

Table of Contents

Overview	1
Data Preprocessing and Data Loading	1
Develop the Image Classification Models (at least TWO)	2
Evaluate models using Test images.....	16
Summary.....	19

Overview

The project aims to apply classification model techniques to classify a specific set of 10 food items. It requires selecting and comparing multiple models, to determine which performs best in terms of accuracy and loss. The goal is not only to create a reliable model for classification but also to ensure that it can be validated and tested using real-world data. This project will involve a detailed analysis of the model's performance through metrics such as accuracy and loss curves and metrics like precision, recall, and F1-scores for the final 2 models chosen. The problem statement is to develop a food classification model to classify this set of 10 assigned food items (edamame, ice cream, bread pudding, grilled salmon, poutine, tiramisu, fish and chips, takoyaki, churros, and devilled eggs) as accurately as possible.

Data Preprocessing and Data Loading

I preprocessed and loaded the data in a separate jupyter notebook file before I could start with modelling. Firstly, I was trying to start by setting up the base and image directories. The base directory is `base_dir` and the image directory is `image_dir`. Essentially, this is so that we can define the `base_dir` is the current working directory and the `image_dir` directs to where the food images are stored.

Next, I then created more directories but for data splits like train, validation, and test directories, `train_dir`, `validation_dir`, and `test_dir` respectively. Those new three directories I created in `image_dir` are for storing training, validation, and test images. Ultimately this is just to create separate folders for training, validation, and test datasets to organize your data for model training.

After that, I can finally start to load and assigning the 10 different food categories I was assigned to, reading and then loading them from my `46.txt` file to `'food_list'`. This is so that we can read food categories from a text file into a `'food_list'` to know which images belong to which category

Next, I started with creating separate folders for each food category helps organize the training data. This structure is crucial for ensuring that each category's images are stored together, which makes it easier to manage and access the data during model training. And after that, the first 750 images of each food category are all

copied into their respective folders in the training directory for the training dataset. And my understanding is that, copying the images helps me ensure that the original dataset is untouched and not modified, thus avoiding any altering of the source files.

Then I did the same for validation data and test data and they each have their own folders too. Validation with 200 images and test with 50. Specifically, in validation, I created directories for each food category in validation_dir and then I copied the next 200 images (indices 750 to 950) into these directories. The same goes for test_dir, where I created directories for each food category in test_dir and then copied the remaining 50 images (indices 950 to 1000) into these directories.

Develop the Image Classification Models (at least TWO)

Normal Models Tested for Model 1:

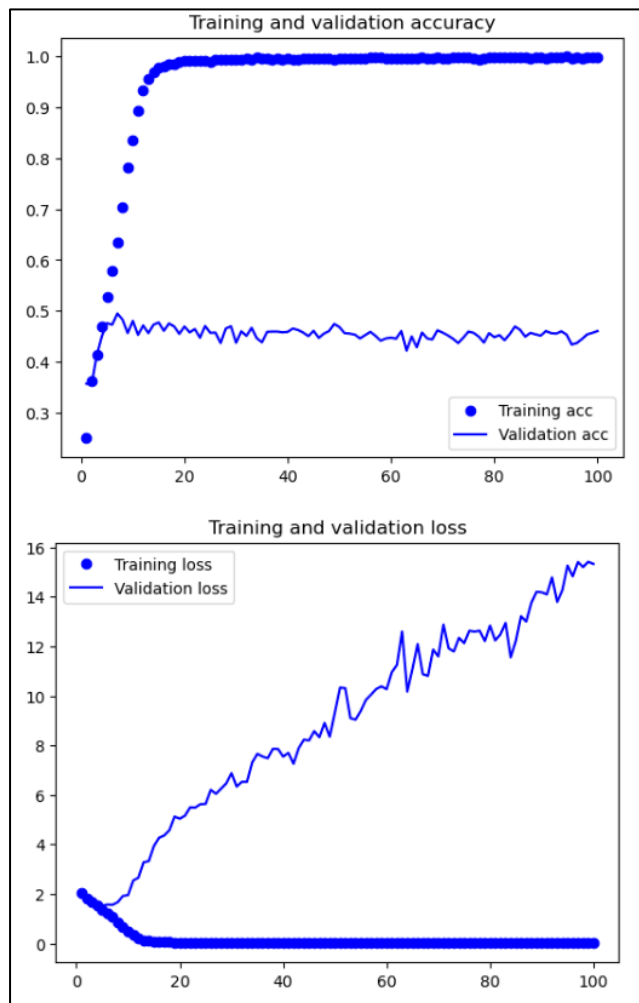
Model no.	Epochs	Filter Size	No. of Filters	L1/L2 Reg/ Nil	Batch Size	Learning rate	Augment (Y/N)	Modified Data Augmentation Code? (Y/N)	Chosen (Y/N), Val Acc, Val loss
1	100	(3x3), (3x3), (3x3), (3x3)	32, 64, 128, 128	Nil	20	2*1e-4	N	N	N, 0.4600, 15.3266
2	100	(3x3), (3x3), (3x3), (3x3)	32, 64, 128, 128	Nil	20	2*1e-4	Y	N	N, 0.7060, 0.9275
3	100	(3x3), (3x3), (3x3), (3x3)	32, 64, 128, 128	Nil	20	2*1e-4	Y	Y	Y, 0.7330, 0.8223
4	100	(3x3), (3x3), (3x3), (3x3)	32, 64, 128, 256	Nil	20	2e-4	Y	Y	N, 0.7075, 1.2727
5	50	(3x3), (3x3), (3x3), (3x3)	32, 64, 128, 256	Nil	20	2e-4	Y	Y	N, 0.6550, 1.2020
6	100	(3x3), (3x3), (3x3), (3x3)	32, 64, 128, 256	L2(0.001) every single layer	20	2e-4	Y	Y	0.7355, 1.1283

Pre-Trained Models Tested for Model 2:

Model no.	Pre-Trained Model Type	Epochs	Batch Size	Learning rate	Augment (Y/N)	Modified Datagen Augmentation Code? (Y/N)	Chosen (Y/N), Val Acc, Val loss
1	Resnet50	100	20	2e-5	Y	N	N, 0.8345, 0.7171
2	Inception	30	25	2e-5	Y	N	N, 0.7675, 0.7686
3	Inception	19	20	2e-5	Y	N	N, 0.7495, 0.7966
4	MobileNet	30	20	2e-5	Y	N	N, 0.8055, 0.7726
5	MobileNetV2	16	20	2e-5	Y	N	Y, 0.8090, 0.7470

Model 1.1:

acc: 0.9975 - loss: 0.0139 - val_acc: 0.4600 - val_loss: 15.3266



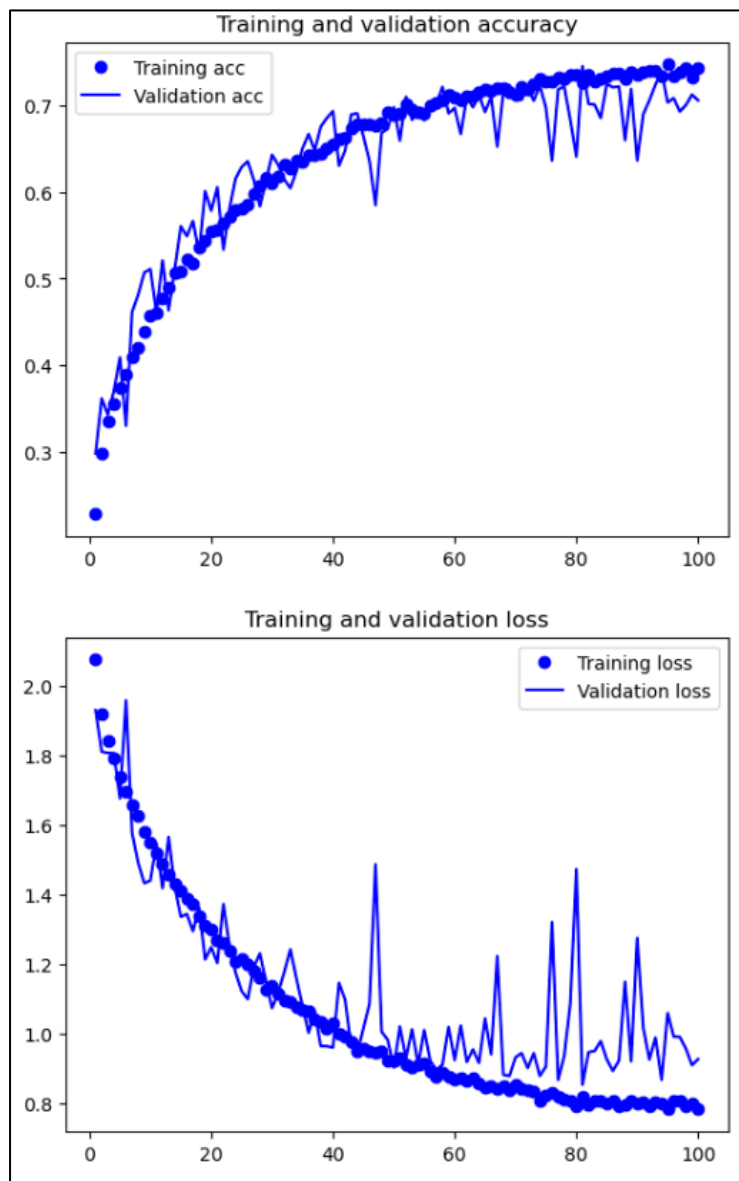
This was my very first model, built without any data augmentation, parameter tuning, or adjustments to the learning rate. My intention was to establish a baseline performance or kind of like a naïve model. As expected, the model showed severe overfitting, as clearly illustrated in this chart.

From the start, the validation accuracy remained constant after fewer than ten epochs, barely improving beyond its initial value. Similarly, the validation loss shot up around the same time, increasing dramatically with each epoch after training began. This sharp increase created a significant gap between the validation and training metrics. While the training accuracy and loss indicated a strong performance, these results were misleading because the model was just memorising the training data rather than learning patterns that could generalise to unseen examples.

This behaviour is a textbook definition of overfitting, where the model focuses solely on the training dataset. Ultimately, causing the validation performance to deteriorate, making the model ineffective for practical applications. This baseline result confirmed the need for strategies like regularisation, learning rate adjustments, or data augmentation to improve the model's generalisation ability and mitigate overfitting.

Model 1.2:

Epoch 100/100
375/375 — 231s 617ms/step - acc: 0.7454 - loss: 0.7858 - val_acc: 0.7060 - val_loss: 0.9275

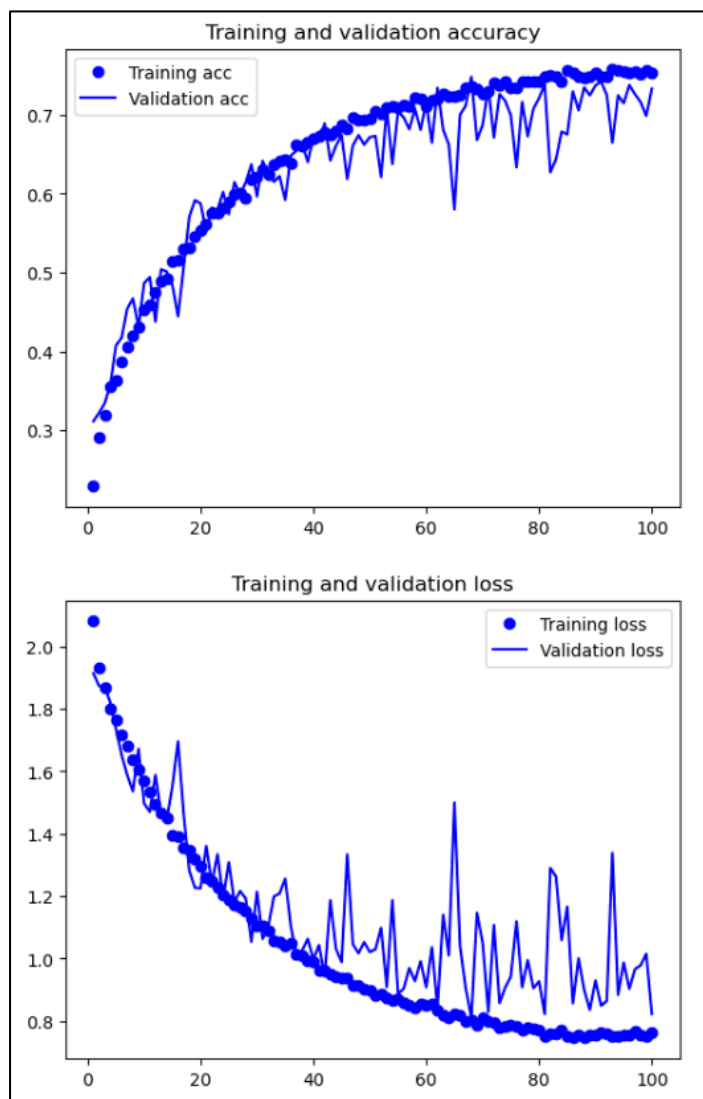


In these charts, the tail end of the validation curves for both loss and accuracy are starting to pull away from the training curves. For accuracy, the validation curve is fluctuating below the training curve. For loss, the validation loss is starting to increase slightly, while the training loss keeps decreasing. This divergence suggests that overfitting is starting to happen, and the model is memorising the training data too much instead of learning patterns that generalise to the validation set.

In this model, the only difference between this and the chosen model for model 1 below is that I didn't modify the data augmentation parameters to have additional settings like adjusting the 'fill_mode' to reflect or adjusting the rotations once again or adjusting the brightness of the images. And thus, this concludes that the model performance is definitely impacted by the data augmentation parameters.

Model 1.3 (chosen Model 1):

Epoch 100/100
375/375 — 161s 428ms/step - acc: 0.7575 - loss: 0.7446 - val_acc: 0.7330 - val_loss: 0.8223



The charts show how well a model performs during training and validation over 100 epochs. The top chart is about accuracy, which kind of displays how often the model gets things right), while the bottom chart shows the loss how far off the model's predictions are from the true answers.

For the training and validation accuracy curve, the top chart, you can see the dots, the training accuracy, steadily going up as the model gets better at learning from the training data. The blue line, the validation accuracy, follows a similar pattern but is bumpier. This bumpiness is normal because validation checks are done on unseen data, so it's more variable. Towards the end, both training and validation accuracy seem to level off around 0.733. This means the model has learned quite a bit, but it's probably hit its limit.

For the training and validation loss curve, the bottom chart, both the dots, the training loss, and the line, the validation loss, go down, which is what we want. Lower loss

means the model is getting better at predicting correctly. The training loss decreases smoothly, but the validation loss has spikes. These spikes could mean that the model sometimes struggles with the unseen data, which potentially is due to noise.

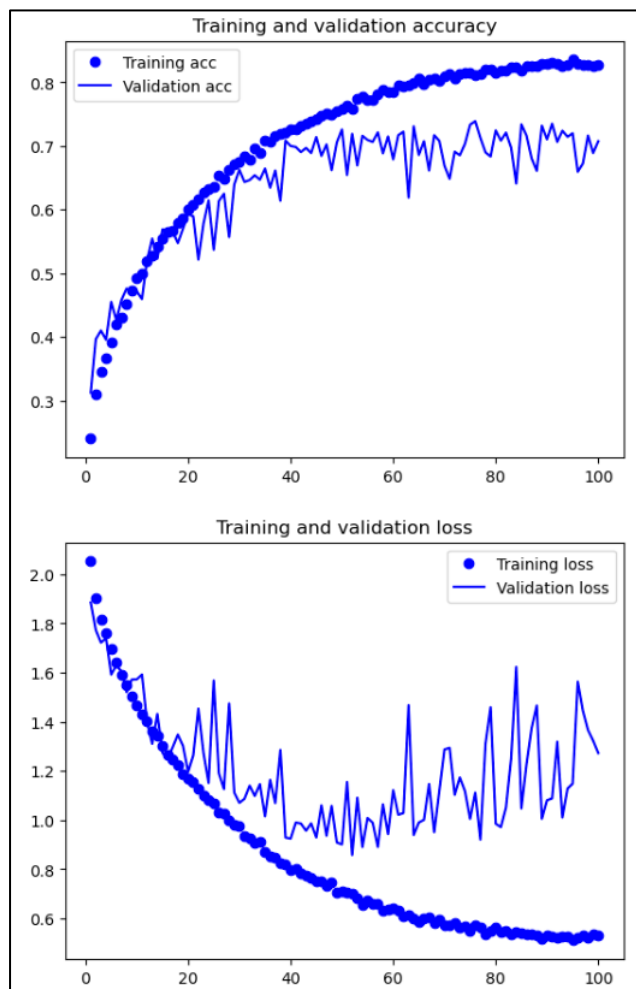
Overall, the model is learning well, it's improving both in accuracy and lowering the loss. But the validation loss's spikes and the bumpy validation accuracy suggest the model might not generalise perfectly to new data. It's not overfitting too badly though, since the validation metrics don't get worse over time, which is a good sign.

Additionally, I've also modified the data augmentation parameters like `fill_mode='reflect'` and adjusting brightness, you're helping the model focus on the important parts of the image while maintaining a realistic appearance of the food. The reflect fill mode is especially helpful because it avoids creating weird artifacts around the edges of images when you apply transformations. Adjusting brightness also ensures that darker or overly bright images don't throw off the model. And the fact that my validation accuracy closely follows training accuracy shows that the augmented images are doing a good job of simulating real-world scenarios.

Overall, since this model produced the best performing score with a low validation loss and a moderately high validation accuracy, while being able to keep the model from overfitting severely, is why I decided to choose this model for my Model 1.

Model 1.4:

Epoch 100/100
375/375 343s 914ms/step - acc: 0.8304 - loss: 0.5130 - val_acc: 0.7075 - val_loss: 1.2727

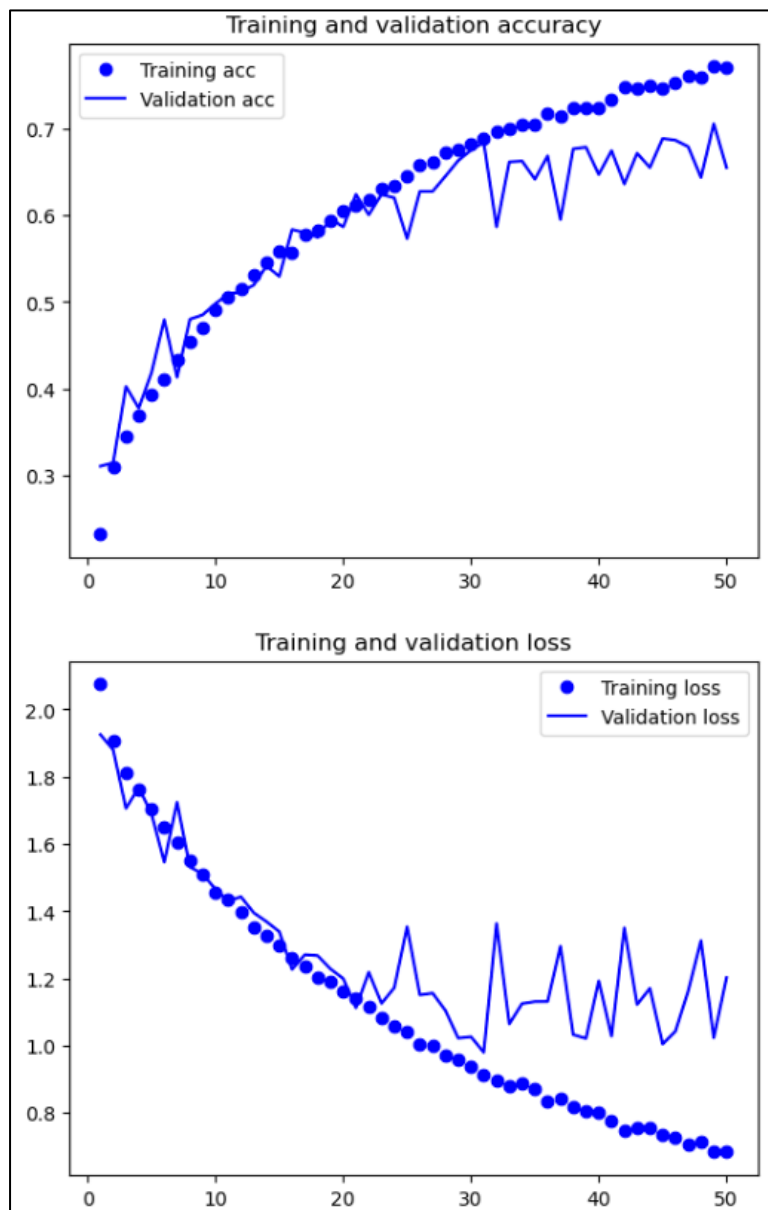


I tried to improve the validation accuracy by making the model more complex. Specifically, I changed the last conv2d layer from 128 to 256. I also kept the learning rate, batch size, and the number of epochs the same, as well as the data augmentation, all parameters stayed the same as the parameters in the chosen model.

Unfortunately, this approach led to a significant difference between the training and validation scores while the model trains over the epochs, with validation accuracy levelling off much earlier than training accuracy, indicating potential overfitting. This suggests that the model is performing well on training data but not generalizing well to new data which is the validation data. Hence, I could only infer that this tuning parameter wasn't effective as the results as it led to the classic issue of when the model becomes too complex for the available data.

Model 1.5:

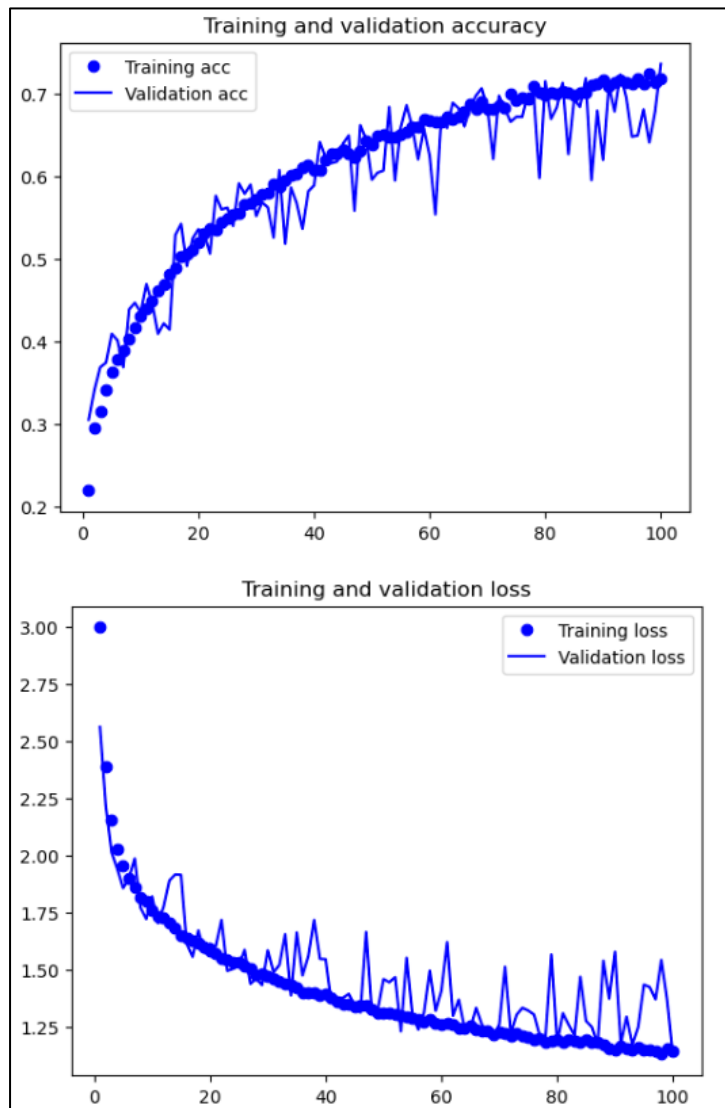
Epoch 50/50
375/375 — 339s 904ms/step - acc: 0.7766 - loss: 0.6785 - val_acc: 0.6550 - val_loss: 1.2020



For this model, I tried to improve the validation accuracy by making it more complex. Specifically, I increased the last Conv2D layer from 128 to 256. I kept the learning rate, batch size, and data augmentation code unchanged. Since I aimed to make the model more complex, I decided to reduce the number of epochs from 100 to 50, thinking it might help reduce overfitting. Unfortunately, that wasn't the case. The training and validation accuracy curves clearly show that the model began to overfit around the 30-epoch mark. Similarly, the training and validation loss curves indicate signs of overfitting appearing a few epochs after the 20th epoch, in the early 20s. As expected, the accuracy scores of this model with 50 epochs were lower compared to the same model trained with 100 epochs. The accuracy score decreased noticeably. For these reasons, I ultimately decided not to choose this model.

Model 1.6:

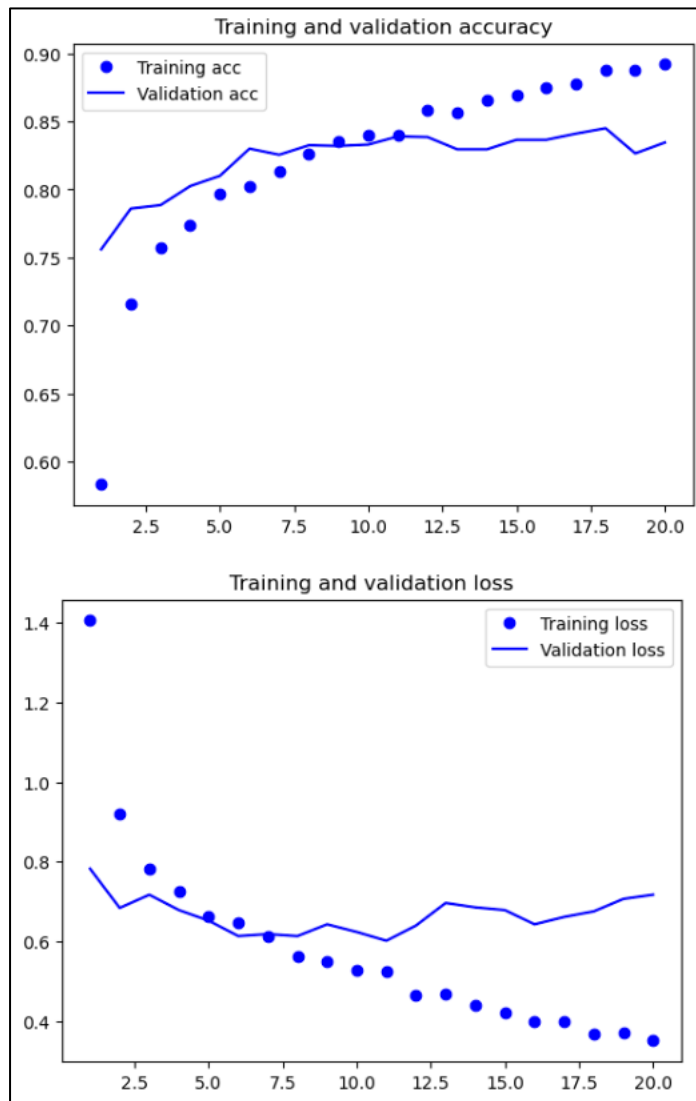
Epoch 100/100
375/375 179s 478ms/step - acc: 0.7206 - loss: 1.1245 - val_acc: 0.7355 - val_loss: 1.1283



After the previous model, I decided to try to use regularisation to curb or reduce overfitting of the model, specifically l2 regularisation. I chose L2 over L1 because all features extracted from the image can contribute to classifying food items. And L1 might discard some of these useful features, reducing model accuracy. Hence, why I chose to use L2 regularization, so that I can ensure that all features are considered but prevents over-reliance on any one feature. And I applied the L2 regularisation to all of the layers, because the overfitting in the previous model was very severe so I believed that it was a good idea to do so. As expected, the overfitted isn't as visible anymore and is very minimal, as there isn't a huge gap between the validation and training accuracy and loss scores. And thanks to the regularisation added, the noise has decreased since the previous model as shown in both the training and validation loss curves. But I didn't choose this chart as the loss scores for the training and validation are still very high, greater than 1.00 which isn't an ideal score. Hence why I didn't choose this model.

Model 2.1:

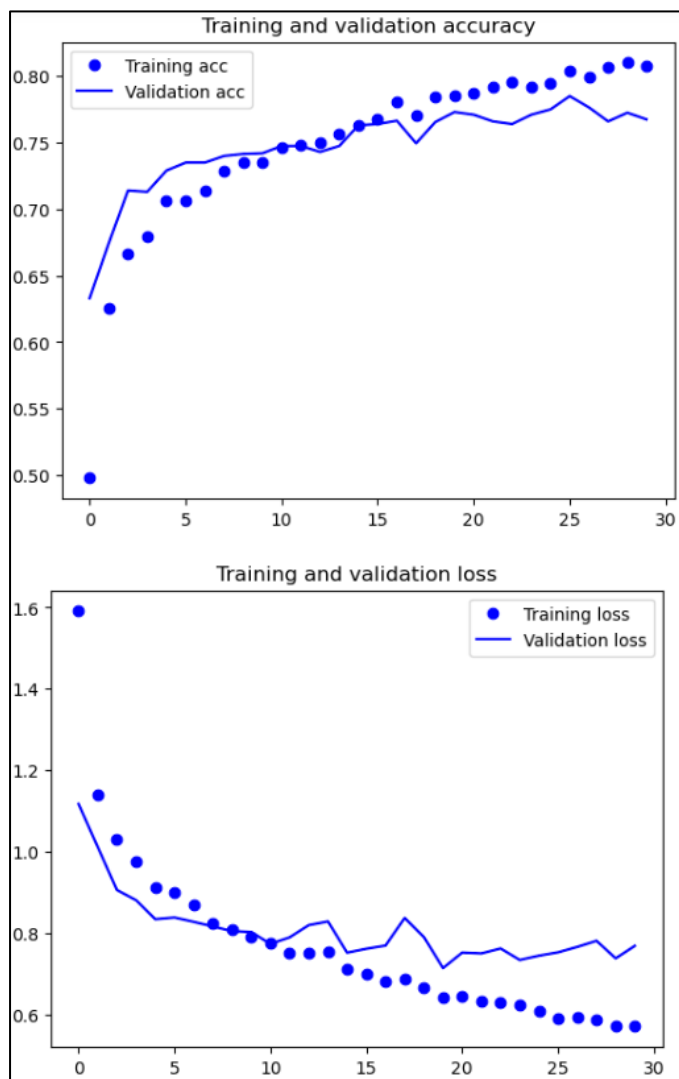
Epoch 20/20
375/375 — 949s 3s/step - acc: 0.8946 - loss: 0.3453 - val_acc: 0.8345 - val_loss: 0.7171



For Resnet50 model, I trained the model over 20 epochs, learning rate of $2e-5$, and the batch size of 20 as per the practical, and modified it to fit the correct shape of this ResNet50 model (5, 5, 2048), with no fine tuning executed yet. Ultimately, I didn't choose this chart because of the observed significant overfitting with the validation loss being more than twice the training loss. Additionally, the overfitting was observed to start after the 7-epoch mark which shows that the model's loss score overfitted early in the model training process. Thus, due to all the significant overfitting and poor generalisation observed, the model and its results were not chosen.

Model 2.2:

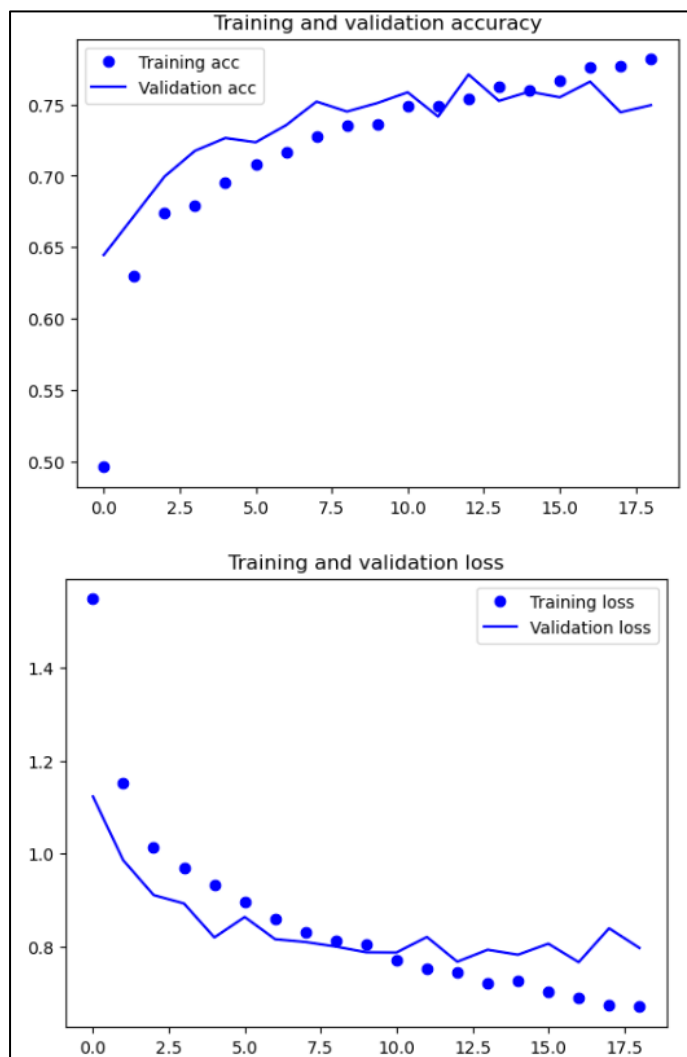
Epoch 30/30
300/300 — 262s 873ms/step - acc: 0.8145 - loss: 0.5635 - val_acc: 0.7675 - val_loss: 0.7686



After that ResNet50 model, I experimented with another model. InceptionV3 model. And I used the same learning rate and, but I tried to make the model more complex to try to increase the accuracy scores of the model I tried doing so by increasing the number of epochs to 30 and the batch sizes to 25. But unfortunately, the model was still unable to show any significant increase in the validation accuracy score. And, we can see that the model result is doing well because the validation loss and accuracy curves trends are generally following the direction of the training loss and accuracy curves.

Model 2.3:

Epoch 19/19
375/375 346s 923ms/step - acc: 0.7757 - loss: 0.6938 - val_acc: 0.7495 - val_loss: 0.7966

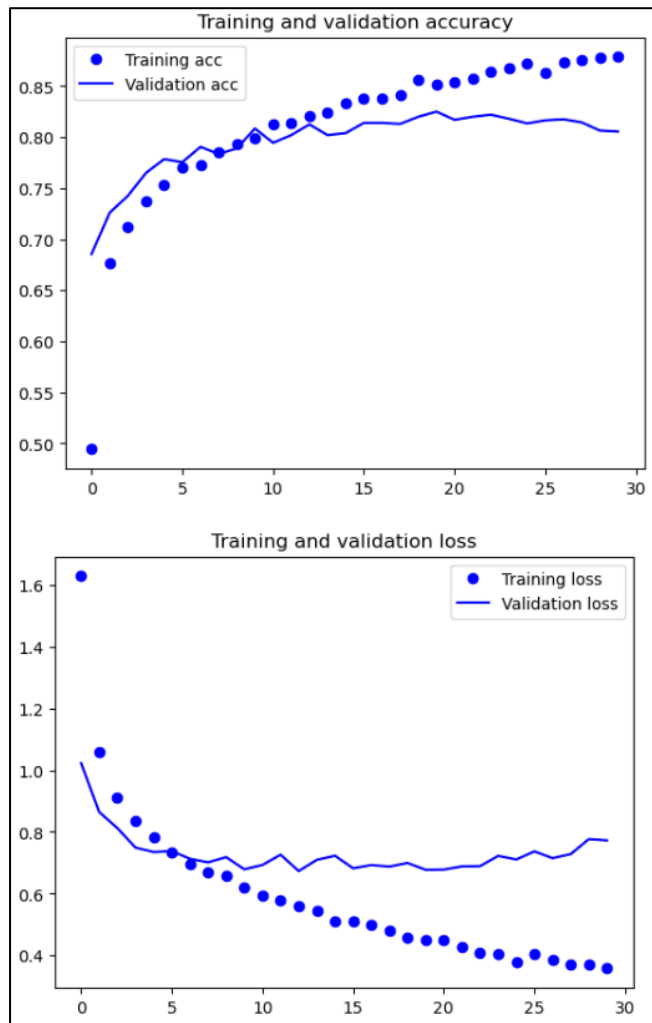


After the previous InceptionV3 model, I analysed the training scores over the 30 epochs and identified that the best model score before the model started to show the validation accuracy curves and loss curves start to show a constant flat trend, which was at the 19-epoch mark. So, I decided to attempt to train the model for 19 epochs, and reduce the model complexity through its batch size. And also, in this model, the overfitting or also known as the gap between the training and validation curves both started to first start to widen in the epochs between the range of 12 to 15 epoch mark, and then from that mark on, the gap just started to widen even more, showing that the model had started to overfit. Also Unfortunately, there isn't an improvement in terms of the training loss and the validation loss scores as instead of reducing the loss, the loss scores increased since the previous model. The same goes for the training accuracy and the validation accuracy, where instead of increasing the accuracy scores, the accuracy scores decreased instead. This showed that the approach to training for fewer epochs may not fully address this issue. And also, this approach to training the model for 19 epochs to stop the model from severely overfitting, may have caused the model to underfit slightly because of the decreased

overall model performance, which would suggest that the cutoff at 19 epochs was too early or too late for optimal performance.

Model 2.4:


Epoch 30/30
375/375 — 141s 375ms/step - acc: 0.8899 - loss: 0.3369 - val_acc: 0.8055 - val_loss: 0.7726



After that model using InceptionV3, I explored other models on Keras's website referenced in the practical and attempted to use the MobileNet model. As compared to the previous model, this model didn't really generalise as well and learn as well as it starts to overfit early in the training process of the model. Overfitting started for validation loss in the epoch range mark between 0 to 5, nearer to 5 where after that epoch, the model's validation loss score didn't decrease any further and just stayed constant. As for the validation accuracy curve, the overfitting started in the epoch range of 5 to 10, the model's validation accuracy score didn't increase any further and just stay at a standstill at 0.80. And while those validation accuracy and loss curves stayed constant after reaching a certain epoch mark, the training accuracy and loss curves continued to improve and learn through the training, which also increased the gap and difference between the validation and train result scores. Thus, which all points to overfitting, severe overfitting in the validation loss score because, the validation loss score is two times greater than the training loss score

which is more severe than the training accuracy and validation accuracy difference which is only has a difference of less than 0.1. Thus, due to this whole overfitting severe overfitting issue and ...I didn't choose this model.

Model 2.5:

Epoch 16/16 375/375		222s 592ms/step - acc: 0.8708 - loss: 0.3860 - val_acc: 0.8090 - val_loss: 0.7470
------------------------	---	---

After that model, I tried another pre train model, MobileNetV2. And MobileNetV2 consistently delivered the best results. Its performance stood out in terms of validation accuracy and overall stability during model training.

For the batch size, I used 20 which was a sample batch size from the practicals, not only that but it was also because while experimenting with other batch sizes like 25 and 30, I found that 20 provided the most stable and accurate performance. Similarly, I opted for a learning rate of $2e-5$ after testing other values and observing that it yielded the best results. This learning rate balanced the speed it took to converge and the performance, ensuring that the model trained effectively without overfitting or underperforming.

I decided to run the model for 16 epochs after much testing. Initially, I experimented with 30 epochs using the same hyperparameters, but the model started overfitting, leading to a decline in validation performance. By analysing the training and validation curves, I identified that 16 epochs produced the best results right before overfitting began. This allowed me to achieve a high validation accuracy while keeping the gap between training and validation scores minimal.

Additionally, I applied feature extraction with data augmentation to address overfitting. Data augmentation, which is well-known for enhancing model generalization, proved crucial in improving performance and ensuring that the model was not overly reliant on the training data.

As a result, this approach led to the best-performing model overall, with a strong balance between training and validation accuracy and minimal overfitting compared to other models and configurations.


Evaluate models using Test images

After choosing the 2 models out of all the models I've tested with, I'll elaborate more on the testing results, and I'll be focusing more on the precision and the recall scores of each of them.


Generally, the precision scores show that the higher the precision means the fewer the false positives. And the recall also shows that the higher the recall score, the fewer the number of false negatives.

Model 1:

Found 500 images belonging to 10 classes.

50/50  **4s** 69ms/step - acc: 0.6860 - loss: 0.9766


Test accuracy: 0.70

25/25 	2s 83ms/step			
	precision	recall	f1-score	support
bread_pudding	0.09	0.06	0.07	50
churros	0.09	0.06	0.07	50
deviled_eggs	0.08	0.08	0.08	50
edamame	0.09	0.08	0.09	50
fish_and_chips	0.05	0.06	0.06	50
grilled_salmon	0.15	0.12	0.13	50
ice_cream	0.05	0.08	0.06	50
poutine	0.11	0.14	0.12	50
takoyaki	0.07	0.08	0.08	50
tiramisu	0.09	0.08	0.08	50
accuracy			0.08	500
macro avg	0.09	0.08	0.08	500
weighted avg	0.09	0.08	0.08	500

The model's performance during the testing phase showed a moderate or average model performance score of 70% accuracy. grilled_salmon with a precision score 0.15 had the highest precision, meaning the model made relatively fewer false positive predictions for this class. grilled_salmon with the F1-score 0.13 also was the highest, reflecting the best overall balance between precision and recall. Poutine's recall score 0.14 had the highest recall, indicating the model was able to correctly identify a relatively higher proportion of the actual poutine samples. On the other hand, fish_and_chips 0.05 and ice_cream 0.05 have the lowest precision and F1-scores, meaning they had the most false positives. And bread_pudding and churros both 0.06 have the lowest recall, meaning the model missed a large proportion of true instances for these classes. Overall, the grilled_salmon and poutine classes stand out as having relatively better scores across all metrics. However, their performance is still not strong and indicates room for significant improvement in the future. Lastly, in the support column, I can see that all classes have the same support 50 instances, meaning the performance metrics are directly comparable across classes without weighting biases.

Model 2 (Pre-trained):

Found 500 images belonging to 10 classes.

50/50  **7s** 134ms/step - acc: 0.8216 - loss: 0.7069

Test accuracy: 0.83

	25/25  11s 244ms/step			
	precision	recall	f1-score	support
bread_pudding	0.10	0.12	0.11	50
churros	0.04	0.04	0.04	50
deviled_eggs	0.08	0.08	0.08	50
edamame	0.10	0.10	0.10	50
fish_and_chips	0.07	0.08	0.08	50
grilled_salmon	0.17	0.14	0.15	50
ice_cream	0.10	0.10	0.10	50
poutine	0.12	0.10	0.11	50
takoyaki	0.07	0.08	0.07	50
tiramisu	0.12	0.12	0.12	50
accuracy			0.10	500
macro avg	0.10	0.10	0.10	500
weighted avg	0.10	0.10	0.10	500

After doing this pre train model using MobileNetV2, the test accuracy is higher than the model above and this model has a test accuracy of 83%. Now moving onto the other metrics. `grilled_salmon`'s precision score 0.17 was the highest which means the model was relatively better at avoiding false positives for this class, and this class also had the highest F1-score 0.15. `bread_pudding` 0.12, `tiramisu` 0.12, and `grilled_salmon` 0.14 have the best recall. Additionally, the model was able to identify a higher proportion of true instances for these classes compared to others. Now zooming in onto the classes that are not really performing well, `churros` 0.04 has the lowest precision, meaning the model frequently misclassified other classes as `churros` many false positives, and `churros` 0.04 also has the lowest recall, indicating the model missed identifying most actual `churros` instances. And lastly, `churros` 0.04 also has the lowest F1-score, reflecting the poor balance between precision and recall. Some of my key observations would be that `grilled_salmon` definitely stands out as the best-performing class overall with the highest scores in all metrics. And as expected, `churros` performed the worst across all metrics, with very low precision, recall, and F1-score. And these 3 classes `bread_pudding`, `tiramisu`, and `poutine` share similar F1-scores of 0.11–0.12. And also, another group with common metrics is `edamame` and `ice_cream` also have identical metrics precision, recall, and F1-score of 0.10. And the same as the model above, the support column shows that each class has equal support 50 instances, ensuring the metrics for each class are directly comparable without being biased by unequal sample sizes.

Recommend the best model:

To decide which option to choose for my classification model, I evaluated the context and our priorities. Looking at both models' metrics side by side, it's clear that the MobileNetV2 pre-trained model 2 generally outperforms model 1 across several important metrics. For example, it has a higher accuracy of 0.10 compared to 0.08 for model 1, which suggests it's more reliable overall.

Additionally, both the macro and weighted averages for precision, recall, and F1-score are better in the pre-trained model 2, indicating it does a better job at minimizing false positives and identifying true positives. While the pre-trained model 2 struggles with the churros class, its performance with key classes like grilled_salmon is significantly stronger. Therefore, despite some weaknesses, the overall improvements in accuracy and class-level performance make MobileNetV2 pre-trained model 2 the better choice for our needs. Hence, this is why I recommended Model 2, the pre-trained MobileNetV2 model as the best model to be used to test against real life data.

Use the Best Model to perform classification

After selecting my final model which I've evaluated to be the best performing model, I started to test on the model using real life data which also means to use real images of the food items my model has been trained on. I loaded the food list the same way I did in the image processing steps before I started modelling. Then I created 2 new functions for my test predictions on real life data and inputs them into a model by resizing and normalizing them, one for image processing which helps me to process the downloaded real life data images when testing and another function called prediction which mainly handles the prediction using the trained best model to predict the 10 food classes based on processed images and then displays results in the form of probability dataframes which contains the scores and results.

Overall, all the newly inputted images for testing have all been predicted correctly for all of the correct classes. Most of the classes predicted with a 0.97 and above score, with one exception which was the bread_pudding class which only predicted a score of 0.53, this could be because the bread pudding in the picture didn't take up a huge section of the picture thus, a possible reason as to why the bread pudding only predicted a result of 0.53.

Summary

In summary, my final model chosen has a decent accuracy score for train, validation and test at around 82%. And the loss score could be further reduced to half the loss score it has now for better model performance and a better predicted output. The precision, recall, and F1-scores weren't great in the final model despite the moderately high accuracy. Hence why we should explore more ways for further improvements. And these could include, further tuning the model so that the precision, recall, and F1-scores can all be further improved to help with the model's predictive power. Next, further exploration of other pre-trained models and other tuning methods such as early stopping can be further explored because I've only briefly explored and researched on it on my own and didn't implement it because I haven't gotten a proper understanding of how and why to use early stopping, thus this could be further explored in helping improve the model's prediction power.