1 Assignment Specifications

1.1 Objective

The objective of this project is to build an image classification model to recognize and classify 10 different types of food. For this assignment, we will develop **TWO** different models:

- One Model trained from scratch using conv2D & dense layers
- One Model utilizing pre-trained models

For each model, we will analyze the model performance and tune the model hyperparameters during training phase. Next, we evaluate the model using the test images and compare the model performance during testing phase. Finally, we use the best model to make predictions.

1.1.1 Data Preprocessing and Data Loading

The dataset we are using comprises ten different food types. These are:

- Baby back ribs
- Bibimbap
- Cup cakes
- Dumplings
- Fried Calamari
- Garlic bread
- Lasagna
- Pancakes
- Prime rib
- Tiramisu

A sample from the dataset is shown in Figure 1.

The datasets have already been split into the train, test and validation directories respectively. Within each directory, they are placed into 10 different directories that have been named based on their food types.

To preprocess data, we use the class **ImageDataGenerator.** This will generate batches of image data with real-time data augmentation. This is useful for pre-processing images before feeding them to the neural network. We use the method, **flow_from_directory**, to load images from the target directory, preprocess them e.g. rescale, resize, and then feed them into the model in batches, one hot encoded labels for multi-class classification problems.

Three instance of data generators are created, one each for training, validation and test data. This creates the following:

- Training Set: 7500 images belonging to 10 classes, 750 images per food type.
- Validation Set: 2000 images belonging to 10 classes, 200 images per food type.
- Test Set: 500 images belonging to 10 classes, 50 images per food type.

Each data batch shape is: (64, 150, 150, 3) and labels batch shape: (64, 10)

Two sets of data generators are created: one without data augmentation and the other with augmentation. Augmentations create a more diverse dataset, making our model more robust

and improving generalization. The first data augmenter applied to custom neural nets uses the following arguments:

Parameter	Value	Description
rescale	1/255	Normalizes the pixel values to the range [0, 1]
rotation_range	20	Randomly rotates images by up to 20 degrees
width_shift_range	0.2	Shifts images horizontally by up to 20% of the width
height_shift_range	0.2	Shifts images vertically by up to 20% of the height
shear_range	0.2	Applies shear transformations with a magnitude of 0.2
zoom_range	0.2	Zooms in/out on images by up to 20%.
horizontal_flip	True	Randomly flips images horizontally
fill_mode	nearest	Fills in any empty pixels after transformations using the nearest pixel values.

Table 1 – Data augmentation parameters applied to custom neural nets

Only the training dataset is augmented. Validation and test data is not augmented because

1. **Validation Dataset**: This dataset is used to tune the model's hyperparameters and to monitor its performance on unseen data during training. Augmenting the validation dataset could introduce variations that are not representative of the original data, leading to an inaccurate assessment of the model's performance.
2. **Test Dataset**: This dataset is used to evaluate the final model's performance after training is complete. It's essential to use raw, unaltered data for testing to get an accurate measure of how well the model generalizes to real-world, unseen data. Augmenting the test dataset could artificially inflate the model's performance metrics or, conversely, make the model appear less effective.

Augmentation is about enhancing the training dataset to make the model better at handling real-world variability, while the validation and test datasets serve as benchmarks for evaluating the model's true performance.

The second data augmenter is applied to a pre-trained Convolution Net. This will be similar to the parameters in Table 1. The only difference is that rescale parameter is replaced by **preprocessing_function**. Table 2 describes this in more detail.

Parameter	Value	Description
preprocessing_function	preprocess_input	It allows you to apply a specific preprocessing function to each image. If you're using a pre-trained model from the Keras applications module (e.g. InceptionV3, ResNet50), a ***preprocess_input*** function to format images correctly for that model is usually provided. This ensures that the images are in the right format and scale for your chosen model.

Table 2 - Data augmentation parameters applied to pre-trained models

Figure 1 – Sample Images

2 Development of Image Classification Models

Two food image classification models were built to recognize and classify the 10 different food types.

- Model 1: Uses a custom neural net architecture with conv2D and dense layers trained from scratch
- Model 2: Leveraging VGG16 as a pre-trained model and employing transfer learning to address the problem

Table 3 and Table 4 summarises the different tuning parameters for Model 1 and 2 respectively.

Model 1: Experiment #	Data Augmentation	# Epochs	Regularization	Dropout	Early Stopping
1	N	30	N	N	N

Model 1: Experiment #	Data Augmentation	# Epochs	Regularization	Dropout	Early Stopping
2	Y	20	N	N	N
3	Y	30	N	N	N
4	Y	30	Y – L2 with regularization factor 0.001	N	N
5	Y	30	Y – L1 and L2 with regularization factor 0.001	N	N
6	Y	30	N	Y – dropout value 0.5	N
7	Y	30	N	N	Monitor: 'loss' Patience: 3

Table 3 - Model 1 Experiments showing different hyper-parameters

Model 2: Experiment #	Data Augmentation	# Epochs	Fine Tuning - unfreeze
1	N	30	N
2	Y	30	N
3	Y	50	N
4	Y	30	Unfreeze from: 'block5_conv1'. See Figure 2
5	Y	30	Unfreeze from: 'block5_conv3'. See Figure 3

Table 4 - Model 2 Experiments showing different hyper-parameters

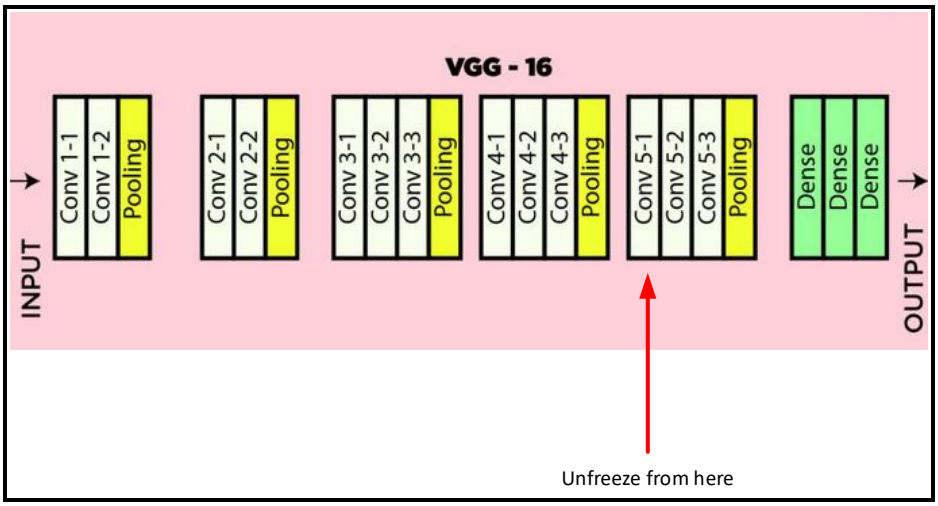


Figure 2 – Unfreeze from layer: block5_conv1 (Source: https://www.geeksforgeeks.org/vgg-16-cnn-model/)

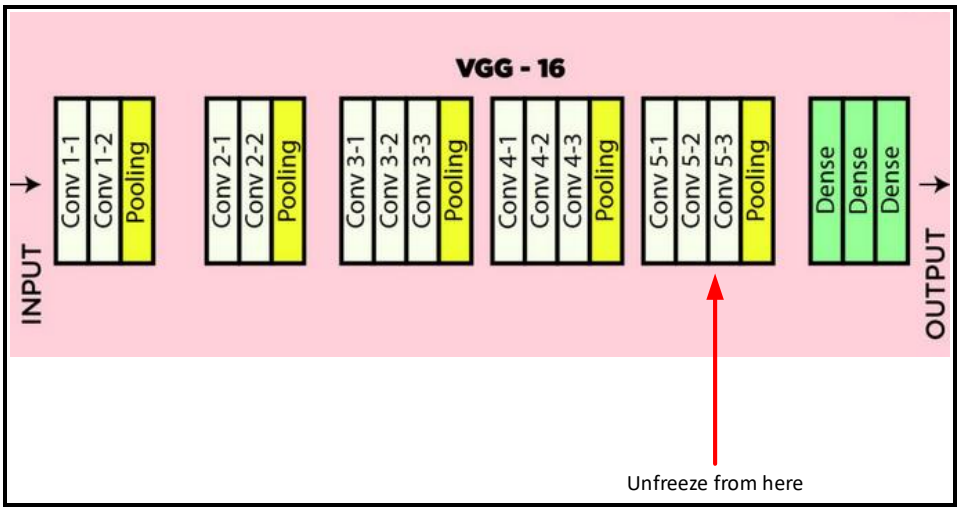


Figure 3 - Unfreeze from layer: block5_conv3 (Source: https://www.geeksforgeeks.org/vgg-16-cnn-model/)

2.1 Model 1 – Custom Neural Net

Here we build the model from scratch using conv2D and dense layers. The architecture of this model comprises multiple convolutional and pooling layers to extract features, followed by fully connected layers to make predictions. The detailed architecture is shown in Table 5. A visual representation of this custom neural net is shown in Figure 4.

Layer #	Layer	Description
Input Layer	keras.Input(shape=(img_size, img_size, 3))	This defines the input shape of the model. It expects images of size (img_size, img_size) with 3 color channels (RGB).
First convolution layer	layers.Conv2D(32, (3, 3), activation='relu')	A 2D convolutional layer with 32 filters, each of size (3, 3). The ReLU activation function is used to introduce non-linearity.
First MaxPooling layer	layers.MaxPooling2D((2, 2))	Reduces the spatial dimensions (height and width) by taking the maximum value in each (2, 2) block.
Second convolution layer	layers.Conv2D(64, (3, 3), activation='relu')	Another convolutional layer with 64 filters, each of size (3, 3)
Second MaxPooling layer	layers.MaxPooling2D((2, 2))	Further reduces the spatial dimensions.
Third convolution layer	layers.Conv2D(128, (3, 3), activation='relu')	A convolutional layer with 128 filters, each of size (3, 3)
Third MaxPooling layer	layers.MaxPooling2D((2, 2))	Further reduces the spatial dimensions.
Fourth convolution layer	layers.Conv2D(256, (3, 3), activation='relu')	A convolutional layer with 256 filters, each of size (3, 3)

Layer #	Layer	Description
Fourth MaxPooling layer	layers.MaxPooling2D((2, 2))	Further reduces the spatial dimensions.
Flatten Layer	layers.Flatten()	Flattens the 3D output from the convolutional layers into a 1D vector.
First Dense Layer	layers.Dense(1024, activation='relu')	A fully connected (dense) layer with 1024 neurons
Second Dense Layer	layers.Dense(512, activation='relu')	Another fully connected layer with 512 neurons
Output Layer	layers.Dense(num_classes, activation='softmax')	The output layer with ***num_classes*** neurons, where each neuron represents a class in a multi-class classification problem. The softmax activation function is used to output class probabilities.

Table 5 – Model 1 built from scratch comprising 4 Conv2D layers, MaxPool layers

Table 6 shows the model training parameters used for the custom neural net.

Model Training Parameters	Value	
optimizer	Adam	The default settings of the Adam class is used.
loss	categorical_crossentropy	Used for multi-class classification problems.
metrics	accuracy	This is used for measuring the performance of binary or multi-class classification functions.

Table 6 – Model 1 Training Parameters for Custom Neural Net

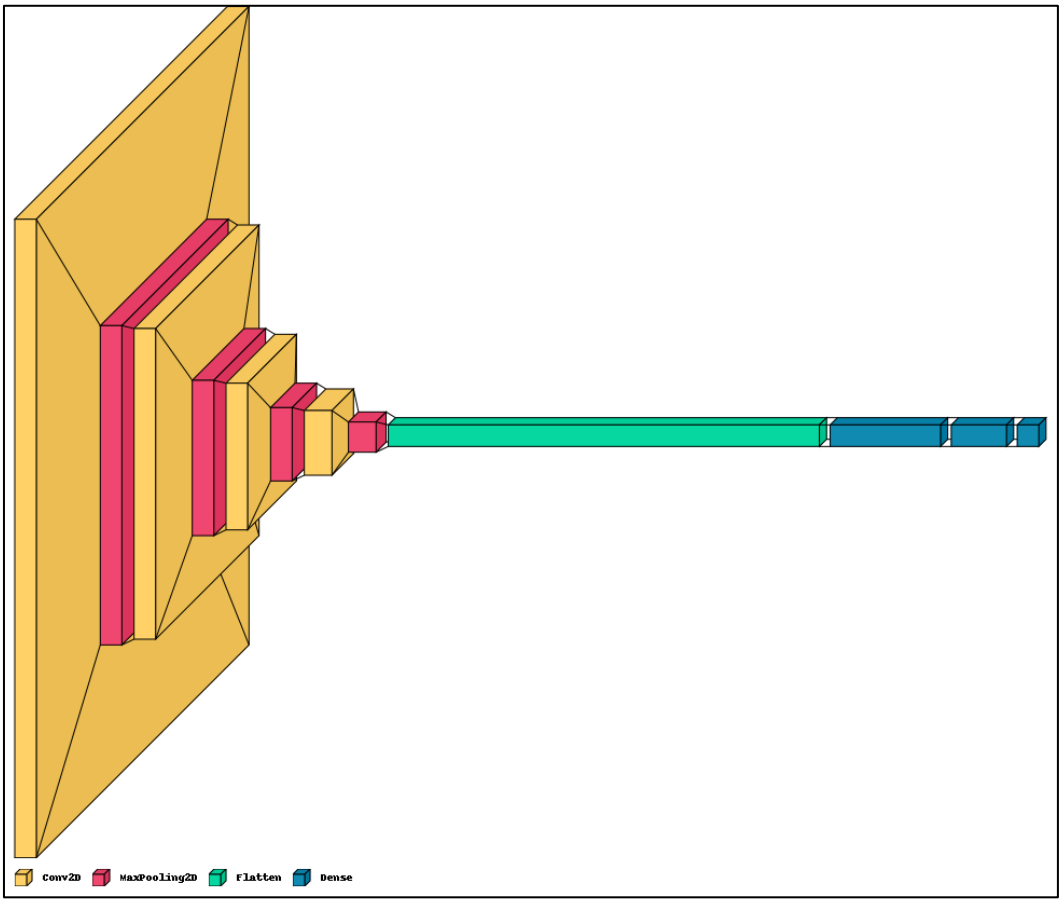


Figure 4 – Custom Model

A summary of the test accuracy results for Model 1 is shown in Table 7. The observations and analysis for each of the different runs for Model 1 are described in Sections 2.1.1 - 2.1.7.

	Experiment	Test Accuracy	Test Loss
1	Baseline - No Augmentation with Epochs = 30	48%	3.74
2	Baseline – Augmentation with Epochs = 20	67%	0.98
3	Baseline – Augmentation with Epochs = 30	67%	1.04
4	Regularization using L2 with regularization factor 0.001	65%	1.26
5	Regularization using L1 & L2 with regularization factor 0.001	50%	3.17
6	Adding Drop Out 0.5	63%	1.12
7	Early Stopping	63%	1.19

Table 7 – Model 1 Evaluation results against the test generator

2.1.1 Experiment 1: 30 Epochs, NO data augmentation

In this run, we train the model for 30 epochs, with no data augmentation. Figure 5 shows the training/validation accuracy and loss plots.

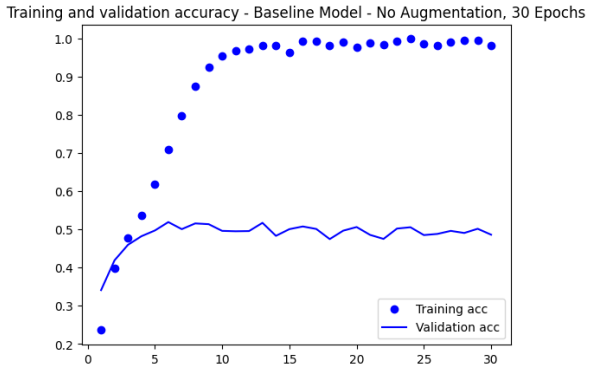
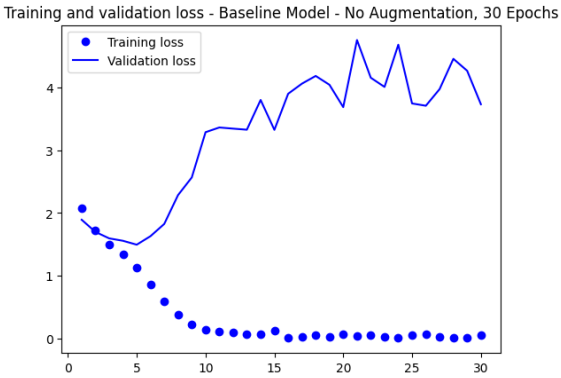
Training/Validation Accuracy	Training/Validation Loss
	

Figure 5 – Training/Validation accuracy and loss plots – Experiment 1: No data augmentation, 30 Epochs

Analysis of the Experiment

1. **Initial Observations:**

- **Epoch 1**: The initial training loss is high (2.08), and the accuracy is low (23.61%). The validation loss is also high (1.89) with an accuracy of 34.1%. This indicates that the model is starting to learn, but there's significant room for improvement.
- **Subsequent Epochs**: Both training and validation losses decrease over the first few epochs, and accuracies improve. By Epoch 5, the training accuracy is 61.85%, and the validation accuracy is 49.7%. This shows that the model is learning effectively initially.

2. **Mid to Late Training:**

- **Epochs 6-10**: The training loss continues to decrease, and the training accuracy improves significantly, reaching 95.31% by Epoch 10. However, the validation loss starts to increase, and the validation accuracy fluctuates around 51.55%, indicating potential overfitting.
- **Epochs 11-20**: The training accuracy continues to improve, reaching 97.72% by Epoch 20. However, the validation loss continues to increase, and the validation accuracy remains around 50.6%, suggesting that the model is overfitting to the training data.
- **Epochs 21-30**: The training accuracy stabilizes around 99.0%, but the validation accuracy fluctuates slightly, ending at 48.6%. The validation loss also fluctuates, indicating that the model is significantly overfitting.

3. **Key Points:**

- **Overfitting**: The significant increase in validation loss and the stagnation of validation accuracy indicate that the model is overfitting. This is a common issue when the model learns the training data too well but fails to generalize to new, unseen data.
- **Model Performance**: While the training accuracy is very high, the validation accuracy is relatively low and does not improve significantly after the initial epochs. This suggests that the model is not generalizing well.

4. **Recommendations:**

- **Data Augmentation**: Use data augmentation to increase the diversity of the training data and improve generalization.

2.1.2 Experiment 2: 20 Epochs with Data Augmentation

In this run, we train the model for 20 epochs, with data augmentation. Figure 6 shows the training/validation accuracy and loss plots.

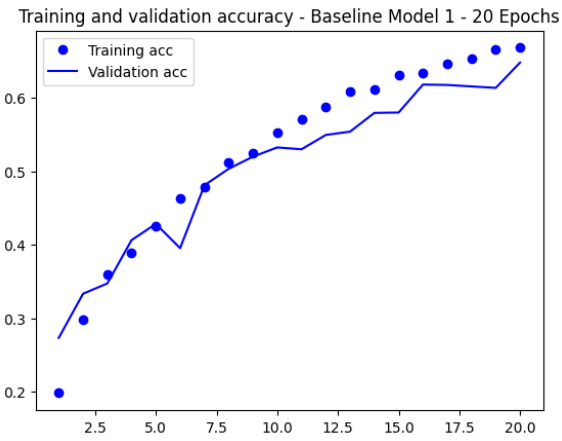
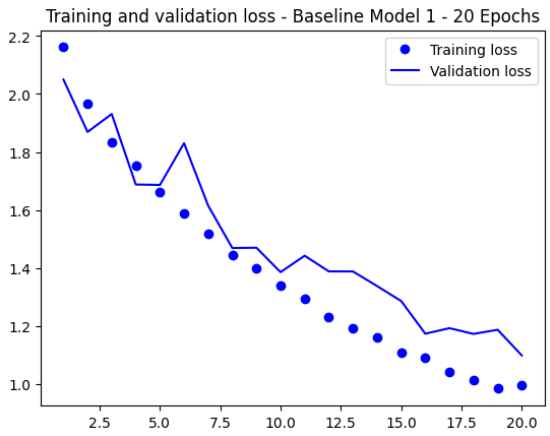
Training/Validation Accuracy	Training/Validation Loss
	

Figure 6 - Training/Validation accuracy and loss plots – Experiment 2: With data augmentation, 20 Epochs

Analysis of the Experiment

1. **Initial Observations:**

- **Epoch 1**: The initial training loss is high (2.16), and the accuracy is low (19.84%). The validation loss is also high (2.05) with an accuracy of 21.35%. This indicates that the model is starting to learn, but there's significant room for improvement.
- **Subsequent Epochs**: Both training and validation losses decrease over the first few epochs, and accuracies improve. By Epoch 5, the training accuracy is 42.6%, and the validation accuracy is 42.85%. This shows that the model is learning effectively initially.

2. **Mid to Late Training:**

- **Epochs 6-10**: The training loss continues to decrease, and the training accuracy improves significantly, reaching 55.24% by Epoch 10. However, the validation loss starts to increase, and the validation accuracy fluctuates around 52%, indicating potential overfitting.
- **Epochs 11-20**: The training accuracy continues to improve, reaching 66.8% by Epoch 20. However, the validation loss continues to increase, and the validation accuracy remains around 61%, suggesting that the model is overfitting to the training data.

3. **Key Points:**

- **Overfitting**: The significant increase in validation loss and the stagnation of validation accuracy indicate that the model is overfitting. This is a common issue when the model learns the training data too well but fails to generalize to new, unseen data.
- **Model Performance**

- While the training accuracy is improving, the validation accuracy is relatively low and does not improve significantly after the initial epochs. This suggests that the model is not generalizing well.
- However, compared to Experiment 1, the overfitting is significantly reduced, comparing Figure 6 vs. Figure 5.

4. **Recommendations:**
   - **Increase the number of epochs** to see if overfitting still occurs.

2.1.3 Experiment 3: 30 Epochs with Data Augmentation

In this run, we train the model for 30 epochs, with data augmentation. Figure 7 shows the training/validation accuracy and loss plots.

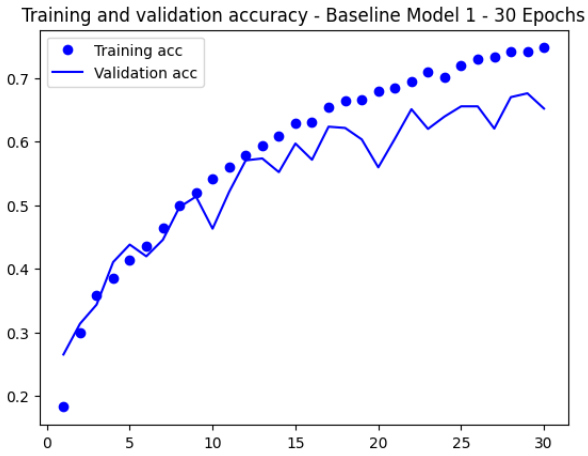
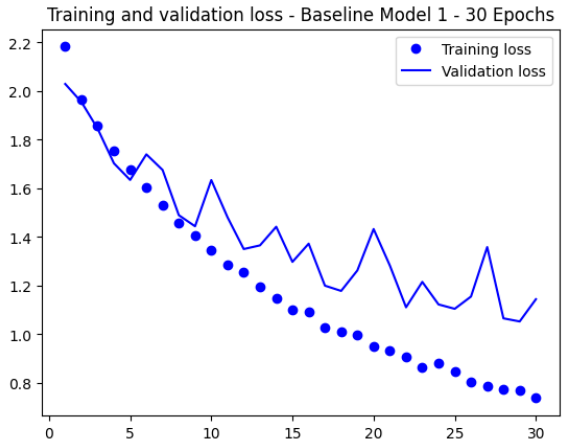
Training/Validation Accuracy	Training/Validation Loss
	

Figure 7 - Training/Validation accuracy and loss plots – Experiment 3: With data augmentation, 30 Epochs

Analysis of the Experiment

1. **Initial Observations:**

- **Epoch 1**: The initial training loss is high (2.18), and the accuracy is low (18.24%). The validation loss is also high (2.03) with an accuracy of 26.5%. This indicates that the model is starting to learn, but there's significant room for improvement.
- **Subsequent Epochs**: Both training and validation losses decrease over the first few epochs, and accuracies improve. By Epoch 5, the training accuracy is 41.44%, and the validation accuracy is 43.8%. This shows that the model is learning effectively initially.

2. **Mid to Late Training:**

- **Epochs 6-10**: The training loss continues to decrease, and the training accuracy improves significantly, reaching 54.17% by Epoch 10. However, the validation loss starts to increase, and the validation accuracy fluctuates around 45%, indicating potential overfitting.
- **Epochs 11-20**: The training accuracy continues to improve, reaching 67.932% by Epoch 20. However, the validation loss continues to increase, and the validation accuracy hovers between 52.1% - 62.2%, suggesting that the model is overfitting to the training data.

- **Epochs 21-30**: The training accuracy stabilizes around 74.8%, but the validation accuracy fluctuates slightly, ending at 65.2%. The validation loss also fluctuates, indicating that the model might be overfitting slightly.

3. **Key Points:**

- **Overfitting**: The significant increase in validation loss and the stagnation of validation accuracy indicate that the model is still overfitting.

4. **Recommendations:**

- **Regularization**: Add regularization techniques such as dropout or L2 regularization to reduce overfitting.

2.1.4 Experiment 4: 30 Epochs with Data Augmentation + L2 Regularization

In this run, we train the model for 30 epochs, with data augmentation and L2 regularization (weight factor: 0.001). Figure 8 shows the training/validation accuracy and loss plots.

L2 regularization is added at the two dense layers to reduce overfitting as shown in the code snippet below:

```
model_1_wr.add(layers.Dense(1024,
kernel_regularizer=regularizers.l2(0.001), activation='relu'))
model_1_wr.add(layers.Dense(512,
kernel_regularizer=regularizers.l2(0.001), activation='relu'))
```

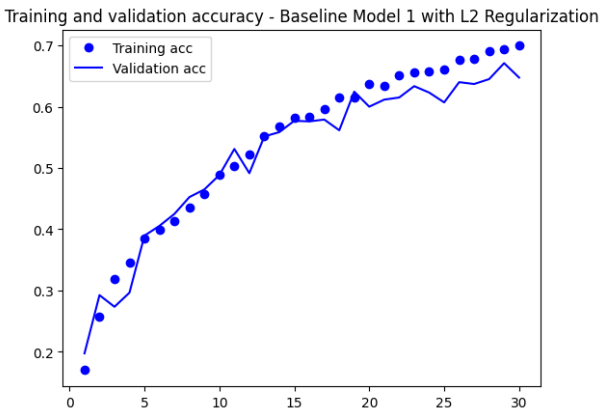
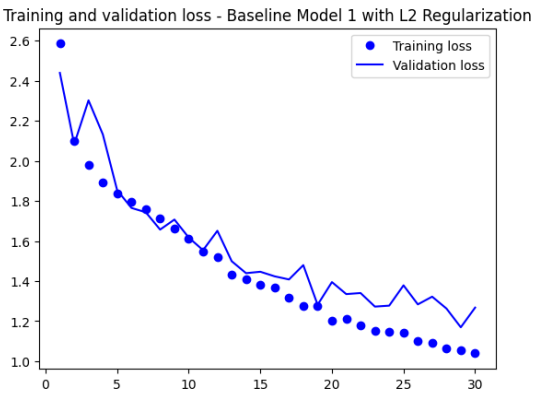
Training/Validation Accuracy	Training/Validation Loss
	

Figure 8 - Training/Validation accuracy and loss plots – Experiment 4: With data augmentation, 30 Epochs, L2 Regularization with weight factor 0.001 added

Analysis of the Experiment

1. **Initial Observations:**

- **Epoch 1**: The initial training loss is very high (2.58), and the accuracy is low (17%). The validation loss is also high (2.44) with an accuracy of 19.75%. This indicates that the model is starting to learn, but there's significant room for improvement.
- **Subsequent Epochs**: Both training and validation losses decrease over the first few epochs, and accuracies improve. By Epoch 5, the training accuracy is 38.43%, and

the validation accuracy is 38.95%. This shows that the model is learning effectively initially.

2. **Mid to Late Training:**

- **Epochs 6-10**: The training loss continues to decrease, and the training accuracy improves significantly, reaching 48.84% by Epoch 10. However, the validation loss starts to increase, and the validation accuracy fluctuates around 48.85%, indicating potential overfitting.
- **Epochs 11-20**: The training accuracy continues to improve, reaching 63.65% by Epoch 20. However, the validation loss continues to increase, and the validation accuracy remains around 60%, suggesting that the model is overfitting to the training data.
- **Epochs 21-30**: The training accuracy stabilizes around 69.93%, but the validation accuracy fluctuates slightly, ending at 64.75%. The validation loss also fluctuates, indicating that the model might be overfitting slightly.

3. **Key Points:**

- **Overfitting**: The model is still overfitting. Compared with Experiment 4, the overfitting is reduced when comparing the curves Figure 7 vs Figure 8.

4. **Recommendations:**

- **Regularization**: Continue using L1 + L2 regularization to still if overfitting can be further reduced.

2.1.5 Experiment 5: 30 Epochs with Data Augmentation with L1+L2 regularization

In this run, we train the model for 30 epochs, with data augmentation and L1+L2 regularization (weight factor: 0.001). Figure 9 shows the training/validation accuracy and loss plots.

L1 & L2 regularizes were added to the 2 dense layers to reduce overfitting as follows:

```
model_1_wr_l1_l2.add(layers.Dense(1024,
kernel_regularizer=regularizers.l1_l2(l1=0.001,l2=0.001),
activation='relu'))

model_1_wr_l1_l2.add(layers.Dense(512,
kernel_regularizer=regularizers.l1_l2(l1=0.001,l2=0.001),
activation='relu'))
```

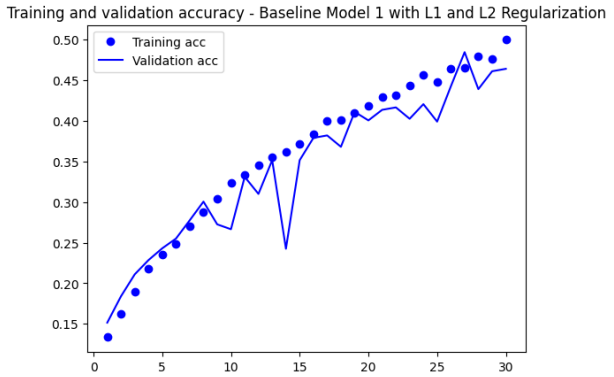
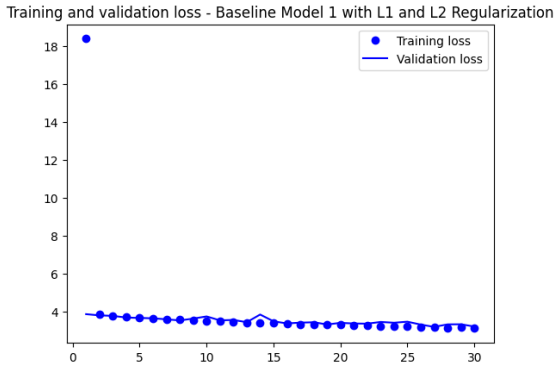
Training/Validation Accuracy	Training/Validation Loss
	

Figure 9 - Training/Validation accuracy and loss plots – Experiment 5: With data augmentation, 30 Epochs, L1 & L2 Regularization with weight factor 0.001 added

Analysis of the Experiment

1. **Initial Observations:**

- **Epoch 1**: The initial training loss is extremely high (18.38), and the accuracy is very low (13.35%). The validation loss is also high (3.87) with an accuracy of 15.15%. This indicates that the model is not learning effectively from the start.
- **Epochs 2-5**:
- By Epoch 5, the training loss decreases to 3.68 and accuracy improves to 23.53%.
- Validation metrics are also improving, though validation loss remains higher than the training loss, indicating potential overfitting

2. **Mid to Late Training:**

- By Epoch 15, training accuracy reaches 37.15%, and validation accuracy improves to 35.15%.
- The slight increase in validation loss around Epochs 9-14 suggests the model might be starting to overfit.
- **Epochs 23-30**:
- Training accuracy steadily increases, reaching 49.96% by Epoch 30.
- Validation accuracy shows consistent improvement, peaking at 48.45% in Epoch 27.
- The model shows a better fit by the end of the run, with validation loss decreasing significantly to 3.21 by Epoch 30.

2. **Key Observations:**

- The model steadily improves in terms of both loss and accuracy over 30 epochs. The use of L1+L2 regularization helps in preventing overfitting to some extent, though there are still signs of overfitting as indicated by the validation loss initially increasing during the mid-epochs.
- Validation Performance: The validation accuracy and loss show that the model is able to generalize quite well. The fluctuation in validation loss suggests that more epochs might be needed or that early stopping could be considered to prevent overfitting.
- Regularization Effect: The combination of L1 and L2 regularization appears to be effective, as indicated by the consistent improvement in accuracy and controlled loss.

3. **Recommendations:**

- Use dropout instead.

2.1.6 Experiment 6: 30 Epochs with Data Augmentation with Dropout

In this run, we train the model for 30 epochs, with data augmentation and Dropout (dropout factor: 0.5). Figure 10 shows the training/validation accuracy and loss plots.

Adding dropout of 0.5 after each of the 2 dense layers:

```
model_1_do.add(layers.Dropout(0.5))
```

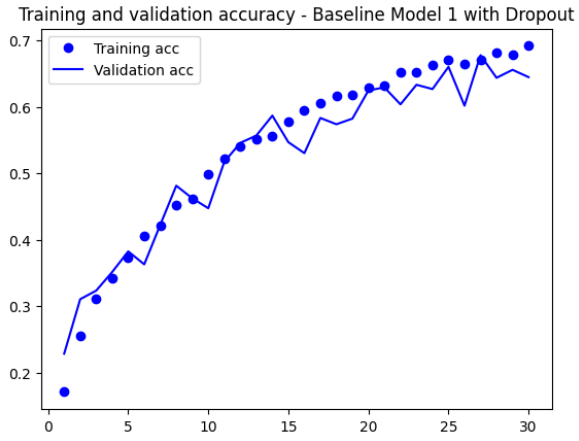
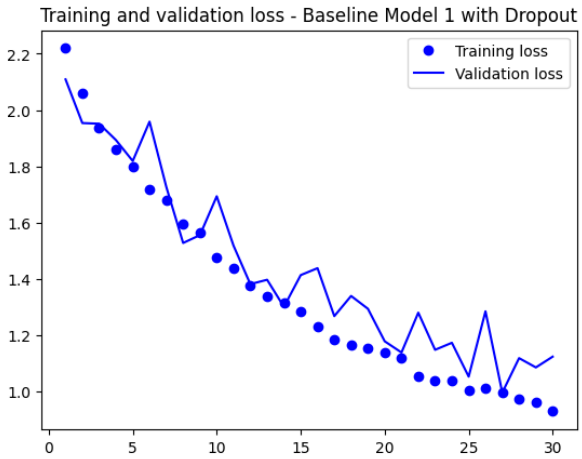
Training/Validation Accuracy	Training/Validation Loss
	

Figure 10 - Training/Validation accuracy and loss plots – Experiment 6: With data augmentation, 30 Epochs, Dropout of 0.5 added

Analysis of the Experiment

1. **Initial Observations:**

- **Epoch 1**: The initial training loss is high (2.22), and the accuracy is low (17.1%). The validation loss is also high (2.11) with an accuracy of 22.85%. This indicates that the model is starting to learn, but there's significant room for improvement.
- **Subsequent Epochs**: Both training and validation losses decrease over the first few epochs, and accuracies improve. By Epoch 5, the training accuracy is 37.37%, and the validation accuracy is 38.25%. This shows that the model is learning effectively initially.

2. **Mid to Late Training:**

- **Epochs 6-10**: The training loss continues to decrease, and the training accuracy improves significantly, reaching 49.81% by Epoch 10. However, the validation loss starts to increase, and the validation accuracy fluctuates around 48.15%, indicating potential overfitting.
- **Epochs 11-20**: The training accuracy continues to improve, reaching 62.84% by Epoch 20. However, the validation loss continues to increase, and the validation accuracy remains around 62.5%, suggesting that the model is overfitting to the training data.
- **Epochs 21-30**: The training accuracy stabilizes around 69.2%, but the validation accuracy fluctuates slightly, ending at 64.5%. The validation loss also fluctuates, indicating that the model might be overfitting slightly.

3. **Key Points:**

- **Overfitting**: The significant increase in validation loss and the stagnation of validation accuracy indicate that the model is overfitting. This is a common issue when the model learns the training data too well but fails to generalize to new, unseen data.
- **Model Performance**: While the training accuracy is improving, the validation accuracy is relatively low and does not improve significantly after the initial epochs. This suggests that the model is not generalizing well.

4. **Recommendations:**

- **Early Stopping**: Implement early stopping to prevent overfitting. Monitor the validation loss and stop training when it stops improving.

2.1.7 Experiment 7: 30 Epochs with Data Augmentation with Early Stopping

In this run, we train the model for 30 epochs, with data augmentation and early stopping. Figure 11 shows the training/validation accuracy and loss plots.

Adding early stopping in the training phase:

```
callback = keras.callbacks.EarlyStopping(monitor='loss', patience=3)

history_1_es = model_1_es.fit(
    train_generator,
    validation_data = validation_generator,
    callbacks=[callback],
    epochs = n_epochs
)
```

monitor: Quantity to be monitored. In this case, it is the training loss.

patience = 3: Number of epochs with no improvement after which training will be stopped. If the monitored metric does not improve for 3 consecutive epochs, the training will be stopped.

Purpose: The early stopping callback is used to stop training the model when a monitored metric has stopped improving. This helps to prevent overfitting and saves computational resources by stopping the training process early.

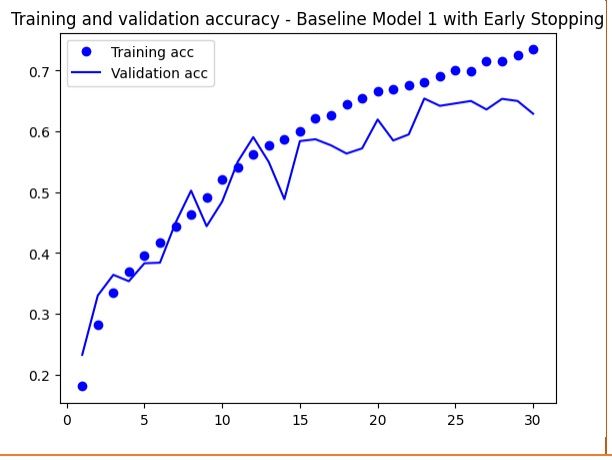
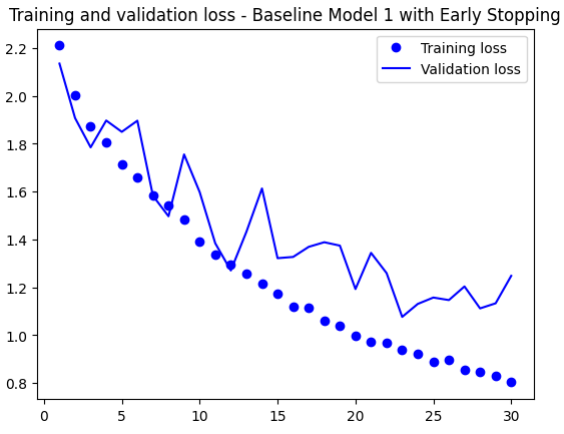
Training/Validation Accuracy	Training/Validation Loss
	

Figure 11 - Training/Validation accuracy and loss plots – Experiment 6: With data augmentation, 30 Epochs, Early Stopping added

Analysis of the Experiment

1. **Initial Observations:**

- **Epoch 1**: The initial training loss is high (2.2), and the accuracy is low (18.05%). The validation loss is also high (2.13) with an accuracy of 23.25%. This indicates that the model is starting to learn, but there's significant room for improvement.
- **Subsequent Epochs**: Both training and validation losses decrease over the first few epochs, and accuracies improve. By Epoch 5, the training accuracy is 39.59%, and the validation accuracy is 38.3%. This shows that the model is learning effectively initially.

2. **Mid to Late Training:**

- **Epochs 6-10**: The training loss continues to decrease, and the training accuracy improves significantly, reaching 52.11% by Epoch 10. However, the validation loss starts to increase, and the validation accuracy fluctuates around 50.25%, indicating potential overfitting.
- **Epochs 11-20**: The training accuracy continues to improve, reaching 66.56% by Epoch 20. However, the validation loss continues to increase, and the validation accuracy remains around 57.7%, suggesting that the model is overfitting to the training data.
- **Epochs 21-30**: The training accuracy stabilizes around 73.45%, but the validation accuracy fluctuates slightly, ending at 62.9%. The validation loss also fluctuates, indicating that the model might be overfitting slightly.

3. **Key Points:**

- **Overfitting**: The significant increase in validation loss and the stagnation of validation accuracy indicate that the model is overfitting. This is a common issue when the model learns the training data too well but fails to generalize to new, unseen data.
- **Model Performance**: While the training accuracy is improving, the validation accuracy is relatively low and does not improve significantly after the initial epochs. This suggests that the model is not generalizing well.

5. **Recommendations:**

- **Use of pre-trained models** to see if the results can be improved.

2.2 Model 2 – Using Pre-trained VGG16

For this experiment, I used a pre-trained model, VGG16 as the base model.

```
conv_base = VGG16(weights='imagenet',
                  include_top=False,
                  input_shape=(img_size, img_size, 3))
```

A visual representation of the VGG16 architecture is shown in Figure 12.

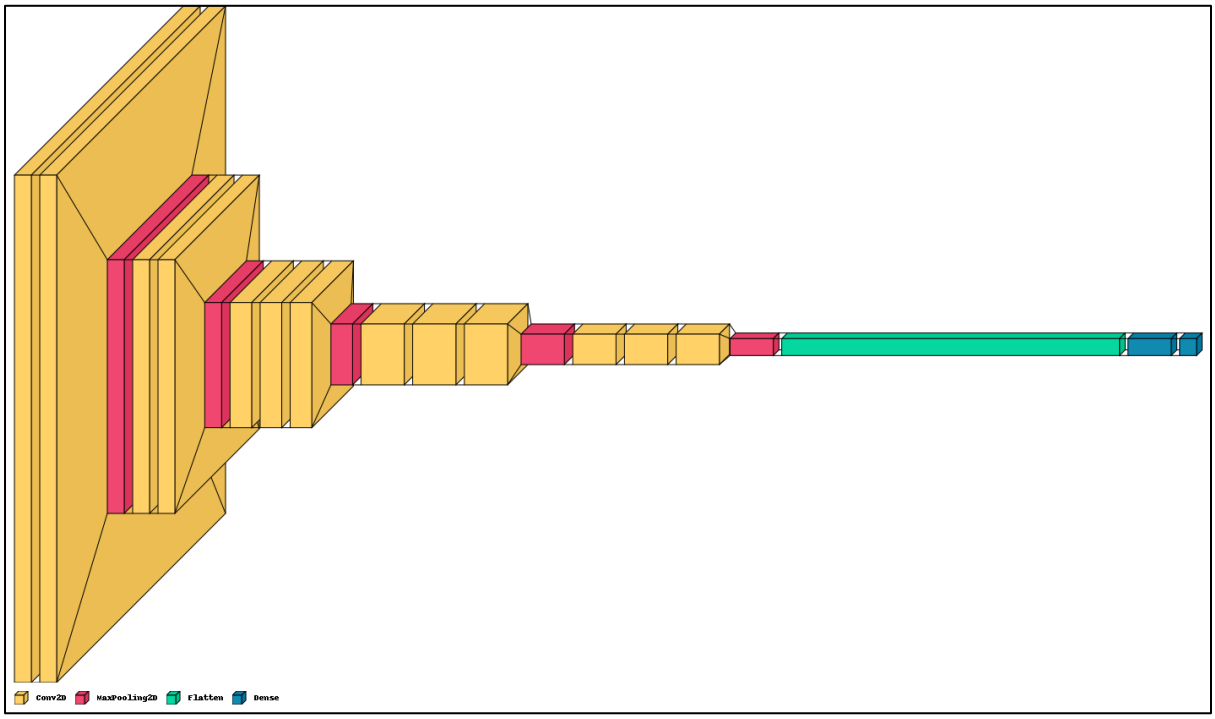


Figure 12 – Using VGG16 as a pre-trained model

We use the same model training parameters as for the custom neural net. The values are shown in Table 8.

Model Training Parameters	Value	
optimizer	Adam	The default settings of the Adam class is used.
loss	categorical_crossentropy	Used for multi-class classification problems.
metrics	accuracy	This is used for measuring the performance of binary or multi-class classification functions.

Table 8 – Model 2 Training Parameters using VGG16 as the pre-trained model

A summary of the test accuracy results for Model 2 is shown in Table 9. The observations and analysis for each of the different runs for Model 2 are described in Sections 2.2.1- 2.2.5.

#	Experiment	Test Accuracy	Test Loss
1	Feature Extraction NO Data Augmentation - Epochs = 30	77%	1.31
2	Feature Extraction with Data Augmentation - Epochs = 30	77%	1.06
3	Feature Extraction with Data Augmentation (Increase # Epochs to 50)	78%	1.20
4	Feature Extraction with Data Augmentation + Fine Tuning (Epochs = 30) by ***unfreezing last 3 layers*** of base network	10%	2.31

#	Experiment	Test Accuracy	Test Loss
5	Feature Extraction with Data Augmentation + Fine Tuning (Epochs = 30) by ***unfreezing last layer*** of base network	82%	0.86

Table 9 – Model 2 Evaluation results against the test generator

2.2.1 Experiment 1: 30 Epochs NO Data Augmentation

In this run, we train the model for 30 epochs, with no data augmentation. Figure 13 shows the training/validation accuracy and loss plots.

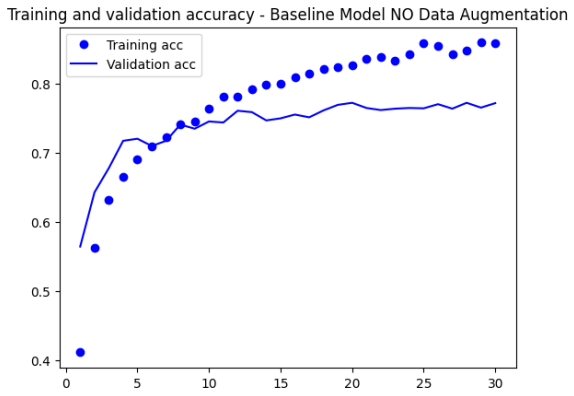
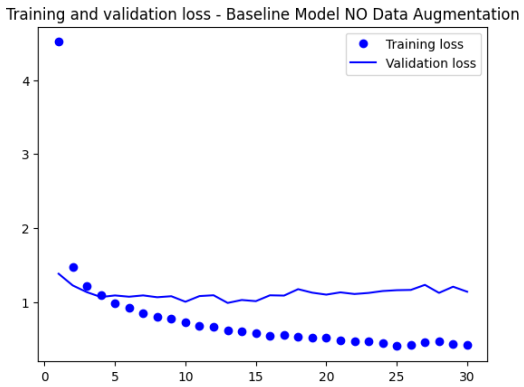
Training/Validation Accuracy	Training/Validation Loss
	

Figure 13 - Training/Validation accuracy and loss plots – Experiment 1: No data augmentation, 30 Epochs

Analysis of the Experiment

1. **Initial Observations:**

- **Epoch 1**: The initial training loss is quite high (4.25), but the accuracy is already at 44.43%. This suggests that the model is starting to learn, but there's a lot of room for improvement.
- **Validation Performance**: The validation loss starts at 1.33 with an accuracy of 59.8%, indicating that the model is generalizing reasonably well from the start.

2. **Progression:**

- **Epochs 2-5**: Both training and validation losses decrease significantly, and accuracies improve. By Epoch 5, the training accuracy is 68.75%, and the validation accuracy is 73.1%. This shows that the model is learning effectively.
- **Epochs 6-10**: The training loss continues to decrease, but the validation loss starts to fluctuate slightly. However, the validation accuracy remains relatively stable around 72-74%, indicating good generalization.

3. **Mid to Late Training:**

- **Epochs 11-20**: The training accuracy continues to improve, reaching 82.4% by Epoch 20. The validation accuracy also improves, peaking at 75.75%. However, the validation loss shows some fluctuations, suggesting potential overfitting.

- **Epochs 21-30**: The training accuracy stabilizes around 83-85%, but the validation accuracy fluctuates slightly, ending at 76.25%. The validation loss also fluctuates, indicating that the model might be overfitting slightly.

1. **Key Points:**

- **Overfitting Signs**: The fluctuations in validation loss and accuracy in the later epochs suggest that the model might be starting to overfit. This is indicated by the validation loss not consistently decreasing and the accuracy not improving significantly.
- **Model Performance**: Overall, the model shows good performance, with a final training accuracy of 85.4% and a validation accuracy of 76.25%. The validation accuracy is relatively close to the training accuracy, indicating decent generalization.

2. **Recommendations:**

- **Data Augmentation**: Use data augmentation to increase the diversity of the training data and improve generalization.

Overall, the experiment shows promising results, but there's room for improvement to enhance the model's generalization and prevent overfitting.

2.2.2 Experiment 2: 30 Epochs with Data Augmentation

In this run, we train the model for 30 epochs, with data augmentation. Figure 14 shows the training/validation accuracy and loss plots.

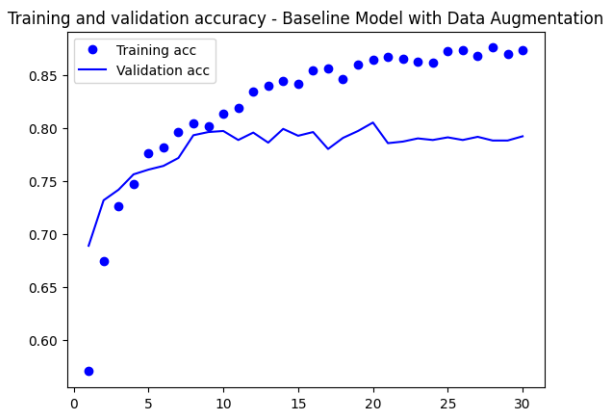
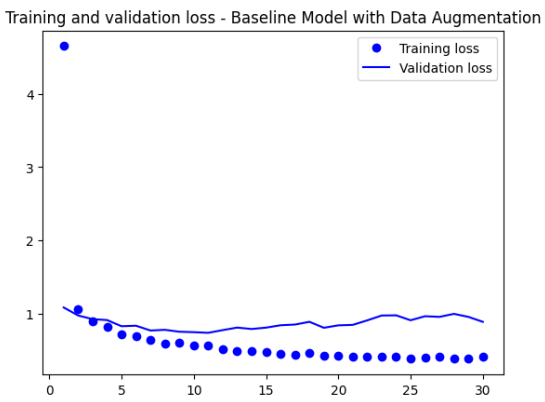
Training/Validation Accuracy	Training/Validation Loss
	

Figure 14 - Training/Validation accuracy and loss plots – Experiment 2: With data augmentation, 30 Epochs

Analysis of the Experiment

1. **Initial Observations:**

- **Epoch 1**: The initial training loss is high (7.44), but the accuracy is already at 54.04%. This suggests that the model is starting to learn, but there's significant room for improvement.
- **Validation Performance**: The validation loss starts at 1.07 with an accuracy of 67.95%, indicating that the model is generalizing reasonably well from the start.

2. **Progression:**

- **Epochs 2-5**: Both training and validation losses decrease significantly, and accuracies improve. By Epoch 5, the training accuracy is 76.95%, and the validation accuracy is 76.50%. This shows that the model is learning effectively.
- **Epochs 6-10**: The training loss continues to decrease, and the validation loss also decreases, indicating good generalization. The validation accuracy improves to around 79.45%.

3. **Mid to Late Training:**

- **Epochs 11-20**: The training accuracy continues to improve, reaching 86.12% by Epoch 20. The validation accuracy also improves, peaking at 79%. However, the validation loss shows some fluctuations, suggesting potential overfitting.
- **Epochs 21-30**: The training accuracy stabilizes around 87.01%, but the validation accuracy fluctuates slightly, ending at 78.6%. The validation loss also fluctuates, indicating that the model might be overfitting slightly.

4. **Key Points:**

- **Overfitting Signs**: The fluctuations in validation loss and accuracy in the later epochs suggest that the model might be starting to overfit. This is indicated by the validation loss not consistently decreasing and the accuracy not improving significantly.
- **Model Performance**: Overall, the model shows good performance, with a final training accuracy of 87.01% and a validation accuracy of 78.6%. The validation accuracy is relatively close to the training accuracy, indicating decent generalization.

5. **Recommendations:**

- **Increase number of epochs** to observe if overfitting becomes pronounced.

Overall, the experiment shows promising results, but there's room for improvement to enhance the model's generalization and prevent overfitting.

2.2.3 Experiment 3: 50 Epochs with Data Augmentation

In this run, we train the model for 50 epochs, with data augmentation. Figure 15 shows the training/validation accuracy and loss plots.

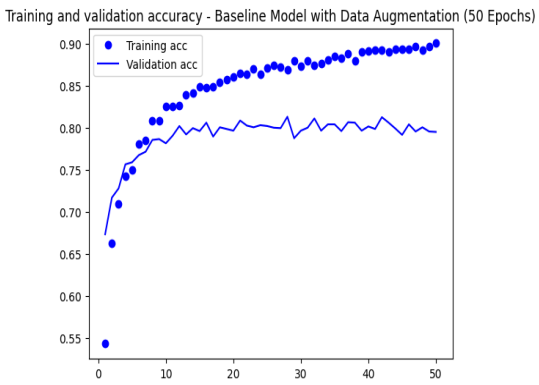
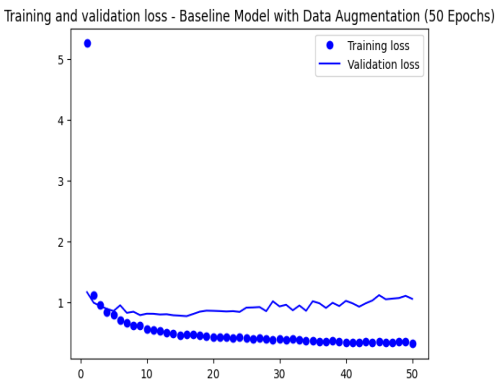
Training/Validation Accuracy	Training/Validation Loss
	

Figure 15 - Training/Validation accuracy and loss plots – Experiment 3: With data augmentation, 50 Epochs

Analysis of the Experiment

1. **Initial Observations:**

- **Epoch 1**: The initial training loss is high (5.25), but the accuracy is already at 53.9%. This suggests that the model is starting to learn, but there's significant room for improvement.
- **Validation Performance**: The validation loss starts at 1.23 with an accuracy of 66.9%, indicating that the model is generalizing reasonably well from the start.

2. **Progression:**

- **Epochs 2-5**: Both training and validation losses decrease significantly, and accuracies improve. By Epoch 5, the training accuracy is 75.92%, and the validation accuracy is 77.1%. This shows that the model is learning effectively.
- **Epochs 6-10**: The training loss continues to decrease, and the validation loss also decreases, indicating good generalization. The validation accuracy improves to around 79.25%.

3. **Mid to Late Training:**

- **Epochs 11-20**: The training accuracy continues to improve, reaching 86% by Epoch 20. The validation accuracy also improves, peaking at 79.35%. However, the validation loss shows some fluctuations, suggesting potential overfitting.
- **Epochs 21-30**: The training accuracy stabilizes around 88.6%, but the validation accuracy fluctuates slightly, ending at 79.65%. The validation loss also fluctuates, indicating that the model might be overfitting slightly.

4. **Late Training:**

- **Epochs 31-50**: The training accuracy continues to improve, reaching 89.95% by Epoch 50. The validation accuracy fluctuates but remains relatively stable around 79% to 80%. The validation loss shows more fluctuations, indicating potential overfitting.

5. **Key Points:**

- **Overfitting Signs**: The fluctuations in validation loss and accuracy in the later epochs suggest that the model might is overfitting. This is indicated by the validation loss not consistently decreasing and the accuracy not improving significantly.
- **Model Performance**: Overall, the model shows good performance, with a final training accuracy of 90% and a validation accuracy of 79.3%. The validation accuracy is relatively close to the training accuracy, indicating decent generalization.

6. **Recommendations:**

- **Fine tuning**: which consists in unfreezing a few of the top layers of a frozen model base used for feature extraction, and jointly training both the newly added part of the model (the fully-connected classifier) and these top layers.

2.2.4 Experiment 4: 30 Epochs with Data Augmentation + Fine Tuning (Unfreeze last 3 layers)

In this run, we train the model for 30 epochs, with data augmentation + fine tuning (unfreeze last 3 layers). Figure 16 shows the training/validation accuracy and loss plots.

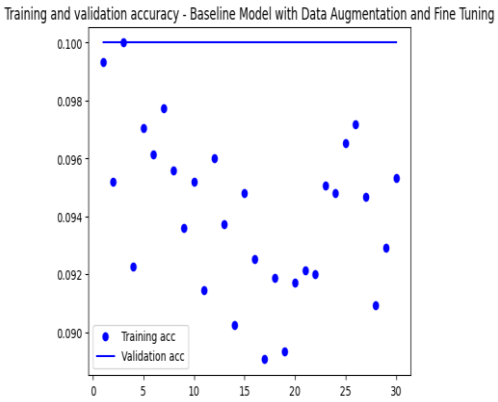
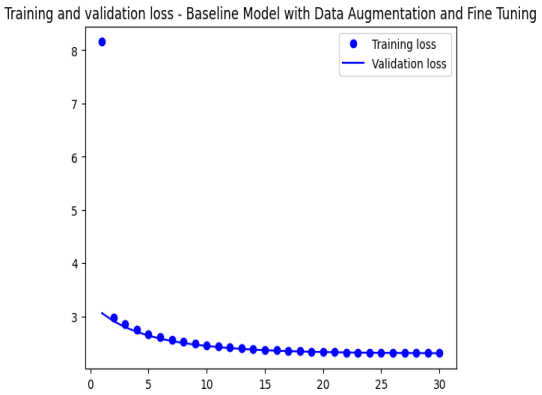
Training/Validation Accuracy	Training/Validation Loss
	

Figure 16 - Training/Validation accuracy and loss plots – Experiment 4: With data augmentation + Fine tuning, 30 Epochs

1. **Initial Observations:**

- **Epoch 1**: The initial training loss is high (3.82), and the accuracy is very low (9.33%). The validation loss is also high (3.05) with an accuracy of 10.00%. This indicates that the model is not learning effectively from the start.
- **Subsequent Epochs**: The loss values plateau around 2.31, and the accuracy remains unchanged. This indicates that the model has reached a point where further training does not significantly improve performance.

2. **Key Issues:**

- **Unfreezing Layers**: Unfreezing the last three layers of VGG16 allows for fine-tuning, but it seems that the model is not benefiting significantly from this adjustment in terms of accuracy. Although the model is learning to reduce the loss, it is not translating into improved accuracy

3. **Recommendations:**

- **Unfreezing less layers**

2.2.5 Experiment 5: 30 Epochs with Data Augmentation + Fine Tuning (Unfreeze last layer)

In this run, we train the model for 30 epochs, with data augmentation + fine tuning (unfreeze last layer). Figure 17 shows the training/validation accuracy and loss plots.

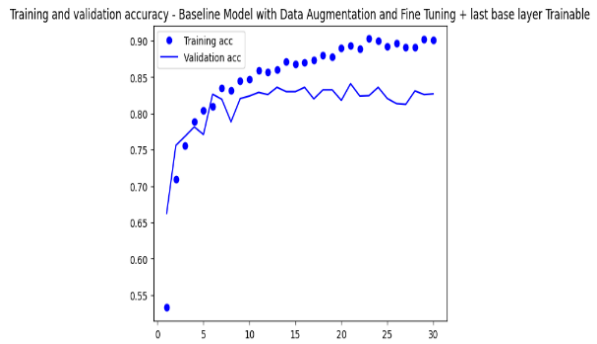
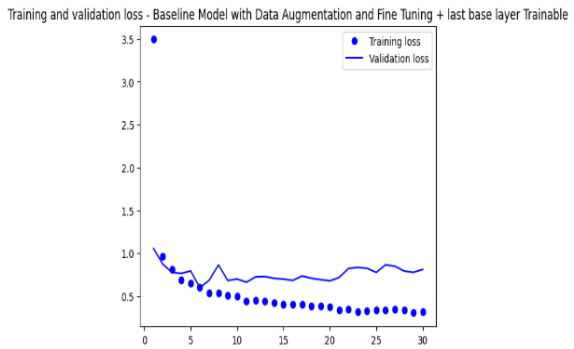
Training/Validation Accuracy	Training/Validation Loss
	

Figure 17 - Training/Validation accuracy and loss plots – Experiment 5: With data augmentation + Fine Tuning, 30 Epochs

Analysis of the Experiment

1. **Initial Observations:**

- **Epoch 1**: The initial training loss is high (4.09), but the accuracy is already at 44.84%. The validation loss is 1.42 with an accuracy of 60.35%. This indicates that the model is starting to learn, but there's significant room for improvement.
- **Subsequent Epochs**: Both training and validation losses decrease significantly over the epochs, and accuracies improve. By Epoch 5, the training accuracy is 76.63%, and the validation accuracy is 74.65%. This shows that the model is learning effectively.

2. **Mid to Late Training:**

- **Epochs 6-10**: The training loss continues to decrease, and the validation loss also decreases, indicating good generalization. The validation accuracy improves to around 80.3%.
- **Epochs 11-20**: The training accuracy continues to improve, reaching 85.12% by Epoch 20. The validation accuracy also improves, peaking at 81%. However, the validation loss shows some fluctuations, suggesting potential overfitting.
- **Epochs 21-30**: The training accuracy stabilizes around 87.1%, but the validation accuracy fluctuates slightly, ending at 82.35%. The validation loss also fluctuates, indicating that the model might be overfitting slightly.

3. **Key Points:**

- **Overfitting Signs**: The fluctuations in validation loss and accuracy in the later epochs suggest that the model might be starting to overfit. This is indicated by the validation loss not consistently decreasing and the accuracy not improving significantly.
- **Model Performance**: Overall, the model shows good performance, with a final training accuracy of 87.1% and a validation accuracy of 82.35%. The validation accuracy is relatively close to the training accuracy, indicating decent generalization.

Explanation:

Unfreezing fewer layers in your pre-trained model helps maintain the generalization capabilities learned from the large dataset, reduces the risk of overfitting by lowering complexity, thus ensuring a stable learning process. The model gradually adapts to the new task with small adjustments in the last unfrozen layer, making it less likely to overfit.

3 Model Evaluation using Test Images

From the different experimental runs of Model 1 in Table 7, experiment 2 (20 epochs, data augmentation only) shows the best results.

For Model 2, experiment 5 using VGG16 as the pre-trained model, with data augmentation, fine tuning by unfreezing only the last layer (see Table 10) shows the best results.

#	Experiment	Test Accuracy	Test Loss
Model 1 – Custom Neural Net	Data Augmentation Only (Epochs = 20)	67.2%	0.98
Model 2 – Using Pre-trained model, VGG16	Feature Extraction with Data Augmentation + Fine Tuning (Epochs = 30) by ***unfreezing last layer*** of base network	81.7%	0.86

Table 10 – Finding the best model

1. **Observations:**

- **Accuracy**: The pre-trained VGG16 model significantly outperforms the custom neural network in terms of test accuracy (81.7% vs. 67.2%). This indicates that the VGG16 model is better at generalizing to new, unseen data.
- **Loss**: The test loss for both models is relatively close (0.86 for VGG16 vs. 0.98 for the custom model). However, the lower loss in the VGG16 model suggests it makes more accurate predictions on average.

2. **Conclusions:**

- **Pre-trained Models**: The use of a pre-trained model like VGG16 provides a substantial performance boost. This is likely due to the extensive training on a large and diverse dataset (ImageNet), which allows the model to learn rich feature representations that are transferable to other tasks.
- **Custom Model Limitations**: While the custom neural network performs reasonably well, it lacks the depth and pre-trained knowledge of VGG16. This results in lower accuracy and slightly higher loss.

3. **Recommendations**:

Leverage Pre-trained Models: For tasks involving image classification, leveraging pre-trained models like VGG16 can provide a significant performance boost. These models have already learned useful features from large datasets, which can be fine-tuned for specific tasks.

We can experiment with other regularization techniques, dropout, changing the optimizer and learning rates to see if there is further improvement in the pre-trained models.

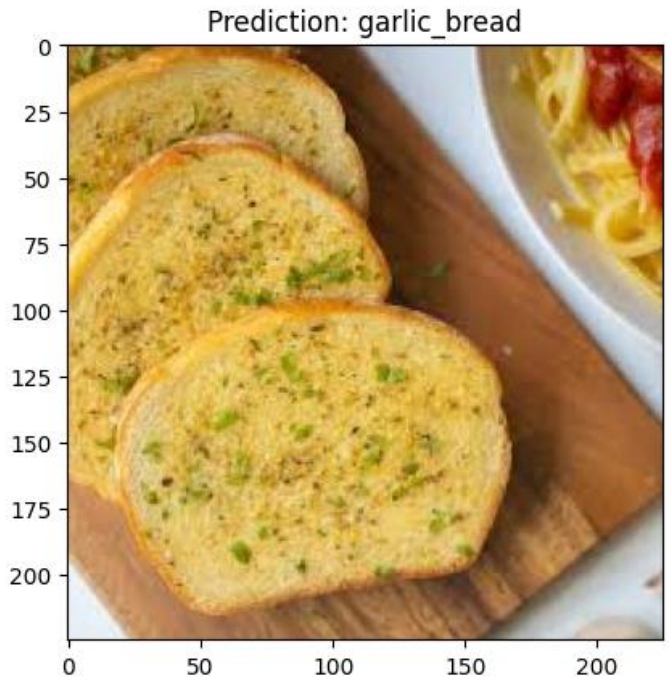
We can also explore using other pre-trained models e.g. DenseNets, EfficientNet if we can get even better accuracy.

4 Use Best Model to make Predictions

A number of food images were downloaded from the internet and placed in the Test_Data directory. We iterate through each image in the test directory and call the prediction function

passing in the best model as an argument. Here is a sampling of the test run. I pick a few results from this test run and this is shown in Table 11.

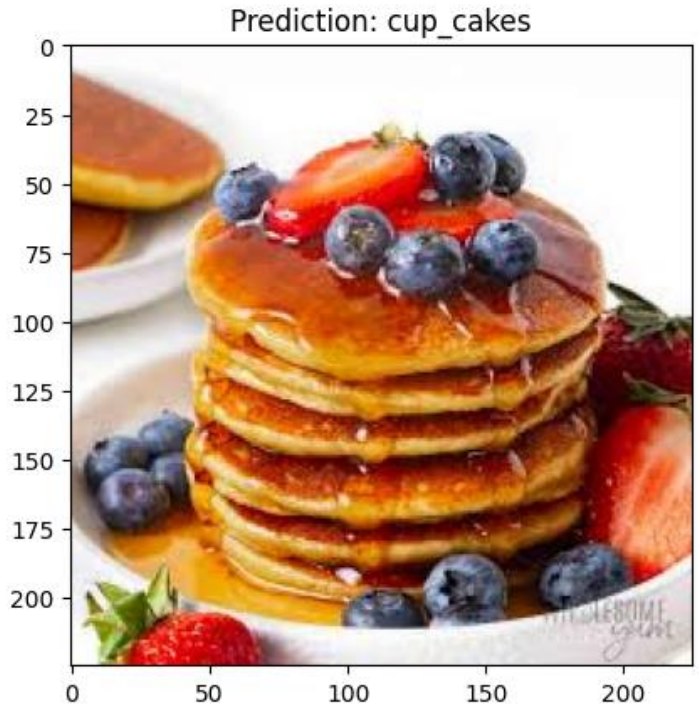
Images/Prediction Results



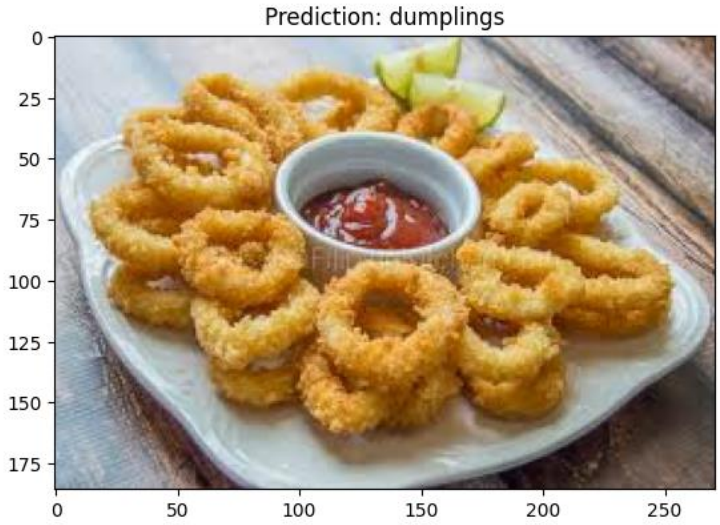
	baby_back_ribs	bibimbap	cup_cakes	dumplings	fried_calamari	garlic_bread	lasagna	pancakes	prime_rib	tiramisu
0	0.070995	0.065985	0.096955	0.115504	0.110709	0.119415	0.116843	0.114718	0.078601	0.110275

Images/Prediction Results

	baby_back_ribs	bibimbap	cup_cakes	dumplings	fried_calamari	garlic_bread	lasagna	pancakes	prime_rib	tiramisu
0	0.075013	0.061898	0.099029	0.120662	0.106468	0.112444	0.127720	0.116125	0.076525	0.104115



	baby_back_ribs	bibimbap	cup_cakes	dumplings	fried_calamari	garlic_bread	lasagna	pancakes	prime_rib	tiramisu
0	0.077408	0.058858	0.138315	0.119846	0.102444	0.102270	0.100232	0.131152	0.072296	0.097179



	baby_back_ribs	bibimbap	cup_cakes	dumplings	fried_calamari	garlic_bread	lasagna	pancakes	prime_rib	tiramisu
0	0.076411	0.054054	0.107655	0.145584	0.119747	0.106315	0.116429	0.127931	0.068609	0.077264

Table 11 – Predicting food types using the best model

5 Conclusions

The best model's prediction was not as good as expected on real world images. This can be explained by:

1. **Data Mismatch**: Pre-trained models are usually trained on large, well-curated datasets like ImageNet. Real-world images often differ significantly in terms of quality, lighting, angles, and backgrounds, leading to poor performance.
2. **Class Imbalance**: If the real-world dataset has an uneven distribution of classes, the model might struggle to correctly classify underrepresented classes.
3. **Noise and Variability**: Real-world images often contain noise, distortions, and variations that the model hasn't encountered during training, affecting its ability to generalize.
4. **Domain Shift**: The distribution of data in the real world can be different from the training data, causing a domain shift that degrades performance.
5. **Overfitting**: Pre-trained models might be overfitted to the training data, making them less adaptable to new, unseen images.

We can also use the following visualization techniques to help analyse model predictions such as:

a) Showing intermediate convnet outputs to give us a better insight to the meaning of the individual convnet filters by understanding how successive convnet layers transform their input.
b) Visualizing convnet filters to understand the pattern each filter is receptive to.
c) Envisioning the heatmaps of class activations in an image will allow us to show how intensely an input image activates the different channels, indicating the importance of each location in the image.

Other model prediction analysis tools we can use are LIME [1]**Error! Reference source not found.**, SHAP [2] to help us understand which parts of the image are influencing the model's decision. This can help identify if the model is focusing on the right features or if it's being misled by noise or irrelevant parts of the image.

6 References

[1]. "Why Should I Trust You?": Explaining the Predictions of Any Classifier, Marco Tulio Ribeiro, Sameer Singh, Carlos Guestrin, Aug 2016

[2]. A Unified Approach to Interpreting Model Predictions, Scott Lundberg, Su-In Lee, Nov 2017