

# School of InfoComm Technology

**Deep Learning Assignment**

Diploma in CSF / FI / IT

April 2021 Semester

**ASSIGNMENT 1**

(30% of DL Module)

11st May 2021 – 13rd Jun 2021

**Submission Deadline:**

**Presentation: 13rd Jun 2021 (Sunday), 11:59PM**

**Report: 13rd Jun 2021 (Sunday), 11:59PM**

|  |  |  |
| --- | --- | --- |
| **Tutorial Group** | **:** | **P02** |
| **Student Name** | **:** | Nicolas Teo Zhi Yong |
| **Student Number** | **:** | S10206093B |

**Penalty for late submission:**

10% of the marks will be deducted every calendar day after the deadline.

**NO** submission will be accepted after 20th Jun 2021 (Sunday), 11:59PM.

# 1. Table of Contents

[1. Table of Contents 2](#_Toc74505352)

[2.1 Problem 2](#_Toc74505353)

[2.2 Objective 3](#_Toc74505354)

[2.3 Approach 3](#_Toc74505355)

[2.4 Extra Features 3](#_Toc74505356)

[3.1 Data Preprocessing 4](#_Toc74505357)

[3.2 Data Loading 4](#_Toc74505358)

[4 Development of image classification models 4](#_Toc74505359)

[4.1.1 First Built from scratch model (Overfitting) 4](#_Toc74505360)

[4.1.2 First Built from scratch model (Prevent overfitting) 5](#_Toc74505361)

[4.2.1 Second Built from scratch model (Overfitting) 9](#_Toc74505362)

[4.2.2 Second Built from scratch model (Prevent Overfitting) 10](#_Toc74505363)

[4.3.1 MobileNetV2 (Overfitting) 12](#_Toc74505364)

[4.3.2 MobileNetV2 (Fine Tuning) 13](#_Toc74505365)

[4.3.3 MobileNetV2 (Prevent Overfitting) 15](#_Toc74505366)

[4.3.4 MobileNetV2 (Prevent Overfitting using Batch Normalization) 17](#_Toc74505367)

[4.3.5 MobileNetV2 (Preventing Overfitting using Layer Normalization) 18](#_Toc74505368)

[5. Evaluate models using Test images 19](#_Toc74505369)

[5.1 Evaluation of models with Test Accuracy 21](#_Toc74505370)

[5.2 Recommendation of Best Model 21](#_Toc74505371)

[6. Classification of Images using Best Model 22](#_Toc74505372)

[6.1 Analysis and Evaluation of Model Prediction 22](#_Toc74505373)

[6.2 Reasons for inconsistencies between Validation and Test accuracy and Predictions 23](#_Toc74505374)

[7. Summary 23](#_Toc74505375)

[References 24](#_Toc74505376)

# 2.1 Problem

This report requires the classification of 10 different food classes across 10,000 images through the use of a convolutional neural network (CNN). CNN is a deep learning algorithm that is able to take in images as input and assign varying importance to different parts of the image, through learnable weights, to differentiate between images (Saha, 2018).CNN does this through the use of ConvNets. ConvNets do this by extracting features from the input image through small squares of input data. The features extracted are then compared to different subsections of the image. This helps the algorithm best classify the different images through the comparison of features.

# 2.2 Objective

The objective of this report is to provide accurate comparisons between self-trained and pre-built CNN models that have the same image size while obtaining the highest possible validation accuracy for each model while reducing any overfitting or underfitting of the curve, while following the principles of universal workflow of machine learning.

# 2.3 Approach

The approach taken to build the models was first to understand that this problem has 10 different answers, but an image can only have 1 possible answer. Thus, this is a multiclass single-label classification problem. Next was to define the measure of success which is accuracy as this is a classification problem. The evaluation protocol used was to maintain a hold-out validation set although using K-fold cross-validation was also a viable choice as the data given was limited. I decided to not use K-fold cross-validation due to a lack of computational power. The process of using K-fold cross-validation would have taken considerably longer compared to using hold-out validation and there already is a small set of data saved aside for validation.

As this is a multiclass, single-label classification problem, it requires a last-layer activation function of softmax with a loss function of categorical crossentropy. Next the approach taken to train the models was to train it until the model overfitted. My goal for this portion was to push the training accuracy to as close to 1.0 as I could before tuning it. After achieving an overfitting graph, I started to prevent overfitting by adding dropout layers, kernel regularizers, batch normalization, layer normalization and changing hyperparameters like the learning rate of the optimizer, the number of epochs, the batch size and the steps per epoch. Due to the lack of data present, I have also used Data augmentation to increase the diversity of data available for training models. Some parameters I used was to rotate or flip the image.

# 2.4 Extra Features

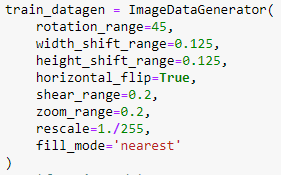
Apart from using Dropout layers and regularizers, another key feature I used was batch normalization and layer normalization. They were used to help with overfitting of the curve. batch normalization helps to improve the stability and performance of the model by restricting the distribution of the input data to any particular activation layer in the network and this helps to produce better gradients for weights update (Vijayrania, 2020). In comparison, layer normalization normalizes the activations along the feature direction compared to batch normalization which does it in mini batches (Vijayrania, 2020). The effects of batch normalization and layer normalization are compared later in the report through the use of the prebuilt MobileNetV2 models.

# 3.1 Data Preprocessing

The 101 food classes present in the Kaggle download were logically separated into the assigned 10 food classes. This was done through the use of the given image preprocessing jupyter notebook file. This resulted in having 1000 images per food class. 750, 200, and 50 images were split between the training, validation, and testing images respectively per food class. In total, there are 7500 training images, 2500 validations images and 500 testing images. I imported and loaded all the relevant libraries into the first cell as this increased my computer’s performance and allowed me to more easily keep track of the modules that I had imported.

# 3.2 Data Loading

For data loading, the image size used is 150. Each image goes through an ImageDataGenerator. The training images go through data augmentation that would rotate the images by 45 degrees, shift the height and width of images by 0.125, horizontally flip the images, zoom the images in and shear the images by a multiplier of 0.2, are resized to 1./255 and the fill mode is set to nearest.



The validation and test images are only rescaled and do not go through any other form of data augmentation.



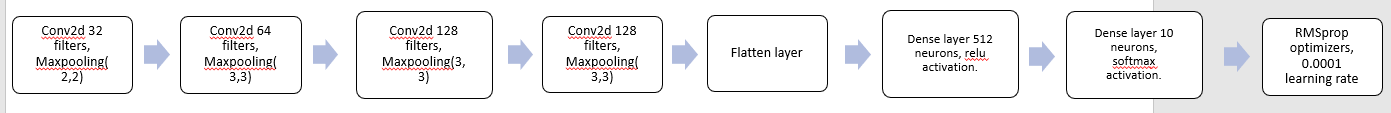
Training, validation, and test generators are also used to feed the relevant images to Keras. 20 images are fed to Keras, through the flow from directory function by Keras, upon request and classification mode is set to categorical as this is a multi-class classification problem.

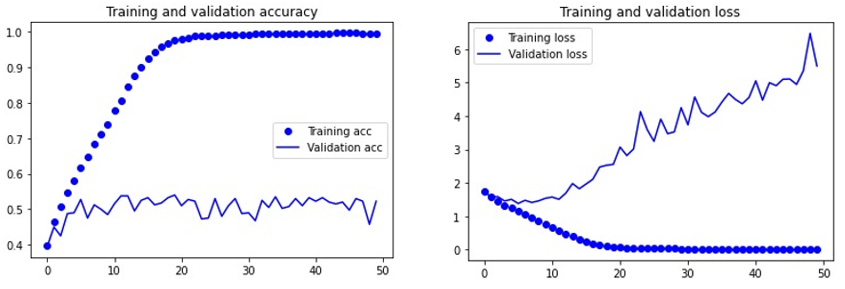
# 4 Development of image classification models

In this section I would be going through 3 different image classification models. Two of which were built from scratch and one which utilize pre-built models. The difference between the 2 image classification models that were built from scratch is how the network is structured. The pre-built model utilized is MobileNetV2.

# 4.1.1 First Built from scratch model (Overfitting)

The first model built very scratch consist of 4 sets of 1 Conv2D layer and 1 Maxpooling layer, followed by flattening the layers. A dense layer with 512 neurons that uses relu activation function is then added and finally ends off with a dense layer of 10 neurons that uses the softmax function. 20 images were fed into the model at a time with each epoch performing 375 steps, fully utilizing the total training image data of 7500 images. As there is a smaller volume of data for validation, only 100 validations steps were needed to fully utilize the total validation image volume of 2000.

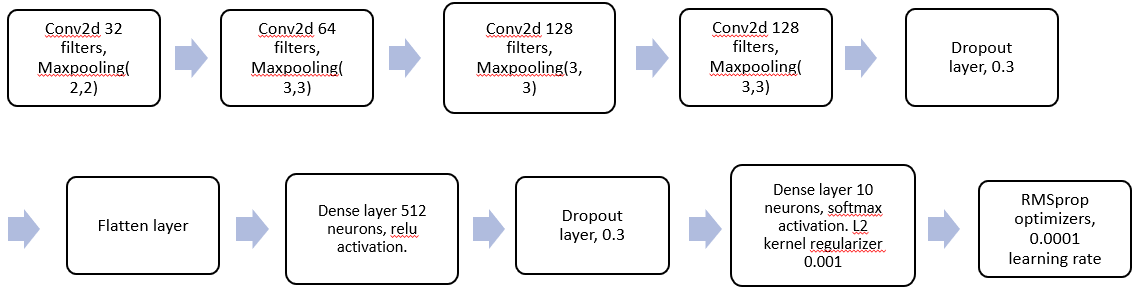


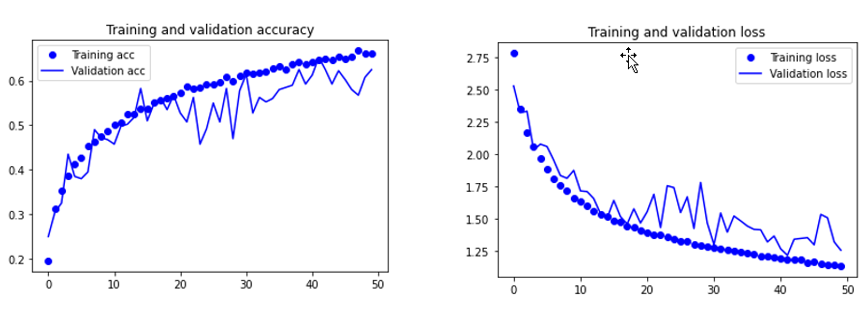


The results obtained shows an overfitting graph which is a good result. The next step is to slowly prevent overfitting.

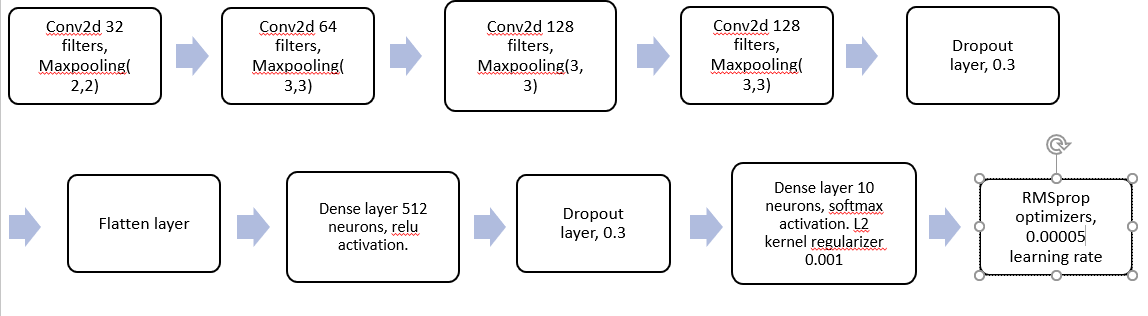
# 4.1.2 First Built from scratch model (Prevent overfitting)

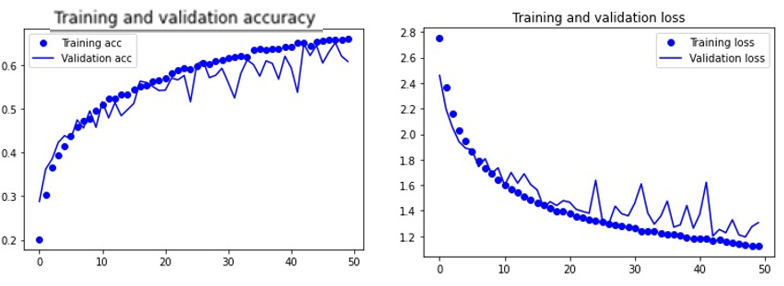
To prevent overfitting, there are 3 main methods. Using dropout layers, using kernel regularizers and using batch normalization (Brownlee, How to Avoid Overfitting in Deep Learning Neural Networks, 2018). For this model, as the model was extremely overfitted, I added 2 dropout layers with a dropout rate of 0.3 and 1 l2 kernel regularizer with a regularizer rate of 0.001.



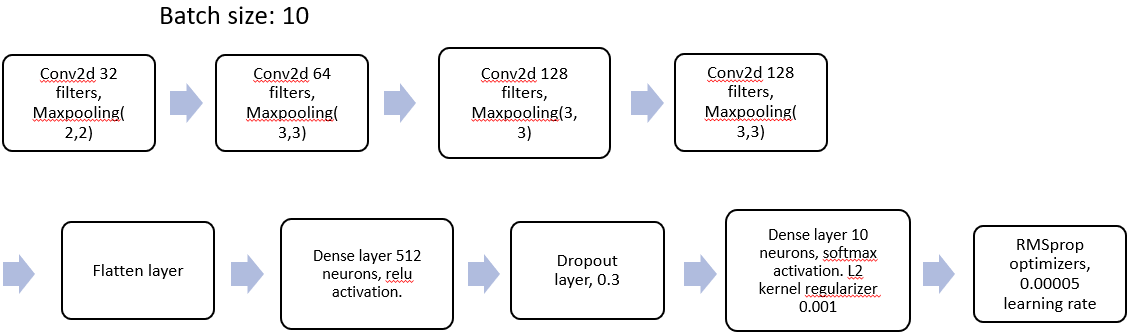


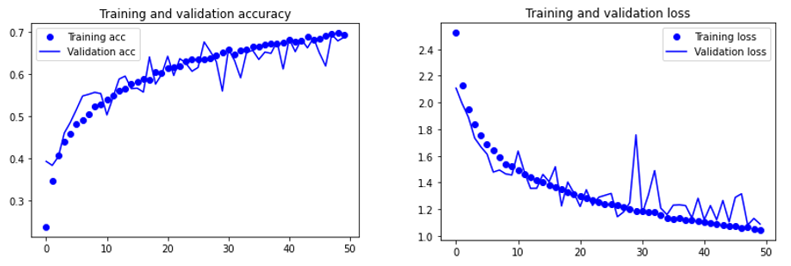
The resulting graph shows a slightly overfitted graph with many fluctuations to the validation accuracy. From the fluctuations and relatively low training and validation accuracy, I know that a low batch size and high learning rate are possible causes to the fluctuations. A good batch size to prevent fluctuations would be 32 (Brownlee, How to Control the Stability of Training Neural Networks With the Batch Size, 2019). However, due to lack of computational resources, a batch size of 32 was not possible so I stuck with 20. Thus, the learning rate of the model had to change. The learning rate was lowered from 0.0001 to 0.00005 (Brownlee, Understand the Impact of Learning Rate on Neural Network Performance, 2019).



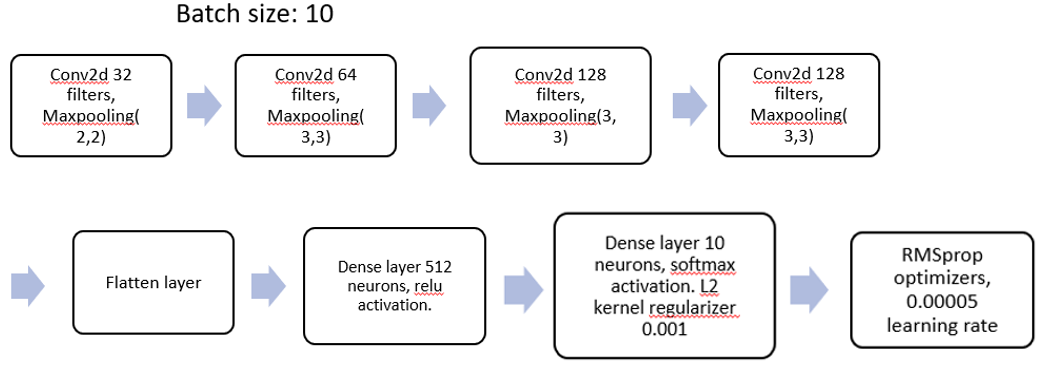


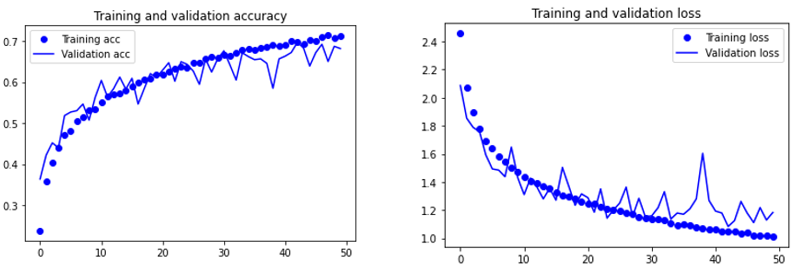
From the graph, we can see that the fluctuations on the validation accuracy are less frequent, yet the training accuracy and validation accuracy are still relatively low. As the graph is of good fit, I decided to remove one dropout layer as well as decrease batch size to 10 to increase validation accuracy (Brownlee, How to Control the Stability of Training Neural Networks With the Batch Size, 2019).



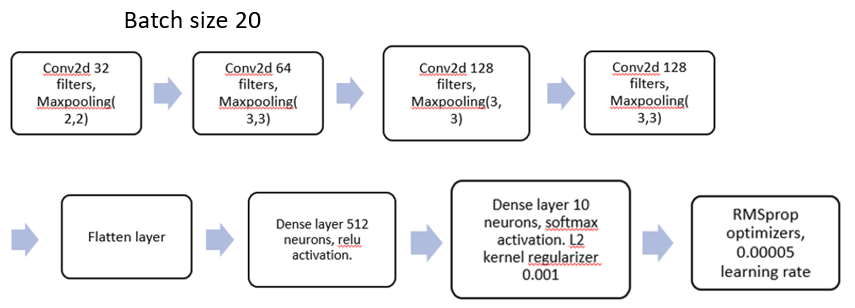


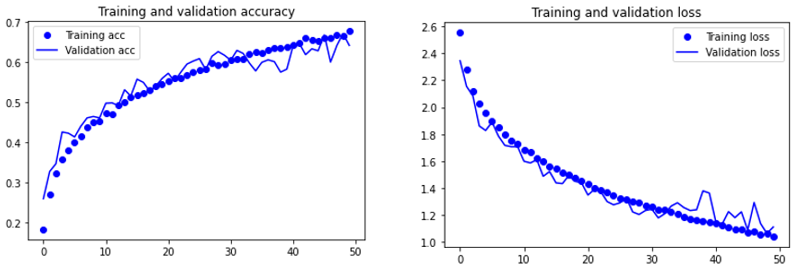
The validation accuracy and training accuracy greatly increased however fluctuations of accuracy also increased. This could also be due to the dropout layer and the randomness of how the dropout layer works (Brownlee, A Gentle Introduction to Dropout for Regularizing Deep Neural Networks, 2018). Furthermore, the graph is of good fit thus I decided to remove another dropout layer.



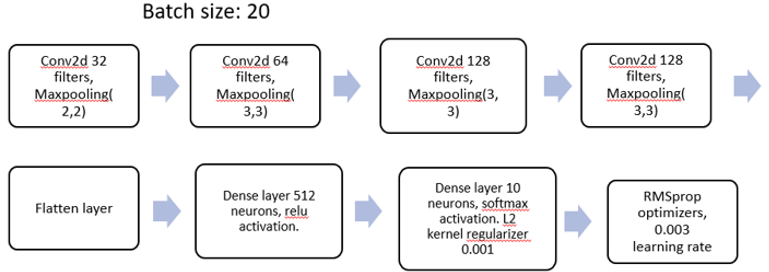


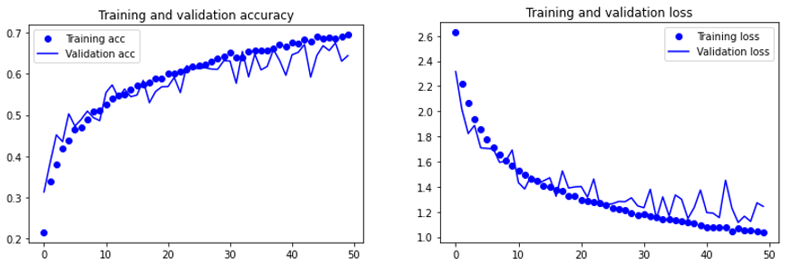
The removal of the dropout layer did not make many changes, thus I decided to increase batch size back to 20 as a small batch size would create more fluctuations in data.



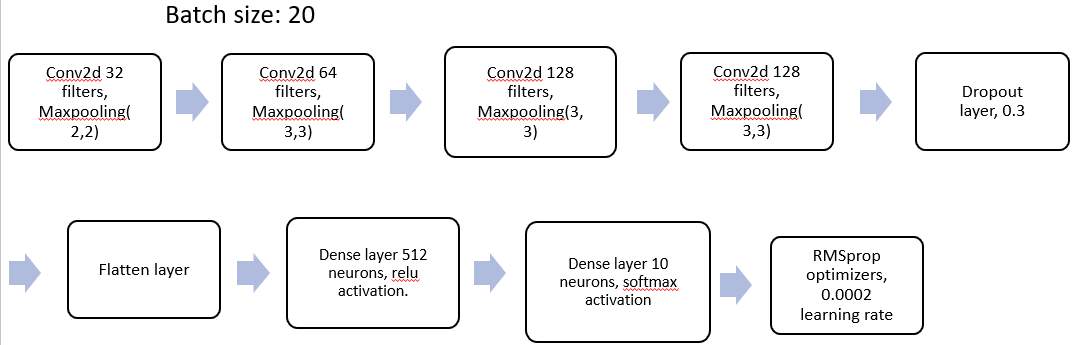


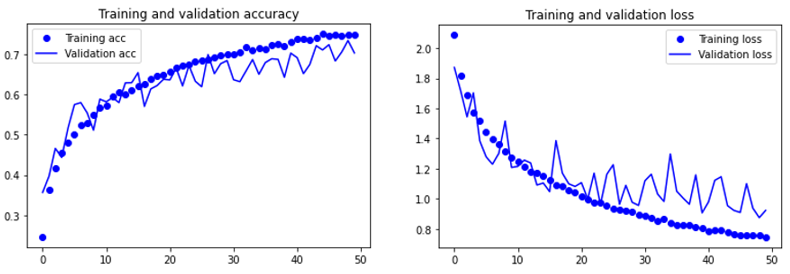
This improved the stability of the model and less fluctuations were noted. I also note that the graph still has an increasing trend, therefore I increased the learning rate to 0.003.





The fluctuations increased again and increasing the learning rate did little to improve my model. After continuous testing, this was my final model. The l2 kernal regularizer was removed, a dropout layer of dropout rate 0.3 was added and, learning rate was tuned down to 0.0002.

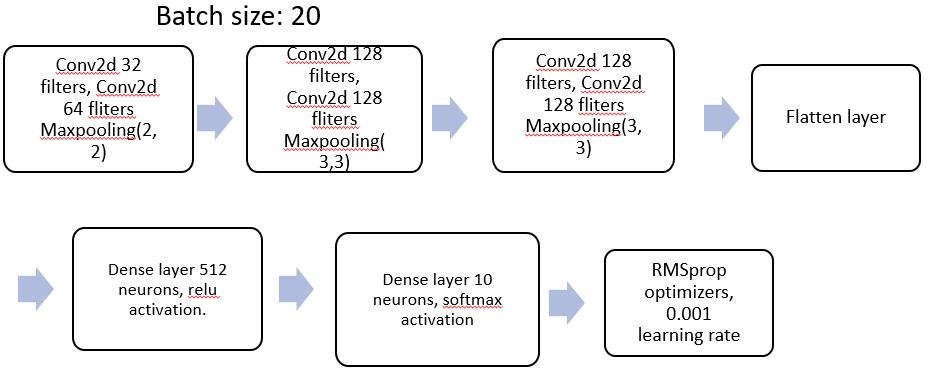


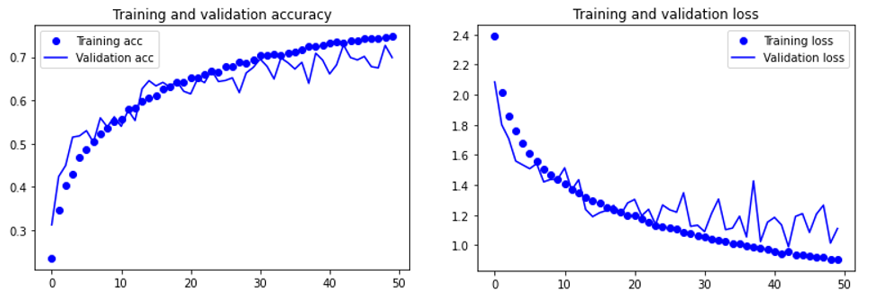


# 4.2.1 Second Built from scratch model (Overfitting)

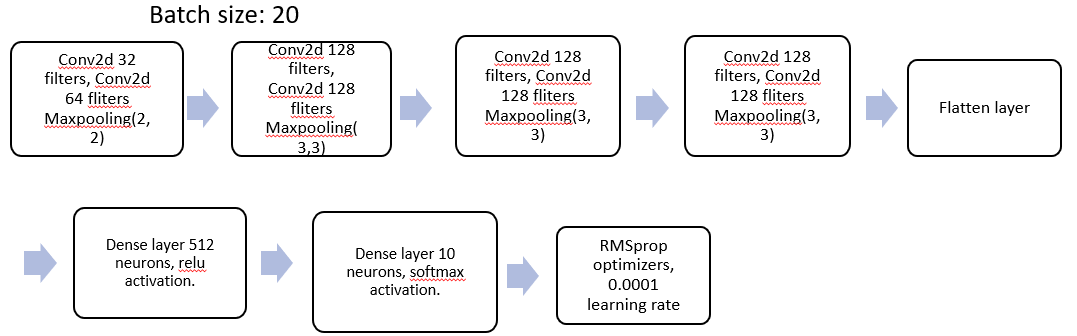
For the second built from scratch model, I decided to put 2 conv2d layers together followed by a maxpooling layer. I did this to increase the network size to increase validation accuracy.

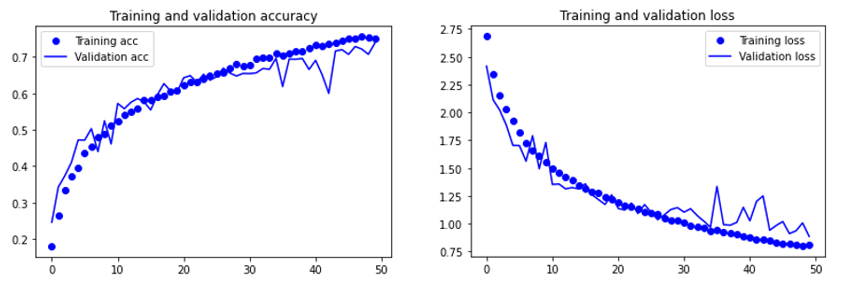
The first model is as follows.





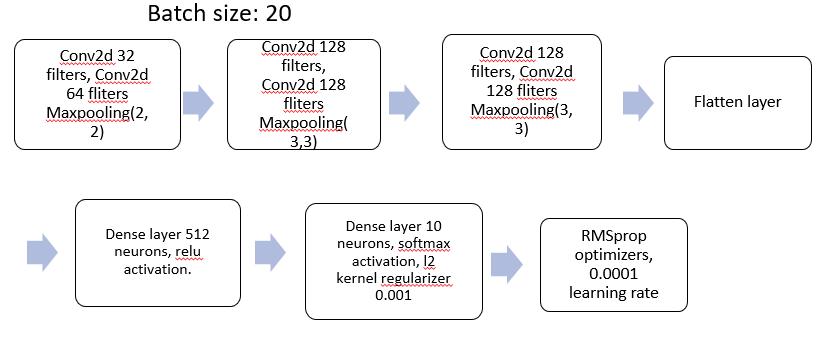
The result shows a model of good fit with a slightly higher accuracy compared to the first built from scratch model. However, the model still looks to be increasing, therefore I added another set of 2 Conv2d layers and a max pooling layer.

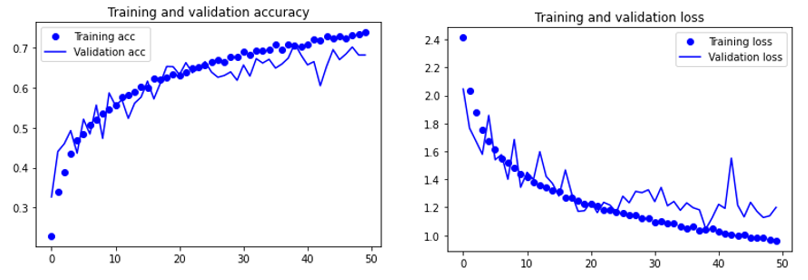




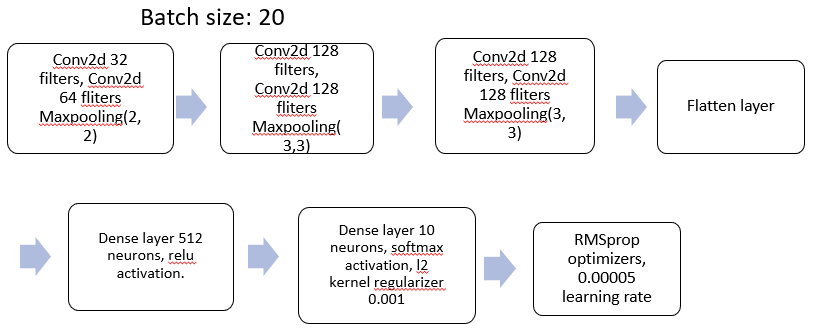
The model is generally a good fit however the addition of the extra Conv2d layers seemed to not have much impact on the model therefore I removed them. Furthermore, towards the end, there seemed to be some fluctuations which made the graph slightly overfit. Hence, I added a l2 regularizer of rate 0.001.

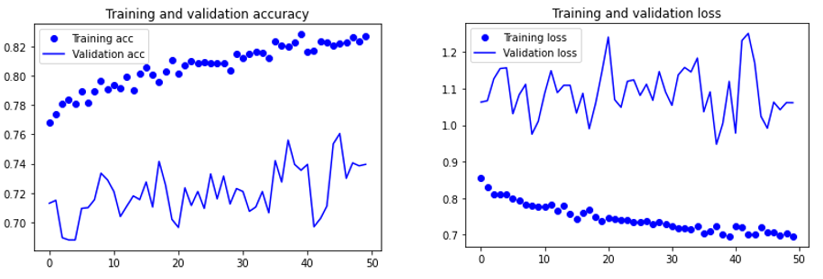
# 4.2.2 Second Built from scratch model (Prevent Overfitting)





The overfitting of the model did not reduce, therefore I decided to experiment with lowering the learning rate to obtain a graph of good fit. I lowered the learning by half to 0.00005.

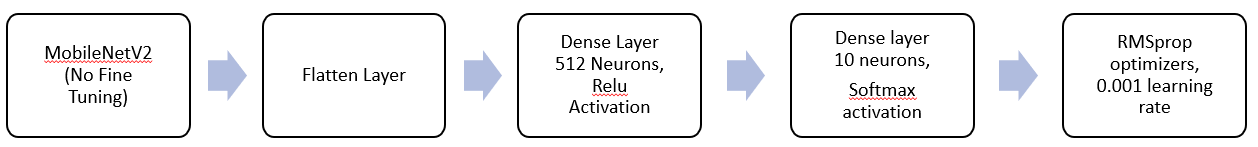


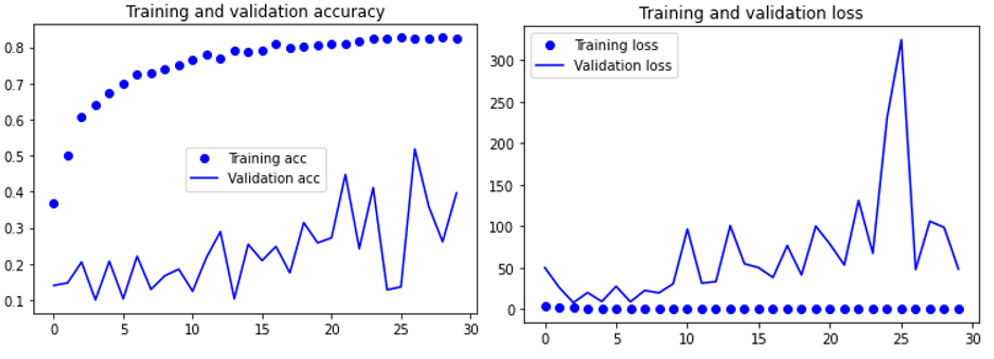


This yielded unexpected results. The training accuracy and validation accuracy increased, and training and validation loss decreased. Even though there looks to be a high fluctuation of validation accuracy, it fluctuates within a small range of 0.70-0.73 and as the graph is extremely scaled in, it is unfair to judge the results as fluctuating. Even though the graph is slightly overfitted, I feel that the overfitting is not extreme and thus I would be using this as my final model for the second built from scratch model.

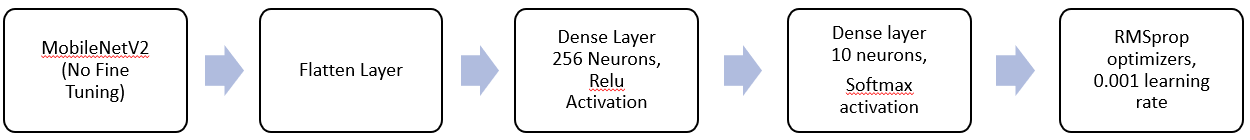
# 4.3.1 MobileNetV2 (Overfitting)

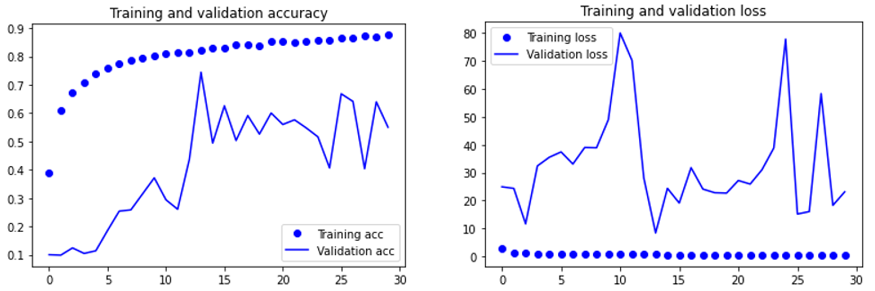
The pre-built convolution base used imagenet weights. The model built consisted of the convolution base followed by a flatten layer, a dense layer of 512 neurons with relu activation function and a dense layer of 10 neurons with softmax activation. The model used RMSprop optimizers with 0.001 learning rate. 30 epochs were used for this model. The resulting model was less than ideal.





Even though the model overfitted, the training accuracy only reached a high of approximately 0.82 while the validation accuracy was constantly fluctuating and inaccurate. Thus, to prevent the fluctuations and the extremely high validation loss, I lowered the the dense layer to 256 neurons instead of 512 neurons.

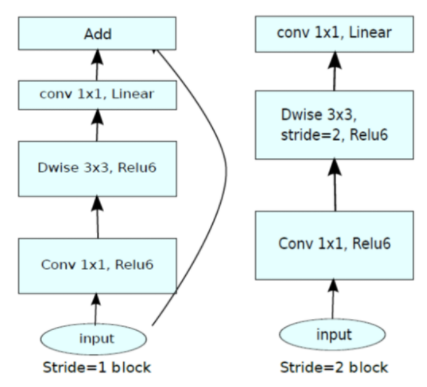




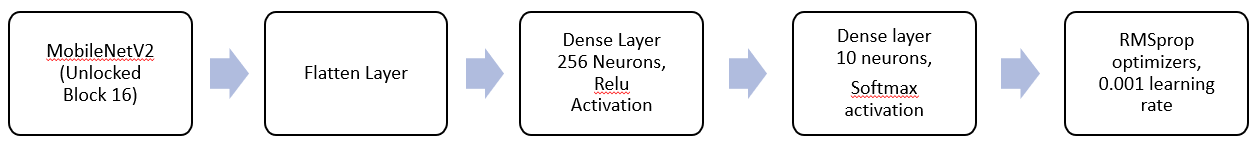
This resulted in less small fluctuations but extreme spikes in results. The training accuracy increased to approximately 0.9 which is what I wanted. Hence, I can now move on to fine tuning the model by unlocking Conv2d blocks and preventing overfiiting.

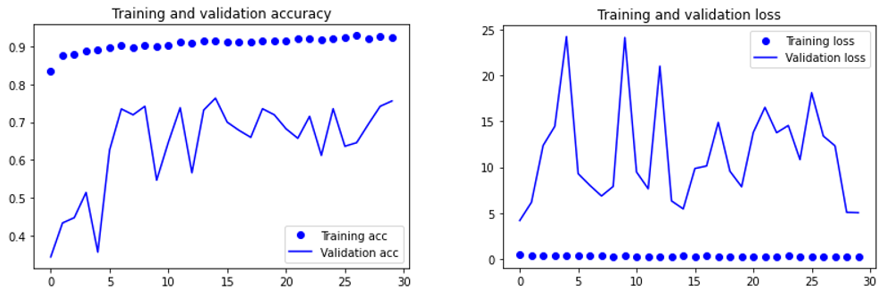
# 4.3.2 MobileNetV2 (Fine Tuning)

MobileNetV2 architecture consist of 3 layers which are made out of 2 types of blocks. A residual block with stride of 1 and another block with stride of 2 for downsizing (Tsang, 2019).

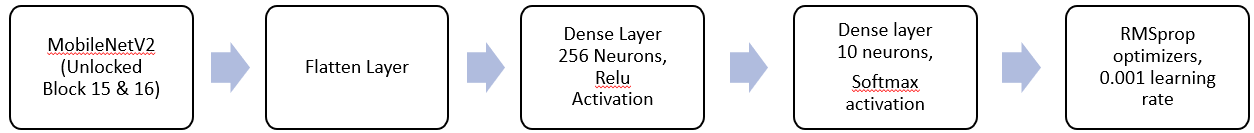
(Tsang, 2019)

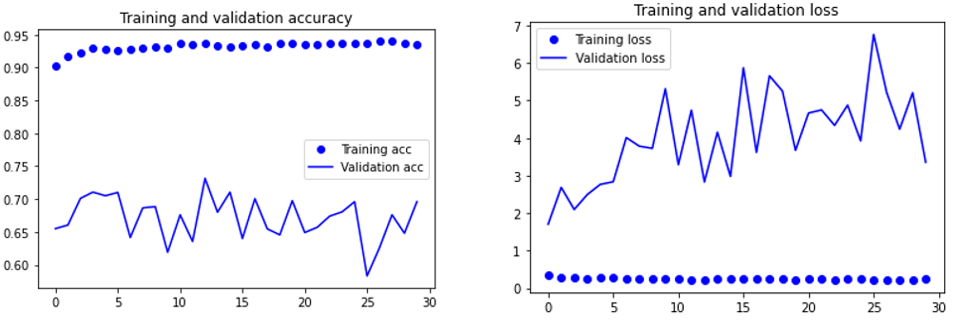
The first layer is a 1x1 convolution with Relu6. Relu6 is used due to its robustness when used with low-precision computation (Liu, 2017). The second layer is the depthwise convolution. The third layer is another 1x1 convolution without any non-linearity (Tsang, 2019). These three layers are made up of 16 different blocks. For fine tuning, I decided to unlock just one block first, block 16 and work my way from there.



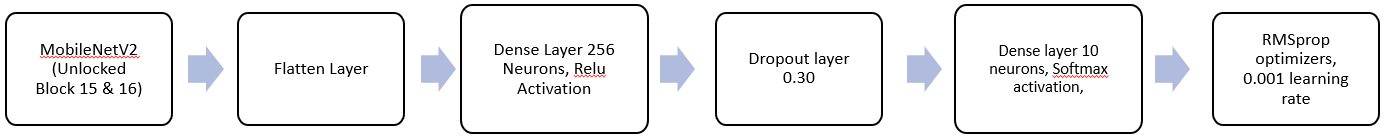


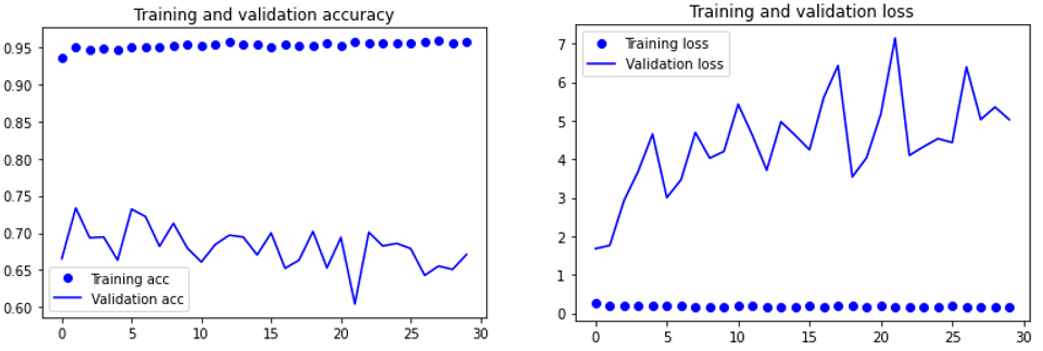
The graph shows that even though training accuracy has improved, fluctuation of validation accuracy still occurs. This tells me that in order to better finetune the validation accuracy to make the graph smoother, I should fine tune more layers (India, 2019). Thus, I also decided to fine tune block 15.



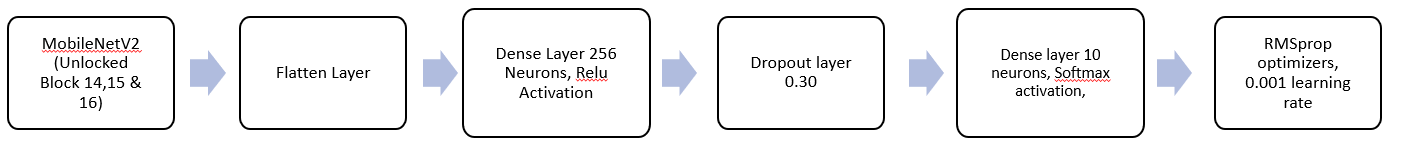


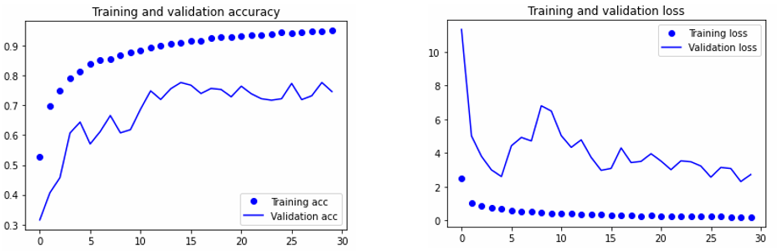
A significant decrease in validation loss from a peak of 24 to 7 shows that fine tuning the extra layer worked. The training accuracy has also slightly increased however the validation accuracy has not moved much. The graph is still very overfitted thus I decided to add a dropout layer of dropout rate 0.3 to reduce the overfitting of the graph.





Little change could be noted apart from lesser fluctuations in validation accuracy. Thus, I decided to tune the learning rate to make the validation accuracy more stable but to little success. Thus, I decided to fine tune the next layer, layer 14.

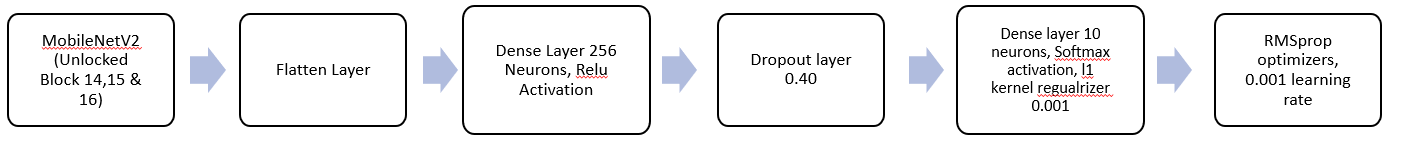


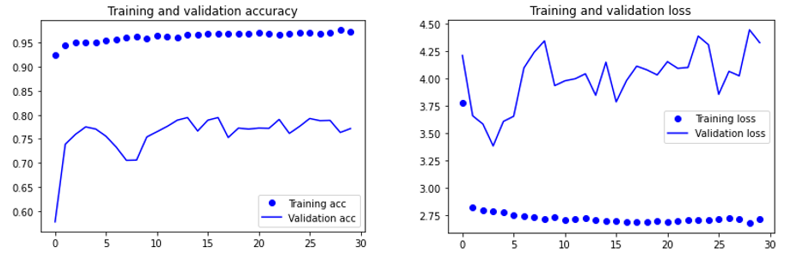


The fine tuning of the next layer shows significant improvement as the graph is not as overfitted anymore and also shows an increasing validation accuracy and training accuracy. From this, I believe that it is time to start to prevent overfitting as the limits of this network have been tested and I do not think I can push the validation accuracy or training accuracy any higher.

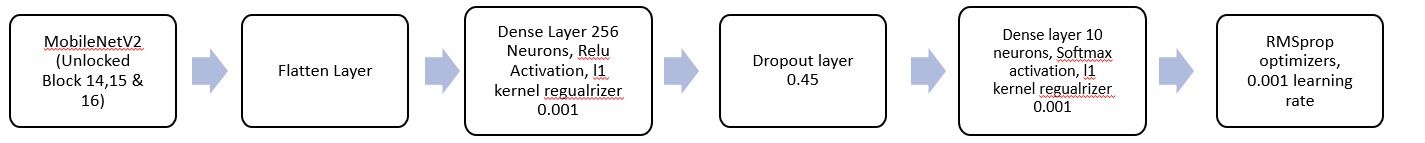
# 4.3.3 MobileNetV2 (Prevent Overfitting)

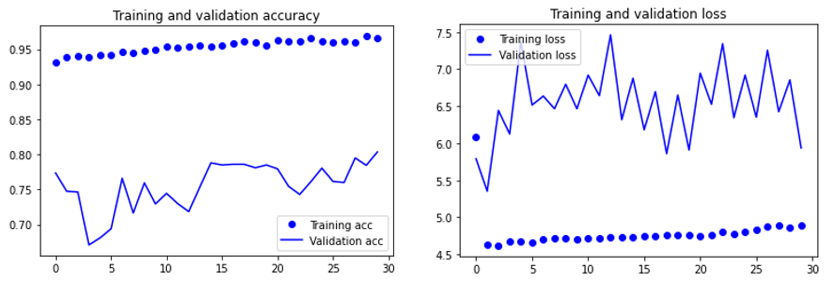
The first step I decided to take was to add a l1 kernel regularizer of rate 0.001 and increase dropout rate to 0.4. This would help to prevent overfitting of the curve as much.



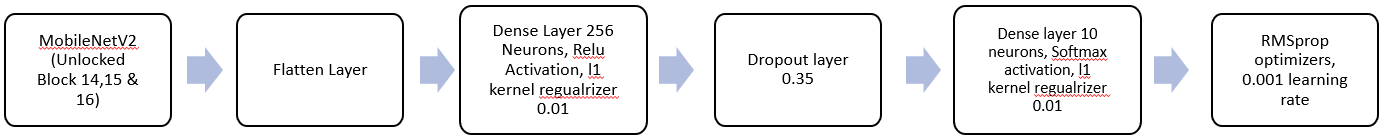


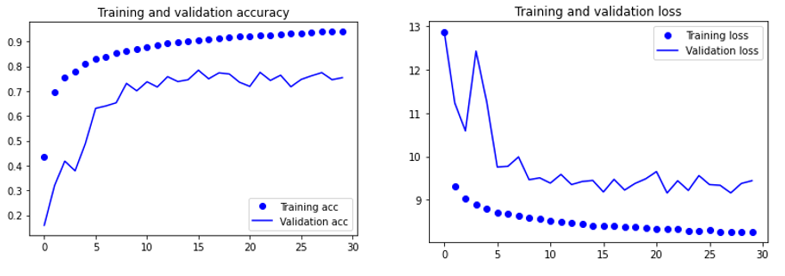
The result, however, was abysmal and even though the validation accuracy faced lesser fluctuations, the problem of overfitting came back. This prompted me to be more aggressive in preventing overfitting and I added another l1 regularizer as well as increased dropout rate to 0.45.





This also came with the downside of a high fluctuating validation loss. After testing with different dropout rates, I have reached a conclusion that any dropout rate higher than 0.4 would not benefit my model and thus I would be sticking to dropout rates of 0.30 or 0.35. I decided to increase kernel regularizer rate instead to 0.01 and dropout rate to 0.35.

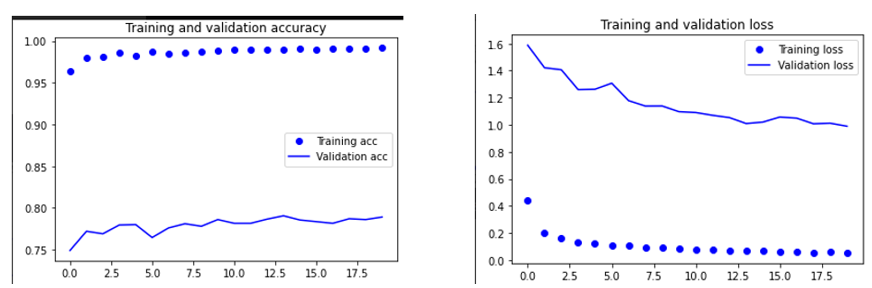
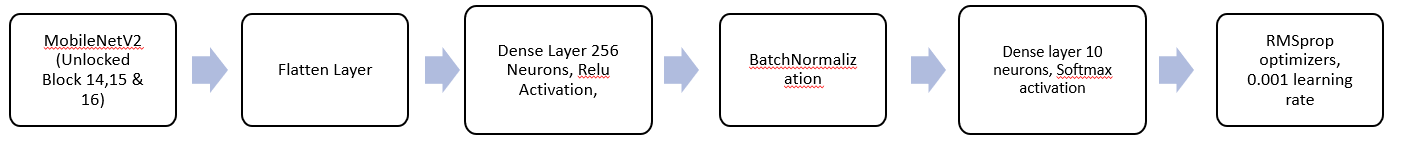




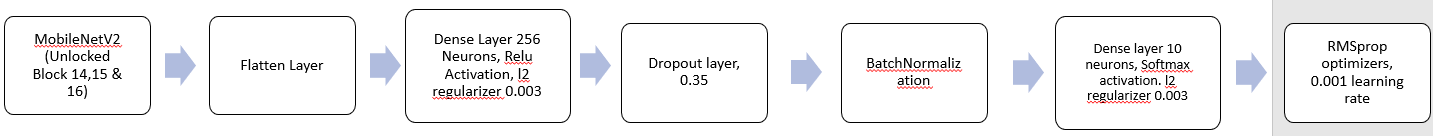
At cost of validation accuracy and training accuracy, the graph is of much better fit now. After experimenting with the hyperparameters more, I found this graph to be the best fit however, there are other hyperparameters I had yet to explore.

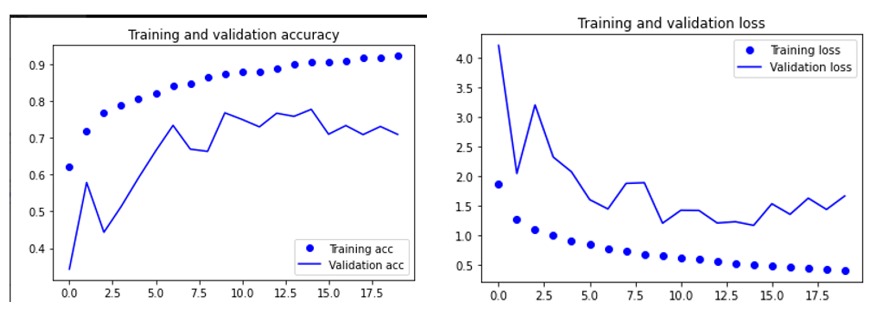
# 4.3.4 MobileNetV2 (Prevent Overfitting using Batch Normalization)

There are 2 other methods to prevent overfitting that I would be using, Batch Normalization and Layer Normalization. My first attempt with Batch Normalization followed the model below.

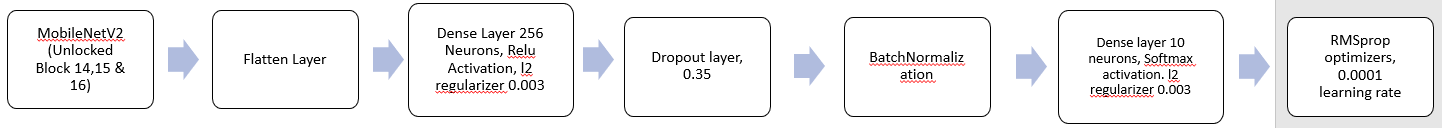


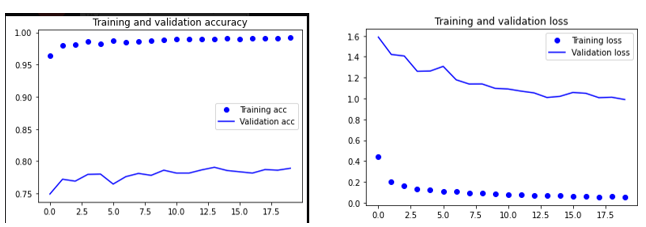
From this graph, not many changes can be noted. The graph still overfits tremendously. Thus, I added hyperparameters of dropout rate 0.35 and l2 regularizers of 0.003. I also made the decision to switch from l1 to l2 regularizers to compare the different effects between l1 and l2 regularizers even though l1 regularizers are noted to work better for feature extraction models like CNNs (Tanuk, 2020).





This addition of hyperparameters managed to prevent overfitting but severely impacted the training and validation accuracy. Hence because of the fluctuations that are still present, I decided to tune down the learning rate to 0.0001 to minimize the fluctuations and inconsistencies in the graph.

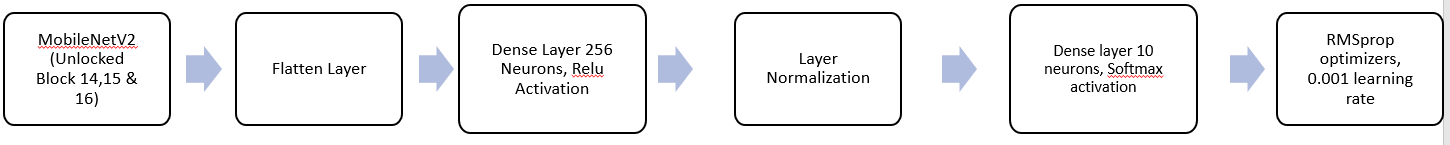


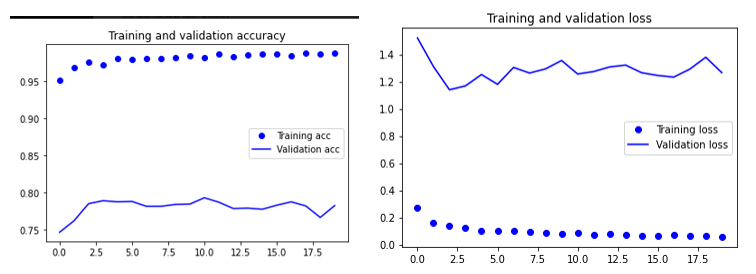


This model, despite being severely overfitted has little fluctuations and has a constant validation accuracy range of 0.77-0.79. I tried to further adjust hyperparameters to make the graph overfit less but to little result. Preventing overfitting of the curve would lead to a significant decrease in validation accuracy.

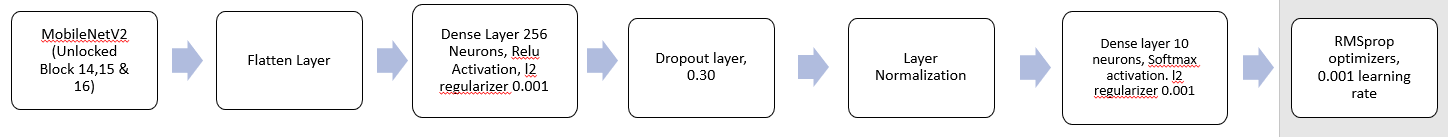
# 4.3.5 MobileNetV2 (Preventing Overfitting using Layer Normalization)

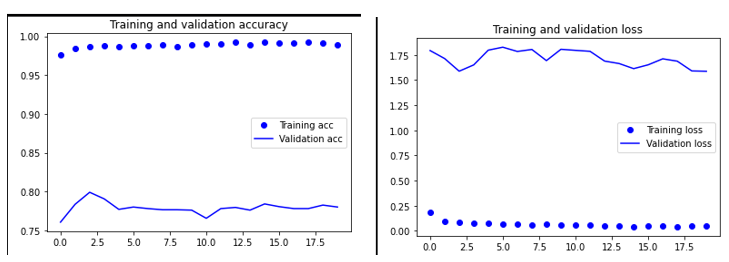
Layer normalization was also used to prevent overfitting. The first model is as follows.



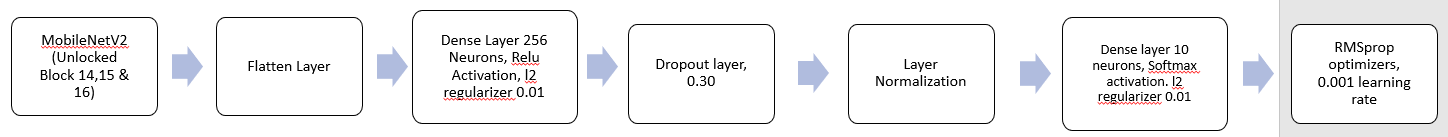


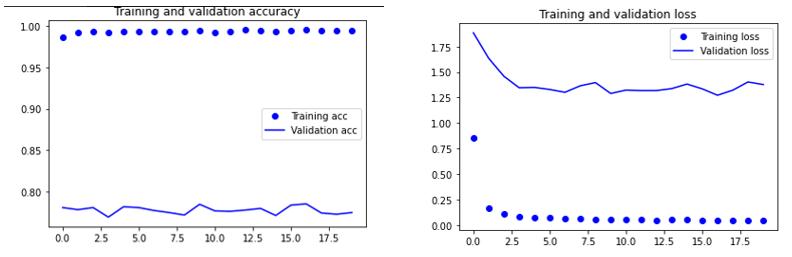
From this graph, layer normalization and batch normalization have similar effects on overfitting and both graphs are extremely similar. I decided to add hyperparameters to see how it would affect the model and its results. I added l2 regularizer of 0.001 and a dropout rate of 0.3.





The graph is generally more stable and validation accuracy hovers around 0.78 – 0.79. I increased the l2 kernel regularizers to 0.01 but to little effect.

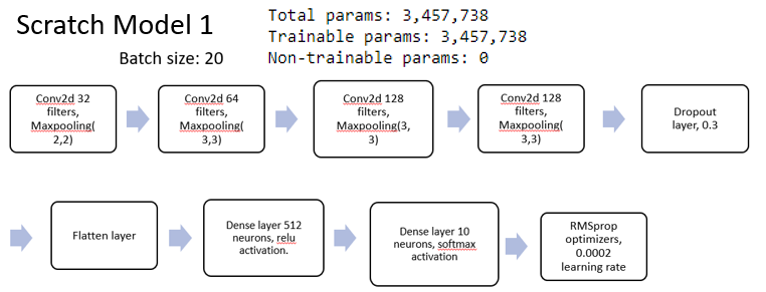


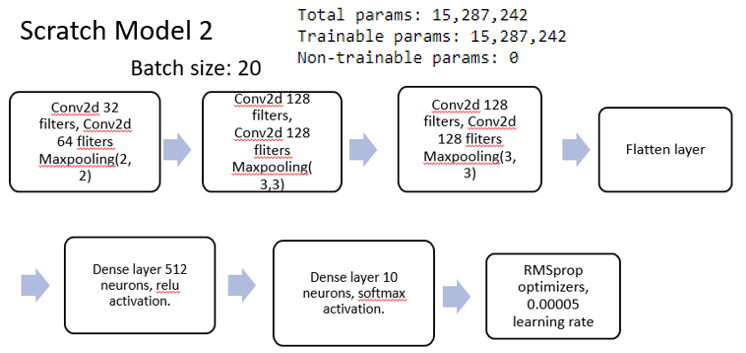


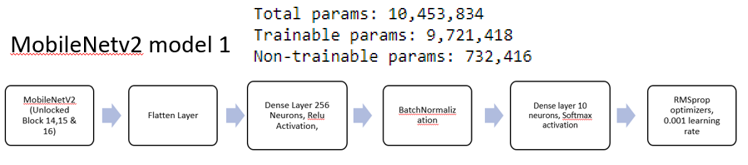
In conclusion, the effect of batch normalization and layer normalization on the prebuilt finetuned MobileNetV2 model are very similar, and both seem to work equally as well, yet overfitting still occurs and tuning of hyperparameters had little effect without compromising validation accuracy. One fix for this could potentially be to unfreeze more layers or all the layers as a bigger network size would result in higher validation accuracy.

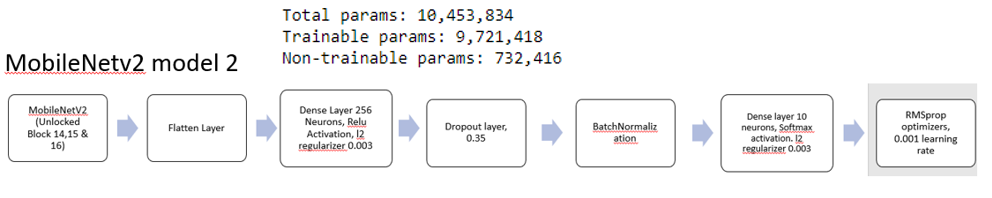
# 5. Evaluate models using Test images

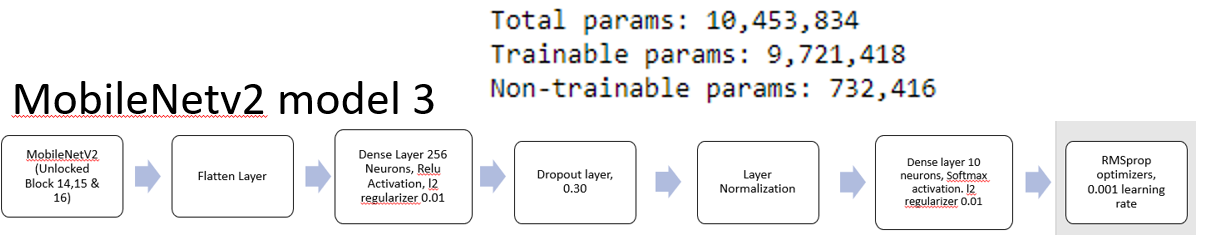
For evaluation of models, I had 5 models prepared. My first built from scratch model, my second built from scratch model and 3 MobileNetv2 models. I tested these models against the testing images data that I had set aside. A total of 500 images were used. The only augmentation of data that occurred was to resize the data to 1./255. The process of testing was done by using a test generator with a batch size of 20. The keras function model.evaluate was used to test the model against the test data.











|  |  |  |
| --- | --- | --- |
| Model | Test accuracy | Total parameters |
| Scratch Model 1 | 0.7160 | 3,457,738 |
| Scratch Model 2 | 0.720 | 15,287,242 |
| MobileNetV2 Model 1 | 0.812 | 10,453,834 |
| MobileNetV2 Model 2 | 0.798 | 10,453,834 |
| MobileNetV2 Model 3 | 0.770 | 10,453,834 |

# 5.1 Evaluation of models with Test Accuracy

From the test accuracy we can see that the prebuilt MobileNetV2 models perform better compared to the built from scratch models. We also see that total parameters play a part in increasing validation accuracy. However, they are not the only statistic that shows how effective a model would be as the Scratch model 2, despite having highest total parameters, were not as effective the MobileNetV2 models. The three MobileNetV2 models differ slightly from each other through the different hyperparameters being adjusted, with MobileNetV2 Model1 not having any dropout layers, MobileNetV2 with a dropout layer of dropout rate 0.3 and MobileNetV3. This tells us that by adding features to prevent overfitting like Dropout layers, the accuracy of the model would be affected. MobileNetV2 model 3 uses layer normalization, and it has the lowest validation accuracy of the three. One reason for this could be due to batch normalization generally being more effective on CNN compared to layer normalization which are generally more effective on recurrent neural networks (RNN) (Vijayrania, 2020).

# 5.2 Recommendation of Best Model

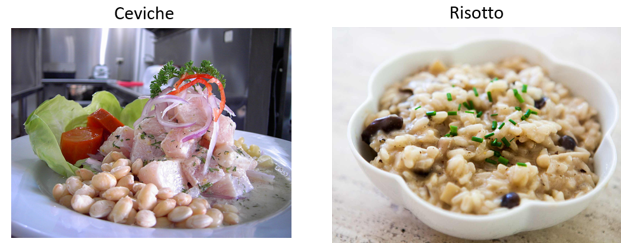
For my Best Model, I choose MobileNetV2 Model 1 as it has the highest test accuracy of the 5 that are being tested.

# 6. Classification of Images using Best Model

To classify images, the best model is first loaded followed by the food list being sorted alphabetically as the prediction function requires the list to be alphabetically sorted. The prediction function consists of a function, keras.model.predict which would use the model to predict what the image is. An image process function is also used to resize images to the appropriate size.

# 6.1 Analysis and Evaluation of Model Prediction

The diagram above shows the model prediction accuracy against images I obtained from the internet. The Best Model used is a finetuned MobileNetV2 model. From the above chart, we can see that the model mainly predicted the food items correctly with high levels of accuracy. The only exception being Risotto. The second highest belonged to ceviche. One reason for this could be the fact that Risotto and Ceviche are extremely similar in color, shape, and texture.



This could result in the neural network having similar features for both ceviche and risotto which has resulted in both accuracies being similar. Apart from that, the model’s prediction is extremely accurate.

# 6.2 Reasons for inconsistencies between Validation and Test accuracy and Predictions

Despite such high accuracies for prediction, it is not reflected on the model’s validation or testing accuracy which were both only slightly above 0.8. There are many factors that could affect this like the lighting of the pictures in the dataset, the presentation of the food as well as the angle the food is taken from. Usually, these problems could be circumvented with a large dataset, however as the dataset given is considerably small, thus it led to such inconsistencies.

# 7. Summary

In conclusion, my models have generally performed well, and huge progress has been made since the first experimental model. Although the validation accuracy is not on the higher ends, factors like a small dataset and quality of data also contributed to the lack of a high validation accuracy. One thing however, that I misunderstood was the impact of network size on validation accuracy. The total number of parameters do not directly corelate with validation accuracy and having a larger network does not necessarily mean that results would be better. As evident from my second scratch model, despite it being significantly bigger than the first scratch and the MobileNetV2 models, the testing accuracy showed us that the second scratch model was only slightly better than the first scratch model and was much worse than the MobileNetV2 models.

Moreover, there are still many features that I have yet to use to further fine-tune my model such as the use of different optimizers, changing the decay rate, experimenting with more pre-train models and the use of K-Fold cross-validation to increase the dataset available and help to increase validation accuracy. Furthermore, more research could have been done on the pre-trained models to learn the intricacies of the model which would help me better finetune the model which would help increase validation accuracy. Different normalization techniques like weight normalization and different pooling layers like average pooling could have also been used to enhance the model.

# References

Brownlee, J. (2018, December 3). *A Gentle Introduction to Dropout for Regularizing Deep Neural Networks*. Retrieved from Machine Learning Mastery: https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/

Brownlee, J. (2018, December 17). *How to Avoid Overfitting in Deep Learning Neural Networks*. Retrieved from Machine Learning Mastery: https://machinelearningmastery.com/introduction-to-regularization-to-reduce-overfitting-and-improve-generalization-error/

Brownlee, J. (2019, January 21). *How to Control the Stability of Training Neural Networks With the Batch Size*. Retrieved from Machine Learning Mastery: https://machinelearningmastery.com/how-to-control-the-speed-and-stability-of-training-neural-networks-with-gradient-descent-batch-size/

Brownlee, J. (2019, January 25). *Understand the Impact of Learning Rate on Neural Network Performance*. Retrieved from Machine Learning Mastery: https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/

India, A. M. (2019, 05 25). *What Does Freezing A Layer Mean And How Does It Help In Fine Tuning Neural Networks*. Retrieved from AnalyticsIndiaMag: https://analyticsindiamag.com/what-does-freezing-a-layer-mean-and-how-does-it-help-in-fine-tuning-neural-networks/

Saha, S. (2018, December 16). *A Comprehensive Guide to Convolutional Neural Networks — the ELI5 way*. Retrieved from Towards Data Science: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53

Tanuk, D. (2020, March 15). *L1 vs L2 Regularization: The intuitive difference*. Retrieved from Analytics Vidhya: https://medium.com/analytics-vidhya/l1-vs-l2-regularization-which-is-better-d01068e6658c

Vijayrania, N. (2020, December 10). *Different Normalization Layers in Deep Learning*. Retrieved from Towards Data Science: https://towardsdatascience.com/different-normalization-layers-in-deep-learning-1a7214ff71d6