A Mobile Application for Australian Politician Recognition Based on Deep Learning

COMP90055 Computing Project (25 Credits)

Type of Project: Software Development Project

Supervisor: Prof. Richard Sinnott

Team Member:

Hanwen Zheng Wenshuang Cao Zhiyuan Chen

Semester 1, 2019

Abstract

Deep learning, part of machine learning methods based on artificial neural networks, has been applied to computer vision with tremendous success for years. With the development of image classification and convolutional neural network models, computer vision problems like facial recognition has achieved better results.

With the 2019 Australian federal election held on 18 May 2019, more and more politicians from various parties have come into people's sight. As most of the politicians are new to Australians, a politician recognition system with mobile application becomes an urgent need. The objective of this project is to implement an Android application that identifies 50 politicians from 5 Australian parties.

In this project, abundant data of politician images were crawled from Internet and videos. After pre-processing, AlexNet and MobileNet were used to train dataset. Finally, an application based on our trained model was implemented to identify politicians.

Key words: Deep Learning, Image Classification, Convolutional Neural Network, AlexNet, MobileNet, TensorFlow, Australian Politicians

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 8694 words in length (excluding text in images, table, bibliographies, and appendices).

Acknowledgement

Firstly, we would like to show our gratitude and appreciation to our supervisor, Professor Richard Sinnott, for giving us the opportunity to finish this meaningful project. Richard also provided us with suggestions in coding and thesis.
Secondly, we would like to thank TensorFlow community for the remarkable APIs and other materials.
Thirdly, we would like to thank Google Image, YouTube, Facebook and Twitter for providing us with data source for the dataset and thesis.
Last but not least, we would like to thank all our friends for their help.

Table of Contents

List of Figures

1. Introduction

With the 2019 Australian federal election held on Saturday 18 May 2019, more and more politicians from different political parties appear in vision of Australian residents. However, most of the politicians cannot be recognized by the general public. Thus, a politician recognition system is needed for people to know better about each politician and his background.

1.1 Deep Learning

Artificial intelligence (AI) has always been one of the most dramatic goals mankind pursuing for years. As one of the technologies and research fields of machine learning, deep learning (or hierarchical learning) accelerates the arrival of AI by establishing artificial neural networks (ANNs) with a hierarchical structure. As ANNs are able to extract and filter useful information from input, deep learning has the ability of representation learning to realize supervised and unsupervised learning from end to end. Furthermore, deep learning can also play an important role in establishing reinforcement learning system to form deep reinforcement learning.

ANNs, inspired by the biological neural networks, are generated by a collection of connected nodes or units (artificial neurons). As is shown in figure 1, a path, started from an input and ended with an output, can be described by a flow graph. In this kind of flow graph, each node represents a fundamental function and an output weight. There are no parent nodes for input nodes and there are no child nodes for output nodes. Then the current output weight becomes the input of the child node. The length of the longest path is the depth of a certain ANN [1].

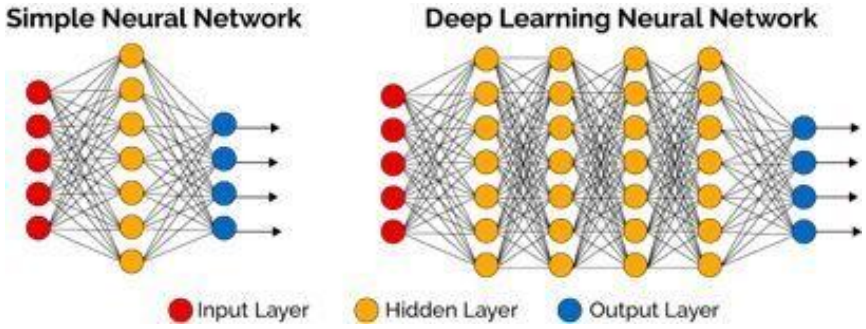


Figure 1. Artificial Neural Networks (ANNs)

Assume that a deep learning system S is established with n layers and an input set I, the system will automatically extract a set of hierarchical features of I by adjusting parameters to make the output is exactly or similar to input I. The method of deep learning is to stack multiple layers, which means the output of the current layer is the input of the next layer. In this way, the system is able to realize the hierarchical expression of input information. The core procedure of deep learning is concluded into three steps: First, use unsupervised learning for the pre-training of each network layer. Then, use unsupervised learning to train only one layer at a time, the training result is the input of next layer. Last, use the top-down supervised algorithm to adjust all layers.

Deep learning architectures such as multilayer perceptron, convolutional neural network, recurrent neural network and convolutional deep belief network are used for training data

with complex structure, large volume and high dimension in areas like computer vison, natural language processing, bioinformatics, automatic control, facial recognition and machine translation, where it has gained some achievements.

1.2 Image Classification

Image classification is an image processing method to classify objects with various labels according to different features extracted from image information (Figure 2). The principle is to quantitively analyze images and classify each image to one of the labels. CNN models are the most commonly used structures for image classification. There are multiple ways for image classification: index based on color features, index based on gray scale histograms, classification by texture features and classification by shapes and spatial relations. Though image classification requires large volume of dataset and high computation complexity, the classification accuracy is always ideal. In general, an image classification model consists of the following four steps:

A. Low-level feature extraction: The model generally extracts local features according to fixed step size and scale. There are several local feature descriptors like Scale-Invariant Feature Transform (SIFT), Histogram of Oriented Gradient (HOG) and Local Binary Pattern (LBP). To prevent from much information loss, a system will use multiple feature descriptors instead of just one.

B. Feature encoding: As the low-level features contain large volume of redundancy and noise, a transform algorithm, used for encoding the low-level features to improve the robustness of feature expression, is feature encoding. The frequently used algorithms for feature encoding are vector quantization coding, sparse coding, local linear constraint coding and Fisher vector coding.

C. Spatial feature constraint: Spatial feature constraint, also called as feature convergence, means to get the maximum or average value of features in each dimension of a specific range of space in order to obtain the expression of the certain invariant feature.

D. Classifier design: The output of an image after the previous operations is a vector with a certain dimension. The next step is to classify the images with a classifier. The most common used classifiers are support vector machine (SVM) and random forest [2].

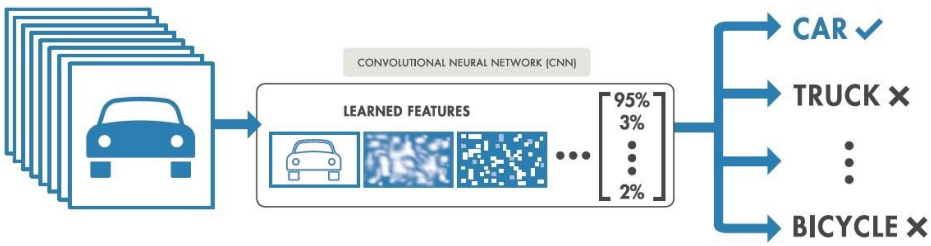


Figure 2. Image Classification Workflow

1.3 TensorFlow

TensorFlow is a free and open-source software library for deep learning and computation using dataflow. To apply this facial recognition system to a practical problem, the authors decided to

use TensorFlow. In this project, TensorFlow structure was used for building CNNs including AlexNet and MobileNet, training the collected dataset, and collecting the results for analysis.

2. Convolutional Neural Networks and Implementation

2.1 Convolutional Neural Network

Convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, which is normally used in deep learning especially in image classification. CNN performs well in representation learning, and it can do shift-invariant classification according to the hierarchical structure of the input information. As a matter of fact, it is also called as shift-invariant artificial neural networks (SIANN) [3]. CNN is inspired by biological visual perception structure for supervised and unsupervised learning. A standard CNN normally consists of five layers: input layer, convolutional layer, pooling layer, fully connected layer and output layer, which are also used in this project. Convolutional layer, pooling layer and fully connected layer comprise a normal hidden layer (Figure 3).

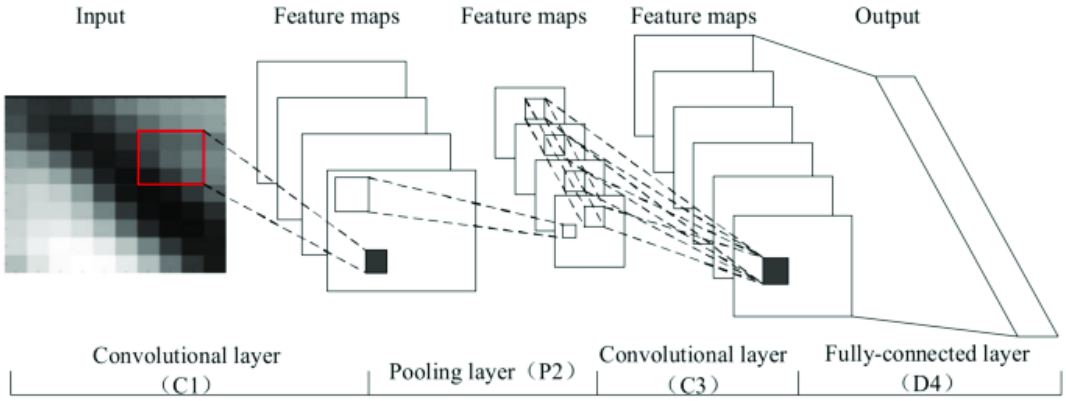


Figure 3. Typical CNN structure

2.1.1 Input layer

The input layer processes image data with multiple dimensions into a matrix of pixel values. In this project, input images have the following dimensions: $[width \times height \times depth]$. The depth represents RGB channels, which is 3 in this case. Thus, the size of images in AlexNet is $227 \times 227 \times 3$, and the size of images in MobileNet is $224 \times 224 \times 3$. As gradient descent is used for learning, the input layer should pre-process the input data. Normally, there are two methods: mean subtraction and normalization. Mean subtraction means that each image's feature should subtract the average feature weight of all the training images so that all the values are centralized to 0. The motivation of mean subtraction is to reduce computation. Normalization is usually the next step of mean subtraction. It means that the value of each dimension should be divided with the standard deviation of data in the specific dimension. The purpose of normalization is to maintain data at a fixed range of change in all dimensions.

2.1.2 Convolutional layer

As one of the most important layers of CNN, convolutional layer extracts features from input

data (Figure 4). There are three important procedures: convolutional kernel, convolutional parameters and activation function.

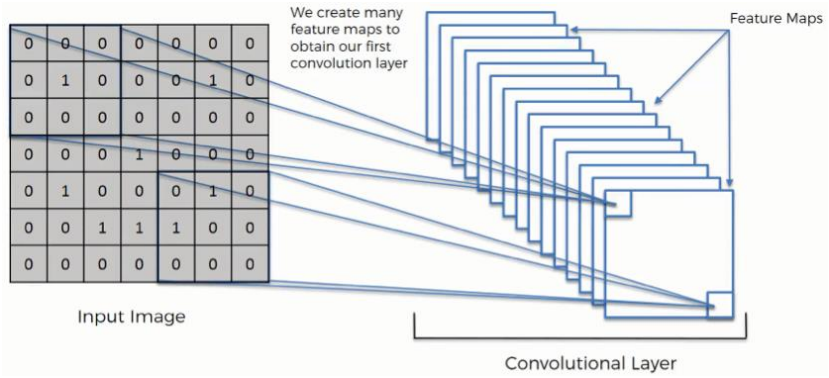


Figure 4. How convolutional layer works

A. Convolutional Kernel

A convolutional layer consists of multiple kernels. Like a neuron of feedforward neural network, each factor of every kernel has a fixed weight coefficient and bias vector. Each neuron is connected with multiple neurons from the previous layer with proximate region. The size of a convolutional kernel depends on the area of proximate region, which is called as receptive field. While working, a kernel will regularly scan the input features, calculate multiplication sum of matrix element of the features in the receptive field and superposition deviation. A layer formed by unit convolutional kernels is called network-in-network (NIN) or multiplayer perceptron convolutional layer. A unit convolutional kernel can not only retain the size of a feature map, but also reduce the computation by decreasing the number of channels. A convolutional layer completely formed by unit kernels is a multi-layer perceptron with parameter sharing.

B. Convolutional parameters

The convolutional parameters include kernel size, step length and padding. The three parameters decide the size of the output feature map. The kernel size can be specified as any value smaller than the input image size. The larger a kernel is, the more complex the extracted features are. The kernel step length defines the distance between positions where the kernel is while scanning the feature map. A convolutional kernel will scan the elements one by one when the step length is 1. When the step length becomes n, the kernel will skip $n-1$ elements for the next scan.

As the feature map size will become smaller with the stack of convolutional layers, padding enlarges the size of feature map in order to offset the effect caused by size shrinkage. The normal padding methods are padding by 0 and replication padding. Padding can be classified into four categories according to the number of layers and motivation. Valid padding means no padding at all. Convolutional kernels can only access the position with a complete receptive field. Same/Half padding means that enough padding to make the sizes of input and output feature maps same. As for full padding, it means filling to make each pixel be accessed with same times. The last one is arbitrary padding, which means padding with a specific value between valid padding and full padding.

C. Activation function

Activation function means a nonlinear mapping for the output result of a convolutional layer to assist complex feature expression. The representation is $A_{i,j,k}^{l} = f(Z_{i,j,k}^{l})$. Rectified linear unit (ReLU) as well as leaky ReLU, parametric ReLU, randomized ReLU and exponential linear ReLU is widely used for activation function. The output will become a linear function of previous input without an activation function. In most cases, an activation function is after the convolutional kernel. On the other hand, it can be before a conventional kernel by some preactivation algorithms.

Apart from linear convolution, there are some CNNs with more complex convolution including tiled convolution, deconvolution and dilated convolution. In tiled convolution, a kernel only scans one part of the feature map. As for the rest parts, they are operated by the kernels in the same layer. Tiled convolution is a better way to capture the shift-invariant features of input images. Dilated convolution, also called as transposed convolution, connects a single input activation with multiple input activations in order to enlarge the input images, which is important in image semantic segmentation and convolutional autoencoder. Dilated convolution introduces expansion rate based on the linear convolution to enlarge the receptive field. As dilated convolution extracts more information of a feature map, it is always used to capture long-range dependency. CNNs with dilated convolution mainly supports areas like machine translation and speech recognition in natural language processing [4].

2.1.3 Pooling layer

After feature extraction from convolutional layer, the output feature map is transformed to pooling layer for feature selection and information filter. A preset pooling algorithm will replace the result of a single point with the statistics of the feature maps from adjacent regions. The pooling procedure also depends on the pooling size, step length and padding control, which is similar with feature extraction. There are three types of pooling: L_p pooling, mixed/stochastic pooling and spectral pooling.

A. L_p pooling

L_p pooling is inspired by the visual cortex hierarchical structure whose normal expression is $A_k^l(i,j) = \left[\sum_{x=1}^{f}\sum_{y=1}^{f} A_k^l(s_0 i + x, s_0 j = y)^p\right]^{1/p}$, where s_0 is the step length, (i,j) is the pixel value, and p is the fixed parameter. Average pooling means $p = 1$. Max pooling means $p \to \infty$. Both poolings are the way to reserve the background texture information by decreasing the size of feature map.

B. Mixed/Stochastic pooling

Mixed pooling is the linear composition of average pooling and max pooling, which can be expressed as $A_k^l = \lambda L_1(A_k^l) + L_\infty(A_k^l),\ \lambda \in [0,1]$. Stochastic pooling means to randomly pick a value in the range of a specific probability distribution, which ensures that part of the activation signal can be transformed to the next layer. Mixed and stochastic pooling perform well in preventing overfitting.

C. Spectral pooling

Spectral pooling is a method based on Fast Fourier Transform (FFT) to establish a CNN based on FFT. Spectral pooling can not only preserve low frequency changing information but control the feature map size efficiently as well. Meanwhile, a spectral pooling based on a mature FFT can complete the task with little computation [5].

2.1.4 Fully connected layer

Fully connected layer, inspired by the hidden layer of traditional feedforward neural network, is always at the end of the hidden layers of a CNN (Figure 5). Feature maps will be no longer 3-dimension and be transformed as vectors to the next layer by the activation function. In some specific CNNs, the function of fully connected layer can be replaced by a global average pooling.

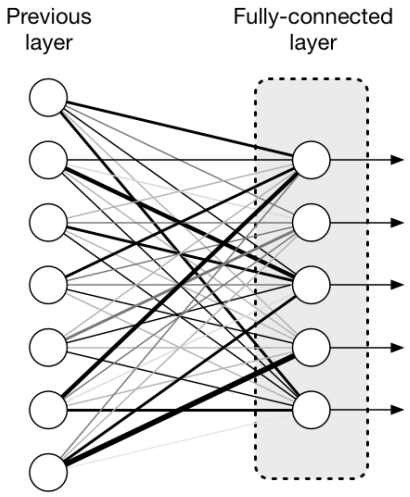


Figure 5. Fully connected layer workflow

2.1.5 Output layer

Normally, an output layer is connected with the fully connected layer. For image classification, the output layer normally uses SoftMax function to output classification labels.

SoftMax function is a function to restrict the range of each element to $(0,1)$, and all the sum equals 1. The expression is $\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}}$, for $j = 1, \ldots, K$. The input of SoftMax is the results from K different linear functions, and the probability of sample vector x belonging to the j_{th} label is $P(y = j|x) = \frac{e^{x^T w_j}}{\sum_{k=1}^{K} e^{x^T w_k}}$. This can be regarded as a composition of K linear SoftMax functions. SoftMax regression model is an algorithm to solve multiple regression problems. It is always combined with cross entropy loss function for classifiers in supervised learning [6].

A CNN always performs three properties: connectivity, representation learning and biological similarity. The connection between convolutional layers is called sparse connection, which means each neuron in the feature map of n_{th} layer is the linear composition of neurons of

receptive field defined by convolutional kernels in $(n-1)_{th}$ layer. Sparse connection not only improves stability and generalization ability to avoid overfitting, but also fasten quick learning and lower memory usage by reducing the sum of weight parameters. Representation learning extracts high order feature by keeping the shift-invariance of input features, which is the main reason for CNN widely used in computer vision area. As for biological similarity, sparse connection is similar to the corresponding neuroscience process – the visual space organization in visual vortex. The term receptive field also comes from biological research. Last but not least, weight sharing closely relates to target-propagation and feedback alignment in brain learning.

2.2 AlexNet

2.2.1 Overview

AlexNet [7] contains eight learned layers in total, including five convolution layers and three fully connected layers (Figure 6). Four types of special layers are used in the process. Firstly, the max pooling layer is used in convolution layer 1, 2, 5. Secondly, local response normalization is used in convolution layer 1 and 2. Thirdly, the ReLU is used in all layers. Finally, dropout layer is applied in fully connected layer 6 and 7. The size of input image is $227 \times 227 \times 3$, the output of the fifth convolution layer is $6 \times 6 \times 256$. Each fully connected layer has 4096 neurons. In order to generate a distribution over 1000 classed, the output of the last full-connected layer is fed to a 1000-way softmax. The original AlexNet was trained on two GPUs which can reduce training time and improve the accuracy. However, our dataset is much smaller than the ImageNet, and with our device's limitation, we only trained on a single CPU.

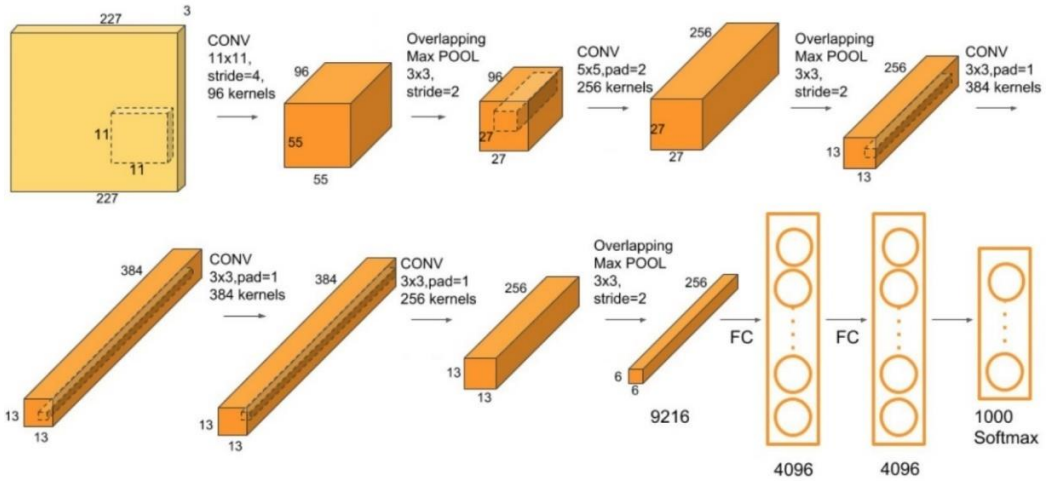


Figure 6. Architecture of AlexNet

2.2.2 ReLU Nonlinearity

ReLU is the activation function used in AlexNet. As a ramp function, it is always expressed as $f(x) = max(0, x)$. Compared with traditional activation functions like logistic sigmoid and tanh hyperbolic function, ReLU has two main advantages. Firstly, ReLU has better response for linear functions. Secondly, as for non-linear functions, ReLU avoids problem of vanishing

gradient because the gradient of non-negative interval is constant. Thus, ReLU not only keeps the convergence speed of the model at a stable state, but also simplifies computing procedure and cost. Furthermore, ReLU realizes the sparse activation which is help for feature mining as well as fitting training data. However, dead ReLU problem might happen during training and that directly result some parameters will never be updated since they are located at negative interval and the gradient is zero. To solve this problem, the learning rate should be adjusted into an appropriate value in avoid of large parameter update during training.

2.2.3 Dropout

In order to reduce overfitting, AlexNet uses two methods, one is data augmentation which will be discussed in dataset section, another one is by adding a dropout layer during fully connected layer. Instead of using regularization like linear models, dropout avoids overfitting by modifying the neural network architecture. This process of dropout contains three steps. The first step is to temporarily sets neurons to zero with the probability 0.5 in hidden layers and keeps the input and output neurons. After that, the modified neuron network keeps propagate forward and then back propagate based on the loss. Secondly, updating the parameters of those un-deleted neurons according to stochastic gradient descent after a batch dataset being trained. Thirdly, retrieving those deleted neurons and then repeating the first step. Moreover, the outputs of all neurons are multiplied by 0.5 which can take the average of the networks. The using of dropout in the first two fully connected layers doubles the iteration number for convergence.

2.2.4 Overlapping max pooling

Normally, the pooling layer used in traditional CNNs is un-overlapped which means the size of pooling unit is the same as the stride. In this way, those neurons after pooling operation will not show up next time. However, the pooling units in AlexNet is set with 3×3 while the stride is set to 2. This approach can keep some neurons because of the overlapping part. The overlapping pooling can improve accuracy as well as the reduce overfitting. Furthermore, it can also enhance the ability of generalization.

2.2.5 Local Response Normalization

Local response normalization (LRN) layer implements “lateral inhibition”. This is a neurobiology concept and it means the capacity of an excited neuron to reduce the activity of its neighbors. AlexNet uses ReLU as its active function which can greatly accelerate the training speed. However, there is no boundary for response of ReLU and that will lead to continuous learning in those neurons which have positive value as an input for ReLU. In order to enhance the sensory perception and extract those high frequency features since they are important elements for CNN, we should normalize the local neighboring parts of an excited neuron. LRN can effectively help for promoting the generalization ability, and it reduces both top-1 and top-5 errors. There are four constant parameters in LRN and their value are defined by using a validating set. The expression of LRN is shown in formula (1), where $\mathrm{k}, \mathrm{n}, \alpha, \beta$ are hyper meters, N is the number of filters, $a_{x,y}^{j}$ is the position of a neuron applied with LRN.

$$b_{x,y}^{i} = a_{x,y}^{i} / \left(k + \alpha \sum_{j=\max\left(0, i-\frac{n}{2}\right)}^{\min\left(N-1, i+\frac{n}{2}\right)} \left(a_{x,y}^{j}\right)^{2} \right)^{\beta} \quad (1)$$

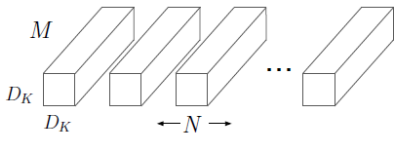
2.3 MobileNet

2.3.1 Overview

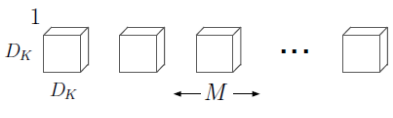
As the development of CNNs, the network structure becomes more sophisticated and the learning becomes deeper and deeper. Even though the accuracy can be improved in this way, the cost of compute as well as training time increases at the same time. However, our goal is to run the facial recognition model on mobile phone and the speed is important for this application. Without decreasing accuracy, MobileNet [8] can reduce the model's parameter size as well as training time. Thus, we decided to use MobileNet as our model for mobile phone application since it is an efficient network both in size and in training speed.

2.3.2 Depthwise Separable Convolution (DSC)

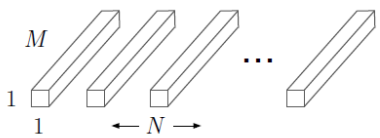
MobileNet introduced a new kind of convolution named depthwise separable convolutions which factorize a standard convolution into two parts (figure 7). The first part is a depthwise convolution, compared to standard convolution, it assigns each input channel an individual filter. The second part is the normal convolution named pointwise convolution which set the size of feature kernel to 1×1. In general, the performance of these two convolutions' combination is almost the same as a standard convolution, but it greatly reduces the compute and parameters. Supposing the size of input feature map is $D_F \times D_F \times M$, the number of filters is N and the size of filter is $D_K \times D_K$, the size of output feature map is $D_F \times D_F \times N$, the computation of standard convolution is $D_K \cdot D_K \cdot M \cdot N \cdot D_F \cdot D_F$ where the DSC's compute cost is $D_K \cdot D_K \cdot M \cdot D_F \cdot D_F + M \cdot N \cdot D_F \cdot D_F$ which is far more smaller than the standard one.



(a) Standard Convolution Filters



(b) Depthwise Convolutional Filters



(c) 1×1 Convolutional Filters called Pointwise Convolution in the context of Depthwise Separable Convolution

Figure 7. Depthwise Separable Convolution (DSC)

2.3.3 Batch Normalization and ReLU

The standard CNNs uses batch normalization as well as ReLU operation after a standard convolution. MobileNet uses batch normalization and ReLU after both depthwise convolution and pointwise convolutions (figure 8). The filter size of depthwise convolution is 3×3.

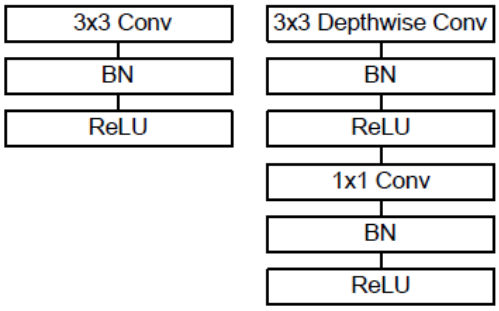


Figure 8. Batch Normalization and ReLU in MobileNet

2.3.4 Network Architecture

MobileNet contains 28 layers including 27 convolution layers as well as a fully connected layer (Figure 9). Furthermore, MobileNet uses an average pooling layer and a softmax layer. The first layer is a standard convolution layer and the other convolution layers are DSC. The depthwise convolution layers and pointwise convolution layers are counted separately.

Table 1. MobileNet Body Architecture

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Figure 9. MobileNet Structure

2.3.5 Width Multiplier and Resolution Multiplier

In order to achieve further reduction of computation cost and network size, MobileNet applies two hyper parameters, one is width multiplies α and another one is resolution multiplier ρ. Width multiplier is used to adjust the number of channels in each layer. Assuming the channel of input is M and the depth of output is N, then the thinned size becomes αM and αN when $\alpha < 1$. The resolution multiplier is used for reducing the input feature map. It sets the resolution of input into ρD_F where $\rho < 1$. After using these two multipliers, the computation cost becomes $D_K \cdot D_K \cdot \alpha M \cdot \rho D_F \cdot \rho D_F + \alpha M \cdot \alpha N \cdot \rho D_F \cdot \rho D_F$.

3 Dataset

Dataset is the foundation of deep learning, as the quality of the dataset have directly influence on learning outcomes. For the purpose of this project, a good object classification dataset should contain a large number of high-resolution images with single faces of every person that are being trained to recognize. There are plenty of open datasets on the Internet that serves the purpose of deep learning, like ImageNet [9] and Oxford-IIIT Pet Dataset [10]. However, there is no pre-processed dataset that suits this project. Therefore, a new dataset needs to be constructed.

There are 226 representative politicians in Australia till mid-2019, including leaders and senators from 10 federal parliamentary parties. Generally, training over more classes means more images for every class, more computing power and more training time are needed to obtain the same accuracy as training over less classes. Due to the time scope and limited human power of this project, we have to choose between developing a system that can recognize all of the politicians in AU or part of them. It is a tradeoff between compatibility and accuracy. We finally decide to target a group of leaders and senators from each party, that is 50 people in total. Leaders and senators are generally the most active people of a party on medias. So, targeting them will maximize facial recognition system's usefulness while limiting the number of classes to maintain high accuracy.

3.1 Dataset sources

A lot of effort were put into searching through the Internet to manually select suitable images and videos due to the special nature of this project which is about specific persons. Our data has mainly three sources: Google Image, YouTube, and Twitter & Facebook. Google Image is a good source of politicians' public photos, and generally comes with high diversity. To obtain data from Google Image, we first include criteria like resolution and size in searching to increase the quality of images. Then we batch download images from Google Image. Total of 1810 images were reserved at the end of Google Image searching.

Google Image covers a lot of images for famous politicians like Scott Morrison or Bill Shorten. However, for less famous ones, less than 10 photos are covered on Google Image. For deep learning, each class needs to have at least hundreds of images to obtain an accurate model during training. Therefore, another format of data is needed. We found that video is in fact a good source of faces. Although it seems quite similar from second to second, the orientation of a person's face is always changing in video. For this reason, YouTube was chosen as the second data source, where most Australian politicians' interview videos could be found. We first search for videos that are as high resolution and contain as many single face footages as possible. After which we download them and do screen shot for every several frames. This procedure yields a total of 2415 images.

The third source is social media platform, such as Twitter and Facebook, where politicians might post images and videos about their daily activities. Considering most of the data on social platforms is taken by phones, which means the resolution is relatively low compared to Google Image and YouTube, only politicians with less than 300 images after searching the first

two sources will be included in this third searching. This procedure yields 1290 images, so total of 5515 raw images were collected after three dataset collection phases.

3.2 Dataset Pre-processing

For the purpose of constructing an ideal dataset for training facial recognition models, the newly collected raw images need to be carefully processed to achieve a better performance in training. This section mainly introduces the methods used in pre-processing the dataset.

3.2.1 Cropping single person

Considering images collected, especially those from social platforms, often contain multiple people other than the targeting politician, manually take out this kind of images from the raw dataset is necessary to purify the dataset. These images will be cropped to only containing the targeting politician if resolution allowed (Figure 10). In other words, the cropped area needs to be at least 227 * 227 effective pixels because the deployment of AlexNet, otherwise the performance of dataset will be compromised.

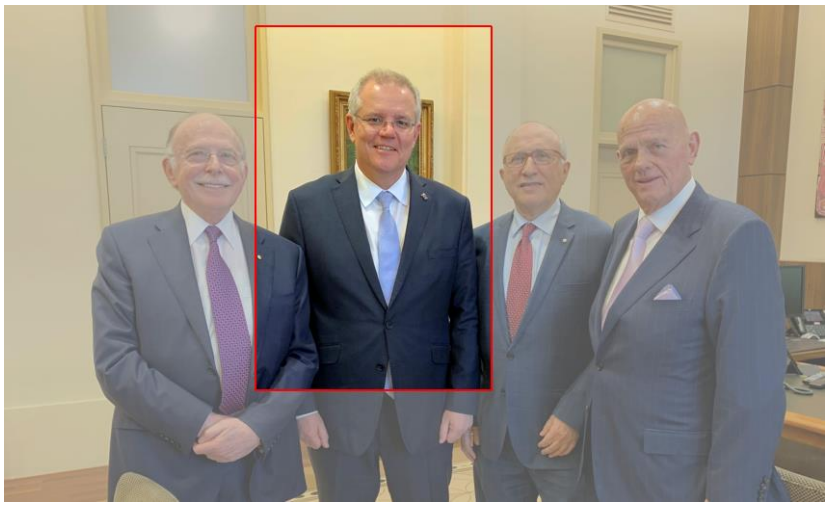


Figure 10. Cropping target from image

3.2.2 Removing “Bad” faces

Taking into account that we are training a recognition algorithm based on representative features on faces, every image should contain a clear face of the person. For example, faces of different people might look similar if they are both from side views or they had makeups. For this reason, another round of inspection is required to remove these “bad” images. In this step, images which containing an unrecognizable head orientation will be removed. And abnormal faces (for instance, heavy makeup or hair covers face) will also be removed from the dataset. This step allows us to further purify our dataset.

3.2.3 Batch face extraction

Because we are trying to train an object classification algorithm for face recognition purpose, the images we pass to training should only contain faces and nothing else. Objects like background and clothing should not be passed to algorithm unless we want algorithm to

recognize the person with specific clothes and background. To serve this purpose, we found two kinds of face detectors: dlib library with its HOG (histogram of oriented gradient) based fact detector [11] and OpenCV library with its pre-trained deep neural networks face detector [12] to batch process the images. Experiments show that the later algorithm had better performance over our dataset (Figure 11). Therefore, we decided to go with OpenCV's face detector model.

Figure 11. Comparison between face detection algorithms

This model automatically detects faces in images and generate information that describes rectangular boxes bounding the faces. It is not perfect and sometimes give false alarms and detect multiple faces on our purified dataset. After we modified it, the algorithm automatically crops faces with higher success rates in the following fashion: A. Faces are cropped as squares instead of the original rectangles because of the un-distortable property of face as discussed in section 3.3.1 B. Faces are cropped for a slightly larger area for data augmentation process in section 3.3.2 (Figure 12). Total of 4794 face images were extracted successfully in this step.

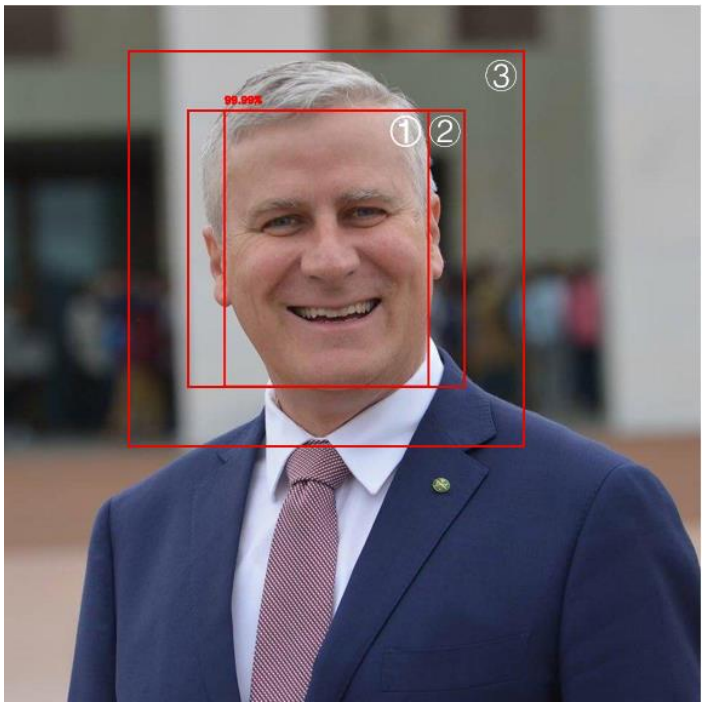


Figure 12. Batch face extraction

3.3 Dataset Augmentation

Despite the fact that we have nearly 90 images on average for every politician, it is still not enough to train a deep learning network and get a model with high accuracy. On average, ImageNet has around 1,000 images for every synset [9]. Therefore, dataset augmentation is needed to increase the number of images to enhance model's generalization ability during training [13]. This section mainly introduces the methods used in expanding the dataset based on the raw images we collected.

3.3.1 Un-distortable property of faces

Because we are focusing on human faces, common distortion methods to increase number of images are not allowed. For example, distortion methods like rotation and rescaling will be useless because variants generated from these methods will not be seen in natural conditions. In contrast, rotation on flowers or cats might be useful because different orientations of the object could be seen in common images. Due to this property of human faces, we need to use other methods to expand our dataset size and be extra careful when rescaling images to feed to models during training.

3.3.2 Random cropping

As presented in face extraction, faces are extracted slightly bigger on purpose to allow random cropping. In this procedure, some face images will be randomly (within a certain range) cropped several times to generate more images (Figure 13). The cropping will be done in such manner that each outcome will contain at least the face area of the targeting person.

Figure 13. Random cropping with features reserved

3.3.3 Gamma adjustment

Gamma adjustment are also used to simulate the brightness change of different shooting/displaying devices. This procedure will take some face images and adjust their gamma within a certain range to produce more images (Figure 14). This will be done in such fashion that each outcome is still recognizable for human eyes.

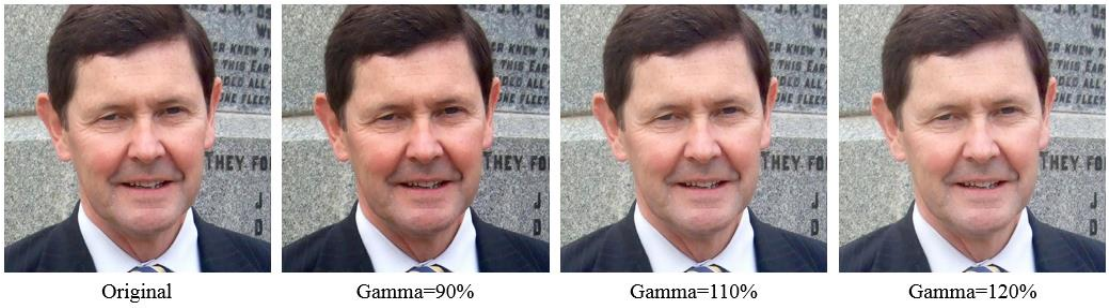


Figure 14. Adjusting an image's gamma

3.3.4 Mirroring

Often seen in tv shows, mirroring is a common method used by media to create a comfortable display layout to audience. Thus, some face images will be flipped horizontally in this procedure to enhance the diversity of dataset.

3.4 Dataset Summary

Total of 5515 raw images were collected from Google Image, YouTube, Twitter, and Facebook. Total of 4794 unique face images were left after dataset pre-processing. Total of 11611 images were generated during dataset augmentation so that is 16405 images in total (Figure 15).

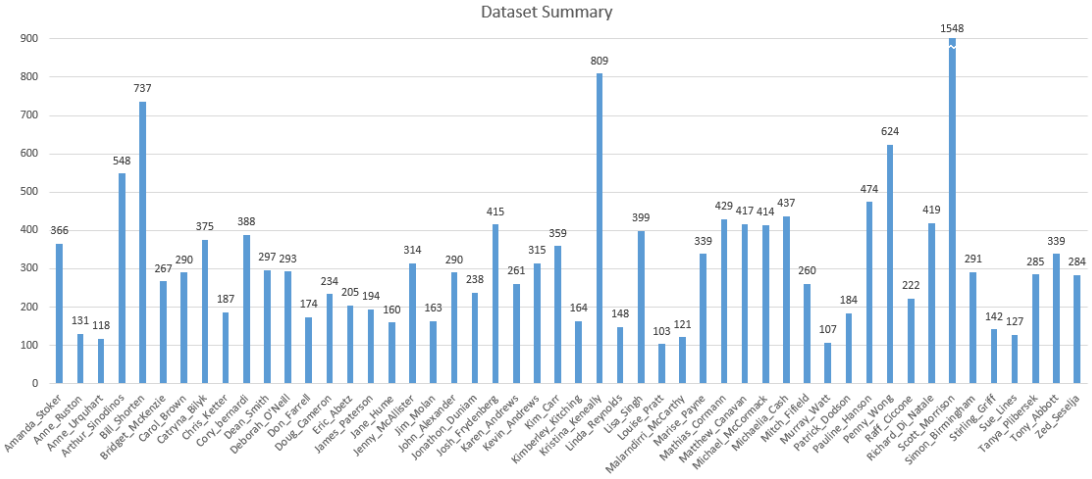


Figure 15. Dataset Summary

3.5 Pre-training processing

Images could be further processed before being handed over to training. Normalization is a way to "center" data by normalizing pixel values. It is done by using the following formula (2), where N is the normalized dataset, X is the original dataset, $\bar{x}$ is the mean of X, and σ is the standard deviation of X. This might convert images to an unrecognizable form to human eyes, but it is proven to help and speed up the learning process of neuron networks [14].

$$N = \frac{X - \bar{x}}{\sigma x} \quad (2)$$

4. Training

This section demonstrates the design choices we made during the training of CNN networks, and present the workflow of fine-tuning them to help understanding how we did it and what are the results we expect.

4.1 Evaluation metrics

The following terms were used in training to evaluate the performance of our deep learning models: accuracy and cross-entropy. It is important to understand these evaluation metrics before training and analyzing the results.

4.1.1 Accuracy and Top-k accuracy

To better understand accuracy, we introduce the concept of multiclass confusion matrix. Multiclass confusion matrix is an n x n matrix containing n^2 conditions that the outcomes of a predicting algorithm might falls in to, where n is the number of classes (Figure 16). Confusion matrix serves as an important role in the field of machine learning [15]. The total accuracy we use here is calculated by taking the sum of correct predictions for each class divided by total predictions made. Total Accuracy might be misleading if the number of images for different classes varies a lot, that is, unbalanced. For example, if a test set is consisting of 95 man and 5 women, a classifier might be considered to be 95% accurate if it predicts all man, where in fact the classifier has 0% accuracy for women. However, this is not the case for our project, so we are safe to use accuracy as one of our evaluation metrics. We also use class accuracy to evaluate the prediction accuracy for specific classes, this is calculated by the correct predictions for a certain class divided by class population.

		Predicted Label				
		1	2	3	...	n
Actual Label	1					
	2					
	3					
	...					
	n					

$$Class\ Accuarcy = \frac{correct\ predictions\ for\ class}{class\ population}$$

$$Total\ Accuarcy = \frac{correct\ predictions}{total\ predictions\ made}$$

Figure 16. Multiclass confusion matrix

In field of computer vision, a neuron network is also considered to be accurate if the target label falls into the top 5 of its predictions. This is called top-k accuracy, where k is the number of top predictions allowed. In this project, top-5 accuracy will be evaluated in results analyzation.

4.1.2 Cross entropy loss

In deep learning, loss functions are used to help effectively training and producing accurate results. The loss function used in our training is cross entropy loss, which is a state of art

measurement of the error between predicted value and true value [16]. It is calculated by the following formula (3), where M is the number of classes, y is the binary indicator if class label c is the correct classification for observation o, p is the predicted probability of observation o for class c. During training, cross entropy loss will be measured at each step and its value will affect the backpropagation of our CNN networks. Thus, we are expecting a decrease of cross entropy loss overtime when training models.

$$-\sum_{c=1}^{M} y_{o.c} \log(p_{o.c}) \tag{3}$$

4.2 Transfer learning

An approach of deep learning is using pre-trained models as starting point of another computer vision task. This is called transfer learning [17]. Transfer learning utilizes weights trained from other datasets and can reduced training time dramatically. Due to the time scope of this project, transfer learning is adopted as it is time efficient. Further training also verified that this approach turns out to be quite useful and produces accurate models. Figure 17 shows the training accuracy of AlexNet with different degrees of transfer learning applied. In this project, we use pretrained weights from Caffe [18] for AlexNet and pretrained weights form Google [19] for MobileNet. The pre-trained weights are both trained by images from ImageNet for over 1000 categories.

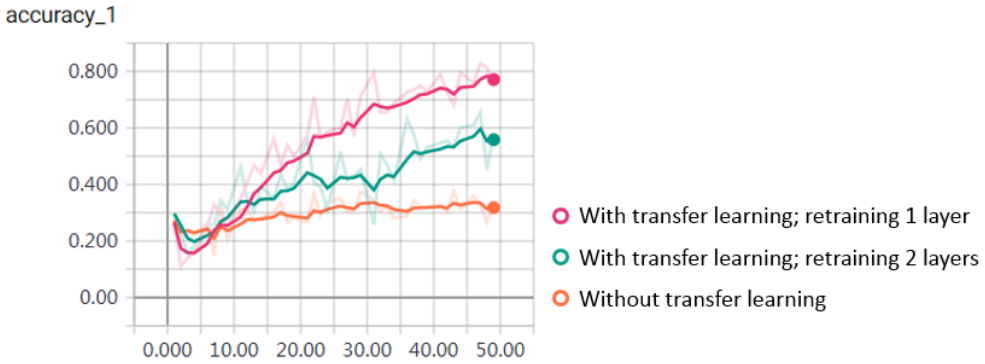


Figure 17. Different degrees of transfer learning over AlexNet

4.3 Hyperparameter Tuning

Hyperparameters are variables set to control the behavior of our training processes. They typically include but not limited to learning rate, batch size, and training epochs. In this project, hyperparameters of AlexNet and MobileNet will be adjusted to see how hyperparameters affects training outcomes, and to optimize our final model's accuracy.

4.4 Tensorboard

Tensorboard is a tool to visualize the training process of networks provided by Tensorflow [20]. It works by reading a log file generated by network during training in real time. In this project, Tensorboard is highly utilized in monitoring the accuracy and cross entropy loss during training and providing us the knowledges for better hyperparameter adjusting.

4.5 Training, Validation, and Testing Sets

Dataset is divided into three sets during training [21]. The largest set is the training set, which is 80% of the original Dataset. Training set is used to feed images into CNN networks during training, and the results will be used in backpropagation to shape networks' weights. In order to avoid overfitting, which is a potential problem in machine learning as discussed in 4.6, the validation set is there to validate if the training process is doing well. Validation set is 10% of the original dataset. Images in validation set will not be used to train models, but only used to validate the classifier on images it has not seen before. This helps monitoring the network's true accuracy over untrained images but not accuracy arising from overfitting. The last one is test set, which is 10% of the original dataset. It is used to test the accuracy of the final version of network.

4.6 Overfitting

When training a network, it is expected to memorize the characteristic features of our targeting objects, instead of memorizing unimportant details of training images to get the right answer. For example, a network might try to remember the pattern of background in training images and guess the label with background patterns. This will give high accuracy on images it has seen before in training but will fail when encountering new images because it did not learn enough about the general characteristics of the object (Figure 18). This problem is called overfitting, which is an inevitable problem in the area of machine learning [22]. To avoid overfitting our model, methodologies like dropout layer [23] are used in this project.

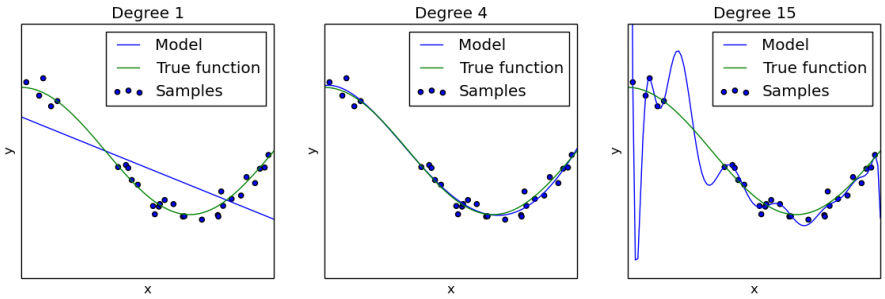


Figure 18. Example of underfitted, desired, and overfitted model

Early stopping is another technique used to avoid overfitting. Because overfitting begins to happen as the training steps increasing, early stopping gets its name from stopping early in training [24]. There are two signs of overfitting to be noticed: when training accuracy is much higher than validation accuracy, and when cross entropy loss stops decreasing (Figure 19). The first situation suggests that the network is doing much better on images it saw then it doesn't, and the second situation suggests that the network has learned everything from the image including irrelevant features like backgrounds. When these signs are spotted during training, we limit the training steps and training epochs to resist overfitting.

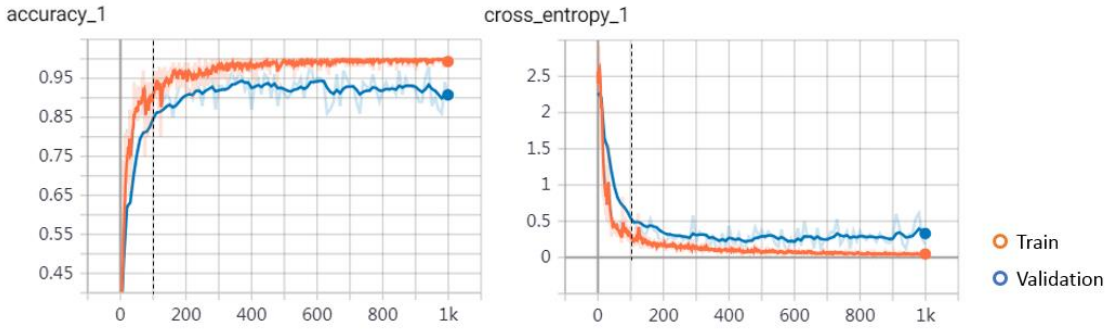


Figure 19. Signs of overfitting at step 100

5. Android Application Implementation

5.1 Motivation

With the rapid development of technology, smartphones have become an inseparable part of people's daily life. On average, people spend more than 3 hours on their phone each day. People rely on smartphones to do business, read news, playing games and more. According to latest statistics, there are more than 2.7 billion smartphone users around the world and around 75.22% of them are android users. Because we want to deliver our Australian politician recognition system as a convenient and user-friendly application, we decide to build an android version of our application.

5.2 Design Choice

In general, there are two ways to deploy our trained model on a mobile phone: client-server approach and on-device approach. The client-server approach, to be more specific, is to build a server which handles all the logics including inference of model, and let the client communicate with server to realize recognition functions. In this approach, client only serves as an information transmitter which takes & sends photo to server and receives & presents label information. This approach avoids the possible delays in inference caused by limited computation power of smart phones. Moreover, the server side, being on the same platform (desktop) with training algorithm, is much easier to deploy comparing to that of on-device approach, which includes format converting at inference.

However, this approach has several disadvantages. First, the server should be constantly maintained and up for 24 hours every day if we want to use it in anytime, which is infeasible for the scope of this project. Second, potential internet delay might influence performance of our application. During peak hours, our applications might be used more often because politicians have a higher chance to appear in peak hours. We definitely do not want our clients to have a bad user experience when they need it, therefore we decide to apply the on-device approach.

5.3 Prepare Model for Serving

After we got our trained model, there are a number of operations required in order to prepare

it for inference and serve it on mobile phone platforms. We choose to use trained MobileNet model because it is more accurate and much faster than AlexNet.

5.3.1 Freezing graph

Freezing graph integrates graph definition file with checkpoint files. Graph definition file contains information about the structure of a neuron network, and checkpoint files contain all the weights we got from training. Freezing a graph gets rid of unnecessary meta-data inside these files and encapsulate them into one single file.

5.3.2 Optimizing for inference

Optimizing for inference reduces the amount of computation needed in network to inference rather than training. For instance, operations used only in training like checkpoint saving and debugging can be removed in this step. This step could reduce the size of model for serving on mobile devices where computing power and storage space are scarce resources.

5.3.3 Format converting

A tool called TensorFlow Lite converter is used in this step to convert trained models into an optimized FlatBuffer [25] format before serving it on mobile devices. FlatBuffer is an efficient open-source cross-platform serialization library provided by Google, which provides a solution to reduce memory allocation and as a result enhancing on-device performance of our trained models.

5.3.4 Quantization

Quantization is a technique to reduce model size and latency introduced by Tensorflow [26]. The model is, however, made less accurate to accomplish improvements in size and latency. It works by quantizing weights from floating point to 8-bits of precision and converting them back at inference.

5.4 Interface Design

5.4.1 Politicians Recognition App

The user interface for this app is concise. There are two components, with the top section being real time camera and the bottom section being prediction presentation box (Figure 20). The logics behind this is: First, real time camera takes photos constantly and pass them to a classifier activity in background. Then, the classifier activity loads our trained model, receives the photos, and do inference to get a list of predictions. Finally. the top 3 predictions are displayed in the prediction presentation box. Tensorflow Lite is a framework dedicated in running Tensorflow like operations on a mobile phone. The framework helps a lot in constructing the classifier activity.

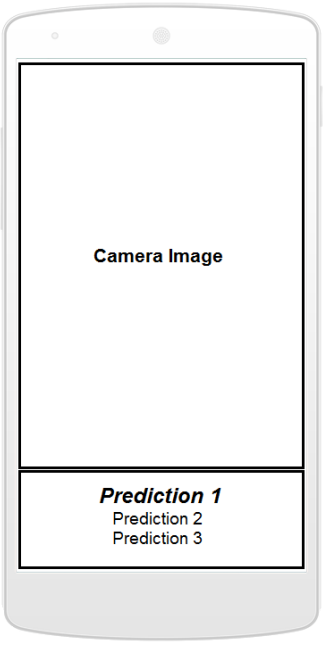


Figure 20. UI of Politicians Recognition App

5.4.2 Politicians Information App

The first user interface is straightforward with a scrollable list containing a picture and information about all of our targeting politicians and their parties as well as information about our team. The columns are clickable. On click, user will be directed to the second interface, and more information will be revealed. This interface shows a portrait and a biography of the selected politician. A small party icon is also presented next to the portrait to indicate the person's current political party (Figure 21).

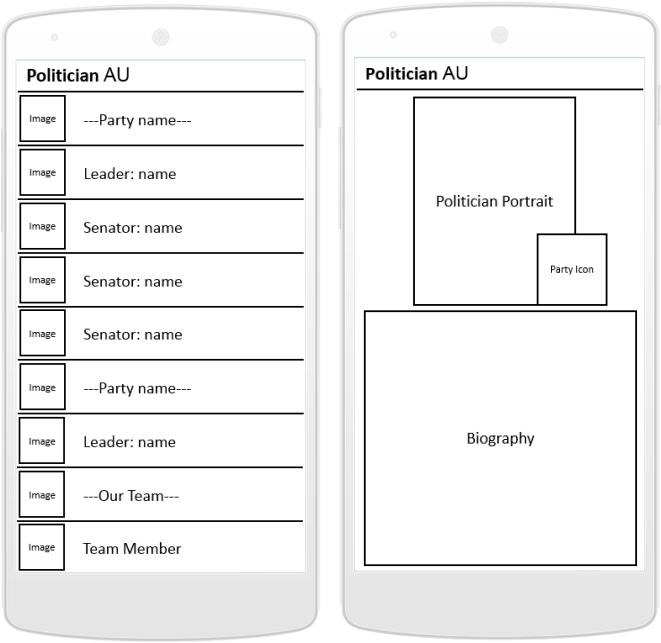


Figure 21. UI of Politicians Information App

6. Results and Analysis

As two CNN architectures (AlexNet and MobileNet) are used to train models for this project, this section will firstly show the training results of each architecture. Then a detail prediction accuracy result of each politician in the mobile application will be given. Finally, a discussion about the prediction result will be presented.

6.1 MobileNet Training results

Figure 22 shows the steps needed for batches with different sizes (256 images per batch, 512 images per batch and 1024 images per batch) to reach the best and stable accuracy. As is shown, the 1024-images batch only uses 200 steps to reach the best accuracy 0.8, while the 256-images batch uses almost 300 steps to do so. It is obvious that the number of steps to get the best and stable accuracy becomes smaller when the batch size becomes larger. The reason is that the features are easier to extract when more images are given to the model at one time.

Figure 23 shows the exact time for each batch with three different sizes to get a best and stable accuracy. As we can see, 256-images batch only uses less than 30 seconds, 512-images batch uses no more than 50 seconds, and 1024-images batch uses more than 80 seconds to reach the goal. The smaller the size of a batch is, the quicker the speed for that batch to get the best result is. Considering both step and time used to reach the best result, our group decided to use 512 images per batch to train to model for MobileNet.

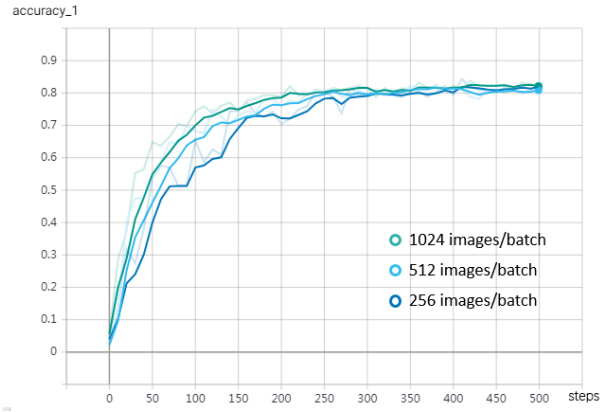


Figure 22. Steps for different batch sizes

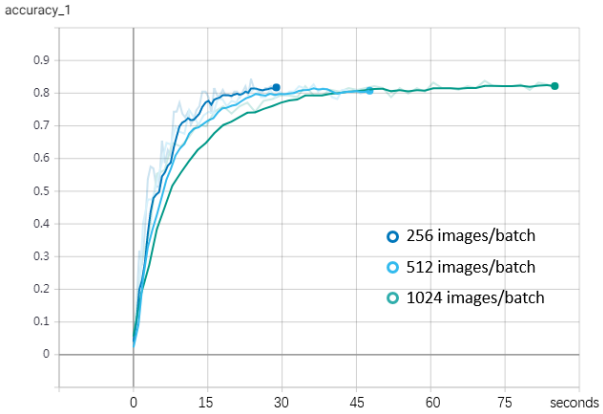


Figure 23. Time for different batch sizes

Figure 24 is the accuracy of MobileNet using 512-images batch. As is shown in the figure, the testing accuracy of the training dataset is 98% and becomes stable after 250 steps. The testing accuracy of the validation dataset is 80.47% and becomes stable after 250 steps. Both testing datasets present a rising trend in accuracy before 250 steps. The accuracy rises sharply in the first 60 steps for both datasets. However, the accuracy then rises smoothly and gradually becomes steady.

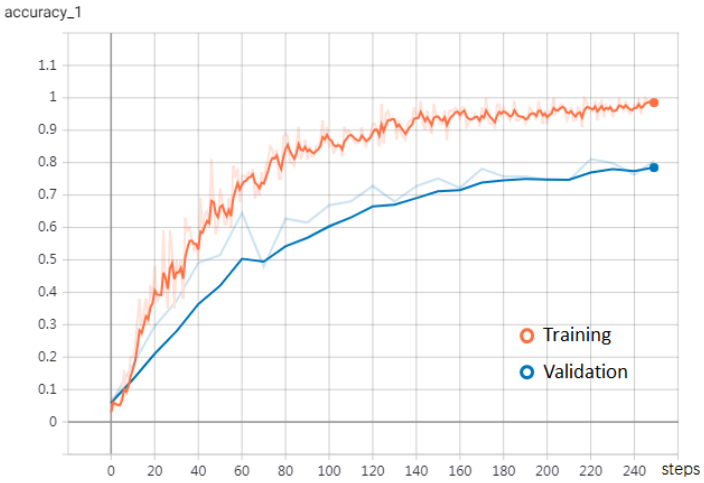


Figure 24. Accuracy result for MobileNet

6.2 AlexNet Training result

Figure 25 shows the AlexNet training result. As is shown, the general trend of both training and validation datasets is rising. Both results rise sharply at the first 60 steps. Then they grow smoothly with little undulation. Finally, the training dataset accuracy reaches 98% and becomes stable after 250 steps. The validation dataset accuracy reaches 73.44% and becomes steady after 250 steps.

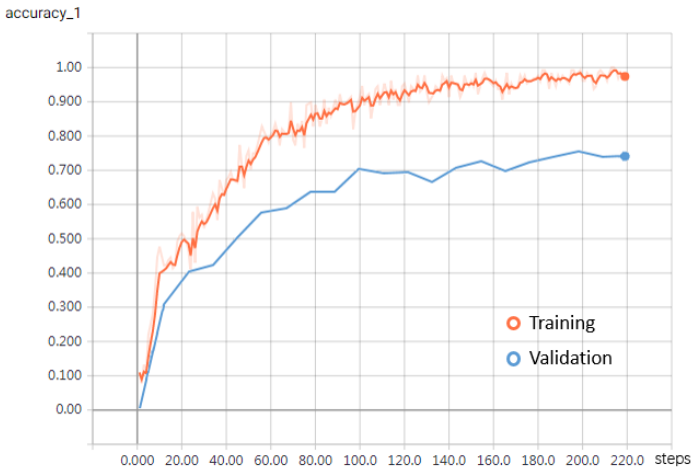


Figure 25. Accuracy result for AlexNet

Both models perform well with training and validation datasets, especially the training dataset accuracy reaches 98%. The reason for our group to choose MobileNet the framework of application is that the architecture is easy to implement and the prediction is accurate with little response time. As for AlexNet, it has a flex layer architecture with various functions to avoid problems like overfitting.

6.3 Mobile application functionality result and analysis

6.3.1 Mobile application functionality result

As the testing and validation datasets only contain images with politicians' pure frontal views, our group decided to test the mobile application with images from Google Image in order to check the actual application functionality. The images chosen for testing are politicians with different angles and actions. For each label, we tested 10 different pictures. A total of 500 images were tested for 50 labels. Figure 26 shows the accuracy results of the first prediction and top 5 predictions. The average first prediction accuracy is 61.2%, and the average top 5 predictions accuracy is 78.6%.

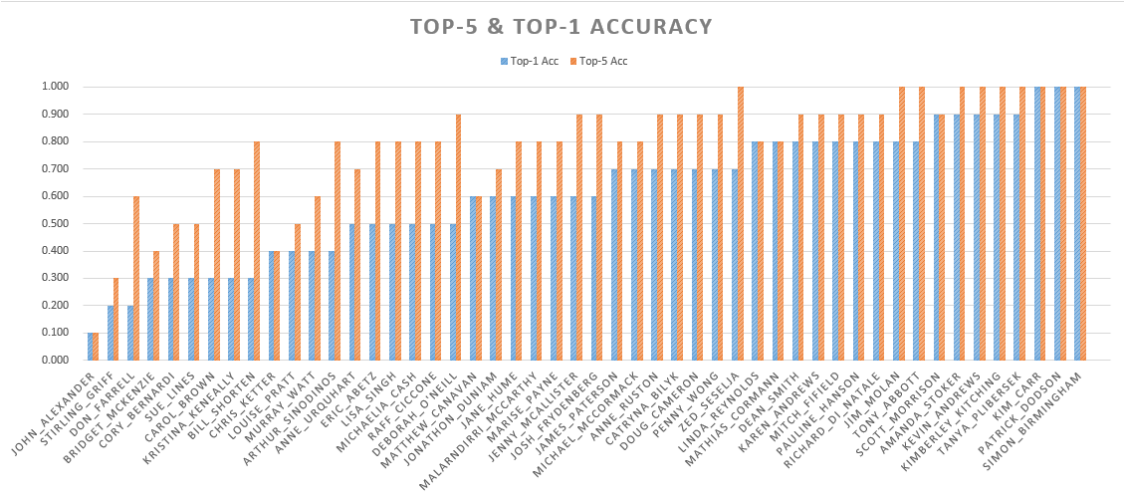


Figure 26. Mobile application functionality result

As is shown in the graph above, there are more than 60% politicians with a more than 60% first prediction accuracy, more than 80% politicians with a more than 70% top 5 predictions accuracy. For this specific test dataset, more than 90% politicians have a top 5 predictions accuracy more than 50%.

6.3.2 Analysis

Different politicians have various correct prediction percentages according to figure 5. Some of them have higher first prediction accuracy, some only present a high top 5 predictions accuracy. The reasons which may lead to this result are listed below:

High first prediction accuracy:

a. Unique appearance: Some people have unique appearances for machines to recognize, such as special beard type, unique face shape and special facial organs with different colors from others. For these people, the application can easily identify the unique appearances and present a high confidence. Figure 27 shows an example of unique appearances in the testing dataset collected from Google Image.

Figure 27. Politicians with unique appearances

b. Pure training dataset: Most of the successful prediction labels are always trained by pure training dataset. A pure dataset has two properties: first, there is no wrong or incomplete images in the dataset. Second, the images in the dataset are in large resolution ratio. The clearer the training pictures are, the more accurate the predictions are.

Low prediction accuracy:

a. Changing appearance: Some of the politicians changed their appearances in recent one or two years. Some of them lost weight, some just raised a beard. However, the training dataset only collects images of 2018 and 2019, while the test dataset is just pictures randomly chosen from Google Image. This is the reason why a low prediction accuracy occurs. Figure 28 is an example of a politician changed his appearance.

Figure 28. Politician changing appearance

b. Bad training dataset: Some obscure politicians only have a small number of blurring pictures on the Internet. Another reason for a bad training dataset is that the video of the politicians is not clear with noises. These factors caused that the training model did not extract enough features. As a matter of fact, the application cannot predict these politicians correctly, even not in the top 5 predictions.

Normal first prediction accuracy and high top 5 prediction accuracy:

a. Good CNN architecture: Even the first prediction is not correct because of multiple reasons like fuzzy dataset, there is still the right prediction inside the top 5 predictions, normally it is the second or the third prediction. The layer design and other functions help the training model

recognize the test images correctly.

b. Normal features: Some of the politicians have the similar appearances. As a matter of fact, the application will predict one politician as another one. With only small details different, the training model cannot extract all the different features.

7. Future Work

Though the mobile application meets the objective of this project, there are still four main tasks for future work.

- The dataset needs to be purer and add more politicians. In this way, the products are able to identify more politicians with larger accuracy.
- As different backgrounds of the images affect the accuracy of classification, we need to try object detection to compare the accuracy with image classification. Find a better training method for this project.
- The CNN architectures used in this project are AlexNet and MobileNet. On the one hand, we need to try other CNN architectures like GoogLeNet, VGG and ResNet, compare the results, and find the best architecture. On the other hand, we can adjust the layer sequence and add new layers with optimization functions to get better results.
- As for the mobile applications, combining the two applications together and improving the user interface are the future work.

8. Conclusion

This thesis firstly introduces the motivation of this project and some important backgrounds including deep learning, image classification and TensorFlow. After a brief introduction of traditional convolutional neural network, the implementation of AlexNet and MobileNet is discussed. Then relevant techniques and functions, which includes data crawling, pre-processing, mean subtraction, cross entropy loss function, are mentioned to apply to the CNN architectures. With an introduction of Android application development, the thesis provides a detailed implementation of the mobile application. Finally, an analysis part of the results and future work are discussed in the paper.

Appendix

Source code: https://github.com/HanwenZheng/PoliticiansAU_Recognition

Demo video: https://www.youtube.com/watch?v=YtuQyDYH6mk

References

[1] Schmidhuber, J., 2015. Deep learning in neural networks: An overview. Neural networks, 61, pp.85-117.

[2] Cireşan, D., Meier, U. and Schmidhuber, J., 2012. Multi-column deep neural networks for image classification. arXiv preprint arXiv:1202.2745.

[3] Shin, H.C., Roth, H.R., Gao, M., Lu, L., Xu, Z., Nogues, I., Yao, J., Mollura, D. and Summers, R.M., 2016. Deep convolutional neural networks for computer-aided detection: CNN architectures, dataset characteristics and transfer learning. IEEE transactions on medical imaging, 35(5), pp.1285-1298.

[4] Krizhevsky, A., Sutskever, I. and Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).

[5] Scherer, D., Müller, A. and Behnke, S., 2010, September. Evaluation of pooling operations in convolutional architectures for object recognition. In International conference on artificial neural networks (pp. 92-101). Springer, Berlin, Heidelberg.

[6] Wang, J., Yang, Y., Mao, J., Huang, Z., Huang, C. and Xu, W., 2016. Cnn-rnn: A unified framework for multi-label image classification. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2285-2294).

[7] Krizhevsky, A., Sutskever, I. and Hinton, G.E., 2012. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 1097-1105).

[8] 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*.

[9] Deng, J., Dong, W., Socher, R., Li, L.J., Li, K. and Fei-Fei, L., 2009, June. Imagenet: A large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition (pp. 248-255). IEEE Computer Society.

[10] Parkhi, O., Vedaldi, A., Zisserman, A. and Jawahar, C.V., 2012. The Oxford-IIIT PET Dataset.

[11] Dalal, N. and Triggs, B., 2005, June. Histograms of oriented gradients for human detection. In international Conference on computer vision & Pattern Recognition (CVPR'05) (Vol. 1, pp. 886-893). IEEE Computer Society.

[12] Guo, L. and WANG, Q.G., 2009. Research of Face Detection Based on Adaboost Algorithm and OpenCV Implementation [J]. Journal of Harbin University of Science and Technology, 5, p.034.

[13] Ahmad, J., Muhammad, K. and Baik, S.W., 2017. Data augmentation-assisted deep learning of hand-drawn partially colored sketches for visual search. PloS one, 12(8), p.e0183838.

[14] Ioffe, S. and Szegedy, C., 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167.

[15] Suominen, H., Pahikkala, T. and Salakoski, T., 2008. Critical points in assessing learning performance via cross-validation. In Proceedings of the 2nd International and Interdisciplinary Conference on Adaptive Knowledge Representation and Reasoning (AKRR 2008), Helsinki University of Technology (pp. 9-22).

[16] Zhang, Z. and Sabuncu, M., 2018. Generalized cross entropy loss for training deep neural networks with noisy labels. In Advances in Neural Information Processing Systems (pp. 8778-8788).
[17] Taylor, M.E. and Stone, P., 2009. Transfer learning for reinforcement learning domains: A survey. Journal of Machine Learning Research, 10(Jul), pp.1633-1685.
[18] Shelhamer, E. 2017. BAIR/BVLC AlexNet Model. Github. Available at: https://github.com/BVLC/caffe/tree/master/models/bvlc_alexnet
[19] Howard, A. and Zhu, M. 2017. MobileNets: Open-Source Models for Efficient On-Device Vision. Google AI Blog. Available at: https://ai.googleblog.com/2017/06/mobilenets-open-source-models-for.html
[20] Mané, D., TensorBoard: TensorFlow's visualization toolkit, 2015.
[21] Guyon, I., 1997. A scaling law for the validation-set training-set size ratio. AT&T Bell Laboratories, pp.1-11.
[22] Lawrence, S., Giles, C.L. and Tsoi, A.C., 1997, July. Lessons in neural network training: Overfitting may be harder than expected. In AAAI/IAAI (pp. 540-545).
[23] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I. and Salakhutdinov, R., 2014. Dropout: a simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research, 15(1), pp.1929-1958.
[24] Prechelt, L., 1998. Early stopping-but when?. In Neural Networks: Tricks of the trade (pp. 55-69). Springer, Berlin, Heidelberg.
[25] Luckcuck, M., 2016. Safety-Critical Java Level 2 Application Model: FlatBuffer.
[26] 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).