.

Table Of Content

List of Figures

List of Tables

1 Introduction

1.1 Project Overview

With advancements in technology, especially in the field of machine learning and AI, powerful technology such as object detection and classification, contribute to raising living standards in society as well as yield further benefits.

The central theme of our project is in the field of Computer Vision. Computer Vision has long established its potential in bringing benefits to, such as enabling autonomous vehicle technology by allowing the vehicle to effectively “see” and promptly respond according to the situation. Furthermore, Computer Vision also has its applications in helping blind people to traverse in their everyday lives by detecting what is in front of the blind person and advising them through speech.

Our group has been tasked with the project of “Flower finder and classifier” as part of Melbourne University’s course COMP90055 Computing Project / COMP90019 Distributed Computing Project under Professor Richard Sinnott. The end goal of this project is to create a mobile app that can detect flowers from an unlabelled image containing flowers with reasonable speed and accuracy.

We endeavored to accomplish the project by making use of tools in industry such as Google Cloud Platform’s Cloud Machine Learning Engine to train our machine learning model with Tensor Processing Units(TPU) processors and employ multiple models in Tensorflow’s model zoo for object detection. By making use of state of the art technology, we can achieve the project’s target with reasonable speed and accuracy as illustrated in the section of “Project Details.”

In this report, we outline our approach to the project, from the theory aspect as well as the actual implementation done(Scraping, cleaning data, training several models, app architecture, and more). From our findings, pre-trained models in model zoo work effectively after being fed with lots of data and using Google’s CMLE TPU’s provided a fast, better alternative to save time for developers in training the model. We note that our model’s performance works well in detecting large objects but have the subpar performance on detecting small objects. Future directions in this research project involve data augmentation algorithms and oversampling of smaller objects to boost our model’s performance in detecting smaller objects.

A noteworthy development in the field to be on watch for is the release of Google Cloud Platform’s Cloud AutoML Vision Object Detection Beta which enables developers to train high-quality machine learning models in which Google handles selection of optimum machine

learning model, and developers need only to upload images and a .csv file outlining the bounding boxes.

1.2 Deep learning

Deep learning is a subset of machine learning which is supported by the layered structure of the ANN. Some complex architectures with various non-linear transformations are put together in order to model data.[4] Low-level features are combined to extract more abstract high-level representatives like attributes or categories, which are used for data classification or detection.[5]

Deep learning methods can automatically learn features of data without manual acquisition by human engineers compared to traditional machine learning methods. For a large dataset or with a complicated variation of data patterns, the capability of conventional machine learning techniques is limited when processing data with their raw form. Sometimes, it could be extremely time and labor consuming to extract and transform the original data into suitable representative features. Hence, those traditional techniques are gradually substituted by modern deep learning approaches which require little feature engineering and are more efficient in dealing with amounts of data. [6]

$$b_j = \Sigma_{i=1}^{m} a_{ij} \qquad (1)$$

$$s = \Sigma_{j=1}^{n} b_j x_j \qquad (2)$$

Deep learning also has the advantage of computing ability. A polynomial of a deep learning algorithm is given in equation (1) and (2) which is expressed as a sum-product form. In contrast, in traditional shallow learning where there is only a simple one-layer structure, the computation is shown as a product and sum in equation (3).

$$s = \Sigma_{i=1}^{n} \Sigma_{j=1}^{m} a_{ij} x_j \qquad (3)$$

Obviously, the computational cost of the double-layer structure is lesser than that of a traditional learning algorithm, which is only O(mn). And the advantage will become greater with the increase of m and n. [5]

Deep learning can provide decent results when working on unstructured data, like images or videos. The multi-layers structure provides it with superior ability in solving more complex AI problems. In some cases, the deep learning algorithm can work better than humans. For example, by using Deep Neural Network (DNN), the accuracy of image recognition that is generated by a computer could be even higher than that achieved by a human. In addition, deep learning methods are generalizable. That is to say, the same neural network approach could be utilized in many different data types and applications. In practice, the deep learning method is scalable when the data increase and the performance can be enhanced as well.

1.3 Convolutional Neural Network (CNN)

CNN Overview

The Convolutional Neural Network (CNN), also called as ConvNet, is one of the most popular models of deep learning techniques in image recognition. The probability of conventional deep learning methods is restricted in dealing with image data because of the fully connected layer structure. When dealing with an enormous number of parameters, the computing complexity could be extremely high and the problem of overfitting may occur. The CNN is superior compared to traditional deep learning approaches as it needs fewer parameters, which decreases the time of learning and also the amount of data provided for training the model. CNN is widely used in terms of image classification or speech recognition. The great power of CNN in feature extraction was proved by AlexNets and the technique is widely adopted in the industry. For example, VGG gained good performance results after using the small convolutional kernel to deepen the network. [7]

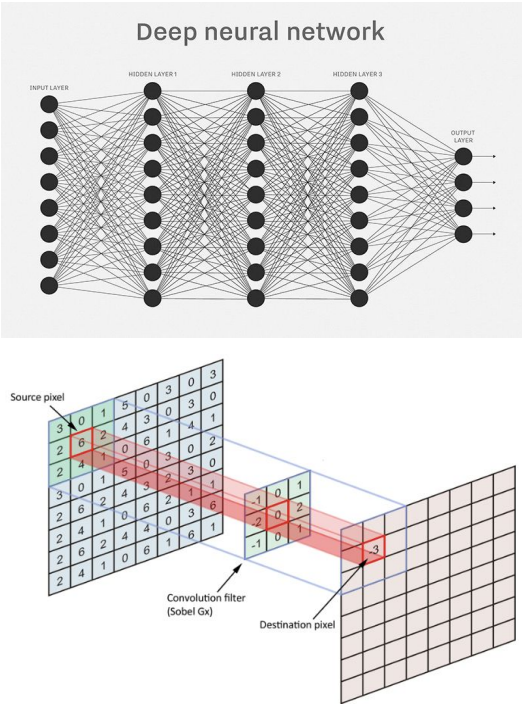


Figure 1. Structures of DNN and CNN [34]

CNN Architecture

In a nutshell, the CNN is composed of three major layers, the input layer, the feature extraction layer and the classification layer[8]:

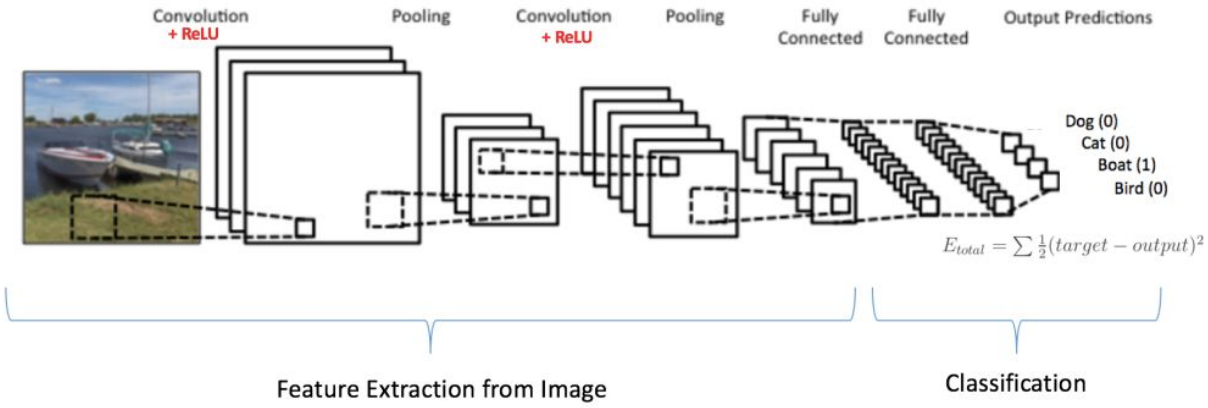


Figure 2. Components of the CNN [35]

1. Input layer

The first layer of CNN is the input layer, working as the entryway of the neural net to receive the original image data. An image is represented by an array of pixel matrix with dimensions of height, width, and depth. When refers to the depth, it is the same as the number of channels or the color specification of the image. For example, a colored image in RGB mode has a depth of 3. Whereas the depth of a grayscale image is 1.[9]

2. Feature extraction layer

Actually, there are multiple sub-layers of the feature extraction layer and each of them is implemented with different operations.

2.1 Convolutional layer

The convolutional layer (CONV) is directly linked to the input layer where arbitrary filtering operations are applied to the image. The primary task of the CONV is to detect features of images such as edges, curves, basic colors and more. The convolution process is realized by applying some filters, also named as kernels, on the input matrix and generates the feature map as the output. The operation mainly involves the math calculations over the image matrix and filters. A filter is actually a matrix with the same depth as the input matrix, which allows the math operation to be executed successfully. In an intuitive way, the filter works as a flashlight, scanning through the whole area of the input pixel matrix in sequence. The operation starts from the top left corner and shifts for several strides which are the number of pixels specified for sliding. Each pixel value in a particular receptive field is multiplied by values of the filter and all of those multiplications will be summed up to a single value. As the kernel keeps moving, a corresponding output matrix will be produced at last.

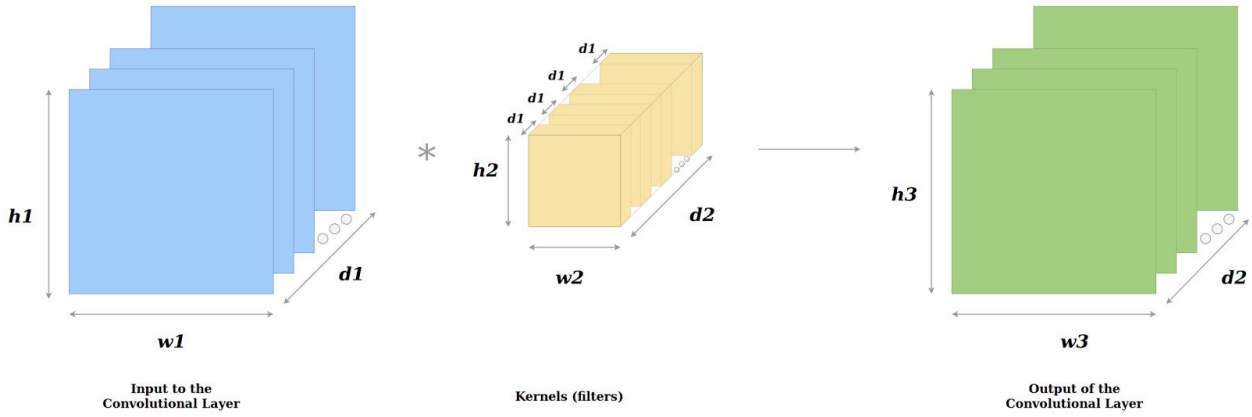


Figure 3. Convolutional layers [36]

2.2 Activation layer

It is not uncommon in CNN to use a nonlinear operation after convolving the input data. The ReLU, standing for the Rectified Linear Units, is one of the most popular functions used in the neural network. The feature map provided by the convolutional layer is applied with the activation function to remove the negative values in the matrix. The function is defined as f(x) = max (0, x), using only the positive values of its arguments. And this helps the model get more useful features to identify the object. Compared to other nonlinear functions like the sigmoid, the ReLU works better since it is more effective in training when performing on a complicated and large dataset.[10]

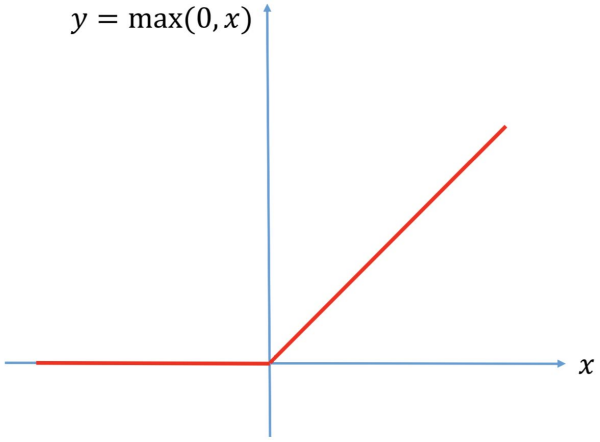


Figure 4. The ReLU function [37]

2.3 Pooling layer

The pooling layer is another important layer in CNN which performs the down-sampling operations. After getting the rectified activation map into the pooling layer, a window with the arbitrary size is applied on it to reduce the dimensionality. There are typically two types of operations, that is, the maximum and average pooling of which the max and average value are taken, respectively. The max pooling is simply choosing the maximum value for each 2*2 window. Whereas the average pooling calculates the average value of each group. [11] The main purpose of this layer is to reduce parameters and also

remain the spatial relationship invariant. The decrease of dimensions contributes to overcoming the overfitting problem and keeps the representative features at the same time, enabling the accuracy of detection.

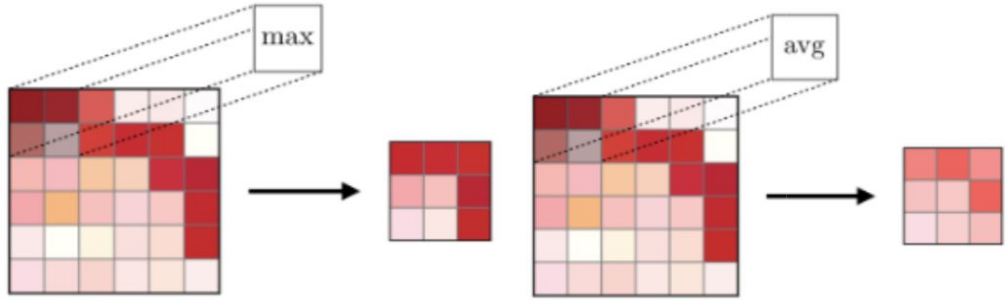


Figure 5. Comparison between max and average pooling [11]

3. Classification layer

In the classification layer, the previous data will forward to the fully connected layer and then being classified into certain classes with different possibilities.

3.1 Fully connected layer

The output of whether the convolutional layer or the final pooling layer is fed into the fully connected layer as input. The term fully connected in CNN refers to connecting each neuron in the last layer to that in the next layer. The output generated from the previous layer represents some high-level features of the image which will be used by the fully connected layer to look for correlated classes. The input volume is flattened first which transforms each value in the pooled feature map into a vector. Then after being attached to the fully connected layer, an N-dimensional vector is generated as the output. And N represents the number of classes defined from the very beginning. Finally, an activation function named softmax is used in the last layer, serving as a classifier to get the possibilities of the input belonging to a certain class.[12] The softmax function ensures that the sum of all probabilities is 1 and the confidence in each class could be seen intuitively based on the probability value.

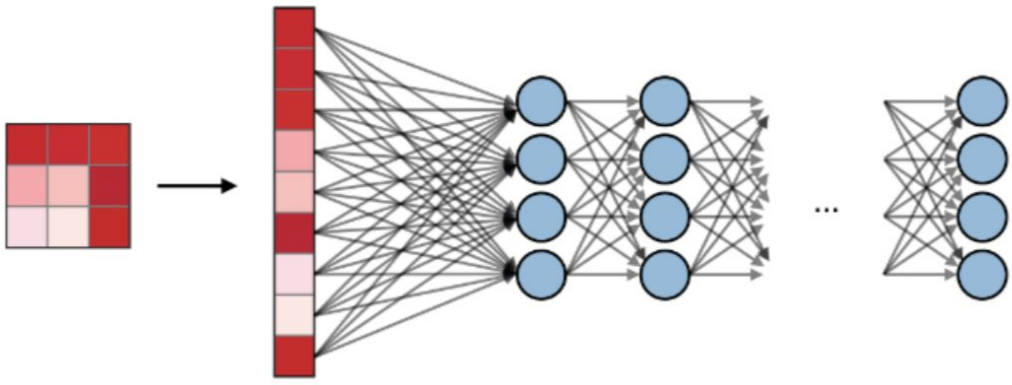


Figure 6. Structure of the fully connected layer [11]

1.4 Transfer learning

In the scenario of deep learning, we assume that a well-prepared dataset and enough computational power is provided for training the model. However, it is common that either of these factors is given in reality, making it infeasible to train a model from scratch.[13] Hence, the idea of transfer learning is introduced in order to lessen the requirement of data. Generally, the term transfer learning is defined as using the knowledge gained from the learning task in the source domain and apply them to solve the problem in the target domain. [14]

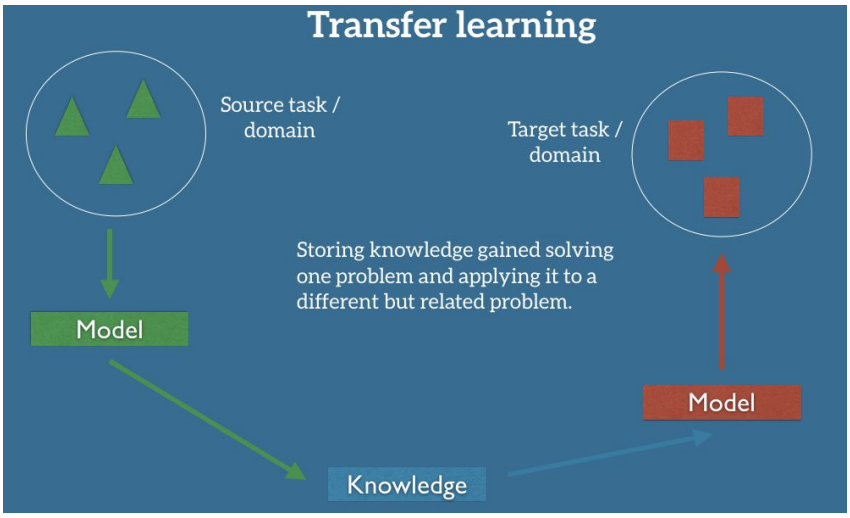


Figure 7. Transfer learning [38]

To be more specific, in the concept of deep learning, a model developed for one task can be taken as a prototype model for another task. The prototype model is called the pre-trained model which has been trained on a large dataset.[15] And the parameters or weights of the model could be adjusted based on your dataset. The whole process of transfer learning in solving object detection problem is implemented in several steps. Firstly, choosing a reasonable prototype model from a set of pre-trained models based on the target task. There are different types of models available from public literature, such as the MobileNet, VGG or Inception, selecting the most appropriate one accordingly. The next step is to consider the similarity between your dataset and the dataset used by the pre-trained model. A size-similarity matrix may be used as the basis.

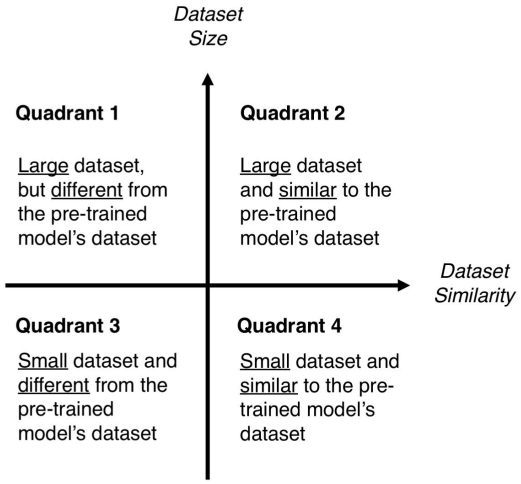


Figure 8. Size similarity matrix [39]

At last, it is optional to decide whether to repurpose the model or not based on your own needs. Several strategies are accessible to fine-tune the model. It is possible to train the entire pre-trained model and apply it on the training data. This kind of strategy requires a large dataset and enough computational power. Another choice is to adjust several layers while leaving others frozen. The number of changes made on the layers depends on the task. The third option is of an extreme situation where the entire convolutional base is remaining frozen and the model is used as a fixed feature extractor. This kind of method is only used when the dataset is small or the task is quite similar to that solved by the pre-trained model.

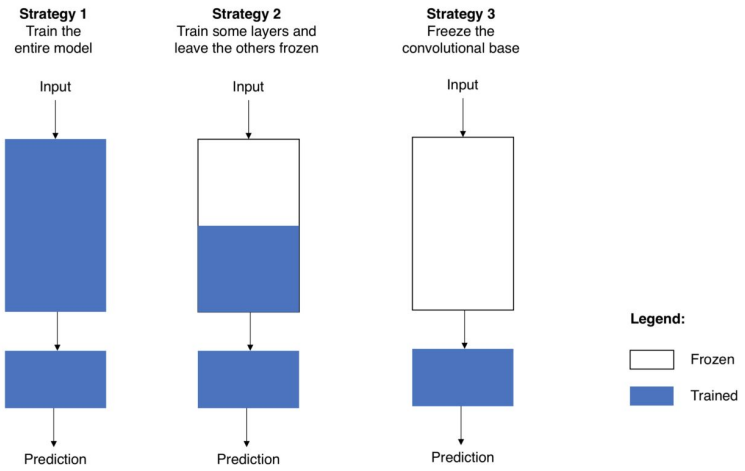


Figure 9. Strategies for tuning the model [39]

1.5 Related Work Section

By being able to count number of instances for each object classes which appear in everyday, natural objects, this would enable these counts to be used as supporting signals in helping vision tasks such as detection[51]. This is attributed to the idea that by having an estimate how many objects were present in the image, the aforementioned knowledge would then be useful as a feature to object detection models[51].

A similar task to the field of object counting is the Visual Question Answering(VQA) task in which the problem statement is to implement a system which is able to answer natural language questions pertaining to the image such as but not limited to “How many cars are in this picture?”, “Which person is wearing glasses?”. Questions with regards to count account for most of the questions in this task and end-to-end networks have been reported to be under-performing for these counting questions[54],[55]. The reason for end-to-end networks not performing as expected is identified to be because of the inherent setup to ensure cross-entropy classification loss is minimal for the correct answer to a question thereby not acknowledge ordinal structure to counting objects[51].

In the paper by Chattopadhyay et al[51], they have researched various methods in counting number of instances for each object classes appearing everyday and have combined the methods to count for natural images. The methods are discussed as below:

Counting by Detection: A straightforward idea would be that with absolute detection of objects, a perfect count would be realized. However, a challenge in this idea is that localizing objects might introduce inaccuracy as there could be unnecessary details in the object detection models such as pixel-accurate segmentation or problems in handling occlusions. Furthermore, object detection has also been proven to not work well on objects at smaller scales[51].

Counting by Glancing: By using Deep Convolutional Neural Networks, representation can be extracted and then applied to several problems for scene understanding such as fine-grained recognition, scene-classification, etc. The glancing models estimate a global count for the whole scene in a single forward pass. Conversely, detection models would increment the counts of each class sequentially for each detected object. The benefit of glance models would be that this model is specifically trained to count instead of detection models optimized for localization. [51]

Counting by Subitizing: Subitizing is a popular phenomenon recognized in Psychology in which students have been observed to directly map a perceptual signal to a numerical estimate such as determining number of pips on a face of a die despite not counting them. A problem in this method was that subitizing is difficult to achieve on high counts so a divide and conquer strategy was implemented. This means that images were divided to non-overlapping cells and accordingly subitized to obtain the total count. For images, context was further incorporated across the cells.[51]

2 Theory in Computer Vision

2.1 Faster R-CNN

Faster R-CNN Overview

Faster R-CNN outperforms R-CNN and Fast R-CNN in terms of the running time for training the model. It takes similar architecture as the Fast R-CNN except for the method used in the region proposals. Instead of using an external region proposal like selective search on the feature map, Faster R-CNN adapts an internal network, called Regional Proposal Network (RPN), which is more efficient and faster compared to that used in Fast R-CNN.[16]

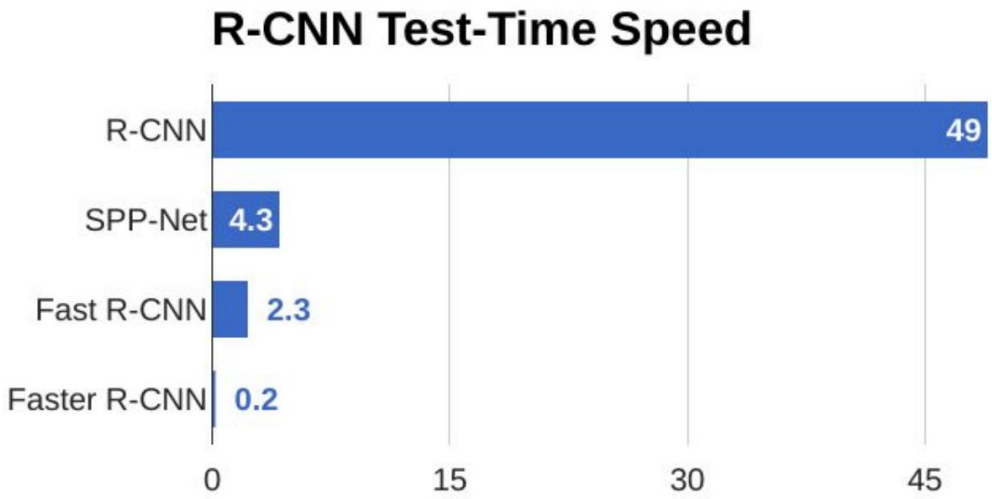


Figure 10. Comparison of speed [40]

Faster R-CNN Architecture

There are mainly two parts of networks in the Faster R-CNN, one is the RPN which is inserted into the model to generate proposals based on features and another is the network used for object detection. The same convolutional network is applied to both tasks. The same idea is taken like the Fast R-CNN, an image initially goes into the conv network as input and generated its corresponding feature map. Then, a spatial window is sliding over the whole feature map and k anchors are used for each location. A bunch of rectangular object proposals is provided as outputs, each of which has contained an object-ness score. Overall, the RPN is used for pre-checking the possible locations where the object may be located. And then passing the predicted locations to the next

layer for object class detection and getting back the bounding box.[17] The remaining work is realized by the detection network. The Region of Interest (RoI) pooling layer starts first to reshape the proposed regions. Then the pooled image is sent to two fully connected branches for image classification and bounding box generation.

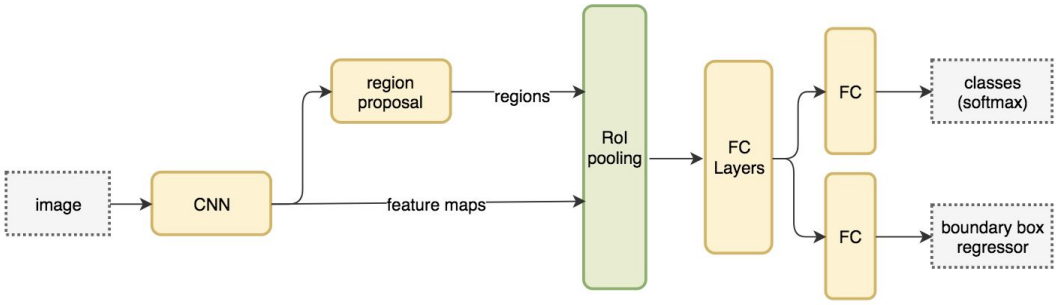


Figure 11. The flow of Faster R-CNN [41]

2.2 YOLO

YOLO Overview

YOLO is an object detection model in a really straightforward way. As the name implies, which is You Only Look Once, it only runs a single convolutional network over the image. Unlike other object detection methods, YOLO only has a simple architecture and makes the prediction based on the entire image.

YOLO Architecture

The workflow of YOLO is that when an image comes in, it is divided into an S*S grid of cells. Each grid cell is responsible for predicting the object that falls into the centre of the cell. Multiple bounding boxes are made within one cell and for each of them, both a probability and an offset value will be generated as outputs by the network. Only those bounding boxes with probability over the threshold can be chosen to detect the object presented on the image.

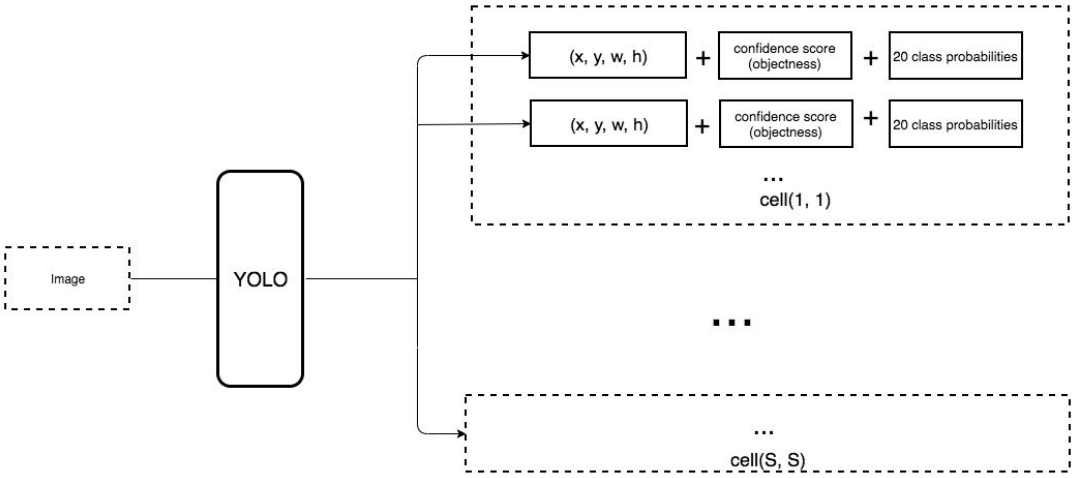


Figure 12. The flow of the YOLO [42]

YOLO has some prominent benefits that make it widely adapted in the industry. One of the biggest advantages is the processing speed which can reach to 45 frames per second (FPS), even faster than real time. Secondly, since YOLO makes a prediction based on the full image, it could not only encode the appearance of the object, but also the contextual information between classes. In addition, YOLO has the ability to understand generalized representations of objects. This property allows it to be accurate when training the network on unexpected images. However, YOLO has relatively low detection accuracy compared to other systems. And since the spatial constraints of the model, it struggles to detect small items or objects that are too close. In addition, it shows that YOLO is more easily to generate localization errors in contrast to Faster R-CNN. Because of all shortcomings of the original YOLO, also called as YOLO v1, an advanced version named YOLO v2 was created which optimized in the accuracy and detection speed. YOLO v2 has improved in various aspects such as adding batch normalization for better convergence, tuning the detection network to improve the resolution and more. All improvements are presented in Table 1.

	YOLO								YOLOv2
batch norm?		✓	✓	✓	✓	✓	✓	✓	✓
hi-res classifier?			✓	✓	✓	✓	✓	✓	✓
convolutional?				✓	✓	✓	✓	✓	✓
anchor boxes?				✓	✓				
new network?					✓	✓	✓	✓	✓
dimension priors?						✓	✓	✓	✓
location prediction?						✓	✓	✓	✓
passthrough?							✓	✓	✓
multi-scale?								✓	✓
hi-res detector?									✓
VOC2007 mAP	63.4	65.8	69.5	69.2	69.6	74.4	75.4	76.8	**78.6**

Table 1. YOLO to YOLOv2 Changes [43]

The YOLO v3 makes a further step based on the version 2. It gains incremental improvements in terms of bounding box prediction, class predictions, prediction scales and feature extractor.[18] The YOLO v3 works even better than the Faster RCNN for small objects detection. And the localization errors have been decreased as well. In addition, the performance of predicting the same object in different scales has been optimised.

2.3 Single Shot MultiBoxDetector(SSD)

SSD Overview

By 2016, State-of-the-art object detection systems were mainly built with extra modifications based upon a common foundation which was to hypothesize bounding boxes, resample pixels or features for each box and eventually apply a classifier with high accuracy. The works of such systems had demonstrated strong capabilities by dominating detection benchmarks such as PASCAL VOC, COCO, and ILSVRC in which most of the approaches were based on Faster R-CNN. Despite being accurate, those approaches require far too much computational capabilities from embedded systems which were deemed too slow for real-time applications even with high-end hardware[20]. Henceforth, various approaches have been proposed to build faster detectors by re-working each stage of the object detection pipeline albeit with noteworthy decline in accuracy by increasing the model's speed.

To overcome this issue of Single Shot MultiBoxDetector(SSD) model was proposed by Szegedy et al[20] which detects objects in images by making use of a single deep neural network. The SSD model performs by discretizing the output space of bounding boxes towards a set of default boxes with differing aspect ratios and scales for each feature map location. During the moment of prediction, the neural network will output scores if an object is detected to be available for each object category in the default boxes, and then generate box adjustments to better fit the object shape. Furthermore, the neural network in SSD will integrate predictions from various feature maps with unique resolutions to manage objects of different sizes naturally.

SSD Architecture

The architecture of SSD is relatively straightforward and simple relative to past methods which require object proposals simply because the SSD model will encapsulate all computations in a single network hence eliminating the need for proposal generation, and consequent pixel or feature resampling stages. Therefore, SSD is a model which is relatively easy to train and compatible with systems which require an object detection component to output results in a reasonably fast speed. Moreover, results upon experimentation on PASCAL, VOC, COCO and

ILSVRC datasets affirm that SSD has high competitive accuracy with greater speed as compared to methods which utilize additional object proposal steps, whilst provide an integrated framework towards both training and inference. In particular, for a 300 x 300 input, SSD achieves 74.3% mAP on the VOC2008 test performing at 59 FPS on a NVIDIA Titan X and for 515 x 512 input, the SSD model achieves 76.9% mAP thereby beating the then state-of-the-art Faster R-CNN model.

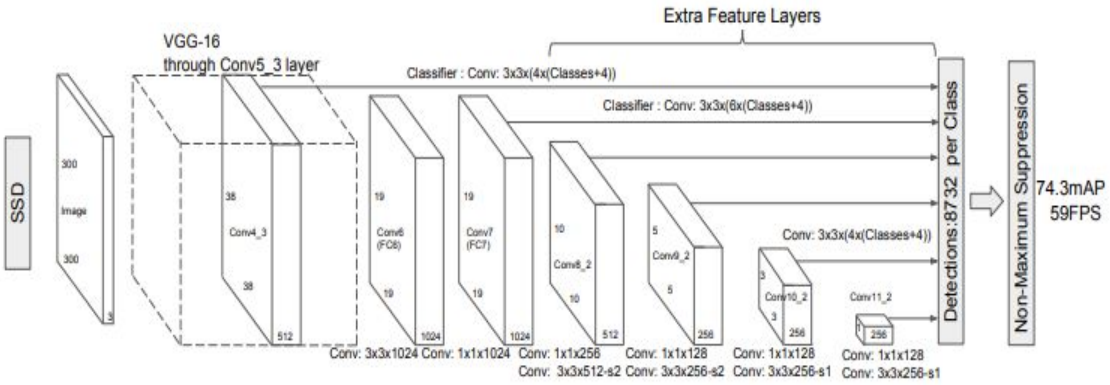


Figure 13. SSD model architecture [20]

For the architecture of SSD as illustrated in Figure 13, SSD is founded upon a feed-forward convolutional neural network which will generate a constant size collection of bounding boxes and outputs scores for the relevant object in the object class instance in the aforementioned boxes, which consequently a non-maximum suppression step ensues to bring in the result of the final detections. VGG-16 has been implemented as the base network is due to its considerable performance upon image classification however, in this architecture, the fully connected layers have been discarded. Instead, supplementary structures have been added to the network in which detections are produced with the following essential features:

Multi-scale feature maps for detection:
Convolutional feature layers have been appended to the truncated base network which reduce in size moving forward and enable predictions of detections at varying scales. To predict detections, each feature layer in the convolutional model is made different.

Convolutional predictors for detection:
For each additional feature layer (otherwise optionally existing feature layer in the base network) as illustrated in Figure 13 , a constant set of detection predictions can be made by employing a set of convolutional filters. By having a feature layer of size m x n along with p channels, the elementary element in which to predict parameters of a likely detection would be a 3 x 3 x p small kernel which outputs either a score of a category or a shape offset relevant to the default box coordinates. For each m x n location in which the kernel was applied, an output value is generated. Thereby the bounding box offset output value is determined relative to a default box position towards each feature map location which employs an intermediate fully connected layers as opposed to a convolutional filter.

Default boxes and aspect ratios:
With various feature maps at the top of the network, each cell of the feature maps have an associated default bounding boxes. The importance ot eh default boxes is such that it tiles the feature map in convolutional manner, which affixes the relevant cells to the position of each box. In each feature map cell, offsets are predicted corresponding to a class instance in each of those boxes. Algorithmically, for each box of k at a specific location, c class scores and 4 offsets relative to the original default box shape were computed. Therefore yielding a total of (c+ 4)k filters that are applied roughly at each location in the feature map, outputting (c + 4) kmn outputs with a m x n feature map. By having unique default box shapes in various feature maps, spaces of potential output box shapes can be then efficiently discretized.

2.4 Residual Neural Network (Resnet)

It is highly desired for applying more layers in the DNN, which enhances the ability in feature extraction and integration. However, as the network going deeper, a problem of vanishing gradient is exposed that decreases the training performance largely[45]. The ResNet aims to address the vanishing gradient issue by introducing “shortcut connection” that skips several layers for training efficiency[45].

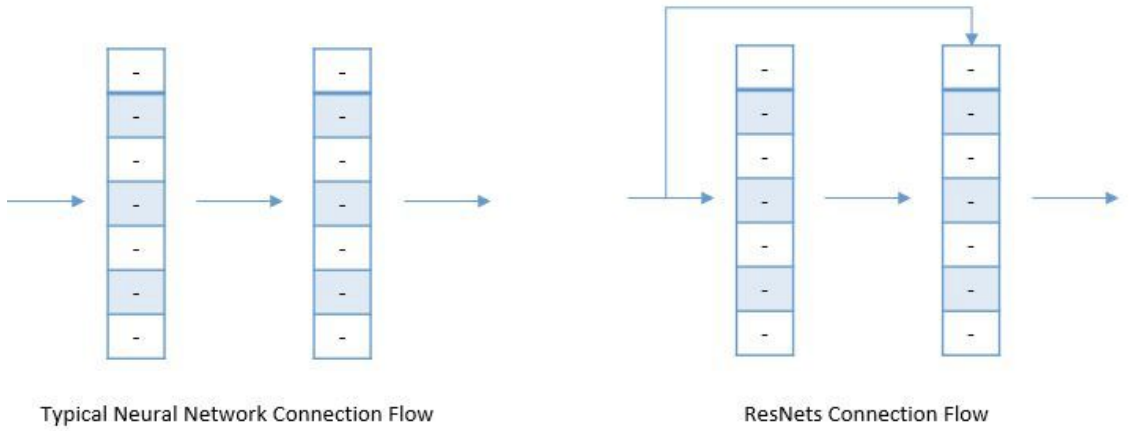


Figure 14. ResNets Vs Plain Neural Network [47]

And the shortcut connections are realized through a residual block which is shown in Figure 15. It allows the gradient signal to go back to previous layers so that make it possible for adding more layers in the network[46].

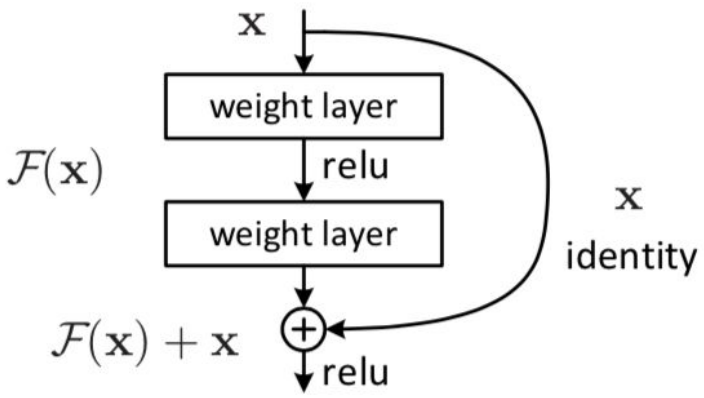


Figure 15. A residual block [45]

2.5 Feature Pyramid Network(FPN)

In the paper by Lin et al[27], deep convolutional networks were leveraged for its inherent properties of multi-scale, pyramidal hierarchy to achieve development of feature pyramids with slight extra costs. The development for high-level semantic feature maps resulted in a top-down architecture with lateral connections suitable for all scales.

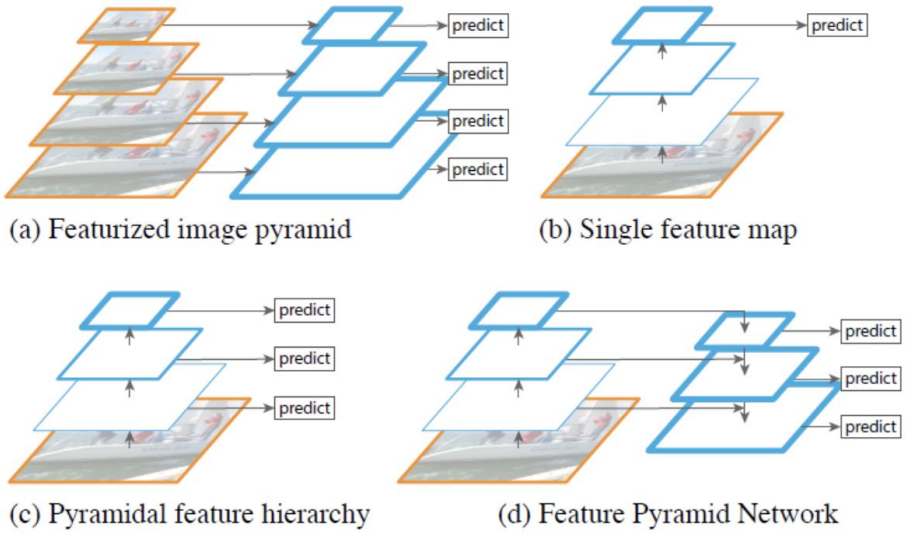


Figure 16. Different pyramidal architectures for detection [27]

In Figure 16, we observe different pyramidal architectures and their characteristics[27] are outlined as follows:

(a) Featurized image pyramid
An image pyramid is used to build a feature pyramid in which features are computed individually on each image scales. This approach consumes too much time and was a popular approach for hand-engineered features.

(b) Single feature map
This architecture is the standard for ConvNet solutions on a single input image which only performs predictions at the very end of the network. The benefit of this architecture is for faster detection speed.

(c) Pyramidal Feature Hierarchy

For each layer in the system, a prediction is made. This architecture benefits from reusing multi-scale feature maps from various, unique layers computed in the forward pass which therefore comes with zero cost. The disadvantage in this architecture is that it does not make use of the opportunity in making use of higher-resolution maps from the feature hierarchy. Thereby, consequentially neglecting detection of smaller objects.

(d) Feature Pyramid Network
The Feature Pyramid network takes into account low-resolution, semantically significant features with high resolution, semantically weak features from top-down pathway as well as lateral connections. The network performs as fast as (b) and (c) but comes with additional accuracy in it's performance.

Overall, FPN works by substituting image pyramid features with in-network feature pyramids and boasts no performance degradation in power, speed or even memory. The abilities of FPN is then demonstrated by achieving higher single-model results on the COCO detection benchmark with Faster R-CNN as the meta architecture[27].

2.6 Mobilenet

The MobileNet model is considered as an efficient architecture which can be easily implemented in mobile devices and embedded vision applications.[44] In contrast to a normal convolution, depth-wise separable convolutions are used in the MobileNet, which are composed of a depth-wise convolution and a pointwise convolution.[44] The realization of the factorization has dramatically decreased the computation complexity and make it a light-weight model. The comparison between the standard convolution and the depth-wise convolutions are presented in Figure 17. A filter is utilized in the depth-wise layer, where outputs are provided and fed into the pointwise convolution layer for combination. And batch normalization and ReLU functions are adapted in both layers.

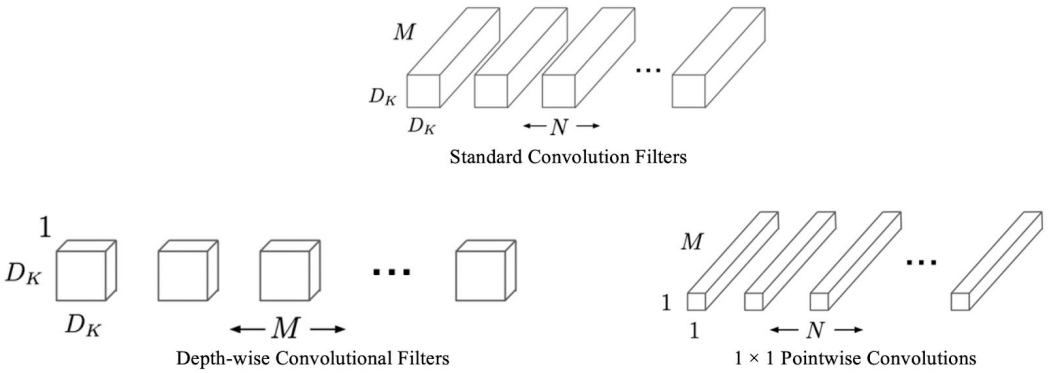


Figure 17. Comparison between standard and depth-wise convolutions [44]

3 Project details

3.1 Dataset

The dataset used for training and testing has been gathered from different sources over the internet. Many different datasets were explored, including playing with web crawlers. A massive kaggle dataset from a 2018 FGCVx Challenge was found and used as a starting point.

Since there are only few existing datasets available, we used Google Image and Flickr as additional sources to get more flower bouquets. Additionally, 3 of the classes were found with annotations in Google's Open Images v4 Dataset. A script was used to extract the required classes from the massive dataset. Several problems have to be solved during the crawling process. Although we are provided with lots of flowers data, only few of them contain our target flowers. Google Image offers abundant bouquets of flowers, but many of them do not have flowers that we are about to detect. In addition, since one type of flower may look different in many aspects, some mis-recognition may occur during the data crawling. Hence, some purifying work are required for crawled data. Flickr provides images with higher resolution which are beneficial for the model to learn more detailed features. But it is limited by the data volume. It is vital for us to make a trade-off between the data volume and data quality. The collected data for each kind of flower is shown in the table below.

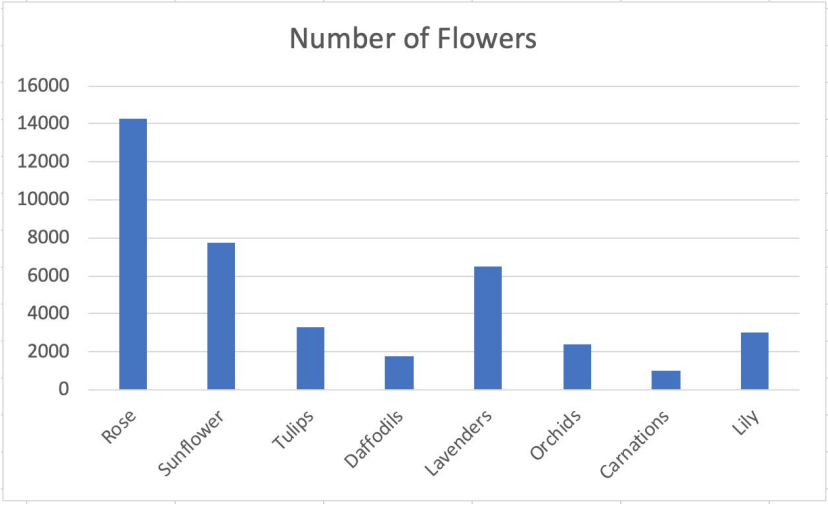


Figure 18. Distribution of Dataset

All of the flowers are contained in 11,270 image files, where most images contain more than one flower. This dataset was approximately split into a typical 70:30 ratio, where 70% of the dataset was used for training, and the other 30% for testing.

3.2 Data Augmentation

Data augmentation To make the model more robust to various input object sizes and shapes, each training image is randomly sampled by one of the following options:

In this project, we employ data augmentation techniques to further improve on our model's performance. Data augmentation technique enables our model to be more robust with respect to varying input object sizes and shapes. Each of our training image is subject to the following techniques.

Horizontal flip

Figure 19. Horizontal flip demonstration on Daffodils

Horizontal flip is a popular choice for data augmentation as it adds value by introducing images in a different perspective to the model. This technique is also fairly simple to employ in terms of programming or in the model configurations and is as illustrated in Figure 19.

SSD Random Crop

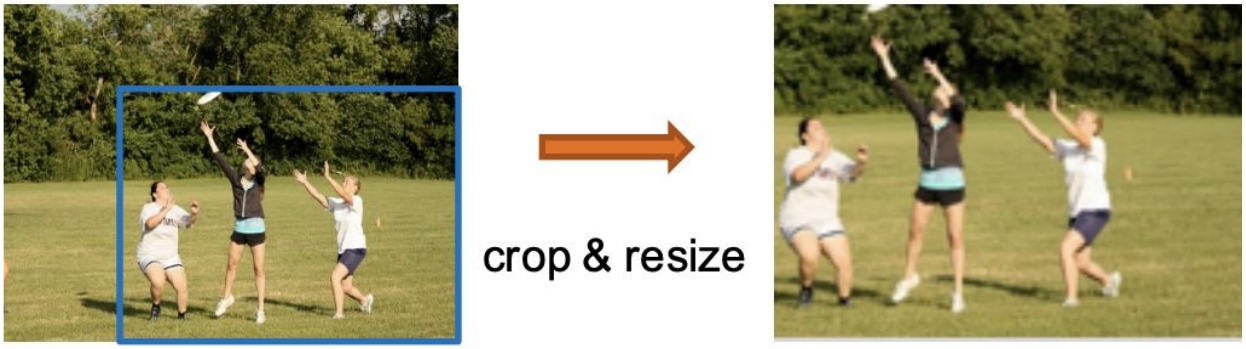


Figure 20. Random Crop example [20]

For this data augmentation technique introduced by Wei et al [20], a patch is cropped in the base image such that the minimum jaccard overlap with objects is 0.1, 0.3, 0.5, 0.7 or 0.9 respectively. A patch is then randomly sampled.

The range of the size for every sampled patch is between 0.1 and 1 with respect to the original image size. The aspect ratio of the image 0.5 and 2. If the center of the overlapped section of the ground truth box is within the sampled patch, then it would be saved. Following the sampling step, every sample patch is then made to a fixed size by resizing it and would then potentially be flipped horizontally with a chance of 0.5, followed with the application of photo-metric distortions described in [21][20]. This is illustrated in Figure 20.

3.3 Models in GCP

Tensorflow han open source Model-Zoo[25] which gives a collection of detections model already pre-trained on datasets such as COCO, Kitti, Open Images, AVA v2.1 and the Inaturalist Species Detection dataset. We leverage the following models in the Model-Zoo to perform transfer learning for our dataset on flowers curated as mentioned in Section 3.1. The selection was based on Speed, mAP on its respective dataset, and compatibility with TPU on Google Cloud.

SSD Mobilenet Depth Quantized

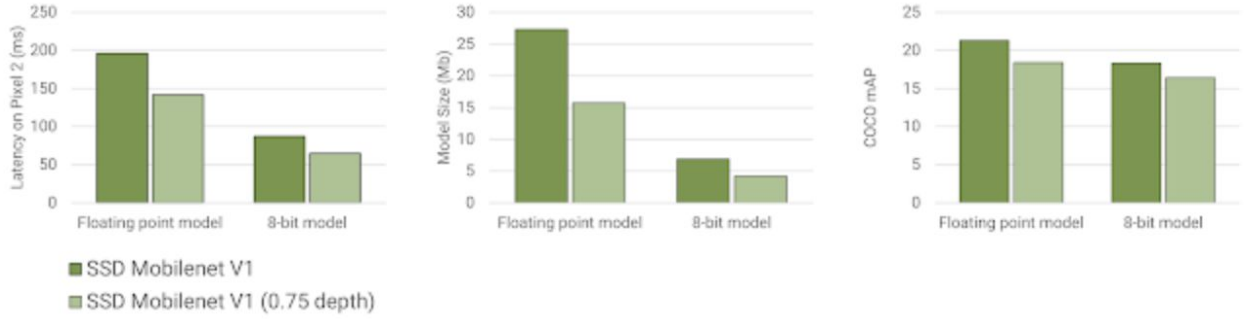


Figure 21. Results displaying Quantized model being smaller, require less latency and with minimal loss in detection results [24]

The SSD Mobilenet Depth Quantized model is configured to be a scalable approach to accomodate for large inference requests with low latency. Quantization is a technique in which weights and activations are compressed to an 8-bit point representation which therefore outputs a model with much smaller in size. A model with smaller size would then require lower latency for the purpose of object detection hence be able to work with high number of requests. As

illustrated in Figure 21, the results of the depth quantized model is similar to the Floating point model in terms of COCO mAP however has much smaller size in the Model and the results can be served with lower latency. The technique of model quantization is based upon the work by Jacob et al[22] and Krishnamoorthi[23].

SSD FPN Resnet_50 COCO and SSD FPN Resnet OID

A known trait of the SSD model is that there is a trade-off between it's accuracy and speed in which Faster RCNN is known to be the more accurate model. To improve SSD model's performance, typically more feature maps are implemented. However, Jeong et al[26] noted that addition of feature maps does not improve performance of the model per se. Instead, performance of the model was enhanced by replacing VGGNet in the SSD architecture with Resnet[31] which works by substituting the structure relevant to the classifier network. The network with Resnet is reported to have better results and faster speed than Faster_RCNN and RFCN[26].

Furthermore, Feature Pyramid Network (FPN) by Lin et al[27] has been implemented in this model which leverages a top-down architecture with lateral connection built to create high-level semantic feature maps across all scales. This feature of FPN, demonstrates clear improvements by being a better feature extractor across several applications[27]. Further details are as outlined in Section 2.5.

The 2 models of SSD FPN Resnet COCO and SSD FPN Resnet OID differ from each other as data used for the pre-training phase of the model are different. The datasets used were the COCO[28] or Open Images Dataset[29] accordingly.

The drawback in these modes however is that, the Resnet model will take up much more memory than typical SSD models. We endeavoured to explore these model so as to investigate models which can bring about high accuracy in object detection.

SSD FPN Mobilenet COCO

Similar to the models of SSD FPN Resnet COCO, the SSD FPN Mobilenet COCO leverages on FPN to perform better detections and was trained on the COCO dataset.

Furthermore, Mobilenet employs depth-wise separable convolutions in creating lightweight deep neural networks. Two hyper-parameters in MobileNet enable model building by selecting appropriately sized model for the application based upon problem constraints. These hyper-parameters are significant due to its importance in effectively determining the tradeoff between accuracy and latency. Effectiveness of Mobilenets were demonstrated through various

applications and use cases such as but not limited to: object detection, fine-grain classification, face attributes and large scale geo-localization[30].

We decided to investigate the results of this model in our dataset due to its reputation for being a smaller, faster model with decent accuracy.

3.4 YOLO Implementation

The official YOLO implementation has been designed to work on 'darknet', which is an open source neural network framework written in C and CUDA. Since we decided to use the tensorflow framework for training and deployment, different customs ports and implementations were considered for training a YOLOv3 detector on tensorflow.A list of considered options has been attached to Appendix B. The implementation selected utilises TensorFlow 1.11s GPU version, to augment, train and test the model[48]. This particular implementation was selected due to the descriptiveness of the author, along with version compatibility with

The model was pre-trained using all 80 classes of the COCO[28] dataset for transfer learning, and was then trained for 30 epochs on the data-set with a batch size of 32 for each step. The training was performed on a custom GCP Compute Engine Instance with 4 vCPUs, 16GB of Memory and an Nvidia Tesla P4 GPU.

3.5 Results

Precision

Models	mAP	mAP(large)	mAP(medium)	mAP(small)	mAP@.50IOU	mAP@.75IOU
SSD Mobilenet Depth Quantized	0.2431	0.3338	0.08479	0.01395	0.4534	0.2173
SSD FPN Resnet 50 COCO	0.3376	0.4221	0.1999	0.01883	0.5596	0.3457
SSD FPN Resnet 101 OID	0.3463	0.4336	0.204	0.02045	0.5814	0.3484
SSD FPN Mobilenet	0.3403	0.4272	0.1932	0.0249	0.5763	0.3401

COCO						
TF YOLOv3	0.021	-	-	-	0.0529	0.012

Table 2. Mean Average Precision results for different models implemented in the project

Notes:

- 'DetectionBoxes_Precision/mAP@.50IOU': mean average precision at 50% IOU
- 'DetectionBoxes_Precision/mAP@.75IOU': mean average precision at 75% IOU
- 'DetectionBoxes_Precision/mAP (small)': mean average precision for small objects (area < 32^2 pixels).
- 'DetectionBoxes_Precision/mAP (medium)': mean average precision for medium sized objects (32^2 pixels < area < 96^2 pixels).
- 'DetectionBoxes_Precision/mAP (large)': mean average precision for large objects (96^2 pixels < area < 10000^2 pixels).

Recall

Models	AR@1	AR@10	AR@100	AR@100(large)	AR@100(medium)	AR@100(small)
SSD Mobilenet Depth Quantized	0.1449	0.3184	0.4154	0.5102	0.264	0.03475
SSD FPN Resnet 50 COCO	0.169	0.401	0.4739	0.5662	0.3261	0.0787
SSD FPN Resnet 101 OID	0.166	0.4056	0.4814	0.5716	0.3414	0.08054
SSD FPN Mobilenet COCO	0.163	0.401	0.4738	0.5597	0.3394	0.1217

Table 3. Average Recall results for different models implemented in the project

Notes

- 'DetectionBoxes_Recall/AR@1': average recall with 1 detection.
- 'DetectionBoxes_Recall/AR@10': average recall with 10 detections.
- 'DetectionBoxes_Recall/AR@100': average recall with 100 detections.
- 'DetectionBoxes_Recall/AR@100 (small)': average recall for small objects with 100.
- 'DetectionBoxes_Recall/AR@100 (medium)': average recall for medium objects with 100.
- 'DetectionBoxes_Recall/AR@100 (large)': average recall for large objects with 100 detections.
- Precision-Recall graphs for YOLOv3 have been attached to Appendix C. Average Recall figures have not been generated for YOLOv3.

Output of training

Models	Size of Saved Model (MB)	Training Time
SSD Mobilenet Depth Quantized	13.8	3 hr 43 min
SSD FPN Resnet 50 COCO	128.2	6 hr 23 min
SSD FPN Resnet 101 OID	197.6	8 hr 47 min
SSD FPN Mobilenet COCO	44.9	4 hr 56 min
TF Yolov3	247.8	7 hr 37 min

Table 4. Training results for different models implemented in the project

Loss in Training

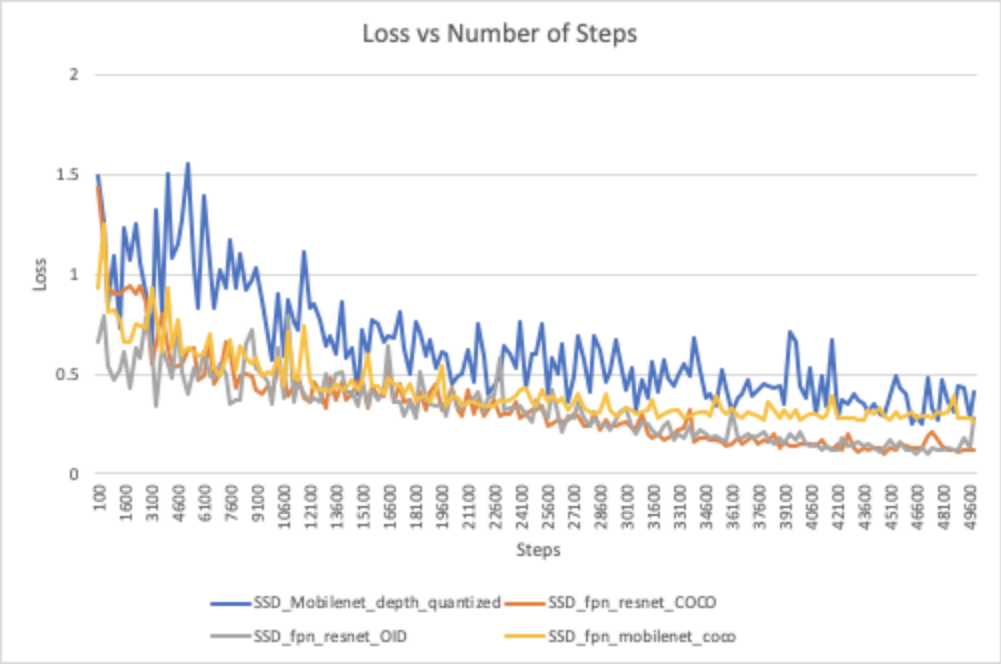


Figure 22: Loss vs Number of Steps for all SSD models

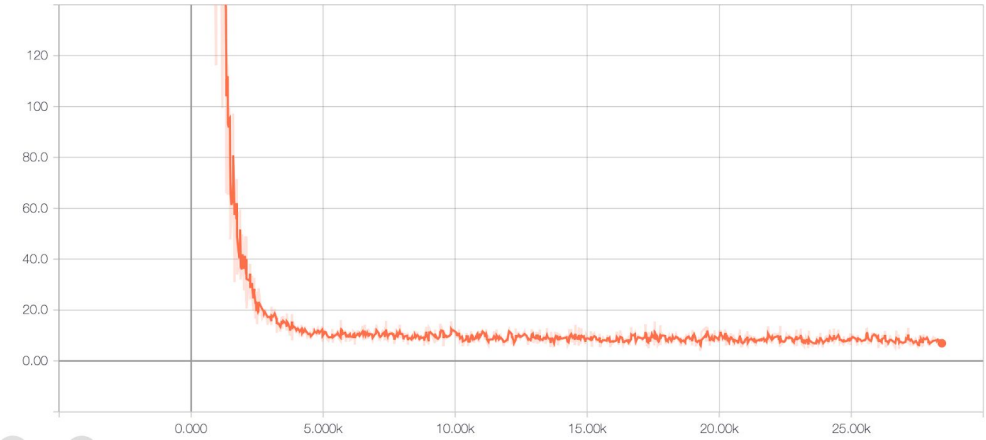


Figure 23 : Loss vs Number of Steps for YOLO v3

Analysis of Results

We observe that we have results which are comparatively similar to results reported by Lin et al[32] in terms of values for each category in Precision and Recall. In particular, we note that SSD FPN Resnet and SSD FPN Mobilenet reported higher values in precision by almost 0.10 in every category as compared to the SSD MobileNet Depth Quantized model. However, the size of the SSD MobileNet Depth Quantized is only at 13.8 MB which is 10x smaller than the smallest Resnet model used in this project. For the mobile app, when we attempted to use Resnet models, we face difficulties to the model being too large and the speed in which the results returned was significantly slower than Mobilenet models.

From the loss graph shown in the section above, SSD MobileNet Depth Quantized had displayed much more volatility and generally recorded higher loss values for each training steps. Both Mobilenet models had significantly higher levels of loss than the Resnet models towards the end of the 50,000 training steps. Consequently, we then observe Resnet models showing greater performance than Mobilenet models. Nevertheless, the performance of Resnet models was not exceedingly greater as the differences were about 0.01 for precision and recall as compared to the SSD FPN MobileNet model.

The poorer performance of SSD Mobilenet Depth Quantized lies in it's theoretical foundation in which quantization was employed. With quantization, the model would have been trained with full precision for certain number of steps which would then switch to quantized training(Weights and activations are compressed to an 8-bit fixed point representation)[24]. The main purpose of this model being quantized was to enable support for accelerated training in TPU and having the ability to have quick, low-latency inferences on images for object detection.

SSD models have an inherent weakness in detecting small objects as reported in our results. This issue is attributed to SSD using feature maps which have low resolution in which small object features would be too small to detect. Increasing input size may help to detect small objects however there is still much more room for improvement as reported by Liu et al[20].

After testing our models in the mobile app, the average time taken for an object to be detected in the app for SSD Mobilenet Depth Quantized and SSD FPN Mobilenet are 9.2s and 12.2s respectively (Resnet models are too large and detection took 30s on average). Therefore, we believe the SSD FPN Mobilenet model was the best model in our investigation as it was the second fastest model to compute training, yield second highest mAP@.50IOU of 0.5763, recorded 0.4738 in the Average Recall for 100 detections, and detections were steadily done at 12.2 seconds on average.

Inference of Test Results

A manual run over a small sample in the test set compared against its ground truth shows that the models are able to detect the flowers that are completely visible, along with partials as long as the top/shape of the flower is visible. It is also able to detect partials with invisible shapes but with lower confidences which get omitted. Since SSD works well with larger objects, it is able to detect all flowers when the number of flowers in the image are less, making the objects more apparent.

$$RMSE_k = \sqrt{\frac{1}{N}\sum_{i=1}^{N}(\hat{c_{ik}} - c_{ik})^2}$$

where:

- RMSE = Root Mean Square Error
- N = Number of Images
- $\hat{c_{ik}}$ = Prediction Counts
- c_{ik} = Ground Truth Counts

Approach	**mRMSE(non-zero)**
always-0	3.03
always-1	2.39
mean	2.96
category-mean	2.97
Mean: SSD FPN MobileNet	**2.99**

Table 5: Mean RMSE results for our mean compared to results reported in [51].

We compare our approach to the following baselines evaluated in [51]:

- always-1: predict the most frequent non-zero value (1) for all classes.
- category-mean: predict the average count per category on Count-val.
- always-0: predict most-frequent ground truth count (0).
- mean: predict the average ground truth count on the Countval set

The table above shows the detection results from the paper[51] and compares it with the mean Root Mean Square Error value for the flowers test set. It shows that our model performs in-line with the performance achieved by Chattopadhyay et Al in his experiment.

4. Mobile Application Architecture

This section discusses the architectural decision process, front end Application, along with the architectural design for predictions.

4.1 Architectural Decisions

At the time of project inception, the team concluded that it is imperative to make use of tools that are platform agnostic and do not lock us into any particular platform. At that point, Google's TensorFlow library seemed like the most logical option to start with, since models on TensorFlow can be trained and deployed on any platform including Google's own Cloud Platform (GCP)[50].
Apple's Vision library running on top of CoreML is intuitive to use but would lock the model down to the Apple platform[52], while the on-device lightweight port of TensorFlow called TensorFlow Lite works across multiple mobile platforms, including iOS[53]. We trained the models using TensorFlow on Cloud TPUs and then looked into deploying the models into an application, where we concluded that converting a full-sized model to a mobile platform would require an understanding of the underlying neural network implementations[57] and would also result in accuracy drops even if we got the conversion right. Also, the TensorFlow lite library was still in developer preview and would not guarantee results. At this point, we decided to use Google's online Cloud Machine Learning Engine (CMLE) to deploy the model, and Firebase Cloud Functions FaaS platform to interact with the CMLE. Any device with Firebase support could (in theory) work with our architecture, making the system platform independent, but would require the predictions to be done online.

4.2 iOS Application Front End

A native iOS Application has been written in Swift 4, which allows users to:

- Click or Select an image from Photo Library
- Select the desired model for detection and classification
    - Two of the five trained models have been selected for deployment. A user can select one of the two options to get detection results, and/or use both one at a time to contrast between the results. The options are:
        - SSD Depth Quantized
        - SSD FPN MobileNet
- Upload image to server for detection (Explained in the Architecture Section)
- View the detection results
    - Each detected flower is annotated and labelled with an id in the image returned. A list of the labelled flowers along with their detection confidences are listed below the image.

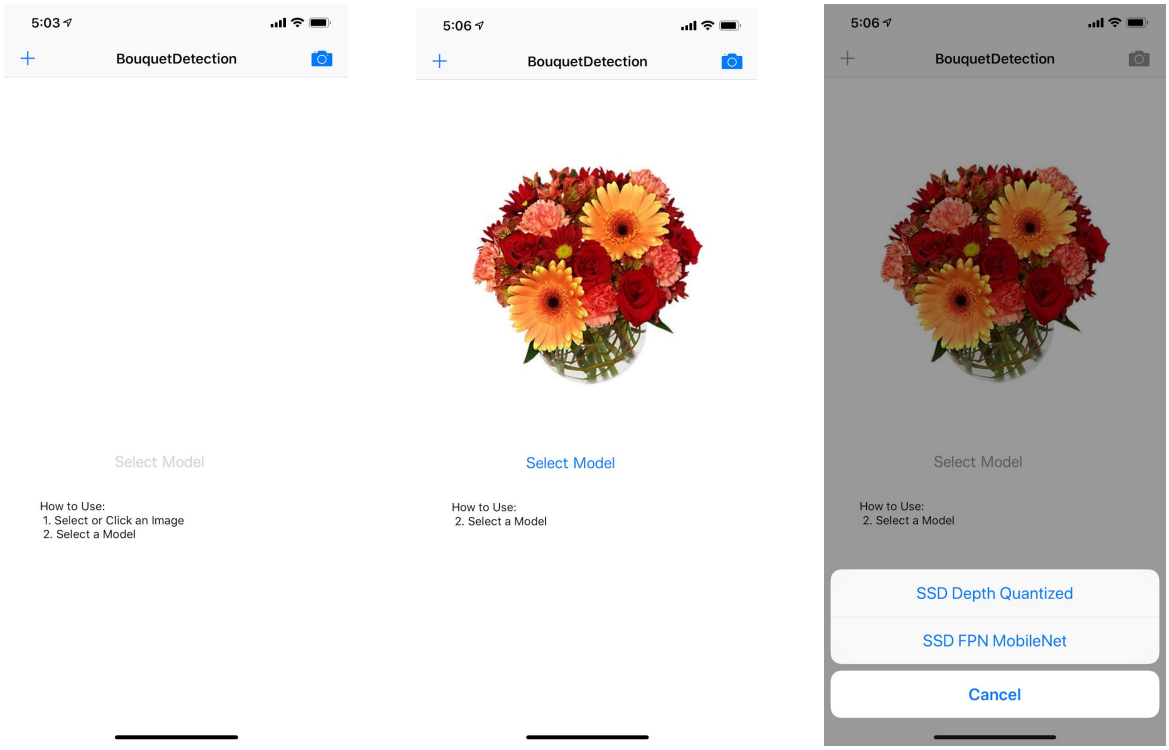


Figure 24: Screenshots of the iOS App

4.3 Server Details and Client Server Communications

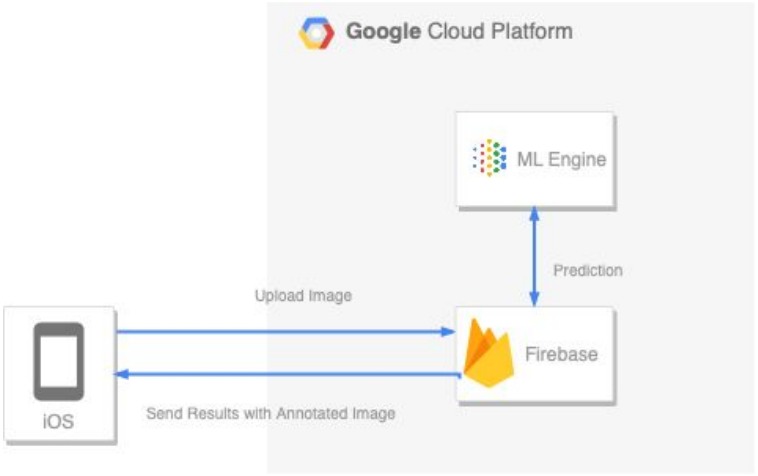


Figure 25: Application Architecture

Google Cloud Platform (GCP)

GCP is a suite of cloud computing services offered by Google. It is a complete package, offering various IaaS, PaaS and SaaS services[50]. The following services have been leveraged for the application:

- Firebase

Firebase is a standalone Google MBaaS (Mobile Backend as a Service)[49] platform which has steadily been integrated into the GCP. It provides a plethora of services which aids Mobile and Web App developers to build Client-Server Applications without having to worry about manually creating a back-end or a communication layer.

- ML Engine

This service allows developers to train and deploy machine learning models on Google's massive infrastructure using simple JSON requests. It can leverage other services within the GCP, like G Storage Buckets for storing and retrieving data, including models and datasets.

Client-Server Architecture

The Application is based on a simple Client-Server design, where the iOS App uploads the image to the Google Storage Bucket and then triggers a Firebase Cloud Function written in Node JS. This program retrieves the image, encodes it into a Base64 string and then encapsulates the request into a JSON structure. This JSON is passed to the ML Engine for prediction and then waits for a response. The result JSON is parsed, which contains the number of detections, along with the bounding boxes and their respective confidences. If a confidence score is higher than 50%, then it's respective bounding box is drawn on the image using a command line image editor called ImageMagick. The final image is stored to a location in the Storage Bucket, whose URL along with label confidences are sent to the client app. The client then fetches the final image from the URL it receives and presents the results.

5. Future work

- Looking for more data
  The performance of the model is influenced by both the quality and quantity of the data to a great extent. Adding new data into the network can provide more features for the model to learn and making it more generalizable in different cases. Various ways could be tried to expand the dataset. Except for crawling some qualified data from the website or through the API, it is also considered to create our dataset. Besides, other augmentation methods such as random flipping can be used as well.

- Combining multiple models
  There are numerous training models used in CNN, and the performance for each of them varies in solving different problems. It is hard to tell which one is the best with a constant variance of results. However, the idea of using ensemble methods can fix this problem. The concept is simple: rather than using a single model, combining multiple models into one network.[19] It helps to reduce the bias as well as the variance and improve the performance of making predictions.

- Optimizing detection of small objects
  From our results, we obtain strong performance in detecting large objects but poor performance for small objects. We infer that two factors can be attributed for this weakness which are namely, lacking of small object images as well as SSD models not being optimized in detecting small objects. To boost our model's performance, we look to the works by Kisantal et al [33] in which they demonstrated better performance on detecting small objects by oversampling images which contains small objects during training. Also, an augmentation based algorithm focused on copy-pasting small objects was implemented and their experimentation showed a 9.7% improvement relative to instance segmentation coupled with 7.1% improvement for object detection of small objects on MS COCO using Mask R-CNN.

- Optimizing application
  A client-server architecture is selected in the mobile app development. Since all data and models are stored in the server, the mobile app only implements some essential functions with a simple user interface (UI). But this architecture may break down when suffering from some network problems. Therefore, a more complicated and robust app can be developed in the future, realizing all prediction tasks locally.

6 Conclusion

This report reflects upon the process of building an Object Detector and Classifier for different types of Flowers from a Mobile Application. The trained models can classify eight different flowers.

One of the most challenging parts of the project was finding appropriate data to build the train and test sets. Since this was an Object detection task, the flowers need to have bounding boxes, which did not exist on most images crawled through. Close to 2000 images were manually annotated to increase the size of the dataset. The images were also augmented to increase the size of the training set.

Multiple object detectors, classifiers, and feature extractors have been identified, out of which a select few were experimented with, to contrast between the performance of all combinations with our dataset.

The trained models can detect certain flowers better than others. This difference may be possible due to the varying numbers of images for each flower in the dataset.
Out of all the models trained, SSD FPN MobileNet seems to be the most suitable model for our application. This model has a relatively small size, which makes it lightweight on cloud resources and can be ported to TensorFlow Lite / MLKit for on-device prediction. It's precision, recall, and accuracy figures are also close to the best results achieved across all the trained models.

The two lightest models have been deployed as Google CMLE models, which are invoked using Firebase Functions. A simple iOS App has been designed to interact with the Firebase functions, which involves selecting a model for prediction and uploading the image to view prediction results.

7 References

[1] Stats and Bots. (2019). *Neural Networks for Beginners: Popular Types and Applications*. [online] Available at: https://blog.statsbot.co/neural-networks-for-beginners-d99f2235efca [Accessed 2 Jun. 2019].
[2] Tetko, I. (2002). Neural Network Studies. 4. Introduction to Associative Neural Networks. *Journal of Chemical Information and Computer Sciences*, 42(3), pp.717-728.
[3] Dkriesel.com. (2019). *A Brief Introduction to Neural Networks [D. Kriesel]*. [online] Available at: http://www.dkriesel.com/en/science/neural_networks [Accessed 2 Jun. 2019].
[4] Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, 61, pp.85-117.
[5] Wang, J., Liu, C., Fu, T. and Zheng, L. (2019). Research on automatic target detection and recognition based on deep learning. *Journal of Visual Communication and Image Representation*, 60, pp.44-50.
[6] Link.springer.com. (2019). [online] Available at: https://link.springer.com/content/pdf/10.1186/s13638-017-0993-1.pdf [Accessed 2 Jun. 2019].
[7] 'Image Classification Using Convolutional Neural Networks and Kernel Extreme Learning Machines' (2018) *2018 25th IEEE International Conference on Image Processing (ICIP), Image Processing (ICIP), 2018 25th IEEE International Conference on*, p. 3009. doi: 10.1109/ICIP.2018.8451560.
[8] Prabhu. 2019. Understanding of Convolutional Neural Network (CNN) — Deep Learning. [ONLINE] Available at: https://medium.com/@RaghavPrabhu/understanding-of-convolutional-neural-network-cnn-deep-learning-99760835f148. [Accessed 2 Jun. 2019].
[9] Adel Nehme. 2019. Understanding Convolutional Neural Networks – Towards Data Science. [ONLINE] Available at: https://towardsdatascience.com/understanding-convolutional-neural-networks-221930904a8e. [Accessed 2 Jun. 2019].
[10] O'Shea, K. and Nash, R. (2015) 'An Introduction to Convolutional Neural Networks'. Available at: https://search-ebscohost-com.ezp.lib.unimelb.edu.au/login.aspx?direct=true&db=edsarx&AN=edsarx.1511.08458&site=eds-live&scope=site [Accessed 2 Jun. 2019].
[11] GitHub. 2019. Data-Science--Cheat-Sheet/super-cheatsheet-deep-learning.pdf at master · abhat222/Data-Science--Cheat-Sheet · GitHub. [ONLINE] Available at: https://github.com/abhat222/Data-Science--Cheat-Sheet/blob/master/Deep%20Learning/super-cheatsheet-deep-learning.pdf. [Accessed 2 Jun. 2019].
[12] CS231n Convolutional Neural Networks for Visual Recognition. 2019. CS231n Convolutional Neural Networks for Visual Recognition. [ONLINE] Available at: http://cs231n.github.io/linear-classify/#softmax. [Accessed 2 Jun. 2019].
[13] Garcia-Gasulla, D. *et al.* (2017) 'An Out-of-the-box Full-network Embedding for Convolutional Neural Networks'. Available at: https://search-ebscohost-com.ezp.lib.unimelb.edu.au/login.aspx?direct=true&db=edsarx&AN=edsarx.1705.07706&site=eds-live&scope=site [Accessed 2 Jun. 2019].
[14] 'A Survey on Transfer Learning' (2010) *IEEE Transactions on Knowledge and Data Engineering, Knowledge and Data Engineering, IEEE Transactions on, IEEE Trans. Knowl. Data Eng*, (10), p. 1345. doi: 10.1109/TKDE.2009.191.
[15] Adit Deshpande. 2019. A Beginner's Guide To Understanding Convolutional Neural Networks Part 2 – Adit Deshpande – CS Undergrad at UCLA ('19). [ONLINE] Available at: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks-Part-2/. [Accessed 2 Jun. 2019].

[16] Uijlings, J. *et al.* (2013) 'Selective Search for Object Recognition', *International Journal of Computer Vision*, 104(2), pp. 154–171. doi: 10.1007/s11263-013-0620-5.
[17] 'Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks' (2017) *IEEE Transactions on Pattern Analysis and Machine Intelligence, Pattern Analysis and Machine Intelligence, IEEE Transactions on, IEEE Trans. Pattern Anal. Mach. Intell*, (6), p. 1137. doi: 10.1109/TPAMI.2016.2577031.
[18] Redmon, J. and Farhadi, A. (2018) 'YOLOv3: An Incremental Improvement'. Available at: https://search-ebscohost-com.ezp.lib.unimelb.edu.au/login.aspx?direct=true&db=edsarx&AN=edsarx.1804.02767&site=eds-live&scope=site [Accessed 2 Jun. 2019].
[19] CiteSeerX — On Optimization Methods for Deep Learning. 2019. CiteSeerX — On Optimization Methods for Deep Learning. [ONLINE] Available at: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.220.8705. [Accessed 3 Jun. 2019].
[20] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.Y. and Berg, A.C., 2016, October. Ssd: Single shot multibox detector. In *European conference on computer vision* (pp. 21-37). Springer, Cham.
[21] Howard, A.G., 2013. Some improvements on deep convolutional neural network based image classification. *arXiv preprint arXiv:1312.5402*.
[22] Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H. and Kalenichenko, D., 2018. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 2704-2713).
[23] Krishnamoorthi, R., 2018. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342*.
[24] Chow, D. (2019). *Accelerated Training and Inference with the Tensorflow Object Detection API*. [online] Google AI Blog. Available at: https://ai.googleblog.com/2018/07/accelerated-training-and-inference-with.html [Accessed 2 Jun. 2019].
[25] GitHub. (2019). *tensorflow/models*. [online] Available at: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md [Accessed 2 Jun. 2019].
[26] Jeong, J., Park, H. and Kwak, N., 2017. Enhancement of SSD by concatenating feature maps for object detection. *arXiv preprint arXiv:1705.09587*.
[27] Lin, T.Y., Dollár, P., Girshick, R., He, K., Hariharan, B. and Belongie, S., 2017. Feature pyramid networks for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 2117-2125).
[28] Lin, T.Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P. and Zitnick, C.L., 2014, September. Microsoft coco: Common objects in context. In *European conference on computer vision* (pp. 740-755). Springer, Cham.
[29] Kuznetsova, A., Rom, H., Alldrin, N., Uijlings, J., Krasin, I., Pont-Tuset, J., Kamali, S., Popov, S., Malloci, M., Duerig, T. and Ferrari, V., 2018. The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale. *arXiv preprint arXiv:1811.00982*.
[30] Howard, A.G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M. and Adam, H., 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.
[31] He, K., Zhang, X., Ren, S. and Sun, J., 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 770-778).
[32] Lin, T.Y., Goyal, P., Girshick, R., He, K. and Dollár, P., 2017. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision* (pp. 2980-2988).

[33] Kisantal, M., Wojna, Z., Murawski, J., Naruniec, J., & Cho, K. ,2019. Augmentation for small object detection. ArXiv, abs/1902.07296.
[34] Everything about Data Analytics. 2019. Interpretability of Neural Networks – Everything about Data Analytics. [ONLINE] Available at: https://datawarrior.wordpress.com/2017/10/31/interpretability-of-neural-networks/. [Accessed 2 Jun. 2019].
[35] Towards Data Science. (2019). *Understanding Convolutional Neural Networks*. [ONLINE] Available at: https://towardsdatascience.com/understanding-convolutional-neural-networks-221930904a8e [Accessed 2 Jun. 2019].
[36] Towards Data Science. (2019). *Convolutional Neural Network*. [ONLINE] Available at: https://towardsdatascience.com/convolutional-neural-network-17fb77e76c05 [Accessed 2 Jun. 2019].
[37] Introduction to Convolutional Neural Networks - Semantic Scholar. (2019). *Introduction to Convolutional Neural Networks - Semantic Scholar.* [ONLINE] Available at: https://www.semanticscholar.org/paper/Introduction-to-Convolutional-Neural-Networks-Wu/450ca19932fcef1ca6d0442cbf52fec38fb9d1e5. [Accessed 2 Jun. 2019].
[38] Sebastian Ruder. (2019). *Transfer Learning - Machine Learning's Next Frontier*. [ONLINE] Available at: http://ruder.io/transfer-learning/index.html#fn6 [Accessed 2 Jun. 2019].
[39] Towards Data Science. (2019). *Transfer learning from pre-trained models*. [ONLINE] Available at: https://towardsdatascience.com/transfer-learning-from-pre-trained-models-f2393f124751 [Accessed 2 Jun. 2019].
[40] Towards Data Science. (2019). *R-CNN, Fast R-CNN, Faster R-CNN, YOLO — Object Detection Algorithms*. [ONLINE] Available at: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e [Accessed 2 Jun. 2019].
[41] Medium. (2019). *What do we learn from single shot object detectors (SSD, YOLOv3), FPN & Focal loss (RetinaNet)?*. [ONLINE] Available at: https://medium.com/@jonathan_hui/what-do-we-learn-from-single-shot-object-detectors-ssd-yolo-fpn-focal-loss-3888677c5f4d [Accessed 2 Jun. 2019].
[42] Jonathan Hui. (2019). *Real-time Object Detection with YOLO, YOLOv2 and now YOLOv3.* [ONLINE] Available at: https://medium.com/@jonathan_hui/real-time-object-detection-with-yolo-yolov2-28b1b93e2088. [Accessed 2 Jun. 2019].
[43] 'You Only Look Once: Unified, Real-Time Object Detection' (2016) *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on*, p. 779. doi: 10.1109/CVPR.2016.91.
[44] Howard, A. G. *et al.* (2017) 'MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications'. Available at: https://search-ebscohost-com.ezp.lib.unimelb.edu.au/login.aspx?direct=true&db=edsarx&AN=edsarx.1704.04861&site=eds-live&scope=site [Accessed 2 Jun. 2019].
ResNet
[45] 'Deep Residual Learning for Image Recognition' (2016) *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Computer Vision and Pattern Recognition (CVPR), 2016 IEEE Conference on*, p. 770. doi: 10.1109/CVPR.2016.90.
[46] Szegedy, C. *et al.* (2014) 'Going Deeper with Convolutions'. Available at: https://search-ebscohost-com.ezp.lib.unimelb.edu.au/login.aspx?direct=true&db=edsarx&AN=edsarx.1409.4842&site=eds-live&scope=site [Accessed 2 Jun. 2019].

[47] ‘Learning long-term dependencies with gradient descent is difficult’ (1994) *IEEE Transactions on Neural Networks, Neural Networks, IEEE Transactions on, IEEE Trans. Neural Netw*, (2), p. 157. doi: 10.1109/72.279181.
[48] Yun, Y. (2019). *YunYang1994/tensorflow-yolov3*. [online] GitHub. Available at: https://github.com/YunYang1994/tensorflow-yolov3 [Accessed 15 Mar. 2019].
[49] S, C. (2019). *Intro to Firebase 2.0: Mobile-Backend-as-a-Service (MBaaS) from Google*. [online] Meetup. Available at: https://www.meetup.com/en-AU/acloud/events/rtntpyvjbsb/ [Accessed 4 Jun. 2019].
[50] Ombura, M. (2019). *GCP: The Google Cloud Platform Compute Stack Explained.*. [online] Medium. Available at: https://medium.com/google-cloud/gcp-the-google-cloud-platform-compute-stack-explained-c4ebdccd299b [Accessed 1 Jun. 2019].
[51] Chattopadhyay, P., Vedantam, R., Selvaraju, R.R., Batra, D. and Parikh, D., 2017. Counting everyday objects in everyday scenes. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 1135-1144).
[52] Thakkar, M., 2019. Introduction to Core ML Framework. In *Beginning Machine Learning in iOS* (pp. 15-49). Apress, Berkeley, CA.
[53]Medium. (2019). *TensorFlow Lite Now Faster with Mobile GPUs (Developer Preview)*. [online] Available at:https://medium.com/tensorflow/tensorflow-lite-now-faster-with-mobile-gpus-developer-preview-e15797e6dee7 [Accessed 15 Feb. 2019].
[54] M. Ren, R. Kiros, and R. Zemel. Exploring models and data for image question answering. In Advances in Neural Information Processing Systems, pages 2953–2961, 2015.
[55] M. Malinowski, M. Rohrbach, and M. Fritz. Ask your neurons: A neural-based approach to answering questions about images. In Proceedings of the IEEE International Conference on Computer Vision, pages 1–9, 2015.
[56] M. Malinowski, M. Rohrbach, and M. Fritz. Ask your neurons: A neural-based approach to answering questions about images. In Proceedings of the IEEE International Conference on Computer Vision, pages 1–9, 2015.
[57]TensorFlow. (2019). *Select TensorFlow operators to use in TensorFlow Lite | TensorFlow Lite | TensorFlow*. [online] Available at: https://www.tensorflow.org/lite/guide/ops_select [Accessed 9 May 2019].

8 Appendices

Appendix A

Link to GitHub Repository: https://github.com/kkkkkaran/BouquetFlowers

Link to Video Demonstration: https://www.youtube.com/watch?v=3rUYZAN9OrM

Appendix B

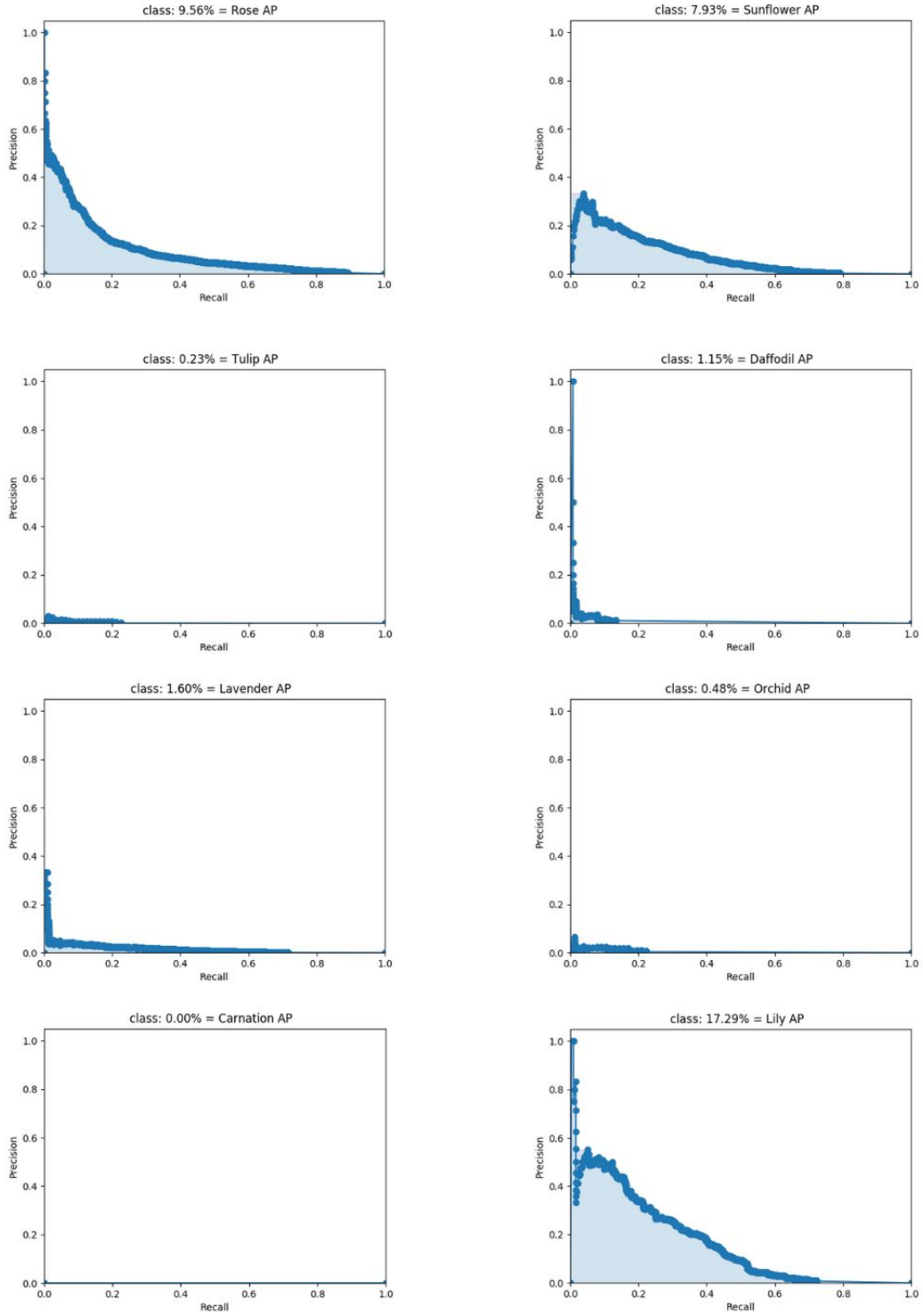
Yun, Y. (2019). *YunYang1994/tensorflow-yolov3*. [online] GitHub. Available at: https://github.com/YunYang1994/tensorflow-yolov3 [Accessed 15 Mar. 2019].

Kapica, P. (2019). *Implementing YOLO v3 in Tensorflow (TF-Slim)*. [online] ITNEXT. Available at: https://itnext.io/implementing-yolo-v3-in-tensorflow-tf-slim-c3c55ff59dbe [Accessed 3 Mar. 2019].

Linder-Noren, E. (2019). *eriklindernoren/PyTorch-YOLOv3*. [online] GitHub. Available at: https://github.com/eriklindernoren/PyTorch-YOLOv3 [Accessed 3 Apr. 2019].

Zhang, Z. (2019). *zzh8829/yolov3-tf2*. [online] GitHub. Available at: https://github.com/zzh8829/yolov3-tf2 [Accessed 5 Apr. 2019].

Appendix C



Precision, Recall and Average Precisions for each class tested on the YOLOv3 trained model.