DUBLIN CITY UNIVERSITY

ELECTRONIC AND COMPUTER ENGINEERING

EE544 - Computer Vision

Assignment 1

Author

Michael Lenehan   michael.lenehan4@mail.dcu.ie

Student Number: 15410402

15/04/2020

Declaration

I declare that this material, which I now submit for assessment, is entirely my own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my work. I understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. I have read and understood the Assignment Regulations set out in the module documentation. I have identified and included the source of all facts, ideas, opinions, and viewpoints of others in the assignment references. Direct quotations from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged and the source cited are identified in the assignment references. This assignment, or any part of it, has not been previously submitted by me or any other person for assessment on this or any other course of study.

I have read and understood the DCU Academic Integrity and Plagiarism at `https://www4.dcu.ie/sites/default/files/policy/1%20-%20integrity_and_plagiarism_ovpaa_v3.pdf` and IEEE referencing guidelines found at `https://loop.dcu.ie/mod/url/view.php?id=448779`.

Signed: _____   Date: 15/04/2020

Michael Lenehan

Contents

1 Introduction

Using Keras, two separate image classification problems will be solved. The first of these problems involves developing a classifier for a subset of the ImageNet dataset. This classifier is based on the VGG-16D neural network.

The second problem involves performing transfer learning on the ResNet50 CNN. The lower 141 layers of the base model must not be modified, with only the upper 33 layers being modified. This is done using the Tensorflow Keras ResNet50 model, and the freeze functionality which is built in.

Google Colab is used as the development environment for this assignment. With access to a GPU, training can be accelerated. Colab allows for the importing of data from a mounted Google Drive volume, the importing of Python libraries, and the saving and exporting of the trained models.

A Google Drive Folder containing the code used, and the saved models can be found at the following URL:

`https://drive.google.com/open?id=1XRTe3RMW5K9cYNi3EcyVj6cwRYOXZX8n.`

2 Question 1

The aim of section 1 of this assignment is to create an "ImageNet" style classifier. The provided dataset contains pictures divided into 6 classes, which are "Church", "English Springer", "Garbage Truck", "Gas Pump", "Parachute", and "Tench". The data has been provided in three subsets, training, validation, and test subsets. This removes the need to do test-train splitting. The dataset is relatively small, with a size of 1350 images per class, totaling 8100 images, compared to the 14 million images in the original "ImageNet" dataset.

The provided CNN structure is based on the "VGG" neural network. The base network has a total of 9 layers. The input images are of size 224x224, and the CNN must output a classification decision based on the image classes. This section of the assignment is divided into five sections. The first involves building the base CNN. Subsequent sections build on top of this, adding dropout, data augmentation, and batch normalization to the neural network in order to assess the performance impact of these features. The final section involves creating a final version of the neural network, which has the highest achievable performance.

2.1 Part a

2.1.1 Introduction

For Part a, the given "VGG-lite" CNN structure (Figure 1) must be implemented. This model will act as a base for each of the subsequent parts in section 1. The given structure has 9 layers, takes the input images, which are of size 224x224, and outputs a classification based on one of the six classes.

Layer	Number of output filters
2D (3x3) convolution layer	32
2D (3x3) convolution layer	32
2D (2x2) Pooling	
2D (3x3) convolution layer	64
2D (3x3) convolution layer	64
2D (2x2) Pooling	
Flatten	
Fully-connected NN layer	512
Fully-connected NN layer (Prediction)	Number of classes

Convolution layers should use Xavier (glorot) uniform kernel initialization, a dilation rate of 1, 'same' padding, strides of (height,width)= (1,1) and 'relu' activation. Note the first layer will need to account for the input shape of the data been examined.

Pooling layers will use (2x2) max pooling with (ie a pool size of (vertical, horizontal) = (2,2)).

Fully-connected layers will also use Xavier (glorot) uniform kernel initialization and 'relu' activations.

Figure 1: Given VGG-lite Structure

2.1.2 Design

The first steps in implementing this system is loading the data into Google Colab. This functionality is available through the "google.colab" "drive" library. The required directory must be mounted to the session, and the zipped data must be extracted. The code implementation in Figure 2 shows this functionality.

Figure 2: Extract Data from ZIP File

To load data from a directory, Tensorflow Keras provides the "flow_from_directory" method. This method takes a directory, image size, and class mode. For the given dataset, the directories are "train", "validation", and "test". The target size for each of the images is 224x224, and the class mode is "categorical", as there are multiple classes of output. For the test data, the class mode must be set to "None", and the data must not be shuffled, as specified by the argument "shuffle=False".

Figure 3: Load in Data from Directory

To construct the neural network, the "model.add" method is called to add layers. Layers are added as previously specified. The 2D convolutional layers have 32 output filters, a 3x3 kernel, a stride of 1, padding "same", "relu" activation, and "glorot_uniform" kernel initialization. There is a difference with the input layer, which must specify the input shape in the form (rows, columns, channels). In this case, the input shape is 244x244x3. These layers are added as Keras "Conv2D" layers.

Following the 2D convolutional layer is a 2D pooling layer. This is implemented with the "MaxPooling2D" function, which takes a pool size as an input. In this case, the specified pool size is 2x2. There are two additional 2D convolutional layers, with 64 output filters, followed by another 2D pooling layer.

A Flatten layer is added before the fully connected layers of the model. The Keras "Flatten" function takes no input arguments.

Each of the final Dense layers use the "glorot_uniform" kernel initializer. The first Dense layer has 512 output units and "relu" activation. The final layer must have 6 output units, as there are 6 total classification classes. It uses "softmax" activation. This must also be used, as ReLU has very poor performance as an output layer.

The finalized neural network is shown in Figure 4.

```
model = tf.keras.Sequential()
model.add(layers.Conv2D(
    filters=32, kernel_size=(3,3), input_shape=(224,224,3),
    strides=(1,1), padding='same',
    dilation_rate=(1,1), activation='relu',
    kernel_initializer='glorot_uniform',
    kernel_regularizer=None
))
model.add(layers.Conv2D(
    filters=32, kernel_size=(3,3), strides=(1,1), padding='same',
    dilation_rate=(1,1), activation='relu',
    kernel_initializer='glorot_uniform',
    kernel_regularizer=None
))
model.add(layers.MaxPooling2D(
    pool_size=(2,2)
))
model.add(layers.Conv2D(
    filters=64, kernel_size=(3,3), strides=(1,1), padding='same',
    dilation_rate=(1,1), activation='relu',
    kernel_initializer='glorot_uniform',
    kernel_regularizer=None
))
model.add(layers.Conv2D(
    filters=64, kernel_size=(3,3), strides=(1,1), padding='same',
    dilation_rate=(1,1), activation='relu',
    kernel_initializer='glorot_uniform',
    kernel_regularizer=None
))
model.add(layers.MaxPooling2D(
    pool_size=(2,2), strides=None
))
model.add(layers.Flatten())
model.add(layers.Dense(
    units=512,
    activation='relu',
    kernel_initializer='glorot_uniform',
    kernel_regularizer=regularizers.l2(0.00025)
))
model.add(layers.Dense(
    units=6,
    activation='softmax',
    kernel_initializer='glorot_uniform'
))
```

Figure 4: Question 1 Base Model

The model is compiled using the "Adam" optimizer, "categorical_crossentropy" loss model, and has a metrics argument which defines the metric on which the model is evaluated on as the "accuracy".

```
model.compile(
    optimizer=optimizers.Adam(learning_rate=0.0001),
    loss = 'categorical_crossentropy',
    metrics=['accuracy']
)
```

Figure 5: Compile Function

Following the compile is the code required to train and evaluate the model. This code can be found in the listings within the appendices.

2.1.3 Testing

There are a number of hyperparameters which may be tuned for this model. Each "Conv2D" and "Dense" layer have activity regularizers, kernel regularizers, and bias regularizers which can be modified in order to tune the performance of the neural network.

Manual tuning was performed for these hyperparameters for various input values. This was done due to a lack of understanding of the "kerastuner" tool, which allows for automated hyperparameter selection.In order to shorten the amount of time taken for testing, early stopping was implemented, which stops training when there has been no change in the validation loss in a set number of epochs. 5 epochs was chosen as the limit for this.

Kernel regularizers were the hyperparameters which were utilised for the purpose of tuning. A number of values for the hyperparameters were tested, with the validation accuracy used as the metric by which they were compared.

First, the kernel regularizer for the final "relu" activated Dense layer was tested, this is labelled as "Regularizer Value" within the following table. The regularizer chosen

for the purpose of this section was the "l2" "weight decay" regularizer. The kernel regularizer for each other layers was set to a value of 0.01, while the Dense layer regularizer was modified in order to obtain the highest possible accuracy.

Table 1: Kernel Regularizer Hyperparameter Tuning (Final "ReLU" Dense Layer)

Regularizer Value	Validation Accuracy
0.00005	69.17
0.00001	71.83
0.0005	76
0.00025	77
0.0001	70

As given by the output listed above, the regularizer value of 0.00025 gave the highest validation accuracy, and as such was used moving forward. The kernel regularizer value of the other layers were then tested, as shown in the table below.

Table 2: Kernel Regularizer Hyperparameter Tuning (Other Layers)

Regularizer Value	Validation Accuracy
0.001	74
0.005	75
0.0005	73
"None"	77

As the optimal value is shown to be "None", i.e. no regularizer is used, this is the option used for the model. While these options are the optimum arrived at using manual hyperparameter tuning, there may be more optimal choices.

Another hyperparameter which must be chosen is the learning rate of the chosen optimizer. For this section, the "Adam" optimizer was chosen, as the hyperparameter tuning is relatively easy, especially when being done manually.

Table 3: Learning Rate Hyperparameter Tuning

Learning Rate	Validation Accuracy
0.01	16.67
0.001	71.33
0.0001	74.83
0.00001	74.33

The learning rate value of 0.0001 was selected, as it gave the highest value for validation accuracy. With this value, the final model could be tested.

For the purposes of training, the model was set to train for 30 epochs, however, the value for early stopping patience was increased from 5 to 7, in order to prevent overfitting without stopping too early to train correctly.

2.1.4 Results

The correct output for the neural network structure was printed using the "model.summary" function in keras, as seen in Figure 6. This shows the output shape of the final layer as 6, which is the number of possible classifications, along with all of the specified outputs in all other layers.

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_8 (Conv2D)            (None, 224, 224, 32)      896
_____
conv2d_9 (Conv2D)            (None, 224, 224, 32)      9248
_____
max_pooling2d_4 (MaxPooling2 (None, 112, 112, 32)      0
_____
conv2d_10 (Conv2D)           (None, 112, 112, 64)      18496
_____
conv2d_11 (Conv2D)           (None, 112, 112, 64)      36928
_____
max_pooling2d_5 (MaxPooling2 (None, 56, 56, 64)        0
_____
flatten_2 (Flatten)          (None, 200704)            0
_____
dense_4 (Dense)              (None, 512)               102760960
_____
dense_5 (Dense)              (None, 6)                 3078
=================================================================
Total params: 102,829,606
Trainable params: 102,829,606
Non-trainable params: 0
```

Figure 6: Model Summary

It is clear from both Figures 7 and 8 that overfitting does not occur within this model. The validation accuracy of the model reaches approximately 80%, however, the loss value reaches approximately 2.00. The final output, as shown in Figure 9 shows a final validation accuracy of 79.50%.
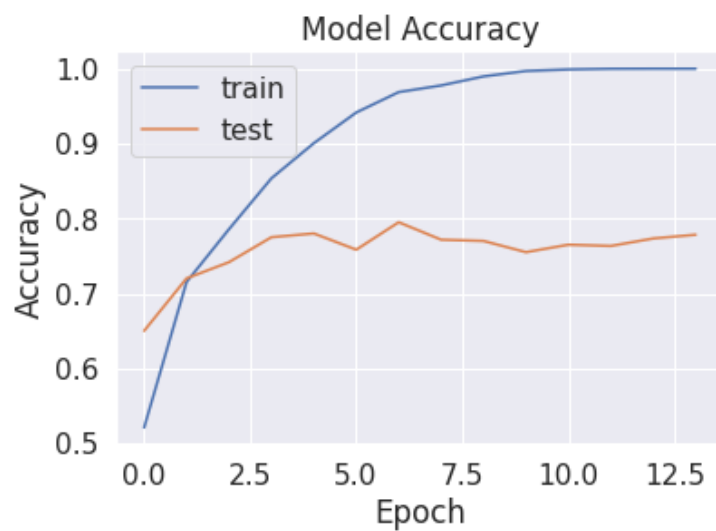
Figure 7: Validation and Training Accuracy

While the validation accuracy is trending upwards, it is possible that better tuning of the hyperparameters would give a higher validation accuracy.
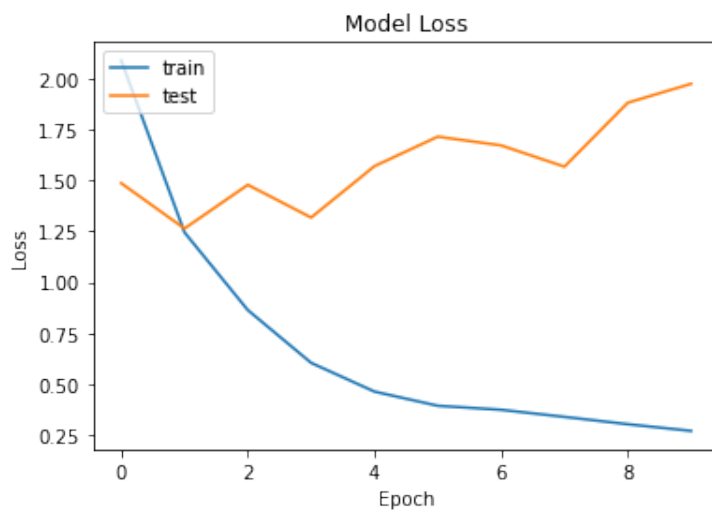
Figure 8: Validation and Training Loss

The validation loss continues to rise. It is again possible that this could be attributed to incorrect hyperparameter tuning.

The final output shows an average precision and recall of 81%, with the highest values attributed to the "Parachute" class, and the lowest values attributed to the "English Springer" class.

Figure 9: Model Testing Results

The final confusion matrix, as shown in Figure 10, shows the number of correctly identified samples in testing, and as previously noted, the class with the highest precision and recall is "Parachute" labelled as "4" within the matrix.
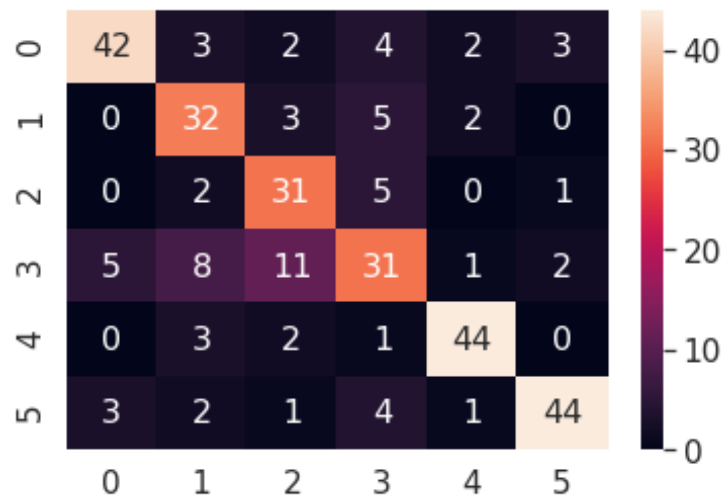
Figure 10: Confusion Matrix

15

Using early stopping, the model ran for a total of 14 epochs. The training time for the model was 436.109 seconds. Given that 1 hour of training on an Nvidia GPU produces approximately 0.25lbs of CO_2 equivalent emissions, this training has an emission output of 1.817lbs CO_2 equivalent emissions.

2.2 Part b

2.2.1 Introduction

The aim of Part b is to apply "Dropout" to the neural network designed in Part A. The hyperperameters, i.e. the dropout rate, needs to be tuned in order to obtain an optimal performance from the network.

2.2.2 Rational

Dropout is a technique used in Neural Networks in order to prevent over-fitting. By dropping certain neurons at random during testing, the network must learn features which are considered "robust". This will lead to a greater test accuracy.

2.2.3 Design

Dropout layers are typically added to the end of a block and before the output classifier. There are a total of three dropout layers added to the model, as shown in Figure 11. No other changes were made to the code, and a full listing of this code can be found in the appendices.

```python
model = tf.keras.Sequential()
model.add(layers.Conv2D(
    filters=32, kernel_size=(3,3), input_shape=(224,224,3),
    strides=(1,1), padding='same',
    dilation_rate=(1,1), activation='relu',
    kernel_initializer='glorot_uniform',
    kernel_regularizer=None
))
model.add(layers.Conv2D(
    filters=32, kernel_size=(3,3), strides=(1,1), padding='same',
    dilation_rate=(1,1), activation='relu',
    kernel_initializer='glorot_uniform',
    kernel_regularizer=None
))
model.add(layers.MaxPooling2D(
    pool_size=(2,2)
))
model.add(layers.Dropout(0.1))
model.add(layers.Conv2D(
    filters=64, kernel_size=(3,3), strides=(1,1), padding='same',
    dilation_rate=(1,1), activation='relu',
    kernel_initializer='glorot_uniform',
    kernel_regularizer=None
))
model.add(layers.Conv2D(
    filters=64, kernel_size=(3,3), strides=(1,1), padding='same',
    dilation_rate=(1,1), activation='relu',
    kernel_initializer='glorot_uniform',
    kernel_regularizer=None
))
model.add(layers.MaxPooling2D(
    pool_size=(2,2), strides=None
))
model.add(layers.Dropout(0.1))
model.add(layers.Flatten())
model.add(layers.Dense(
    units=512,
    activation='relu',
    kernel_initializer='glorot_uniform',
    kernel_regularizer=regularizers.l2(0.00025)
))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(
    units=6,
    activation='softmax',
    kernel_initializer='glorot_uniform'
))
```

Figure 11: Model including Dropout

2.2.4 Testing

For the purposes of testing, the dropout rate was the hyperparameter being tuned. The dropout rate of the first two dropout layers was set to 0.1 as a baseline measurement, while the rate of the dropout rate before the final "Dense" layer was modified. This

resulted in an optimum value of 0.5 given for the final dropout layer, as shown in the table below.

Table 4: Dropout Hyperparameter Tuning

Dropout Rate	Validation Accuracy
0.55	77.83
0.5	78.5
0.25	74.7
0.1	76.17

Setting this dropout rate, the dropout rates of the other two dropout layers are changed simultaneously. The optimal value given by the testing was 0.1. This value is not shown in the table below, as it had previously been tested in the table above.

Table 5: Dropout Hyperparameter Tuning

Dropout Rate	Validation Accuracy
0.55	73.83
0.5	77.83
0.25	77

Using this value for the dropout rate, the early stopping patience value was increased to 7, and the training was set to run for 30 epochs.

2.2.5 Results

The output for the neural network structure was again printed using the "model.summary" function in keras, as seen in Figure 12. This shows the placement of the Dropout layers, as previously discussed.

```
Layer (type)                    Output Shape            Param #
=================================================================
conv2d_32 (Conv2D)              (None, 224, 224, 32)    896
_____
conv2d_33 (Conv2D)              (None, 224, 224, 32)    9248
_____
max_pooling2d_16 (MaxPooling (None, 112, 112, 32)       0
_____
dropout_24 (Dropout)            (None, 112, 112, 32)    0
_____
conv2d_34 (Conv2D)              (None, 112, 112, 64)    18496
_____
conv2d_35 (Conv2D)              (None, 112, 112, 64)    36928
_____
max_pooling2d_17 (MaxPooling (None, 56, 56, 64)         0
_____
dropout_25 (Dropout)            (None, 56, 56, 64)      0
_____
flatten_8 (Flatten)             (None, 200704)          0
_____
dense_16 (Dense)                (None, 512)             102760960
_____
dropout_26 (Dropout)            (None, 512)             0
_____
dense_17 (Dense)                (None, 6)               3078
=================================================================
Total params: 102,829,606
Trainable params: 102,829,606
Non-trainable params: 0
```

Figure 12: Model Summary

Figures 13 and 14 again show that overfitting is not occurring. While the accuracy is on an upward trend, it does begin to trail downwards towards the final epochs.
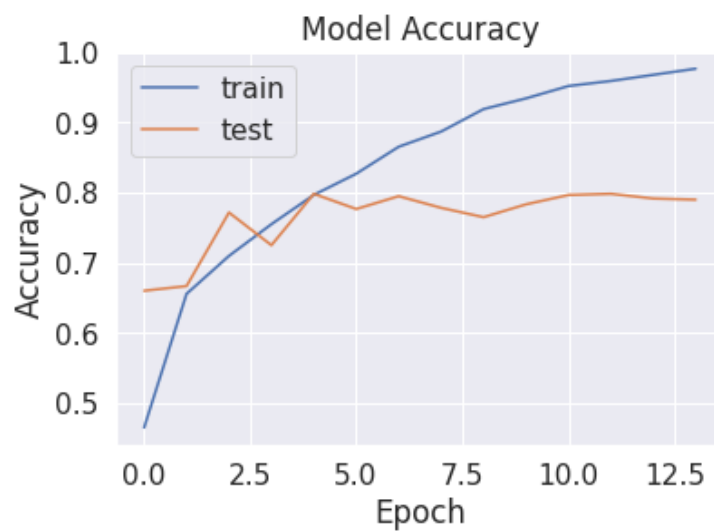
Figure 13: Validation and Training Accuracy

Again, the loss value is much higher than that of the training, however it is lower than in Part a, with a value of approximately 1/2 that of Part a.
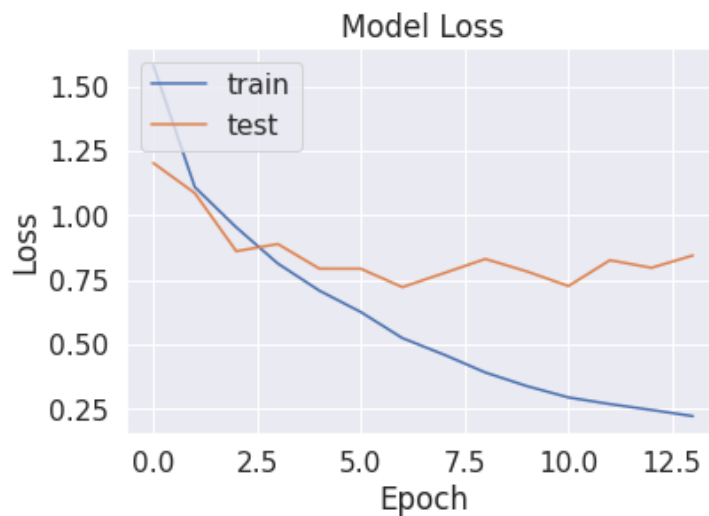

Figure 14: Validation and Training Loss

The final output shows an average precision and recall of 83% and 84% respectively, with the highest values attributed to the "Parachute" class, and the lowest values attributed to the "English Springer" class.

```
12/12 [==============================] - 2s 158ms/step - loss: 0.9417 - accuracy: 0.7600
CNN Accuracy: 76.00%
CNN Error: 24.00%
300/300 [==============================] - 1s 5ms/step
                 precision    recall  f1-score   support

         Church       0.92      0.84      0.88        55
English Springer       0.80      0.65      0.71        62
  Garbage Truck       0.80      0.82      0.81        49
       Gas Pump       0.64      0.91      0.75        35
      Parachute       0.94      0.96      0.95        49
          Tench       0.86      0.86      0.86        50

       accuracy                           0.83       300
      macro avg       0.83      0.84      0.83       300
   weighted avg       0.84      0.83      0.83       300
```

Figure 15: Model Testing Results

The confusion matrix shows the number of correctly identified samples during testing. It is clear that the class with the highest precision and recall is "Parachute", labelled as "4" within the matrix.
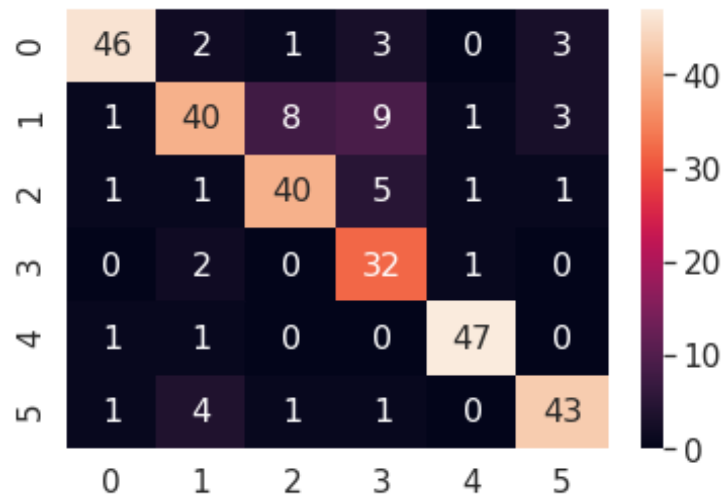

Figure 16: Confusion Matrix

Using early stopping, the model ran for a total of 14 epochs. The training time for

the model was 493.8769 seconds, which gives an emission output of 2.058lbs CO_2 equivalent emissions. This value is higher than the value in Part a.

2.3 Part c

2.3.1 Introduction

The aim of this section is to demonstrate the effects of data augmentation on the performance of the neural network.

2.3.2 Rational

Data augmentation is a technique typically used in order to increase the size of small datasets. Having more images, with unique features allows for the training of a more robust neural network. By rotating, mirroring, and scaling images the neural network can become more robust in real use cases to these variances in input image.

2.3.3 Design

For this section, the values in the "testDataGenerator" were modified (Figure 17. The width shift range and height shift range were given values of 0.1, while the horizontal and vertical flip arguments were given "True" values. The width shift range and height shift range take a float value and shift the image the input fraction of the width/height, in this case 1/10th. The horizontal and vertical flip randomly mirror the image along the horizontal or vertical planes.

No other changes were made to the code from Part a, and a full listing of the code can be seen in the appendices.

```
trainDataGenerator = tf.keras.preprocessing.image.ImageDataGenerator(
    rotation_range=0.,
    width_shift_range=0.1,
    height_shift_range=0.1,
    zoom_range=0.,
    horizontal_flip=True,
    vertical_flip=True,
    rescale=1./255
)
```

Figure 17: Data Augmentation Modifications

2.3.4 Testing

As there were no hyperparameters being tested, there was no testing which took place. The code was executed to extract final data, discussed in the Results section.

2.3.5 Results

The output for the neural network structure is the same as that of Part a, as no change is being made to the CNN structure, only to the input test data.

```
Layer (type)                   Output Shape            Param #
=================================================================
conv2d_4 (Conv2D)              (None, 224, 224, 32)    896
_____
conv2d_5 (Conv2D)              (None, 224, 224, 32)    9248
_____
max_pooling2d_2 (MaxPooling2   (None, 112, 112, 32)    0
_____
conv2d_6 (Conv2D)              (None, 112, 112, 64)    18496
_____
conv2d_7 (Conv2D)              (None, 112, 112, 64)    36928
_____
max_pooling2d_3 (MaxPooling2   (None, 56, 56, 64)      0
_____
flatten_1 (Flatten)            (None, 200704)          0
_____
dense_2 (Dense)                (None, 512)             102760960
_____
dense_3 (Dense)                (None, 6)               3078
=================================================================
```

Figure 18: Model Summary

Figures 19 and 20 show that overfitting is not occurring. While the validation accuracy
is quite close to the testing accuracy, the best fit line through the points on the graph
would be below that of the training accuracy.
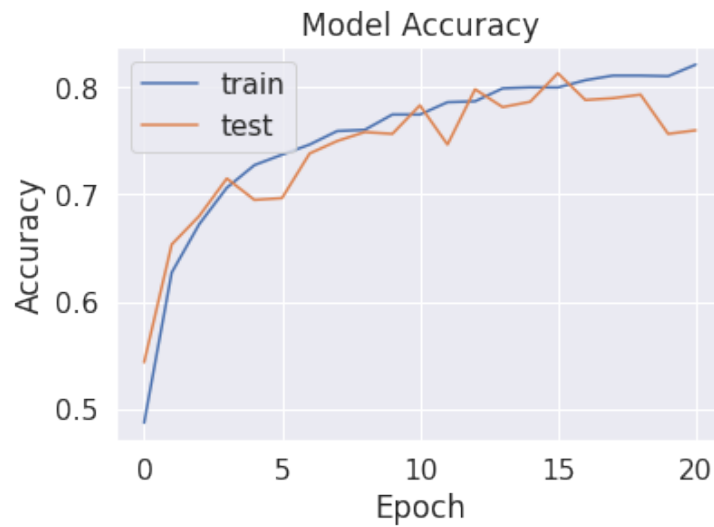
Figure 19: Validation and Training Accuracy

The loss value improves yet again, with a value of approximately 0.8 in the validation set.
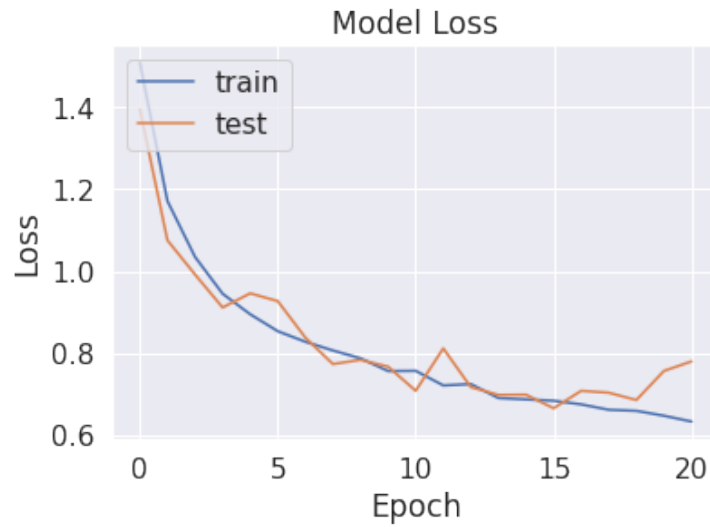
Figure 20: Validation and Training Loss

The final output shows an average precision and recall of 85% and 86% respectively , with the highest values attributed to the "Parachute" class, and the lowest values associated with the "English Springer" class.

Figure 21: Model Testing Results

The confusion matrix again demonstrates the number of correctly identified samples,

with the highest precision class being "Parachute", labelled as "4" within the matrix.

Figure 22: Confusion Matrix

Using early stopping, the model ran for a total of 21 epochs. The training time for this model was 2185.3824 seconds, giving an emission output of 9.106lbs CO_2 equivalent emissions. This is, yet again, much higher than Part a, and is also higher than Part b.

2.4 Part d

2.4.1 Introduction

This section aims to add batch normalization to the neural network designed in Part A. The effect on the neural network should be investigated in order to determine if it increases the performance of the network.

2.4.2 Rational

Batch normalization works to alleviate the effects of covariate shift within a CNN. Batch normalization allows for the use of higher training rates, and also the use of less dropout within the network, due to the regularization effects it has.

2.4.3 Design

Batch normalization was originally proposed as a layer to be placed before the activation function, however, it has been noted that performance increases can be found by placing the layer after the activation function. As the base model is defined with the activation functions within the layer definitions, the normalization layers will be placed after the activation functions.

```python
model = tf.keras.Sequential()
model.add(layers.Conv2D(
    filters=32, kernel_size=(3,3), input_shape=(224,224,3),
    strides=(1,1), padding='same',
    dilation_rate=(1,1), activation='relu',
    kernel_initializer='glorot_uniform',
    kernel_regularizer=None
))
model.add(layers.Conv2D(
    filters=32, kernel_size=(3,3), strides=(1,1), padding='same',
    dilation_rate=(1,1), activation='relu',
    kernel_initializer='glorot_uniform',
    kernel_regularizer=None
))
model.add(layers.BatchNormalization(axis=1))
model.add(layers.MaxPooling2D(
    pool_size=(2,2)
))
model.add(layers.Conv2D(
    filters=64, kernel_size=(3,3), strides=(1,1), padding='same',
    dilation_rate=(1,1), activation='relu',
    kernel_initializer='glorot_uniform',
    kernel_regularizer=None
))
model.add(layers.Conv2D(
    filters=64, kernel_size=(3,3), strides=(1,1), padding='same',
    dilation_rate=(1,1), activation='relu',
    kernel_initializer='glorot_uniform',
    kernel_regularizer=None
))
model.add(layers.BatchNormalization(axis=1))
model.add(layers.MaxPooling2D(
    pool_size=(2,2), strides=None
))
model.add(layers.Flatten())
model.add(layers.Dense(
    units=512,
    activation='relu',
    kernel_initializer='glorot_uniform',
    kernel_regularizer=regularizers.l2(0.00025)
))
model.add(layers.BatchNormalization(axis=1))
model.add(layers.Dense(
    units=6,
    activation='softmax',
    kernel_initializer='glorot_uniform'
))
```

Figure 23: Batch Normalization Model

2.4.4 Testing

As previously mentioned, three batch normalization layers were added to the model. The hyperparameter being tested is the momentum, which, by default is set to 0.99. The value of 0.5 and 0.0 were also tested, as outlined below.

Table 6: Batch Normalisation Hyperparameter Tuning

Momentum	Validation Accuracy
0.99	78.83
0.5	78.50
0.0	74.33

The value of 0.0 for momentum was proposed in the original paper in which batch normalisation was introduced, however, the default value of 0.99 is shown to have the best accuracy. As such this is the value which will be tested.

Again, the number of epochs was increased to 30, however, the early stopping patience value left at 5, as any higher and the model would train for all 30 epochs, and give low accuracy.

2.4.5 Results

The model structure, as output via the "model.summary" function in keras, can be seen below. The three batch normalisation layers can clearly be seen.

```
Layer (type)                    Output Shape              Param #
=================================================================
conv2d_32 (Conv2D)              (None, 224, 224, 32)      896
_____
conv2d_33 (Conv2D)              (None, 224, 224, 32)      9248
_____
batch_normalization_23 (Batc    (None, 224, 224, 32)      896
_____
max_pooling2d_16 (MaxPooling    (None, 112, 112, 32)      0
_____
conv2d_34 (Conv2D)              (None, 112, 112, 64)      18496
_____
conv2d_35 (Conv2D)              (None, 112, 112, 64)      36928
_____
batch_normalization_24 (Batc    (None, 112, 112, 64)      448
_____
max_pooling2d_17 (MaxPooling    (None, 56, 56, 64)        0
_____
flatten_8 (Flatten)             (None, 200704)            0
_____
dense_16 (Dense)                (None, 512)               102760960
_____
batch_normalization_25 (Batc    (None, 512)               2048
_____
dense_17 (Dense)                (None, 6)                 3078
=================================================================
Total params: 102,832,998
Trainable params: 102,831,302
Non-trainable params: 1,696
_____
None
```

Figure 24: Model Summary

The plots in Figures 25 and 26 show the validation and training accuracies and losses respectively. The validation accuracy shows that the model is not overfitting, and the loss is relatively low.
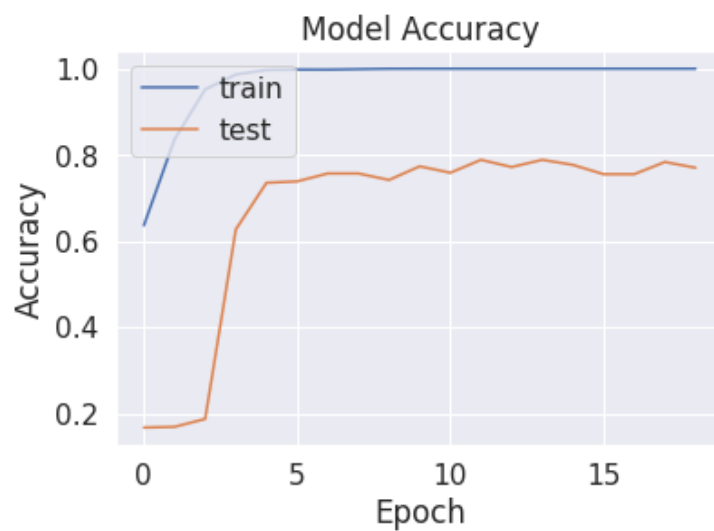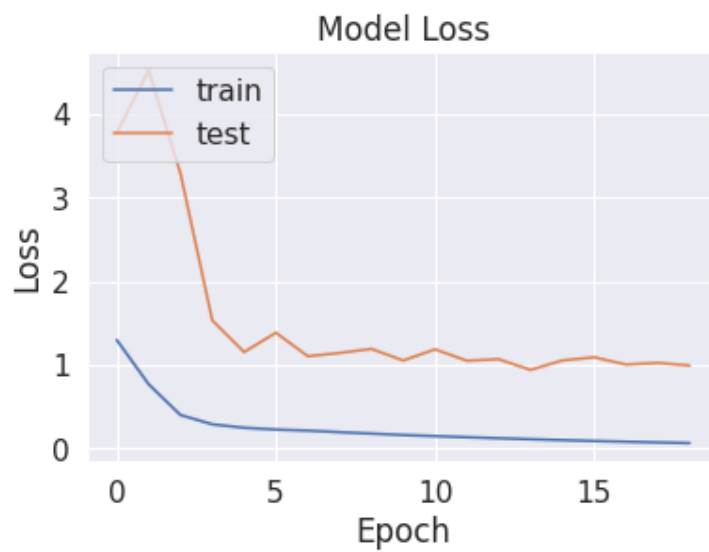
Figure 25: Validation and Training Accuracy

Figure 26: Validation and Training Loss

The test output gives an average precision and recall of 80%, with the highest values

attributed to the "Parachute" class, and the lowest values attributed to the "English Springer" class.

```
12/12 [==============================] - 2s 137ms/step - loss: 1.0121 - accuracy: 0.7850
CNN Accuracy: 78.50%
CNN Error: 21.50%
300/300 [==============================] - 1s 4ms/step
                 precision    recall  f1-score   support

         Church       0.82      0.80      0.81        51
English Springer       0.72      0.72      0.72        50
  Garbage Truck       0.80      0.80      0.80        50
       Gas Pump       0.60      0.81      0.69        37
       Parachute       0.96      0.83      0.89        58
          Tench       0.88      0.81      0.85        54

       accuracy                           0.80       300
      macro avg       0.80      0.80      0.79       300
   weighted avg       0.81      0.80      0.80       300
```

Figure 27: Model Testing Results

The confusion matrix shows the number of correctly identified test samples, with the highest precision class being "Parachute", labelled as "4" within the matrix.

Figure 28: Confusion Matrix

Using early stopping, the model ran for a total of 19 epochs, The training time for this model was 536.67 seconds, giving an emission output of 2.236lbs CO2 equivalent emissions. While this value is lower than Part c, it is higher than that of Part a and Part

32

b, showing the overhead involved with batch normalisation.

2.5   Part e

2.5.1   Introduction

Part e aims to combine the previous sections, applying a combination of dropout, data augmentation and batch normalization to the baseline model from Part a. The aim is to develop a CNN with the optimal accuracy.

2.5.2   Rational

By combining the advantages of the techniques used in the previous sections, a CNN can be developed which outperforms the baseline model developed in Part a. While it is not necessary to utilise all of the techniques, combinations of data augmentation, dropout, and batch normalisation can lead to improved accuracy for the neural network.

2.5.3   Design

The design of this section came about from the combination of the techniques in each of the previous sections. As discussed in the Testing section below, different combinations of the techniques were tested in order to obtain the best possible accuracy from the model. The final code for this section can be seen in the appendices.

```python
model = tf.keras.Sequential()
model.add(layers.Conv2D(
    filters=32, kernel_size=(3,3), input_shape=(224,224,3),
    strides=(1,1), padding='same',
    dilation_rate=(1,1), activation='relu',
    kernel_initializer='glorot_uniform',
    kernel_regularizer=None
))
model.add(layers.Conv2D(
    filters=32, kernel_size=(3,3), strides=(1,1), padding='same',
    dilation_rate=(1,1), activation='relu',
    kernel_initializer='glorot_uniform',
    kernel_regularizer=None
))
# model.add(layers.BatchNormalization(axis=1))
model.add(layers.Dropout(0.25))
model.add(layers.MaxPooling2D(
    pool_size=(2,2)
))
model.add(layers.Conv2D(
    filters=64, kernel_size=(3,3), strides=(1,1), padding='same',
    dilation_rate=(1,1), activation='relu',
    kernel_initializer='glorot_uniform',
    kernel_regularizer=None
))
model.add(layers.Conv2D(
    filters=64, kernel_size=(3,3), strides=(1,1), padding='same',
    dilation_rate=(1,1), activation='relu',
    kernel_initializer='glorot_uniform',
    kernel_regularizer=None
))
# model.add(layers.BatchNormalization(axis=1))
model.add(layers.Dropout(0.25))
model.add(layers.MaxPooling2D(
    pool_size=(2,2), strides=None
))
model.add(layers.Flatten())
model.add(layers.Dense(
    units=512,
    activation='relu',
    kernel_initializer='glorot_uniform',
    kernel_regularizer=regularizers.l2(0.00025)
))
# model.add(layers.BatchNormalization(axis=1))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(
    units=6,
    activation='softmax',
    kernel_initializer='glorot_uniform'
))
```

Figure 29: Combined Technique Model

2.5.4   Testing

Testing was again done manually, and as such, the obtained result is suboptimal.

Table 7: Layer and HyperParameter Testing

CNN Description	Validation Accuracy
Combined Part a-d	76.17
Batch Normalisation + Data Augmentation	76.17
Dropout + Data Augmentation	71.17
Batch Normalisation + Dropout	70.5
No Batch Normalisation in the output layers	72.5
No Dropout in the output layers	66.2

By using only Dropout and Data Augmentation , and a dropout rate of 0.25 for the non-output Dropout layers, a highest validation accuracy of 78.17% was achieved. This is far from optimal, however it is approximately on par with the results obtained from using Dropout and Data Augmentation separately. This is undoubtably due to poor hyperparameter tuning.

2.5.5 Results

The model summary shown below displays the use of dropout within the model. As mentioned, data augmentation was also applied to the input images in order to achieve the performance that is seen in the results.

```
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_28 (Conv2D)           (None, 224, 224, 32)      896
_____
conv2d_29 (Conv2D)           (None, 224, 224, 32)      9248
_____
dropout_17 (Dropout)         (None, 224, 224, 32)      0
_____
max_pooling2d_14 (MaxPooling (None, 112, 112, 32)      0
_____
conv2d_30 (Conv2D)           (None, 112, 112, 64)      18496
_____
conv2d_31 (Conv2D)           (None, 112, 112, 64)      36928
_____
dropout_18 (Dropout)         (None, 112, 112, 64)      0
_____
max_pooling2d_15 (MaxPooling (None, 56, 56, 64)        0
_____
flatten_7 (Flatten)          (None, 200704)            0
_____
dense_14 (Dense)             (None, 512)               102760960
_____
dropout_19 (Dropout)         (None, 512)               0
_____
dense_15 (Dense)             (None, 6)                 3078
=================================================================
Total params: 102,829,606
Trainable params: 102,829,606
Non-trainable params: 0
```

Figure 30: Model Summary

The model seems to be overfitting to the training set. This is an issue as the training set does not have extremely high accuracy, and this is likely causing issues with the predictions of the test set.

While the loss is relatively low, the overall accuracy does not benefit from this improvement in the system.
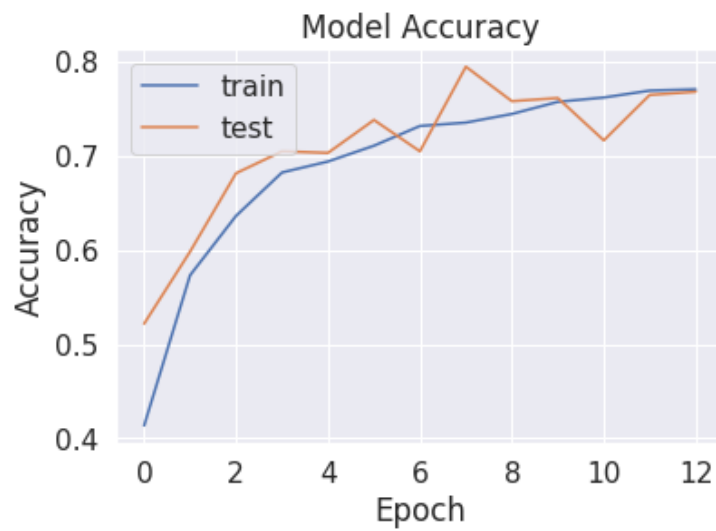
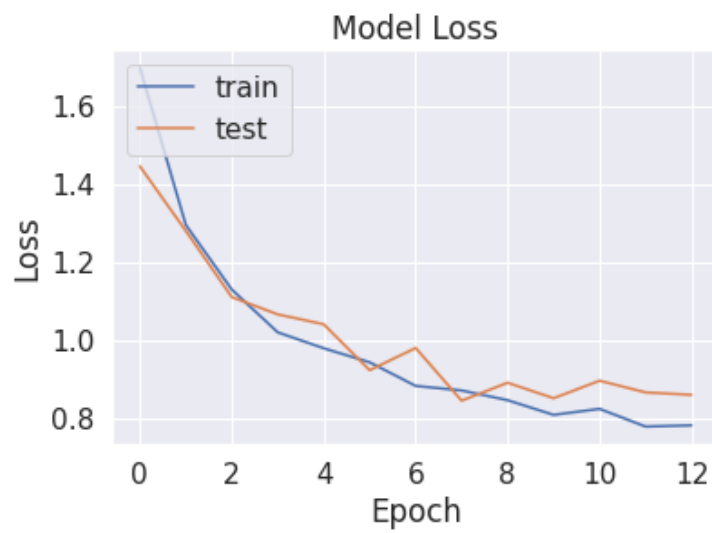Figure 31: Validation and Training Accuracy

Figure 32: Validation and Training Loss

The test results can be seen in Figure 33. The average precision is 82%, while the average recall is 82%. These values are below the values achieved using only Data

Augmentation techniques. Again, this is likely due to the poor implementation of hyperparameter tuning, i.e. trial-and-error.


```
12/12 [==============================] - 2s 138ms/step - loss: 0.8387 - accuracy: 0.7817
CNN Accuracy: 78.17%
CNN Error: 21.83%
300/300 [==============================] - 1s 4ms/step
                 precision    recall  f1-score   support

         Church       0.90      0.83      0.87        54
English Springer       0.76      0.79      0.78        48
  Garbage Truck       0.88      0.69      0.77        64
       Gas Pump       0.48      0.92      0.63        26
       Parachute       0.90      0.96      0.93        47
          Tench       0.92      0.75      0.83        61

       accuracy                           0.81       300
      macro avg       0.81      0.82      0.80       300
   weighted avg       0.84      0.81      0.81       300
```

Figure 33: Model Testing Results

The confusion matrix in Figure 34 shows that the highest number of correctly identified samples belonged to the "Church" class.
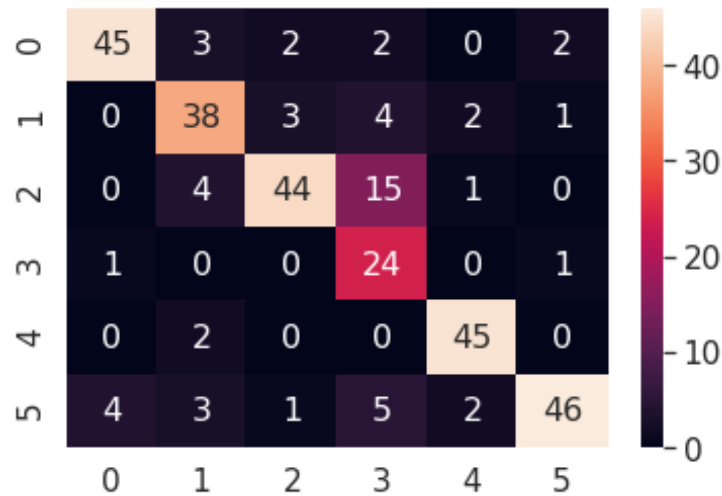

Figure 34: Confusion Matrix

The training time of this model was 1138.7972 seconds, which is 4.745lbs of CO_2 equivalent emissions.

3 Question 2

The aim of section 2 of this assignment is to implement "Transfer Learning" using a Resnet-50 model. The model must be adapted to the "Food-101" classification task, classifying images of foods which fall into the following classes; Chicken Curry, Hamburger, Omelette, and Waffles. The dataset is yet again pre-divided into training, validation, and test sets, therefore requiring no pre-processing in order to be divided.

The Keras "ResNet50" model is used for this section of the assignment. This model has 174 layers, as is specified in the assignment brief.

3.1 Part a

3.1.1 Introduction

The aim of part A is to implement the optimisation of the classification task using the pretrained ResNet50 model. Fine-tuning based transfer learning is to be utilised in order to apply the pretrained model to the required task, i.e. classifying the Food-101 dataset.

3.1.2 Rational

Utilising a pretrained model can be useful as a fast way of deploying a neural network. When using a small dataset, getting high test accuracy can be difficult to achieve. Transfer learning can be utilised in order to improve the achieved accuracy. By training only the output layers of the model on the new dataset, the model can be trained to classify the new dataset to a high degree of accuracy.

Another benefit of transfer learning is the training time. As the model is pretrained, and only certain layers are required to be retrained on the new dataset, the model can be trained in much fewer epochs, meaning that it will train much quicker than an untrained

or newly developed model.

3.1.3 Design

The Food-101 dataset is loaded from Google drive in the same way as Question 1. Once loaded, the "ImageDataGenerator" for the training data must have the "resnet50.preprocess_input" function applied. No preprocessing is to be done to the validation or the test images. The "flow_from_directory" function is again used to pull the images from google drive.

The base model for this problem is the keras "ResNet50" model. This model takes the input arguments "include_top", "weights", and "input_shape". Include top defines whether the classifier section of the model is included. As this problem calls for transfer learning to be applied, this value is set to False. The weights argument defines what data the model is trained on. Again, the problem specifies that the ResNet50 model is trained on the "imagenet" weights. Finally, the input shape is the dimensions of the input data, which in this case is 224x224x3.

```python
base_model = tf.keras.applications.ResNet50(
    include_top=False,
    weights='imagenet',
    input_shape=(224,224,3)
)

x = base_model.output
x = layers.Flatten()(x)
preds = layers.Dense(4, activation='softmax')(x)

model = tf.keras.models.Model(inputs=base_model.input, outputs=preds)
```

Figure 35: Transfer Learning Model

Freezing was used in order to ensure that only the top "res5c" block, and the added classifier block were trainable. This can be seen implemented in Figure 36.

```
for layer in model.layers[:143]:
    layer.trainable = False

for layer in model.layers[143:]:
    layer.trainable = True
```

Figure 36: Freezing the Lower ResNet Layers

3.1.4 Testing

Testing of the transfer learning portion of this assignment involved only the addition of the "Flatten" and "Dense" layers, and as such, no comparisons between hyperparameters have been made.

3.1.5 Results

The final number of layers of the model can be seen in Figure 37. The added layers, i.e. the "Flatten" and "Dense" layers are the final output layers, with the Dense layer classifying the images into one of four classes.

conv5_block3_1_bn (BatchNormali	(None, 7, 7, 512)	2048	conv5_block3_1_conv[0][0]
conv5_block3_1_relu (Activation	(None, 7, 7, 512)	0	conv5_block3_1_bn[0][0]
conv5_block3_2_conv (Conv2D)	(None, 7, 7, 512)	2359808	conv5_block3_1_relu[0][0]
conv5_block3_2_bn (BatchNormali	(None, 7, 7, 512)	2048	conv5_block3_2_conv[0][0]
conv5_block3_2_relu (Activation	(None, 7, 7, 512)	0	conv5_block3_2_bn[0][0]
conv5_block3_3_conv (Conv2D)	(None, 7, 7, 2048)	1050624	conv5_block3_2_relu[0][0]
conv5_block3_3_bn (BatchNormali	(None, 7, 7, 2048)	8192	conv5_block3_3_conv[0][0]
conv5_block3_add (Add)	(None, 7, 7, 2048)	0	conv5_block2_out[0][0] conv5_block3_3_bn[0][0]
conv5_block3_out (Activation)	(None, 7, 7, 2048)	0	conv5_block3_add[0][0]
flatten_2 (Flatten)	(None, 100352)	0	conv5_block3_out[0][0]
dense_2 (Dense)	(None, 4)	401412	flatten_2[0][0]

```
Total params: 23,989,124
Trainable params: 15,377,412
Non-trainable params: 8,611,712
```

Figure 37: Model Summary (Final Layers)

There is an obvious difference between the training accuracy and the validation accuracy, as shown in Figure 38, and it is clear from this graph and from the final test results that the model is not overfitting.
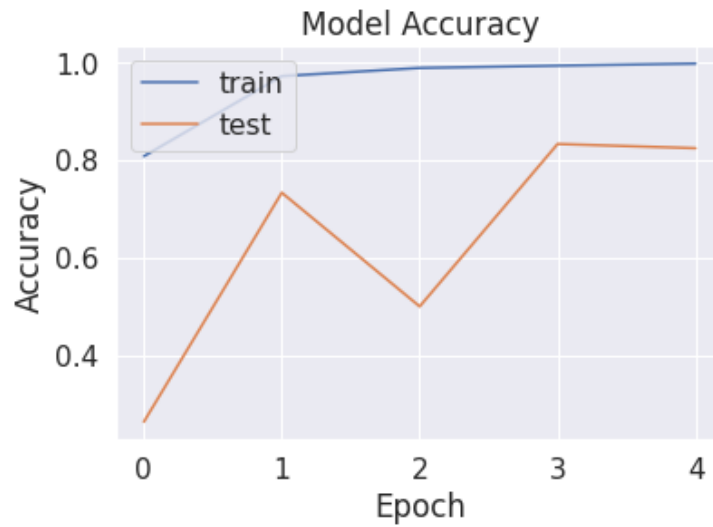

Figure 38: Validation and Training Accuracy

42

The loss for this model, as shown for both the training and validation sets in Figure 39, is very low. Over 5 epochs it can be seen how quickly the value decreases to almost zero.
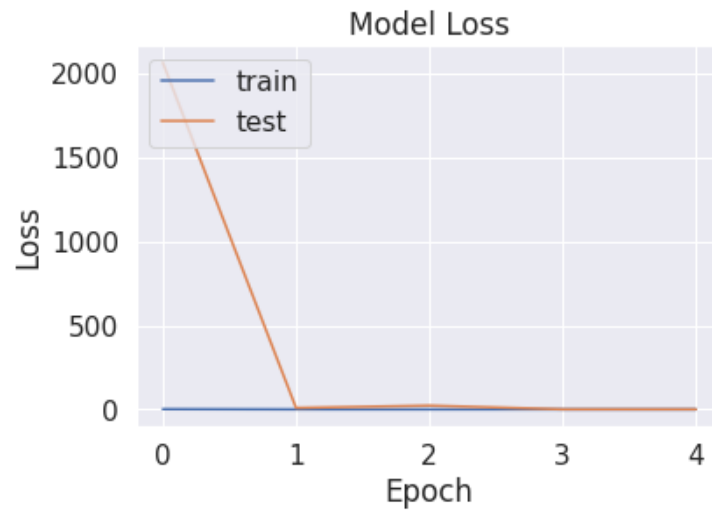

Figure 39: Validation and Training Loss

The final test results from the model can be seen in Figure 40. The precision has an average value of 83%, while the recall has an average value of 86%. For such a short training time, this value is much higher than that of Question 1. The highest precision value is for the "Waffles" class, however, the recall for this class is the lowest. The highest recall value is for the "Omlette" class, but this has the lowest precision value. The highest overall value by f1-score is for the "Hamburger" class, with a precision value of 91%, and a recall value of 89%.

```
44/44 [==============================] - 9s 212ms/step - loss: 0.4778 - accuracy: 0.8232
CNN Accuracy: 82.32%
CNN Error: 17.68%
2128/2128 [==============================] - 21s 10ms/step
                precision    recall  f1-score   support

Chicken Curry       0.88      0.84      0.86       579
    Hamburger       0.91      0.89      0.90       545
     Omelette       0.52      0.97      0.68       276
      Waffles       0.99      0.71      0.83       728

     accuracy                           0.83      2128
    macro avg       0.83      0.86      0.82      2128
 weighted avg       0.88      0.83      0.84      2128
```

Figure 40: Model Testing Results

The confusion matrix shows the "Waffles" class, labeled as "3", as having the greatest number of correctly identified samples, while the "Hamburger" and "Chicken Curry" classes, labelled "1" and "0" respectively, as having the least number of incorrectly identified samples.
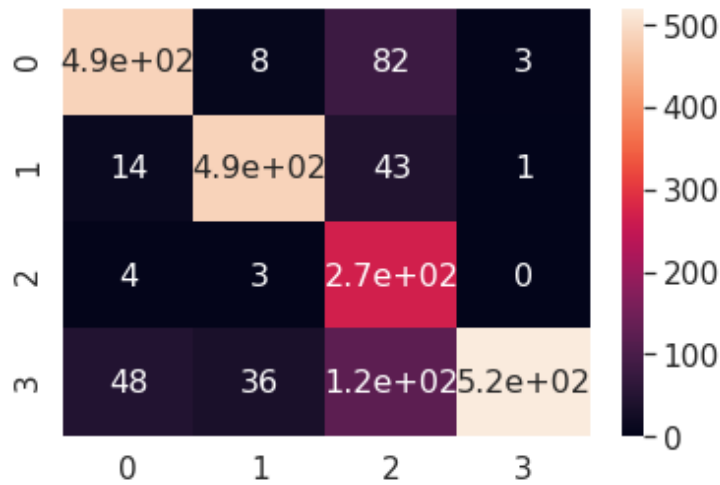

Figure 41: Confusion Matrix

The overall training time of this model was 154.2878 seconds. This would give a value of 0.643lbs of CO_2 equivalent emissions. This is much lower than the developed baseline model in Question 1, and with a much higher output accuracy.

3.2 Part b

Part b was left incomplete due to time constraints and a lack of previously unseen images.

4 Conclusion

From the completion of this assignment, it is clear that the task of training a neural network on a small database is challenging. Attaining high accuracy can be very difficult. It is also challenging to achieve high accuracy when manually tuning hyperparameters. A better understanding of automated tools for hyperperameter tuning, or a better understanding of hyperparameter tuning in general would undoubtedly have yielded better performance from the model.

Transfer learning proves extremely useful in training a model when given a small dataset. The ability to take a pretrained model and train a different final classifier is an excellent way of achieving high accuracy with predictions, as shown in Question 2 Part a.

The differences in CO_2 emissions from training a model with no dropout, data augmentation, and batch normalisation, to training a model with these techniques implemented is quite large. However, there is an even more extreme difference to be seen in using fine-tuning based transfer learning, which had the lowest emission equivalent value by far. This does not, however, take into account the training of the model used, but the ability to retrain for different datasets will cut down on emissions which would otherwise be caused by running hardware to train newly developed models for every classification task.

5 Appendix

5.1 Question 1

5.1.1 Part a

```
%tensorflow_version 2.x

from __future__ import absolute_import, division,
    print_function, unicode_literals
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import optimizers
from tensorflow.keras import losses
from tensorflow.keras import constraints
from tensorflow.keras import regularizers
from tensorflow.keras import callbacks
from google.colab import drive
from sklearn.metrics import classification_report,
    confusion_matrix
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn
import time

print(tf.__version__)
print(tf.test.gpu_device_name())

drive.mount('/content/drive')
```

```python
root_dir = "/content/drive/My\ Drive/"
base_dir = root_dir + 'Colab\ Notebooks/EE544/Assignment/
    '
print('Extracting Data')
!unzip -qo {base_dir + 'imagenette_6class.zip'}
print("Extraction Done!")

trainDataGenerator = tf.keras.preprocessing.image.
    ImageDataGenerator(
    rotation_range=0.,
    width_shift_range=0.,
    height_shift_range=0.,
    zoom_range=0.,
    horizontal_flip=False,
    vertical_flip=False,
    rescale=1./255
)

validDataGenerator = tf.keras.preprocessing.image.
    ImageDataGenerator(
    rotation_range=0.,
    width_shift_range=0.,
    height_shift_range=0.,
    zoom_range=0.,
    horizontal_flip=False,
    vertical_flip=False,
    rescale=1./255
)

testDataGenerator = tf.keras.preprocessing.image.
```

```python
    ImageDataGenerator(
        rotation_range=0.,
        width_shift_range=0.,
        height_shift_range=0.,
        zoom_range=0.,
        horizontal_flip=False,
        vertical_flip=False,
        rescale=1./255
)


trainGenerator = trainDataGenerator.flow_from_directory(
    directory=r"./train/",
    target_size=(224,224),
    batch_size=50,
    class_mode="categorical",
)


validGenerator = validDataGenerator.flow_from_directory(
    directory=r"./validation/",
    target_size=(224,224),
    batch_size=50,
    class_mode="categorical",
)


testGenerator = testDataGenerator.flow_from_directory(
    directory=r"./test/",
    target_size=(224,224),
    batch_size=1,
    class_mode=None,
    shuffle=False
```

```
)

model = tf.keras.Sequential()
model.add(layers.Conv2D(
    filters=32, kernel_size=(3,3), input_shape
        =(224,224,3),
    strides=(1,1), padding='same',
    dilation_rate=(1,1), activation='relu',
    kernel_initializer='glorot_uniform',
    kernel_regularizer=None
))
model.add(layers.Conv2D(
    filters=32, kernel_size=(3,3), strides=(1,1), padding
        ='same',
    dilation_rate=(1,1), activation='relu',
    kernel_initializer='glorot_uniform',
    kernel_regularizer=None
))
model.add(layers.MaxPooling2D(
    pool_size=(2,2)
))
model.add(layers.Conv2D(
    filters=64, kernel_size=(3,3), strides=(1,1), padding
        ='same',
    dilation_rate=(1,1), activation='relu',
    kernel_initializer='glorot_uniform',
    kernel_regularizer=None
))
model.add(layers.Conv2D(
    filters=64, kernel_size=(3,3), strides=(1,1), padding
```

```
        =' same ' ,
    dilation_rate =(1 ,1) , activation =' relu ' ,
    kernel_initializer =' glorot_uniform ' ,
    kernel_regularizer=None
) )
model . add ( layers . MaxPooling2D (
    pool_size =(2 ,2) , strides =None
) )
model . add ( layers . Flatten ( ) )
model . add ( layers . Dense (
    units =512 ,
    activation =' relu ' ,
    kernel_initializer =' glorot_uniform ' ,
    kernel_regularizer=regularizers . l2 (0.00025)
) )
model . add ( layers . Dense (
    units =6 ,
    activation =' softmax ' ,
    kernel_initializer =' glorot_uniform '
) )

model . compile (
    optimizer=optimizers . Adam( learning_rate =0.0001) ,
    loss = ' categorical_crossentropy ' ,
    metrics =[ ' accuracy ' ]
)

model . save ( ' / content / drive /My Drive / Colab Notebooks /EE544
    / Assignment / q1pa . h5 ' )
```

```python
print(model.summary())

trainSteps = trainGenerator.n // trainGenerator.batch_size
validSteps = validGenerator.n // validGenerator.batch_size
testSteps = testGenerator.n // testGenerator.batch_size

es = callbacks.EarlyStopping(
    monitor='val_loss', patience=7, mode='auto'
)

trainTimeStart = time.perf_counter()

history = model.fit(
    x=trainGenerator,
    steps_per_epoch=trainSteps,
    validation_data=validGenerator,
    validation_steps=validSteps,
    epochs=30,
    callbacks=[es]
)

trainTimeEnd = time.perf_counter()

totalTime = trainTimeEnd - trainTimeStart

print("Total Time = %.4f" % totalTime)

print(history.history.keys())
```

```python
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

scores = model.evaluate(
    x=validGenerator,
    steps=validSteps,
)

print('CNN Accuracy: %.2f%%' % (scores[1]*100.0))
print('CNN Error: %.2f%%' % (100-scores[1]*100))

testGenerator.reset()
pred = model.predict(testGenerator, steps=testSteps,
    verbose=1)
predictedClass = np.argmax(pred, axis=-1)
predictedClassIndices = np.argmax(pred, axis=1)
labels = (testGenerator.labels)
```

```
classLabels = (testGenerator.classes)

targetNames = ['Church', 'English Springer', 'Garbage
    Truck', 'Gas Pump',
                'Parachute', 'Tench']
print(classification_report(predictedClassIndices, labels
    , target_names=targetNames))

matrix = confusion_matrix(predictedClassIndices, labels)
print(matrix)
seaborn.set(font_scale=1.4)
seaborn.heatmap(matrix, annot=True, annot_kws={"size":
    16})
plt.show()

drive.flush_and_unmount()
print('All changes made in this colab session should now
    be visible in Drive.')
```

5.1.2 Part b

```
%tensorflow_version 2.x

from __future__ import absolute_import, division,
    print_function, unicode_literals
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import optimizers
from tensorflow.keras import losses
from tensorflow.keras import constraints
```

```python
from tensorflow.keras import regularizers
from tensorflow.keras import callbacks
from google.colab import drive
from sklearn.metrics import classification_report,
    confusion_matrix
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn
import time


print(tf.__version__)
print(tf.test.gpu_device_name())


drive.mount('/content/drive')
root_dir = "/content/drive/My\ Drive/"
base_dir = root_dir + 'Colab\ Notebooks/EE544/Assignment/
    '
print('Extracting Data')
!unzip -qo {base_dir + 'imagenette_6class.zip'}
print("Extraction Done!")


trainDataGenerator = tf.keras.preprocessing.image.
    ImageDataGenerator(
     rotation_range=0.,
     width_shift_range=0.,
     height_shift_range=0.,
     zoom_range=0.,
     horizontal_flip=False,
     vertical_flip=False,
```

```python
    rescale=1./255
)

validDataGenerator = tf.keras.preprocessing.image.
    ImageDataGenerator(
    rotation_range=0.,
    width_shift_range=0.,
    height_shift_range=0.,
    zoom_range=0.,
    horizontal_flip=False,
    vertical_flip=False,
    rescale=1./255
)

testDataGenerator = tf.keras.preprocessing.image.
    ImageDataGenerator(
    rotation_range=0.,
    width_shift_range=0.,
    height_shift_range=0.,
    zoom_range=0.,
    horizontal_flip=False,
    vertical_flip=False,
    rescale=1./255
)

trainGenerator = trainDataGenerator.flow_from_directory(
    directory=r"./train/",
    target_size=(224,224),
    batch_size=50,
    class_mode="categorical",
```

```
)

validGenerator = validDataGenerator.flow_from_directory(
    directory=r"./validation/",
    target_size=(224,224),
    batch_size=50,
    class_mode="categorical",
)

testGenerator = testDataGenerator.flow_from_directory(
    directory=r"./test/",
    target_size=(224,224),
    batch_size=1,
    class_mode=None,
    shuffle=False
)

model = tf.keras.Sequential()
model.add(layers.Conv2D(
    filters=32, kernel_size=(3,3), input_shape
        =(224,224,3),
    strides=(1,1), padding='same',
    dilation_rate=(1,1), activation='relu',
    kernel_initializer='glorot_uniform',
    kernel_regularizer=None
))
model.add(layers.Conv2D(
    filters=32, kernel_size=(3,3), strides=(1,1), padding
        ='same',
    dilation_rate=(1,1), activation='relu',
```

```python
    kernel_initializer='glorot_uniform',
    kernel_regularizer=None
))
model.add(layers.MaxPooling2D(
    pool_size=(2,2)
))
model.add(layers.Dropout(0.1))
model.add(layers.Conv2D(
    filters=64, kernel_size=(3,3), strides=(1,1), padding
        ='same',
    dilation_rate=(1,1), activation='relu',
    kernel_initializer='glorot_uniform',
    kernel_regularizer=None
))
model.add(layers.Conv2D(
    filters=64, kernel_size=(3,3), strides=(1,1), padding
        ='same',
    dilation_rate=(1,1), activation='relu',
    kernel_initializer='glorot_uniform',
    kernel_regularizer=None
))
model.add(layers.MaxPooling2D(
    pool_size=(2,2), strides=None
))
model.add(layers.Dropout(0.1))
model.add(layers.Flatten())
model.add(layers.Dense(
    units=512,
    activation='relu',
    kernel_initializer='glorot_uniform',
```

```python
        kernel_regularizer=regularizers.l2(0.00025)
))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(
    units=6,
    activation='softmax',
    kernel_initializer='glorot_uniform'
))

model.compile(
    optimizer=optimizers.Adam(learning_rate = 0.0001),
    loss = 'categorical_crossentropy',
    metrics=['accuracy']
)

model.save('/content/drive/My Drive/Colab Notebooks/EE544
    /Assignment/q1pb.h5')


print(model.summary())

trainSteps = trainGenerator.n//trainGenerator.batch_size
validSteps = validGenerator.n//validGenerator.batch_size
testSteps = testGenerator.n//testGenerator.batch_size

es = callbacks.EarlyStopping(
    monitor='val_loss', patience=7, mode='auto'
)

trainTimeStart = time.perf_counter()
```

```python
history = model.fit(
    x=trainGenerator,
    steps_per_epoch=trainSteps,
    validation_data=validGenerator,
    validation_steps=validSteps,
    epochs=30,
    callbacks = [es]
)

trainTimeEnd = time.perf_counter()

totalTime = trainTimeEnd - trainTimeStart

print("Total Time = %.4f" % totalTime)

print(history.history.keys())

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
```

```python
plt.xlabel('Epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

scores = model.evaluate(
    x=validGenerator,
    steps=validSteps,
)

print('CNN Accuracy: %.2f%%' % (scores[1]*100.0))
print('CNN Error: %.2f%%' % (100-scores[1]*100))

testGenerator.reset()
pred = model.predict(testGenerator, steps=testSteps,
    verbose=1)
predictedClass = np.argmax(pred, axis=-1)
predictedClassIndices = np.argmax(pred, axis=1)
labels = (testGenerator.labels)
classLabels = (testGenerator.classes)

targetNames = ['Church', 'English Springer', 'Garbage
    Truck', 'Gas Pump',
                'Parachute', 'Tench']
print(classification_report(predictedClassIndices, labels
    , target_names=targetNames))

matrix = confusion_matrix(predictedClassIndices, labels)
print(matrix)
seaborn.set(font_scale=1.4)
seaborn.heatmap(matrix, annot=True, annot_kws={"size":
```

```
   16})
plt.show()


drive.flush_and_unmount()
print('All changes made in this colab session should now
    be visible in Drive.')
```

5.1.3 Part c

```
%tensorflow_version 2.x


from __future__ import absolute_import, division,
    print_function, unicode_literals
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import optimizers
from tensorflow.keras import losses
from tensorflow.keras import constraints
from tensorflow.keras import regularizers
from tensorflow.keras import callbacks
from google.colab import drive
from sklearn.metrics import classification_report,
    confusion_matrix
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn
import time


print(tf.__version__)
```

```python
print(tf.test.gpu_device_name())

drive.mount('/content/drive')
root_dir = "/content/drive/My\ Drive/"
base_dir = root_dir + 'Colab\ Notebooks/EE544/Assignment/
    '
print('Extracting Data')
!unzip −qo {base_dir + 'imagenette_6class.zip'}
print("Extraction Done!")

trainDataGenerator = tf.keras.preprocessing.image.
    ImageDataGenerator(
    rotation_range=0.,
    width_shift_range=0.1,
    height_shift_range=0.1,
    zoom_range=0.,
    horizontal_flip=True,
    vertical_flip=True,
    rescale=1./255
)

validDataGenerator = tf.keras.preprocessing.image.
    ImageDataGenerator(
    rotation_range=0.,
    width_shift_range=0.,
    height_shift_range=0.,
    zoom_range=0.,
    horizontal_flip=False,
    vertical_flip=False,
    rescale=1./255
```

```python
)

testDataGenerator = tf.keras.preprocessing.image.
    ImageDataGenerator(
    rotation_range=0.,
    width_shift_range=0.,
    height_shift_range=0.,
    zoom_range=0.,
    horizontal_flip=False,
    vertical_flip=False,
    rescale=1./255
)

trainGenerator = trainDataGenerator.flow_from_directory(
    directory=r"./train/",
    target_size=(224,224),
    batch_size=50,
    class_mode="categorical",
)

validGenerator = validDataGenerator.flow_from_directory(
    directory=r"./validation/",
    target_size=(224,224),
    batch_size=50,
    class_mode="categorical",
)

testGenerator = testDataGenerator.flow_from_directory(
    directory=r"./test/",
    target_size=(224,224),
```

```python
    batch_size=1,
    class_mode=None,
    shuffle=False
)

model = tf.keras.Sequential()
model.add(layers.Conv2D(
    filters=32, kernel_size=(3,3), input_shape
        =(224,224,3),
    strides=(1,1), padding='same',
    dilation_rate=(1,1), activation='relu',
    kernel_initializer='glorot_uniform',
    kernel_regularizer=None
))
model.add(layers.Conv2D(
    filters=32, kernel_size=(3,3), strides=(1,1), padding
        ='same',
    dilation_rate=(1,1), activation='relu',
    kernel_initializer='glorot_uniform',
    kernel_regularizer=None
))
model.add(layers.MaxPooling2D(
    pool_size=(2,2)
))
model.add(layers.Conv2D(
    filters=64, kernel_size=(3,3), strides=(1,1), padding
        ='same',
    dilation_rate=(1,1), activation='relu',
    kernel_initializer='glorot_uniform',
    kernel_regularizer=None
```

```python
))
model.add(layers.Conv2D(
    filters=64, kernel_size=(3,3), strides=(1,1), padding
        ='same',
    dilation_rate=(1,1), activation='relu',
    kernel_initializer='glorot_uniform',
    kernel_regularizer=None
))
model.add(layers.MaxPooling2D(
    pool_size=(2,2), strides=None
))
model.add(layers.Flatten())
model.add(layers.Dense(
    units=512,
    activation='relu',
    kernel_initializer='glorot_uniform',
    kernel_regularizer=regularizers.l2(0.00025)
))
model.add(layers.Dense(
    units=6,
    activation='softmax',
    kernel_initializer='glorot_uniform'
))

model.compile(
    optimizer=optimizers.Adam(learning_rate = 0.0001),
    loss = 'categorical_crossentropy',
    metrics=['accuracy']
)
```

```python
model.save('/content/drive/My Drive/Colab Notebooks/EE544
    /Assignment/q1pc.h5')


print(model.summary())

trainSteps = trainGenerator.n//trainGenerator.batch_size
validSteps = validGenerator.n//validGenerator.batch_size
testSteps = testGenerator.n//testGenerator.batch_size

es = callbacks.EarlyStopping(
    monitor='val_loss', patience=5, mode='auto'
)

trainTimeStart = time.perf_counter()

history = model.fit(
    x=trainGenerator,
    steps_per_epoch=trainSteps,
    validation_data=validGenerator,
    validation_steps=validSteps,
    epochs=30,
    callbacks = [es]
)

trainTimeEnd = time.perf_counter()

totalTime = trainTimeEnd - trainTimeStart

print("Total Time = %.4f" % totalTime)
```

```python
print(history.history.keys())

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

scores = model.evaluate(
    x=validGenerator,
    steps=validSteps,
)

print('CNN Accuracy: %.2f%%' % (scores[1]*100.0))
print('CNN Error: %.2f%%' % (100-scores[1]*100))

testGenerator.reset()
pred = model.predict(testGenerator, steps=testSteps,
    verbose=1)
```

```
predictedClass = np.argmax(pred, axis=-1)
predictedClassIndices = np.argmax(pred, axis=1)
labels = (testGenerator.labels)
classLabels = (testGenerator.classes)

targetNames = ['Church', 'English Springer', 'Garbage
    Truck', 'Gas Pump',
                'Parachute', 'Tench']
print(classification_report(predictedClassIndices, labels
    , target_names=targetNames))

matrix = confusion_matrix(predictedClassIndices, labels)
print(matrix)
seaborn.set(font_scale=1.4)
seaborn.heatmap(matrix, annot=True, annot_kws={"size":
    16})
plt.show()

drive.flush_and_unmount()
print('All changes made in this colab session should now
    be visible in Drive.')
```

5.1.4 Part d

```
%tensorflow_version 2.x

from __future__ import absolute_import, division,
    print_function, unicode_literals
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
```

```python
from tensorflow.keras import optimizers
from tensorflow.keras import losses
from tensorflow.keras import constraints
from tensorflow.keras import regularizers
from tensorflow.keras import callbacks
from google.colab import drive
from sklearn.metrics import classification_report,
    confusion_matrix
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn
import time


print(tf.__version__)
print(tf.test.gpu_device_name())


drive.mount('/content/drive')
root_dir = "/content/drive/My\ Drive/"
base_dir = root_dir + 'Colab\ Notebooks/EE544/Assignment/
    '
print('Extracting Data')
!unzip -qo {base_dir + 'imagenette_6class.zip'}
print("Extraction Done!")


trainDataGenerator = tf.keras.preprocessing.image.
    ImageDataGenerator(
    rotation_range=0.,
    width_shift_range=0.,
    height_shift_range=0.,
```

```python
    zoom_range =0. ,
    horizontal_flip=False ,
    vertical_flip=False ,
    rescale =1./255
)

validDataGenerator = tf.keras.preprocessing.image.
    ImageDataGenerator (
    rotation_range =0. ,
    width_shift_range =0. ,
    height_shift_range =0. ,
    zoom_range =0. ,
    horizontal_flip=False ,
    vertical_flip=False ,
    rescale =1./255
)

testDataGenerator = tf.keras.preprocessing.image.
    ImageDataGenerator (
    rotation_range =0. ,
    width_shift_range =0. ,
    height_shift_range =0. ,
    zoom_range =0. ,
    horizontal_flip=False ,
    vertical_flip=False ,
    rescale =1./255
)

trainGenerator = trainDataGenerator.flow_from_directory (
    directory=r"./ train /" ,
```

```python
    target_size =(224 ,224) ,
    batch_size =50,
    class_mode="categorical",
)

validGenerator = validDataGenerator . flow_from_directory (
    directory =r"./ validation /",
    target_size =(224 ,224) ,
    batch_size =50,
    class_mode="categorical",
)

testGenerator = testDataGenerator . flow_from_directory (
    directory =r"./ test /",
    target_size =(224 ,224) ,
    batch_size =1,
    class_mode=None ,
    shuffle =False
)

model = tf . keras . Sequential ()
model . add ( layers . Conv2D(
    filters =32, kernel_size =(3 ,3) , input_shape
        =(224 ,224 ,3) ,
    strides =(1 ,1) , padding='same',
    dilation_rate =(1 ,1) , activation ='relu',
    kernel_initializer ='glorot_uniform ',
    kernel_regularizer=None
) )
model . add ( layers . Conv2D(
```

```
        filters =32,  kernel_size =(3,3),  strides =(1,1),  padding
            ='same',
        dilation_rate =(1,1),  activation ='relu',
        kernel_initializer ='glorot_uniform',
        kernel_regularizer=None
))
model.add(layers.BatchNormalization(axis =1))
model.add(layers.MaxPooling2D(
        pool_size =(2,2)
))
model.add(layers.Conv2D(
        filters =64,  kernel_size =(3,3),  strides =(1,1),  padding
            ='same',
        dilation_rate =(1,1),  activation ='relu',
        kernel_initializer ='glorot_uniform',
        kernel_regularizer=None
))
model.add(layers.Conv2D(
        filters =64,  kernel_size =(3,3),  strides =(1,1),  padding
            ='same',
        dilation_rate =(1,1),  activation ='relu',
        kernel_initializer ='glorot_uniform',
        kernel_regularizer=None
))
model.add(layers.BatchNormalization(axis =1))
model.add(layers.MaxPooling2D(
        pool_size =(2,2),  strides=None
))
model.add(layers.Flatten())
model.add(layers.Dense(
```

```python
        units=512,
        activation='relu',
        kernel_initializer='glorot_uniform',
        kernel_regularizer=regularizers.l2(0.00025)
))
model.add(layers.BatchNormalization(axis=1))
model.add(layers.Dense(
        units=6,
        activation='softmax',
        kernel_initializer='glorot_uniform'
))

model.compile(
        optimizer=optimizers.Adam(learning_rate=0.0001),
        loss = 'categorical_crossentropy',
        metrics=['accuracy']
)

model.save('/content/drive/My Drive/Colab Notebooks/EE544
        /Assignment/q1pd.h5')


print(model.summary())

trainSteps = trainGenerator.n//trainGenerator.batch_size
validSteps = validGenerator.n//validGenerator.batch_size
testSteps = testGenerator.n//testGenerator.batch_size

es = callbacks.EarlyStopping(
        monitor='val_loss', patience=5, mode='auto'
```

```
)

trainTimeStart = time.perf_counter()

history = model.fit(
    x=trainGenerator,
    steps_per_epoch=trainSteps,
    validation_data=validGenerator,
    validation_steps=validSteps,
    epochs=30,
    callbacks = [es]
)

trainTimeEnd = time.perf_counter()

totalTime = trainTimeEnd - trainTimeStart

print("Total Time = %.4f" % totalTime)

print(history.history.keys())

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

plt.plot(history.history['loss'])
```

```python
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()


scores = model.evaluate(
    x=validGenerator,
    steps=validSteps,
)


print('CNN Accuracy: %.2f%%' % (scores[1]*100.0))
print('CNN Error: %.2f%%' % (100-scores[1]*100))


testGenerator.reset()
pred = model.predict(testGenerator, steps=testSteps,
    verbose=1)
predictedClass = np.argmax(pred, axis=-1)
predictedClassIndices = np.argmax(pred, axis=1)
labels = (testGenerator.labels)
classLabels = (testGenerator.classes)


targetNames = ['Church', 'English Springer', 'Garbage
    Truck', 'Gas Pump',
                'Parachute', 'Tench']
print(classification_report(predictedClassIndices, labels
    , target_names=targetNames))


matrix = confusion_matrix(predictedClassIndices, labels)
```

```
print ( matrix )
seaborn . set ( font_scale =1.4)
seaborn . heatmap ( matrix , annot=True , annot_kws={" size ":
    16})
plt . show ()

drive . flush_and_unmount ()
print ('All changes made in this colab session should now
    be visible in Drive . ')
```

5.1.5 Part e

```
%tensorflow_version 2 . x

from __future__ import absolute_import , division ,
    print_function , unicode_literals
import tensorflow as tf
from tensorflow import keras
from tensorflow . keras import layers
from tensorflow . keras import optimizers
from tensorflow . keras import losses
from tensorflow . keras import constraints
from tensorflow . keras import regularizers
from tensorflow . keras import callbacks
from google . colab import drive
from sklearn . metrics import classification_report ,
    confusion_matrix
import numpy as np
import pandas as pd
import matplotlib . pyplot as plt
import seaborn
```

```python
import time

print(tf.__version__)
print(tf.test.gpu_device_name())

drive.mount('/content/drive')
root_dir = "/content/drive/My\ Drive/"
base_dir = root_dir + 'Colab\ Notebooks/EE544/Assignment/
    '
print('Extracting Data')
!unzip -qo {base_dir + 'imagenette_6class.zip'}
print("Extraction Done!")

trainDataGenerator = tf.keras.preprocessing.image.
    ImageDataGenerator(
     rotation_range=0.,
     width_shift_range=0.1,
     height_shift_range=0.1,
     zoom_range=0.,
     horizontal_flip=True,
     vertical_flip=True,
     rescale=1./255
)

validDataGenerator = tf.keras.preprocessing.image.
    ImageDataGenerator(
     rotation_range=0.,
     width_shift_range=0.,
     height_shift_range=0.,
     zoom_range=0.,
```

```python
    horizontal_flip=False,
    vertical_flip=False,
    rescale=1./255
)

testDataGenerator = tf.keras.preprocessing.image.
    ImageDataGenerator(
    rotation_range=0.,
    width_shift_range=0.,
    height_shift_range=0.,
    zoom_range=0.,
    horizontal_flip=False,
    vertical_flip=False,
    rescale=1./255
)

trainGenerator = trainDataGenerator.flow_from_directory(
    directory=r"./train/",
    target_size=(224,224),
    batch_size=50,
    class_mode="categorical",
)

validGenerator = validDataGenerator.flow_from_directory(
    directory=r"./validation/",
    target_size=(224,224),
    batch_size=50,
    class_mode="categorical",
)
```

```python
testGenerator = testDataGenerator.flow_from_directory(
    directory=r"./test/",
    target_size=(224,224),
    batch_size=1,
    class_mode=None,
    shuffle=False
)


model = tf.keras.Sequential()
model.add(layers.Conv2D(
    filters=32, kernel_size=(3,3), input_shape
        =(224,224,3),
    strides=(1,1), padding='same',
    dilation_rate=(1,1), activation='relu',
    kernel_initializer='glorot_uniform',
    kernel_regularizer=None
))
model.add(layers.Conv2D(
    filters=32, kernel_size=(3,3), strides=(1,1), padding
        ='same',
    dilation_rate=(1,1), activation='relu',
    kernel_initializer='glorot_uniform',
    kernel_regularizer=None
))
# model.add(layers.BatchNormalization(axis=1))
model.add(layers.Dropout(0.25))
model.add(layers.MaxPooling2D(
    pool_size=(2,2)
))
model.add(layers.Conv2D(
```

```
    filters =64, kernel_size =(3,3), strides =(1,1), padding
        ='same',
    dilation_rate =(1,1), activation ='relu',
    kernel_initializer ='glorot_uniform',
    kernel_regularizer=None
))
model.add(layers.Conv2D(
    filters =64, kernel_size =(3,3), strides =(1,1), padding
        ='same',
    dilation_rate =(1,1), activation ='relu',
    kernel_initializer ='glorot_uniform',
    kernel_regularizer=None
))
# model.add(layers.BatchNormalization(axis=1))
model.add(layers.Dropout(0.25))
model.add(layers.MaxPooling2D(
    pool_size =(2,2), strides=None
))
model.add(layers.Flatten())
model.add(layers.Dense(
    units=512,
    activation ='relu',
    kernel_initializer ='glorot_uniform',
    kernel_regularizer=regularizers.l2(0.00025)
))
# model.add(layers.BatchNormalization(axis=1))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(
    units=6,
    activation ='softmax',
```

```python
        kernel_initializer='glorot_uniform'
))

model.compile(
    optimizer=optimizers.Adam(learning_rate=0.0001),
    loss = 'categorical_crossentropy',
    metrics=['accuracy']
)

model.save('/content/drive/My Drive/Colab Notebooks/EE544
    /Assignment/q1pe.h5')


print(model.summary())

trainSteps = trainGenerator.n//trainGenerator.batch_size
validSteps = validGenerator.n//validGenerator.batch_size
testSteps = testGenerator.n//testGenerator.batch_size

es = callbacks.EarlyStopping(
    monitor='val_loss', patience=5, mode='auto'
)

trainTimeStart = time.perf_counter()

history = model.fit(
    x=trainGenerator,
    steps_per_epoch=trainSteps,
    validation_data=validGenerator,
    validation_steps=validSteps,
```

```
    epochs =15,
    callbacks = [es]
)


trainTimeEnd = time.perf_counter()


totalTime = trainTimeEnd - trainTimeStart


print("Total Time = %.4f" % totalTime)


print(history.history.keys())


plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()


plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()


scores = model.evaluate(
    x=validGenerator,
```

```python
        steps=validSteps,
)

print('CNN Accuracy: %.2f%%' % (scores[1]*100.0))
print('CNN Error: %.2f%%' % (100-scores[1]*100))

testGenerator.reset()
pred = model.predict(testGenerator, steps=testSteps,
    verbose=1)
predictedClass = np.argmax(pred, axis=-1)
predictedClassIndices = np.argmax(pred, axis=1)
labels = (testGenerator.labels)
classLabels = (testGenerator.classes)

targetNames = ['Church', 'English Springer', 'Garbage
    Truck', 'Gas Pump',
                'Parachute', 'Tench']
print(classification_report(predictedClassIndices, labels
    , target_names=targetNames))

matrix = confusion_matrix(predictedClassIndices, labels)
print(matrix)
seaborn.set(font_scale=1.4)
seaborn.heatmap(matrix, annot=True, annot_kws={"size":
    16})
plt.show()

drive.flush_and_unmount()
print('All changes made in this colab session should now
    be visible in Drive.')
```

5.2 Question 2

5.2.1 Part a

```
%tensorflow_version 2.x

from __future__ import absolute_import, division,
    print_function, unicode_literals
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import optimizers
from tensorflow.keras import losses
from tensorflow.keras import constraints
from tensorflow.keras import regularizers
from tensorflow.keras.applications.resnet import ResNet50
    , preprocess_input
from google.colab import drive
from sklearn.metrics import classification_report,
    confusion_matrix
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn
import time

print(tf.__version__)
print(tf.test.gpu_device_name())

drive.mount('/content/drive')
root_dir = "/content/drive/My\ Drive/"
```

```python
base_dir = root_dir + 'Colab\ Notebooks/EE544/Assignment/
    '
print('Extracting Data')
!unzip -qo {base_dir + 'food101_4class.zip'}
print("Extraction Done!")

trainDataGenerator = tf.keras.preprocessing.image.
    ImageDataGenerator(
    preprocessing_function = tf.keras.applications.
        resnet50.preprocess_input
)

validDataGenerator = tf.keras.preprocessing.image.
    ImageDataGenerator()

testDataGenerator = tf.keras.preprocessing.image.
    ImageDataGenerator()

trainGenerator = trainDataGenerator.flow_from_directory(
    directory=r"./train/",
    target_size=(224,224),
    batch_size=50,
    class_mode="categorical",
)

validGenerator = validDataGenerator.flow_from_directory(
    directory=r"./validation/",
    target_size=(224,224),
    batch_size=50,
    class_mode="categorical",
```

```python
)

testGenerator = testDataGenerator.flow_from_directory(
    directory=r"./test/",
    target_size=(224,224),
    batch_size=1,
    class_mode=None,
    shuffle=False
)

base_model = tf.keras.applications.ResNet50(
    include_top=False,
    weights='imagenet',
    input_shape=(224,224,3)
)

x = base_model.output
x = layers.Flatten()(x)
preds = layers.Dense(4, activation='softmax')(x)

model = tf.keras.models.Model(inputs=base_model.input,
    outputs=preds)

for layer in model.layers[:143]:
  layer.trainable = False

for layer in model.layers[143:]:
  layer.trainable = True

trainSteps = trainGenerator.n//trainGenerator.batch_size
```

```python
validSteps = validGenerator.n // validGenerator.batch_size
testSteps = testGenerator.n // testGenerator.batch_size


model.compile(
    optimizer='Adam',
    loss='categorical_crossentropy',
    metrics=['accuracy']
)

model.save('/content/drive/My Drive/Colab Notebooks/EE544
    /Assignment/q2pa.h5')

print(model.summary())

trainTimeStart = time.perf_counter()

history = model.fit(
    trainGenerator,
    steps_per_epoch=trainSteps,
    epochs = 5,
    validation_data = validGenerator,
    validation_steps = validSteps,
  )

trainTimeEnd = time.perf_counter()

totalTime = trainTimeEnd - trainTimeStart

print("Total Time = %.4f" % totalTime)
```

```python
print(history.history.keys())

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model Accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()

scores = model.evaluate(
    x=validGenerator,
    steps=validSteps,
)

print('CNN Accuracy: %.2f%%' % (scores[1]*100.0))
print('CNN Error: %.2f%%' % (100-scores[1]*100))

testGenerator.reset()
pred = model.predict(testGenerator, steps=testSteps,
    verbose=1)
```

```python
predictedClass = np.argmax(pred, axis=-1)
predictedClassIndices = np.argmax(pred, axis=1)
labels = (testGenerator.labels)
classLabels = (testGenerator.classes)

targetNames = ['Chicken Curry', 'Hamburger', 'Omelette',
    'Waffles']
print(classification_report(predictedClassIndices, labels
    , target_names=targetNames))

matrix = confusion_matrix(predictedClassIndices, labels)
print(matrix)
seaborn.set(font_scale=1.4)
seaborn.heatmap(matrix, annot=True, annot_kws={"size":
    16})
plt.show()

drive.flush_and_unmount()
print('All changes made in this colab session should now
    be visible in Drive.')
```