Deep Learning Assignment 1

Table of Contents

[1. Overview 1](#_Toc106267503)

[2. Data Preprocessing and Data Loading 2](#_Toc106267504)

[Data Preprocessing 2](#_Toc106267505)

[Data Loading 2](#_Toc106267506)

[3. Developing the Image Classification Models 2](#_Toc106267507)

[Building and Training the Scratch Model (780) 3](#_Toc106267508)

[Building and Training the Model Using a Pre-trained Model 20](#_Toc106267509)

[4. Evaluating Models Using Test Images 23](#_Toc106267510)

[Comparison of the Models’ Performance During Testing Phase 23](#_Toc106267511)

[5. Using the Best Model to Perform Classification 23](#_Toc106267512)

[Applying the Model on Real Life Images 23](#_Toc106267513)

[Model Prediction Analysis 24](#_Toc106267514)

[6. Summary 26](#_Toc106267515)

[References 26](#_Toc106267516)

# 1. Overview

At least two image classification models were to be built, one trained from scratch using conv2D and dense layers, and another using a pre-trained model. The assigned dataset containing food images that are organized into 10 different food types, needed to be divided into training, validation, and testing samples. The assigned food types are, beet salad, bibimbap, bread pudding, clam chowder, French fries, onion rings, pho, scallops, sushi and Takoyaki. These sample images then needed to be loaded into Jupyter Notebook for the training of the models. The images dataset was downloaded from Kaggle. In this assignment, only two models were built, one scratch model and one using the pre-trained model, Xception.

The objective was to build and train image classification models able to recognize and classify the 10 different types of food assigned. In addition, the model with the best performance was to be selected to make a prediction. Three images of food from the assigned food types were imported from the internet for the prediction. They were then fed into the chosen best model to see if it could classify the food images correctly.

The approach taken to build the models was, first starting with a baseline model, then scaling it up until it overfits, tunning its hyperparameters and finally regularizing the model to overcome overfitting. To find the best configuration of the model, model performance curves were recorded, analyzed, and compared with one another during the training period. Afterwards, the performance of the final configuration of the models was evaluated using test images and the performance of each model was compared with each other.

In this report, further details on how I executed my plan for building the models, my analysis of their performance during training and testing, and the implementation of the best model will be covered.

# 2. Data Preprocessing and Data Loading

## Data Preprocessing

Using the ‘Image\_Preprocessing.ipynb’ template file provided, I first initialized the base directory and the image directory. The base directory is the current directory where I stored the 10 assigned type of food images. The image directory is where I saved all the downloaded 101 types of food images. After that, directories for the training, validation and test image samples were created in the base directory. Next, directories for the 10 types of food assigned are created in each of the directories of the sample types. Finally, the images of each food type are divided into 750, 200 and 50 images into the train, validation, and test folder respectively. As a result, there are 7500 training, 2000 validation, and 500 testing images for all 10 categories of food.

## Data Loading

I initialized the base, train, validation, and test directory. To load the data into the models, I added the train, validation, and test directory as the target directory parameter into the train, validation, and test generator function respectively. Before training, I also resized all the images to the target dimensions of 150 x 150. During the training of the model, the model will be able to access the training images from these directories directly.

# 3. Developing the Image Classification Models

## Building and Training the Scratch Model

Starting off from a base model, I scaled it up until it overfitted by increasing the number of nodes per layer and adding more Conv2D and dense layers. After overfitting occurs, I performed regularization to the model to overcome overfitting by testing adding regularization functions l1 and / or l2 and testing adding dropout after each layer.

For my base model, I started off with 4 Conv2D layers and 2 dense layers.

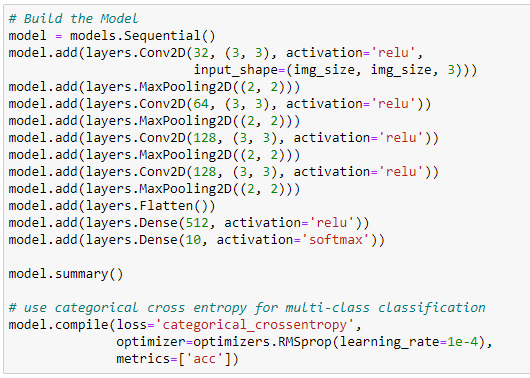


Figure 1.0 Base Model Code

Since the model is for multi-class classification, softmax activation function is used for the output layer and catergorical\_crossentropy is used as the loss function.

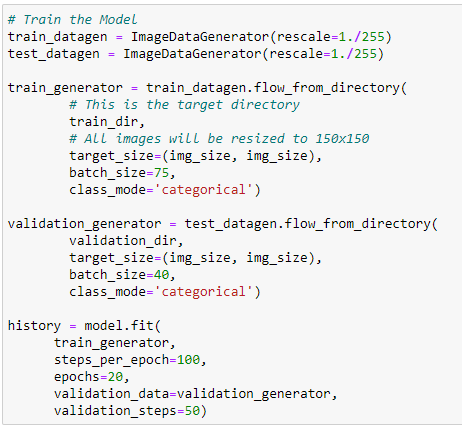


Figure 1.1 Model’s Training Code Without Data Augmentation

The first time I trained the model, I did not implement data augmentation. I used a batch size of 75 and 40 for the training and validation data respectively to get 100 and 50 steps\_per\_epoch and validation steps respectively. These values were derived using the formula:

steps\_per\_epoch = sample size / batch size

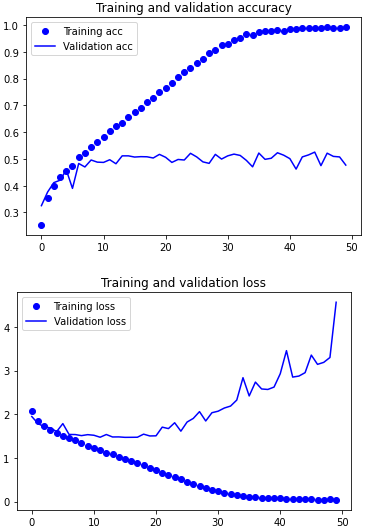


Figure 1.2 Performance Curve of Base Model Without Data Augmentation

After training the model for 50 epochs without data augmentation, it is observed that the model overfits after 20 epochs when the training loss continues to decrease while validation loss increases. The generalization of the model also stagnates as seen from the training accuracy increasing to nearly 1.0 while the validation accuracy stays at about 0.5 after about the fifth epoch.

Next, I implemented data augmentation to the training.

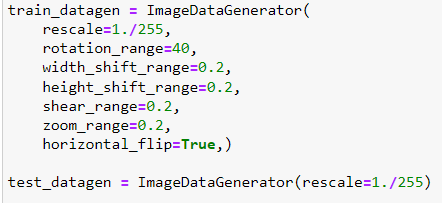


Figure 1.3 Data Augmentation on Training Data Code

More training data by creating variations of images so that the model does not see the same image more than once. Images will be rotated, zoomed in, trimmed, etc. As a result, generalization of the model improves and overcomes overfitting because the model will be exposed to more features of the data.

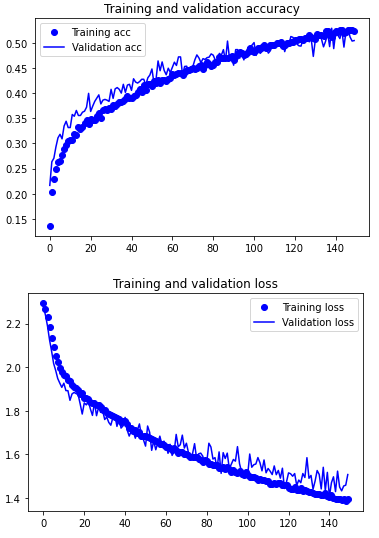
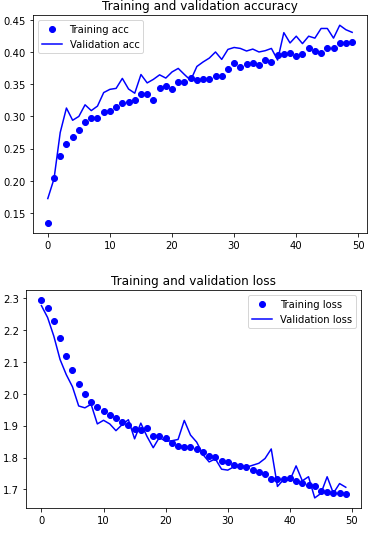


Figure 1.4 and Figure 1.5 Performance Curve of Base Model with Data Augmentation

As seen in the performance curves of the base model with data augmentation, no overfitting occurs even after 150 epochs. Hence, the next step is to scale up the model until overfitting occurs, because we want to find the point where overfitting barely starts to occur. That is the point where the model is the best. I scaled up my model by adding more layers and / or increasing the number of neurons in each layer.

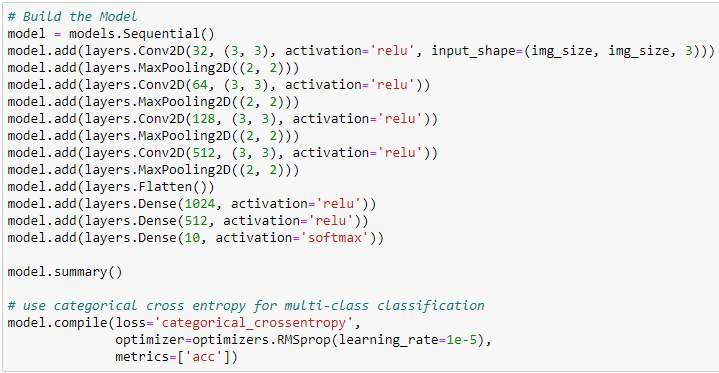


Figure 1.6 Model 1.1 Code

I started by increasing the number of neurons in the last Conv2D layer from 128 to 512 and the adding a dense layer with 1024 neurons.

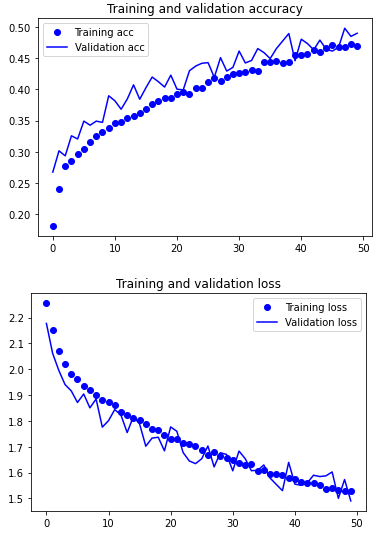


Figure 1.7 Model 1.1 Performance Curve

As seen from the performance curve, overfitting has not occurred yet. The model’s accuracy and performance improved slightly from about 0.43 to about 0.48 and the loss from about 1.7 to 1.5 respectively.

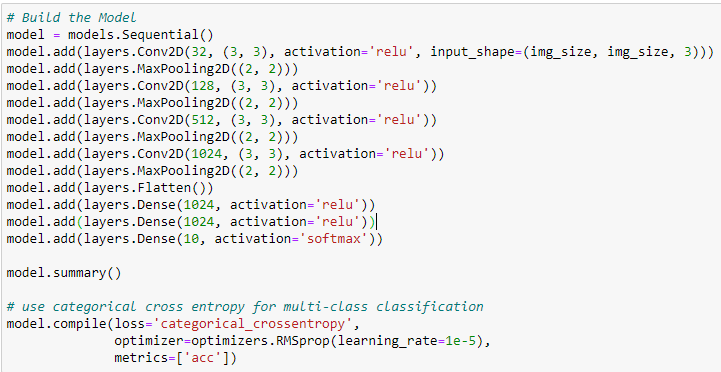


Figure 1.8 Model 1.2 Code

I increased the number of nodes of the last three Conv2D layers and the second last dense layer to 128, 512, 1024 and 1024 respectively.

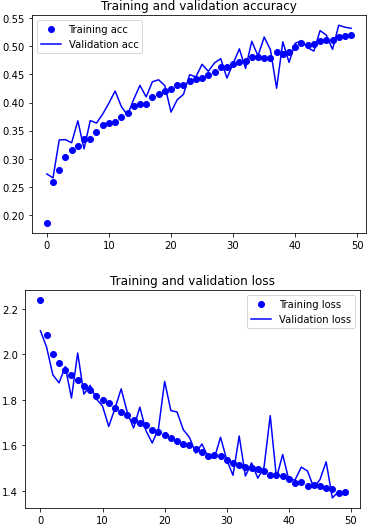


Figure 1.9 Model 1.2 Performance Curve

Overfitting has not occurred yet and the model slightly improved again. The performance and accuracy of the model improved from 0.48 to about 0.53 and 1.5 to about 1.4.

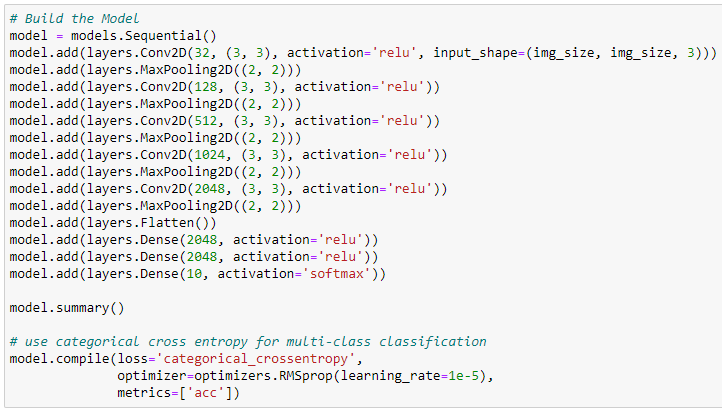


Figure 2.0 Model 1.3 Code

I added a Conv2D layer with 2048 nodes and also increased the number of nodes in the 2 hidden dense layers from 1024 to 2048 respectively.

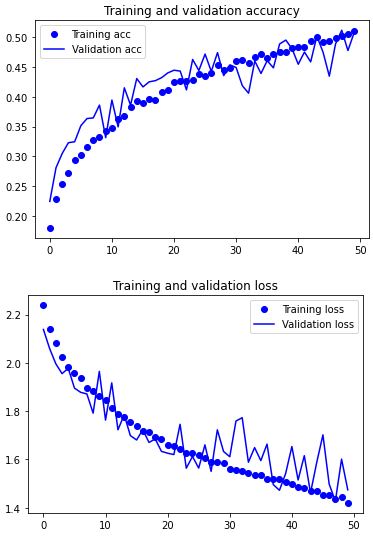


Figure 2.1 Model 1.3 Performance Curve

Overfitting still has not occurred and the model’s accuracy and performance deteriorated. The accuracy and loss are slightly worse than that of the previous model with accuracy of 0.53 and loss of 1.4.

After scaling up the model 3 times, it still didn’t overfit, which meant that the learning rate was too low. Hence, I doubled the learning rate of the model from 1e-5 to 2e-5.

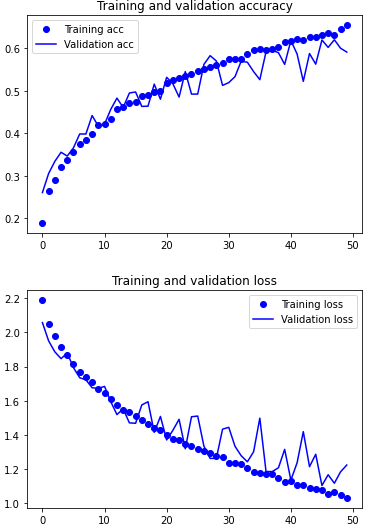


Figure 2.1 Model 1.3 Performance Curve with Double the Learning Rate

The performance of the model improved significantly. The accuracy of the model increased from 0.53 to about slightly more than 0.60 and the loss decreased from 1.4 to about 1.1. Overfitting still did not occur, so I scaled up my model more and trained it longer.

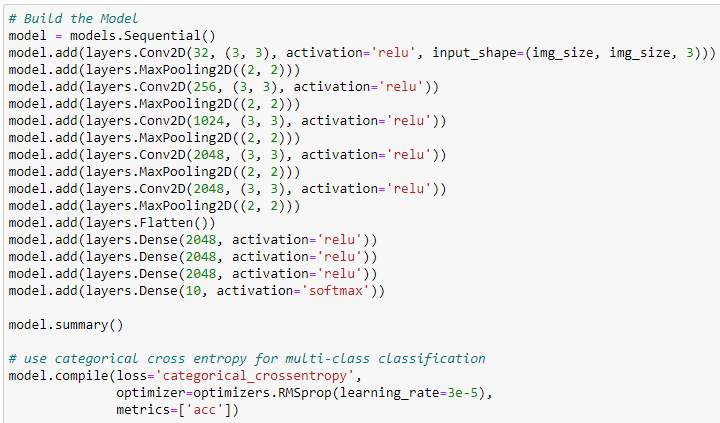
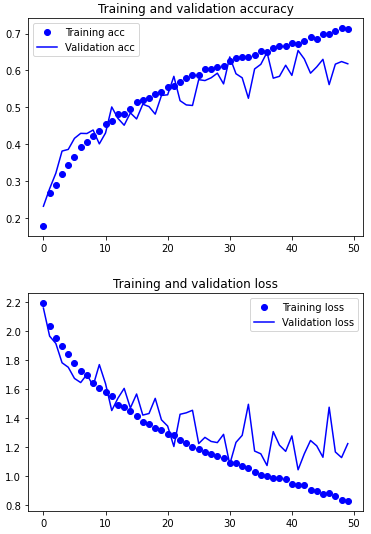


Figure 2.1 Model 1.4 Code

I increased the nodes of the Conv2D layers and added another dense layer. In addition, the learning rate was increased from 2e-5 to 3e-5.

Chart, scatter chart

Description automatically generated

Figures 2.2 and 2.3 Model 1.4 Performance curves

I trained the model for 50 epoch and saw that it might start overfitting if I train it more. After training for 75 epochs as seen in Figure 2.3, it still does not overfit. I went on and scaled up the model and gradually increased the learning rate further until it overfitted, however, it was too large and complex to carry out regularization (resource exhaustion error).

I decided to use a simpler model, model 1.2 and drastically increase the learning rate to 3e-4 from 1e-5.

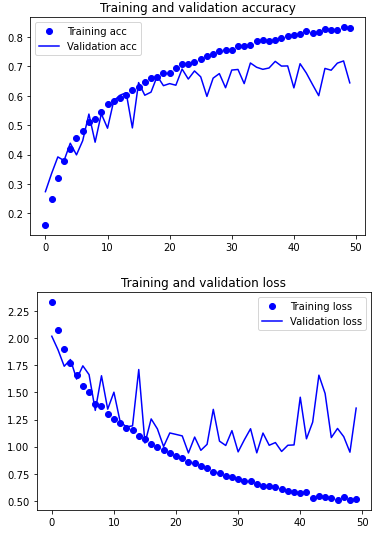


Figure 2.4 Model 1.2 Performance Curve with Increased Learning Rate

As seen from Figure 2.4, the model performance is much better. The accuracy of the model is about 0.7 as compared to that of model 1.4 of 0.65. The loss of the mode is about 1.0 which is lower than that of model 1.4 of about 1.25. Overfitting is also observed roughly after the 40th epoch.

The following step after scaling up the model until it overfits is regularizing the model. The first regularization method used to overcome overfitting was adding Regularizer l2 in the layers.

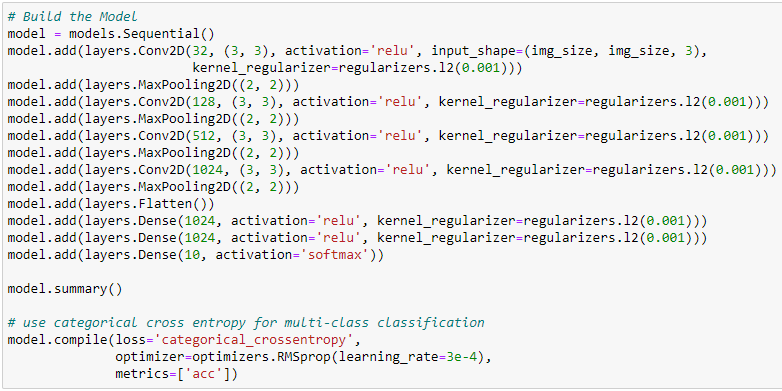


Figure 2.5 Model with Regularizer l2

I set Lambda (regularization rate), the parameter of Regularizer l2 to 0.001. l2 forces weights to become close to zero but not zero. Every coefficient in the weight matrix of the layer will add (lambda \* weight coefficient value) to the total loss of the network.

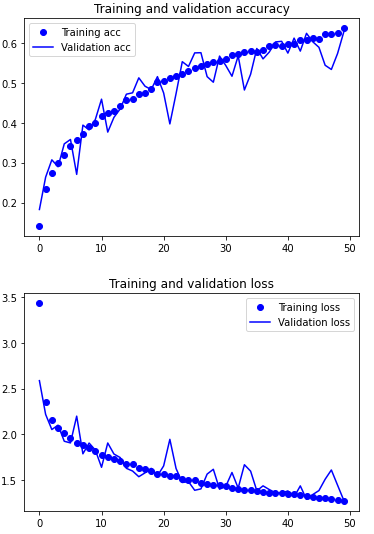


Figure 2.6 Model with Regularizer l2(0.001) Performance Curve

Adding Regularizer l2 overcame overfitting, while slightly penalizing the performance of the model. I then tested the model by decreased lambda from 0.001 to 0.0001.

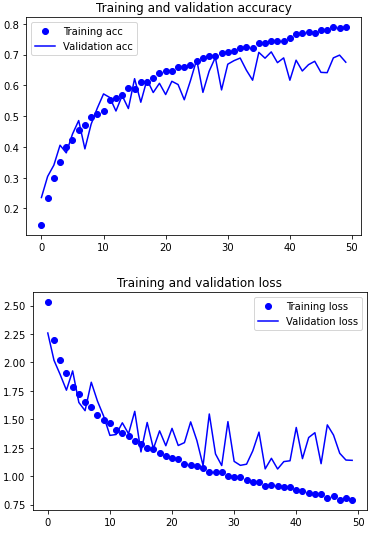


Figure 2.8 Model with Regularizer l2(0.0001) Performance Curve

This overcame overfitting minimally. moreover, the value of lambda is so low that the effect of regularization becomes insignificant. Subsequently, I changed the value of lambda to be 0.0005, in between the first value of 0.001 and second value of 0.0001.

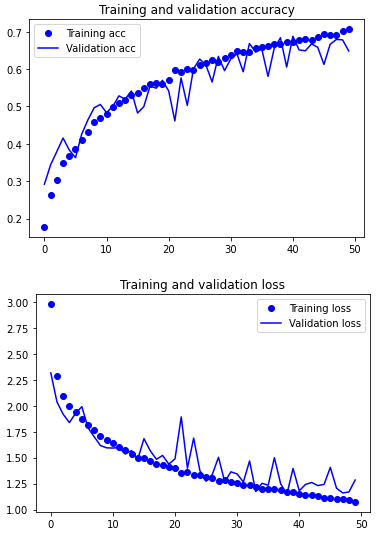


Figure 2.9 Model with Regularizer l2(0.0005) Performance Curve

Overfitting is prevented when using l2(0.0005) and the penalization of the model’s performance is minimal. The accuracy of the model is close to 0.7 while the loss is less than 1.25. Next, I used Regularizer l1 instead.

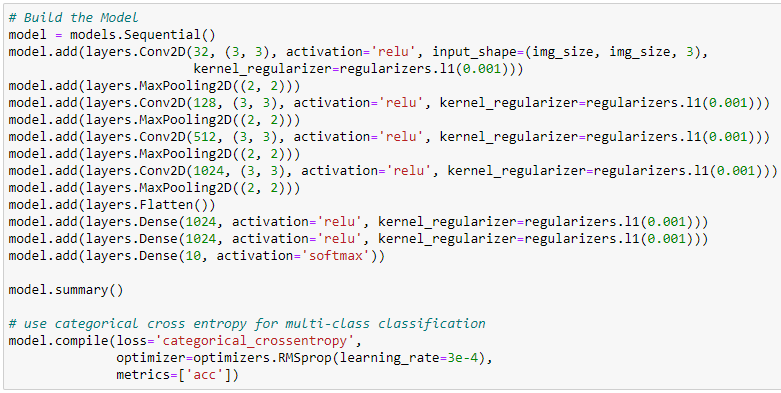


Figure 3.0 Model with Regularizer l1 Code

I set the parameter of l1 to 0.001. For uninformative features, l1 makes their weights zero by subtracting the parameter from the weight every repetition.

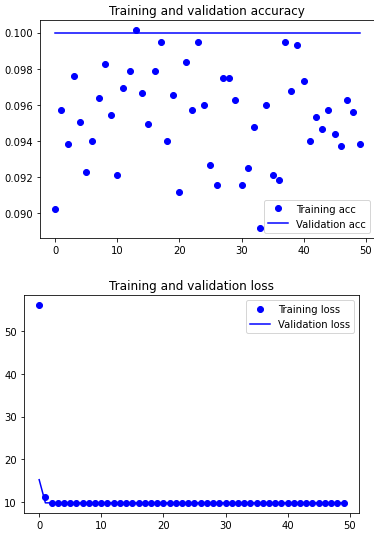


Figure 3.1 Model with l1 Performance curve

Adding l1 regularization to the model penalizes the learning of the model too much. The validation accuracy is stationary at 0.1 and the training accuracy is all over the place. The loss also does not change at all and stays at 10. What I did to avoid this from happening is, reducing the number of layers with l1 regularization by only adding it every other layer.

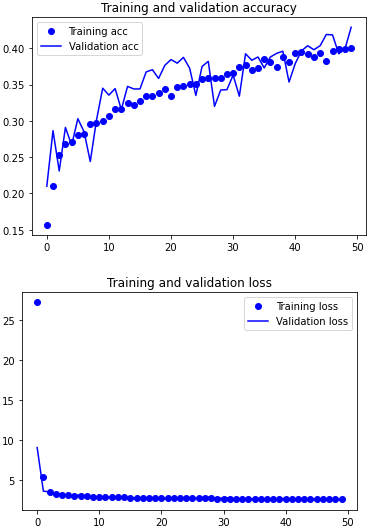


Figure 3.2 Model with l1 in every alternate layer

Based on the performance curve it is an improvement from the previous implementation, however, the model is still underfitting and being heavily penalized. The accuracy of the model is less than 0.5 and the loss is stagnant. Following this, I reduced the parameter value of l1 by half to 0.0005.

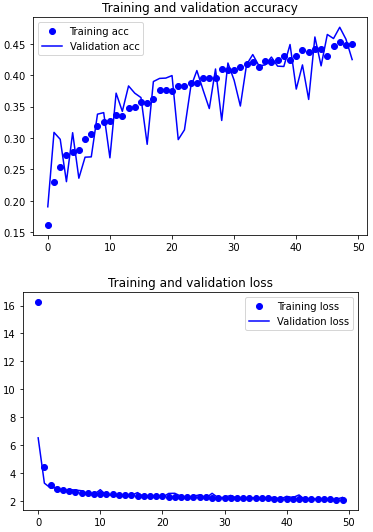


Figure 3.3 Model with l1(0.0005) in every alternate layer

There is little to no improvement from the previous implementation of l1. Finally, I reduced the number of layers with l1 regularization to one, the first dense layer.

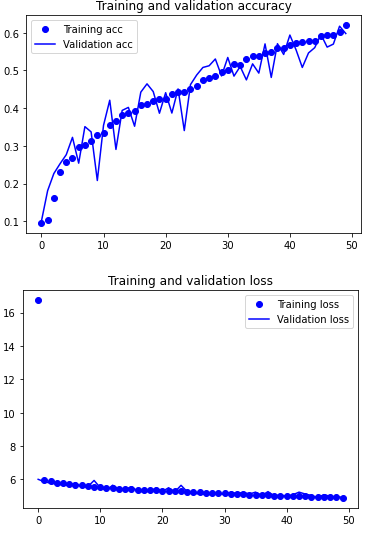


Figure 3.4 Model with one l1 regularized layer

Improvement is seen as the accuracy increased from about 0.45 to about 0.6, however, the model is still underfitting as observed from the loss. This concludes that l1 regularization penalizes the model too much and is not suitable for this model. One of the reasons could be that the data sample is too small. Moreover, the same can be said about combining l1 and l2.

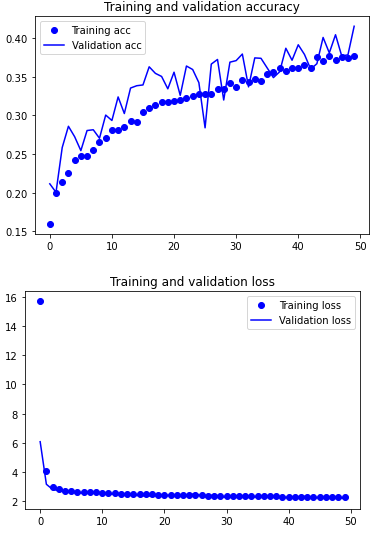


Figure 3.5 Combining l1 and l2 Regularization

The last regularization technique I used to overcome overfitting is adding dropout layers to the model.

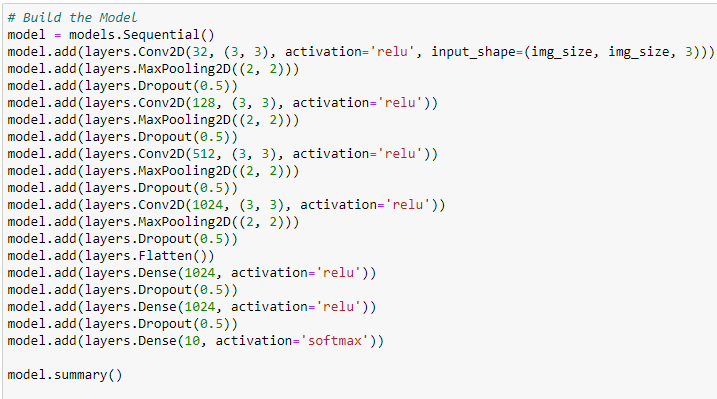


Figure 3.6 Utilizing Dropout

In the starting phase, I added dropout layers after every maxpooling layer and dense layer. I used a dropout rate of 0.5.

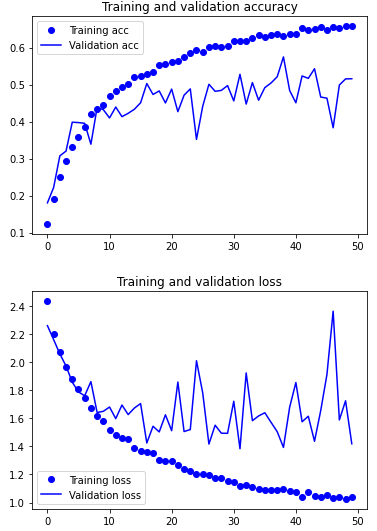


Figure 3.7 Dropout rate 0.5

The dropout layers did not help with overfitting and decreased the accuracy of the model. Dropout drops the output of a certain percentage of neurons in a layer. Too many neurons were dropped during the training which resulted in the model not being exposed to as many features as possible. As a result, I used a lower dropout rate of 0.2 later.

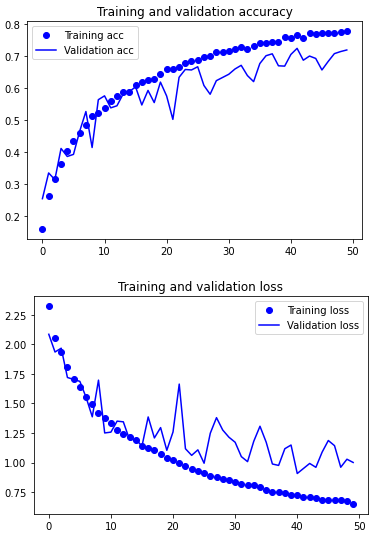


Figure 3.8 Dropout rate 0.2

The performance of the model is significantly better when using a dropout rate of 0.2 instead of 0.5. The accuracy of the model increased from 0.5 to about 0.7 and the loss decreased from about 1.4 to about 1.0. Lastly, I used different dropout rates 0.2 and 0.5 for convolutional layers and dense layers respectively.

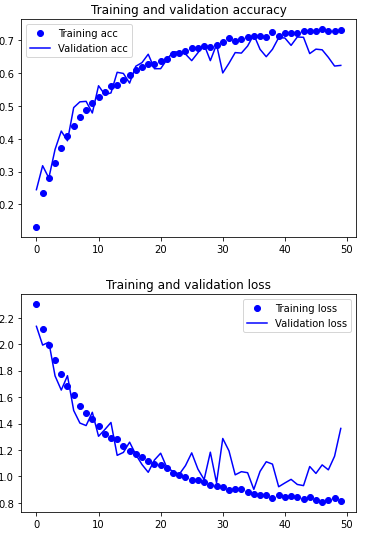


Figure 3.9 Mixed Dropout Rates

This is the best performing model, with an accuracy of about 0.7 and loss of about 0.9 since the lower the loss, the better the performance. The accuracy of the model using dropout is the same as using l2 regularization, but the loss is lower than that of l2 which is about 1.25.

The best performance of the model is in the middle of underfitting and overfitting. According to the validation loss, it is around 40 epochs. The final scratch model is that of model 1.2 with dropout rates of 0.2 and 0.5 in convolutional and dense layers respectively.

## Building and Training the Model Using a Pre-trained Model

The second model utilizes the pre-trained model, Xception, so I needed to import it.

Text

Description automatically generated

Figure 4.0 Parameters of Xception

The first configuration of the model was only using feature extraction without data augmentation. The output of Xception is ‘5,5,2048’ which will be the input of the new model. This is done by reshaping the features to the dimension of Xception’s output shape.

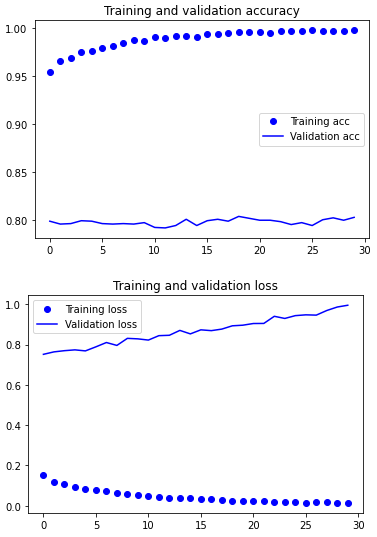


Figure 4.1 Pre-trained Model Using Feature Extraction without Data Augmentation

As seen from the validation accuracy curve, the model’s accuracy is stagnant at 0.80. This is much better than the model trained from scratch with an accuracy of about 0.70. The model also overfitted immediately as seen from the validation loss curve increasing since the first epoch. This is a result of the data set being small, thus, I subsequently trained the model using data augmentation, which extends the Xception model. The Xception blocks of layers are frozen, so that its trained weights do not get updated.

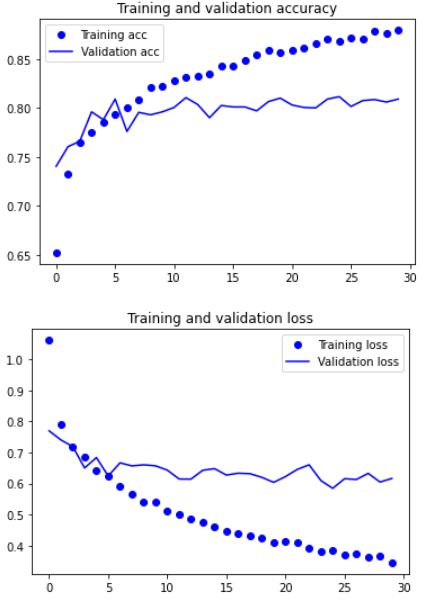


Figure 4.2 Feature Extraction with Data Augmentation

As seen from the validation loss, the model does not overfit, however, the accuracy of the model does not improve at all, staying at 0.8. Lastly, I unfroze the last 2 layers of Xception to fine-tune so that it is modified to be more applicable to the scenario. The reason why the last few layers are tuned instead is so that what the layers already learnt would not be changed significantly.

Chart, histogram, scatter chart

Description automatically generated

Figure 4.3 Fine-tuning

The model accuracy improved from 0.8 to about 0.825 and the loss also decreased from about 0.65 to about 0.6. Fine-tuning improved the overall performance of the model.

# 4. Evaluating Models Using Test Images

## Comparison of the Models’ Performance During Testing Phase

Text

Description automatically generated

Figure 4.4 Testing Phase of Model 1 and 2

Model two’s performance is better than model one’s performance. Model two’s loss of 0.5759 is lower than that of model one’s loss of 0.9031. Loss is a gauge of the model’s performance and the lower the loss, the better the model. Furthermore, the accuracy of model two of 0.8240 is higher than that of model one of 0.7080.

Based on the comparison of the evaluation of the two models developed using testing images, I determined that Model 2 is the best model.

# 5. Using the Best Model to Perform Classification

## Applying the Model on Real Life Images

The best food model and the food list (in alphabetical order) needs to be first loaded. Secondly, related functions for image processing and model prediction needs to be defined. Lastly, the real-life images will be fed into the model for the prediction.

## Model Prediction Analysis

A picture containing text, food, dish

Description automatically generated

Image source: <https://www.allrecipes.com/recipe/7177/bread-pudding-ii/>

Figure 4.5 Bread Pudding Prediction

The model predicted that the image is 98.7939% bread pudding, which is correct. This means that the model can distinguish the features of bread pudding from the features of the rest of the food types very well.

A picture containing text, food, dish, soup

Description automatically generated

Image source: <https://www.simplyrecipes.com/recipes/easy_wok_kissed_beef_pho/>

Figure 4.6 Pho Prediction

The model predicted that the image is 78.2033% Pho, and is correct, however, the percentage is significantly lower than that of predicting bread pudding by more than 20 percentage points. One of the reasons is because the image of the pho has similar features to bibimbap. Based on the prediction, the model predicted that the image is 21.4332% bibimbap, the second highest prediction percentage. Another reason could also be that the model has not been exposed to enough images to differentiate some of the features of pho and bibimbap.

Graphical user interface

Description automatically generated

Image source: <https://www.recipetineats.com/bibimbap/>

Figure 4.7 Bibimbap Prediction

The model predicted that the image is 99.9993% bibimbap, which is correct. This is the highest certainty that the image is one of the food types. In conjunction with the analysis of the pho image prediction, a reason why the model is almost 100% certain that that image is bibimbap, could be that this image has all the distinctive features that the model learnt.

Graphical user interface

Description automatically generated

Image source: <https://www.bokksu.com/blogs/news/all-about-takoyaki>

Figure 4.8 Takoyaki Prediction

The model predicted that the image is 95.5619% Takoyaki, which is correct. This means that the model can distinguish the features of Takoyaki from the features of the rest of the food types very well.

# 6. Summary

The best model’s performance, which uses the pre-trained model Xception was very good. When using the test images for model evaluation, it had an accuracy of 0.8240 with a loss of 0.5759. Additionally, all the real-life images that were loaded into the model were predicted correctly. Majority of the predictions were above 95% with the exception of one with about 78%.

For further improving the generalization and accuracy is increasing the training images sample size. This will allow the model to be exposed to more features and variations of the image types. The model will also be able to increase the number of types of images that it recognizes. With the larger the image sample size, the model can differentiate features of different types of images especially those with very similar ones. Implementing feature selection could also improve the performance of the model. The best subcategory of features is chosen to describe the relationship of independent variables with the target variable better (Ray, 2015).

# References

Ray, S. (2015, December 29). *8 Proven Ways for improving the “Accuracy” of a Machine Learning Model*. Retrieved June 16, 2022, from Analytics Vidhya: https://www.analyticsvidhya.com/blog/2015/12/improve-machine-learning-results/