Ollscoil Chathair
Bhaile Átha Cliath
Dublin City University

Submission Cover Sheet

Please use block capitals

Student Name	**RAFAELA GRACIA NONGRUM**
Student ID No.	20210341
Module Code	**EE544**
Degree	**MASTER'S**
Programme	**MENG IN ELECTRONIC AND COMPUTER ENGINEERING**
Year	**2021-2022**
Date	**04/04/2021**
CA Materials Link (where applicable)	

For use by examiners only (students should not write below this line)

Solution 1a:

Aim: To design and develop a simple VGG-lite baseline convolutional neural network architecture as per the CNN structure illustrated in question to classify 4-Class ImageNette Data.

The simple VGG -lite network was designed and use of the deliverables specified in the question was made, that is, the number of filters, the different parameters and so on. After importing the required libraries and mounting the drive (As the code had been developed in Colab) the training, validation and testing data was loaded for the model. The images were rescaled for ease of computational purposes and shuffled so that the model gets the variation of data for learning. Everything was done as specified by the question. The model was compiled with SGD optimizer and categorical crossentropy as the loss function to update the weights in the model. After testing out the model with different optimizers, use of SGD optimizer was made as it was found to be compatible with the model. It is also known to be a low-cost optimizer. A low learning rate of 0.01 was given as this would make our model to learn slowly and hence enable it to give an accurate prediction. For the last dense layer, SoftMax was used as the activation function as it gave the probabilistic value between the number of classes which are ranging from 0 to 1. The model summary was printed and is shown below:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 224, 224, 32)      896
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 224, 224, 32)      9248
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 112, 112, 32)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 112, 112, 64)      18496
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 112, 112, 64)      36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 56, 56, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 200704)            0
_________________________________________________________________
dense (Dense)                (None, 512)               102760960
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 2052
=================================================================
Total params: 102,828,580
Trainable params: 102,828,580
Non-trainable params: 0
```

Fig 1: Model Summary

From the summary, we can see that the size of the image keeps decreasing as we go down whereas the number of filters increases so as to capture as many combinations for the features as possible. At the input layer, the network receives raw data which is usually noisy, because of this, we let the CNN extract just a few relevant features first. Once these important features have been extracted, we make the CNN extract more complex ones. The model diagram containing all details has been stored in the drive (link for this has been provided at the bottom of this report) under folder 'Model', with file name 'VGG16_model'. Batch size of 32 was given. The model was then compiled and fit. Callbacks were used so as to have an automated control over our model while it was training. In order to save the model after every epoch, ModelCheckpoint callback has been used, validation loss has been monitored here due to which 'mode' has been assigned a minimum value. EarlyStopping was also used with a patience of 7 so that if there was no improvement after 7 epochs, the model would stop training. The model gave a validation accuracy of 75.26 % and a test accuracy of 79.5 %

The following are the plots in graph form:

```
import matplotlib.pyplot as plt
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title("Validation accuracy and Accuracy")
plt.ylabel("Valid Accuracy")
plt.xlabel("Epoch")
plt.legend(["Accuracy","Validation Accuracy"])
plt.show()
```

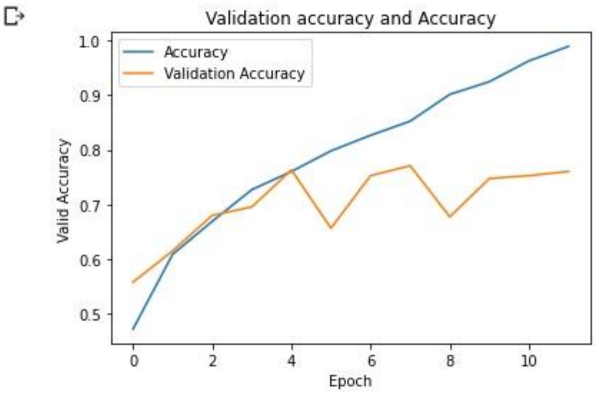


fig2: Validation and Training Accuracy.

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title("Validation Loss and loss")
plt.ylabel("Valid Loss")
plt.xlabel("Epoch")
plt.legend(["Loss","Validation Loss"])
plt.show()
```

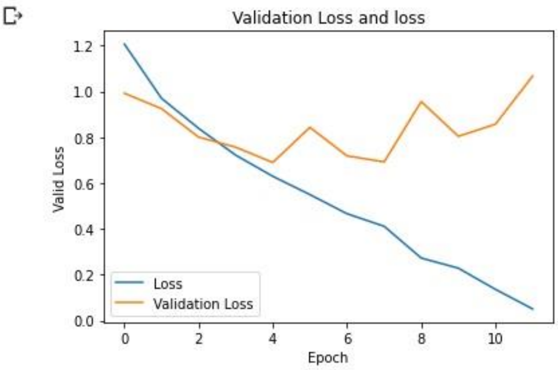


Fig3: Validation and Training Loss.

The Test accuracy and loss was predicted as follows:

```
[ ] print("Validation Loss:",history.history['val_loss'][6])

    Validation Loss: 0.718955934047699
```

```
print("Validation Accuracy:",(history.history['val_accuracy'][6])*100)

Validation Accuracy: 75.26041865348816
```

```
[ ] test_loss, test_acc = model.evaluate(testing_data)
    print('Test Accuracy: ', (test_acc)*100)
    print('Test Loss: ' , test_loss)

    7/7 [==============================] - 60s 10s/step - loss: 0.8347 - accuracy: 0.7950
    Test Accuracy:  79.50000166893005
    Test Loss:  0.8346536159515381
```

Fig 4: Test Accuracy and Loss.

The confusion Matrix was as follows:

```
sn.set(font_scale=1.4) # for label size
sn.heatmap(df_cm, annot=True, annot_kws={"size": 16}) # font size
plt.xlabel("Predicted labels")
plt.ylabel("True labels")
plt.show()
```

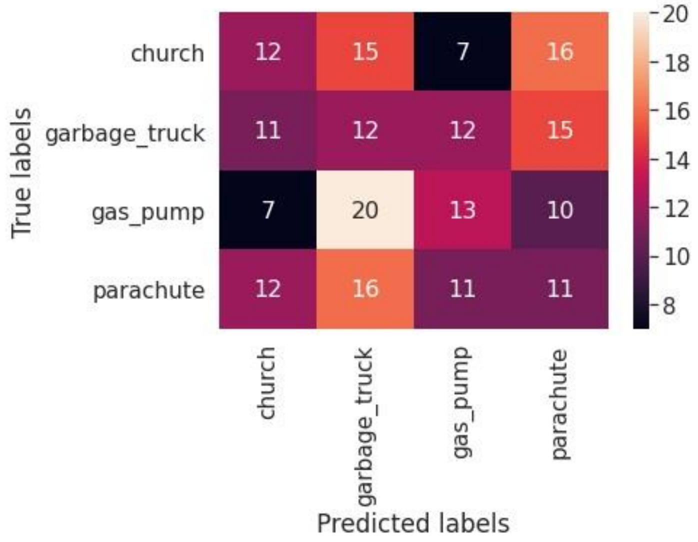


Fig 5: Confusion Matrix.

The Network's Computational Cost was found to be 411311506 in FLOPS:

```
[ ] print("Network FLOPS: ",get_flops(model_path))

    Network FLOPS:  411311506
```

Concluding remarks:

The Model had been built according to the specifications of the question. It was a simple one and hence it did not do a lot of learning. It's performance was average with the exception that it stopped learning after just the fourth epoch. Improvement for this network has been done in solution 1b.

Solution 1b:

Aim: To experiment with ways to improve the baseline VGG-lite network's performance.

The following processes were attempted for improving the previous network's accuracy:

i. <u>Dropout along with Augmentation:</u>

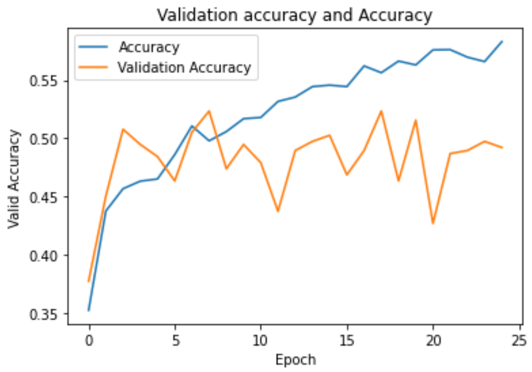
Here, the exact same network was implemented except the training data was augmented along with an additional dropout of 0.05%. As the network stopped learning after a few epochs in the previous network, Augmentation was seen as an option to increase the network's complexity and also to provide variation and extra data for it to learn better. Small amount of dropout was given as the model was neither overfitting nor have a lot of data in the first place, so a high amount would have hindered the performance of the network even further. The following were the outcomes :

a.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 224, 224, 32)      896
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 224, 224, 32)      9248
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 112, 112, 32)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 112, 112, 64)      18496
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 112, 112, 64)      36928
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 56, 56, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 200704)            0
_________________________________________________________________
dropout (Dropout)            (None, 200704)            0
_________________________________________________________________
dense (Dense)                (None, 512)               102760960
_________________________________________________________________
dense_1 (Dense)              (None, 4)                 2052
=================================================================
Total params: 102,828,580
Trainable params: 102,828,580
Non-trainable params: 0
```

<u>Fig 6: Model Summary.</u>

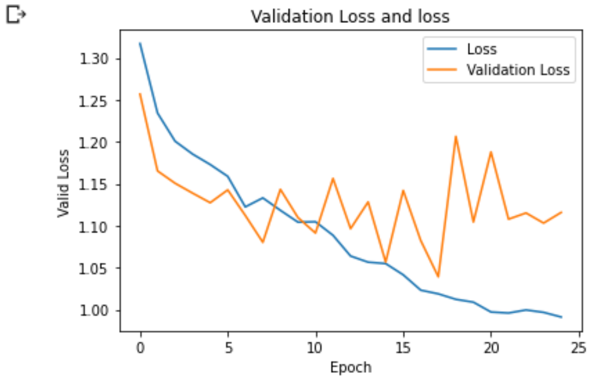
```
[ ] import matplotlib.pyplot as plt
    plt.plot(history.history['accuracy'])
    plt.plot(history.history['val_accuracy'])
    plt.title("Validation accuracy and Accuracy")
    plt.ylabel("Valid Accuracy")
    plt.xlabel("Epoch")
    plt.legend(["Accuracy","Validation Accuracy"])
    plt.show()
```



b.

Fig7: Validation and training accuracy.

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title("Validation Loss and loss")
plt.ylabel("Valid Loss")
plt.xlabel("Epoch")
plt.legend(["Loss","Validation Loss"])
plt.show()
```



c.

Fig 8: Validation and Training Loss.

```
[ ] test_loss, test_acc = model.evaluate(testing_data)
    print('Test Accuracy: ', (test_acc)*100)
    print('Test Loss: ' , test_loss)
```

```
7/7 [==============================] - 70s 12s/step - loss: 1.0200 - accuracy: 0.5700
Test Accuracy:  56.99999928474426
Test Loss:  1.020021915435791
```

d.

Fig 9: Test Accuracy and Loss.

.

e.

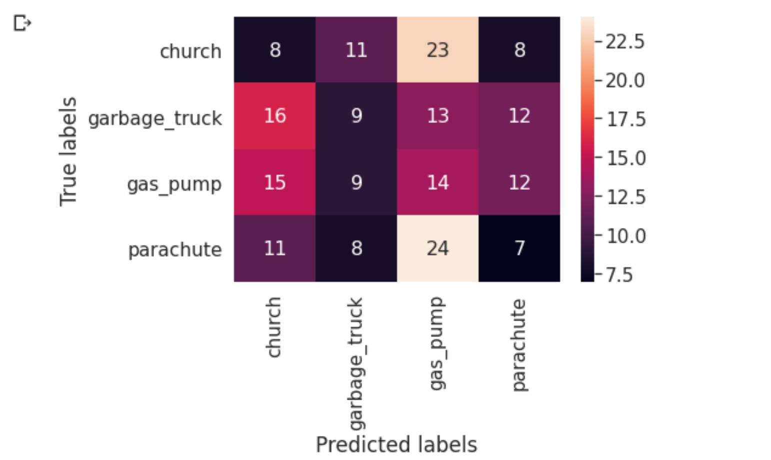


Fig 10: Confusion Matrix.

ii. Network with Batchnormalization and Augmentation.

Batchnormalization was implemented here with the aim of normalizing the inputs and increasing the accuracy.

a.

```
Model: "sequential_2"
_________________________________________________________________
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
_________________________________________________________________
batch_normalization_1 (Batch (None, 200704)            802816
_________________________________________________________________
dense_4 (Dense)              (None, 512)               102760960
_________________________________________________________________
dense_5 (Dense)              (None, 4)                 2052
=================================================================
Total params: 103,631,396
Trainable params: 103,229,988
Non-trainable params: 401,408
_________________________________________________________________
```

Fig 11: Model Summary.

b.

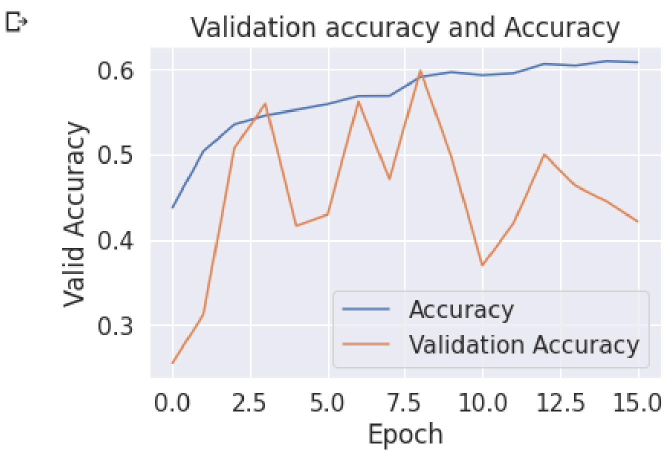
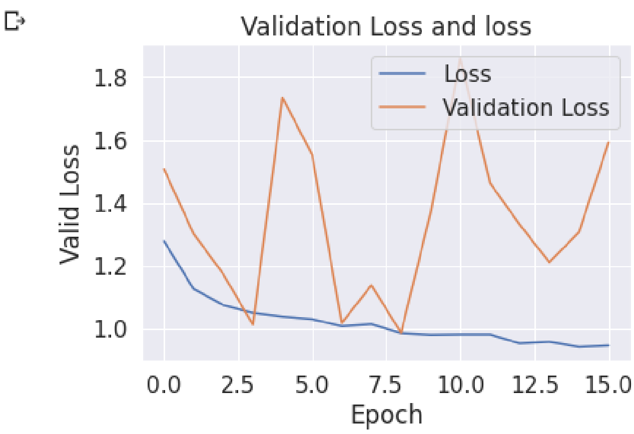


Fig 12: Validation and Training accuracy.



c.

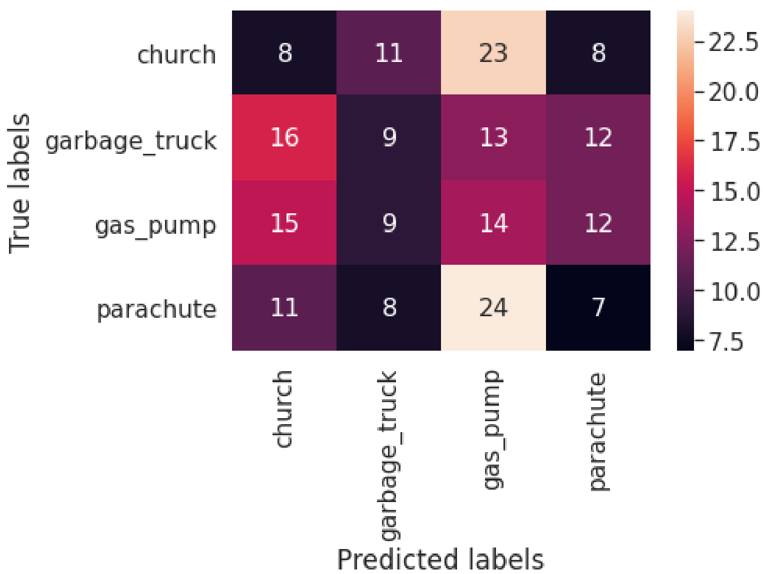
<u>Fig 13: Validation and Training Loss.</u>

```
test_loss, test_acc = model.evaluate(testing_data)
print('Test Accuracy: ', (test_acc)*100)
print('Test Loss: ' , test_loss)
```

```
7/7 [==============================] - 1s 106ms/step - loss: 1.5107 - accuracy: 0.4650
Test Accuracy:  46.50000035762787
Test Loss:  1.5106823444366455
```

d.

<u>Fig 14: Test Accuracy and Loss.</u>



e.

<u>Fig 15: Confusion Matrix.</u>

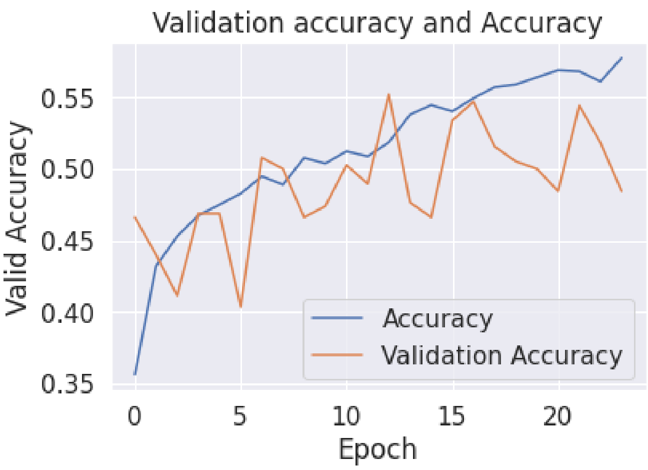
iii. <u>Network with Augmentation alone.</u>

After the second attempt did not prove to be useful, only Augmentation was implemented to check if it would help improve the model by simply adding more data.

```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_16 (Conv2D)           (None, 224, 224, 32)      896
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 224, 224, 32)      9248
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 112, 112, 32)      0
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 112, 112, 64)      18496
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 112, 112, 64)      36928
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 56, 56, 64)        0
_________________________________________________________________
flatten_4 (Flatten)          (None, 200704)            0
_________________________________________________________________
dense_8 (Dense)              (None, 512)               102760960
_________________________________________________________________
dense_9 (Dense)              (None, 4)                 2052
=================================================================
Total params: 102,828,580
Trainable params: 102,828,580
Non-trainable params: 0
_________________________________________________________________
```

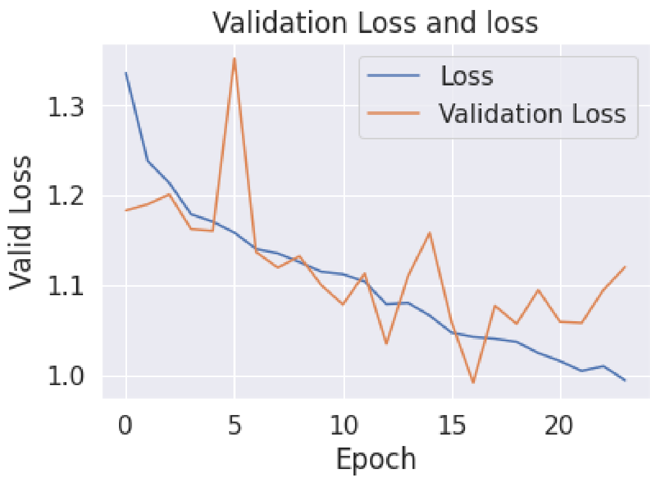
a.

Fig 16: Model Summary.



b.

Fig 17: Training and Validation Accuracy.



c.

Fig 18: Training and Validation Loss.

d.

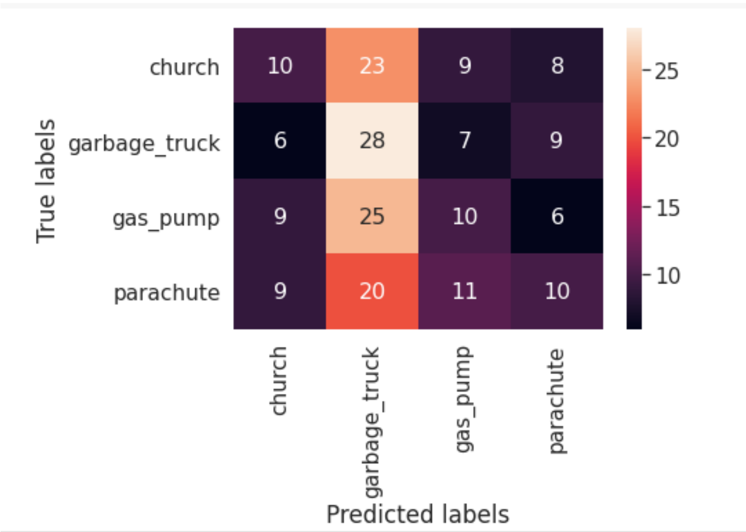


Fig 19: Confusion Matrix.

iv. Network with Regularization and Augmentation.

The previously implemented network seemed to be overfitting quite a bit and hence Regularization was used here with the purpose of trying to improve the network by preventing overfitting.

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_20 (Conv2D)           (None, 224, 224, 32)      896
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 224, 224, 32)      9248
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 112, 112, 32)      0
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 112, 112, 64)      18496
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 112, 112, 64)      36928
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 56, 56, 64)        0
_________________________________________________________________
flatten_5 (Flatten)          (None, 200704)            0
_________________________________________________________________
dense_10 (Dense)             (None, 512)               102760960
_________________________________________________________________
dense_11 (Dense)             (None, 4)                 2052
=================================================================
Total params: 102,828,580
Trainable params: 102,828,580
Non-trainable params: 0
_________________________________________________________________
```

a.

Fig 20: Model Summary.

b.

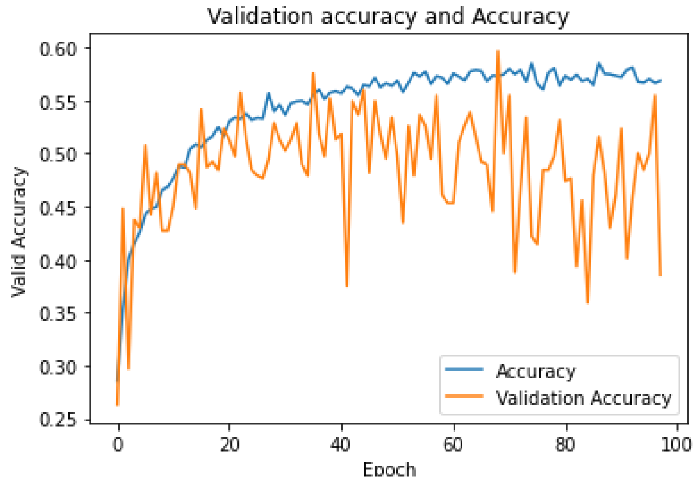


Fig 21: Training and Validation Accuracy.

c.

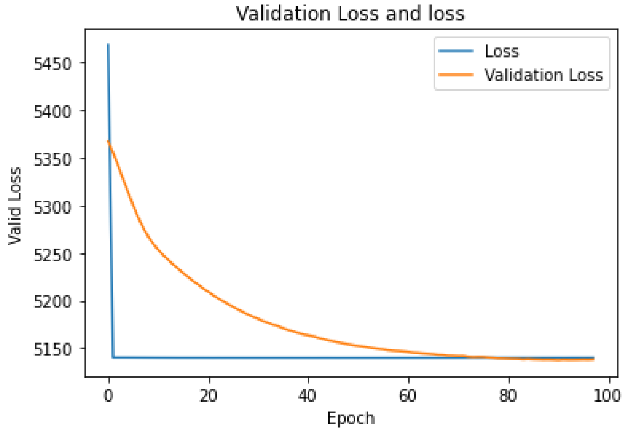


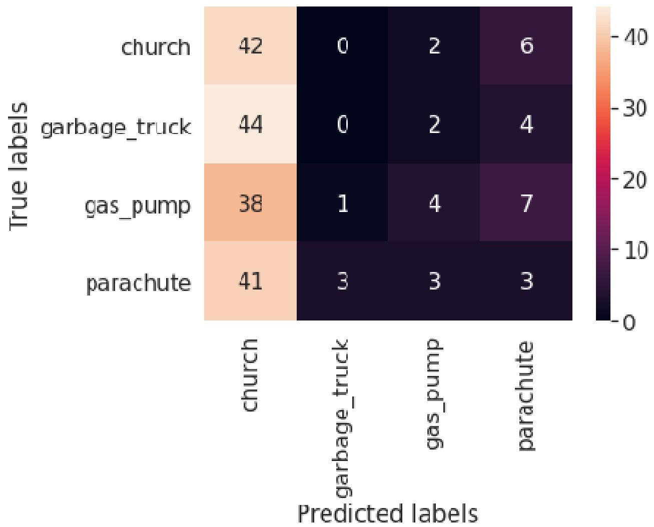
Fig 22: Training and Validation Loss.

```
[ ] test_loss, test_acc = model.evaluate(testing_data)
    print('Test Accuracy: ', (test_acc)*100)
    print('Test Loss: ' , test_loss)

    7/7 [==============================] - 59s 10s/step - loss: 5138.0425 - accuracy: 0.3400
    Test Accuracy:  34.00000035762787
    Test Loss:  5138.04248046875
```

d.

Fig 23: Test Accuracy and Loss.



e.

Fig 24: Confusion Matrix.

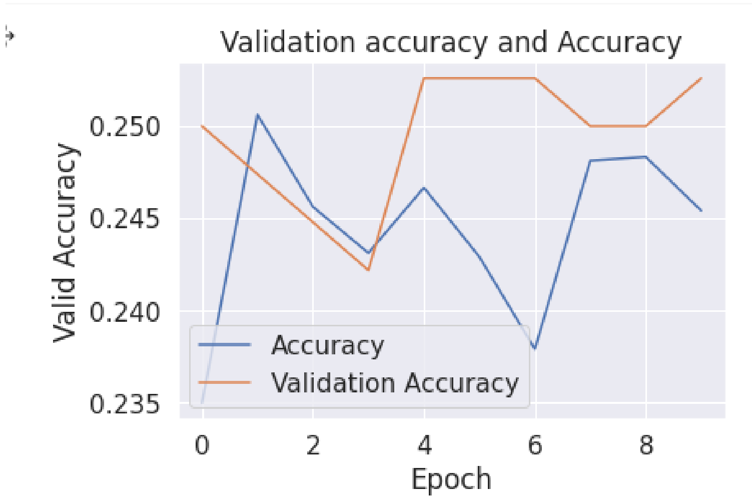
v. Network without Early Stopping and Augmentation

The network still did not seem to perform any better as compared to the one developed in 1a, and hence it came to mind that perhaps an underfitting prevention task should be performed for the last attempt.

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_24 (Conv2D)           (None, 224, 224, 32)      896
_________________________________________________________________
conv2d_25 (Conv2D)           (None, 224, 224, 32)      9248
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 112, 112, 32)      0
_________________________________________________________________
conv2d_26 (Conv2D)           (None, 112, 112, 64)      18496
_________________________________________________________________
conv2d_27 (Conv2D)           (None, 112, 112, 64)      36928
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 56, 56, 64)        0
_________________________________________________________________
flatten_6 (Flatten)          (None, 200704)            0
_________________________________________________________________
dense_12 (Dense)             (None, 512)               102760960
_________________________________________________________________
dense_13 (Dense)             (None, 4)                 2052
=================================================================
Total params: 102,828,580
Trainable params: 102,828,580
Non-trainable params: 0
_________________________________________________________________
```

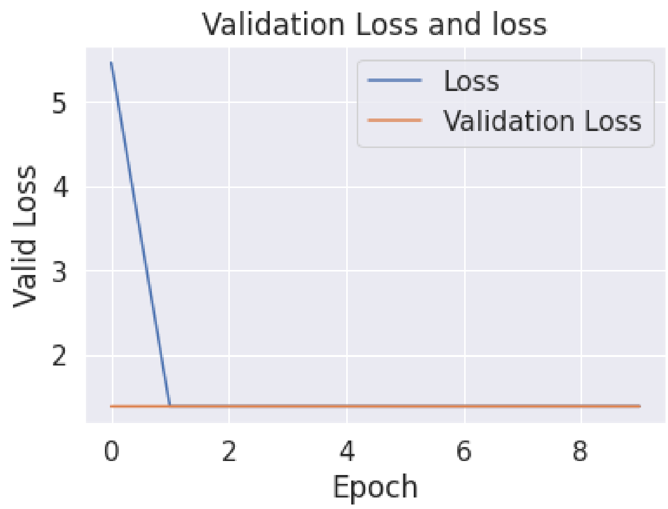
a.

Fig 25: Model Summary.



b.

Fig 26: Validation and Training Accuracy.



c.

Fig 27: Validation and Training Loss.

```
[ ]  test_loss, test_acc = model.evaluate(testing_data)
     print('Test Accuracy: ', (test_acc)*100)
     print('Test Loss: ' , test_loss)

     7/7 [==============================] - 1s 108ms/step - loss: 1.3867 - accuracy: 0.2500
     Test Accuracy:  25.0
     Test Loss:  1.3867219686508179
```

d.

Fig 28: Test Accuracy and Loss.

e.

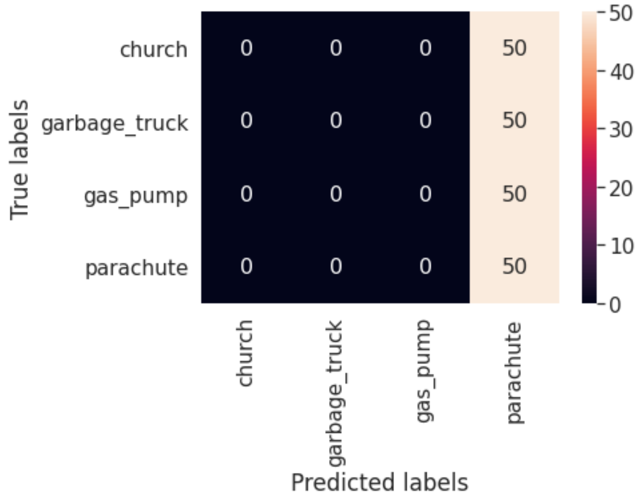


Fig 29 : Confusion Matrix.

Concluding Remarks:

The first attempt in improving the previous network led to a decrease in the validation accuracy to 50.5%, the second, to a validation accuracy of 56.25%, the third 50.78%, the fourth 44.27% and the fifth a validation accuracy of 25.26%. It can be seen that no improvement was made possible with any of the changes made in this network. It only seemed to be performing worse as the changes kept being made. Though in some of the processes the computation time was lesser, it amounted to nothing as there was no increase in the performance rate of the model. Hence, the previous model in 1a itself is a good enough network for the given dataset. Without the addition of extra dense layers, the model cannot perform any better. Links for all the five programs have been provided at the bottom section of this report.

Solution 2a:

Aim: To organize original data into proper Train/Validation/Test split before we train our network models.

The Training data was left untouched as it had enough data to train the model. The original Validation data has been split into a 60:40 ratio with 60% going to the Validation set and 40% going to the test set. The purpose of having more validation data is that it improves the model by updating the weights. More validation data leads to calculating loss function in a more optimal manner which ultimately leads to increased performance of the CNN architecture. The final Data has been stored in folder named SplitData in the Google Drive and the link for the same has been provided.

Solution 2b:

Aim: To implement a Resnet-50 based fine-tuning based transfer learning CNN architecture to optimize the Dog Breed classification task based on the new data split developed in solution 2a.

Three things had been done here. First, the pre-trained Resnet50 was loaded based on ImageWoof weights and these weights helped to extract the features. Second, the first 141 layers were frozen so that as data was passed, the weights would not get updated. Lastly, extra dense layers were added along with average pooling so as to extract the denser features from the data and the later layers that is, the res5c block were re-trained along with the additional layers. Basically, the initial part of this network was learning things that are common to lots of images like edges, corners, junctions and so on. The later layers were more application specific. After this was done, the model summary was shown and also the diagram which has been stored under the folder name 'secondquesbmodel' with file name 'resnet50_model_plot' in the google drive. The model had trained quite well and gave an accuracy in the 80s for validation data and an 81.49% testing accuracy and a loss of 0.718. The graphs of the same have been plotted below:

```
plt.plot(epochs, accuracy_train, 'g', label='Training accuracy')
plt.plot(epochs, accuracy_val, 'b', label='validation accuracy')
plt.title('Training and Validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

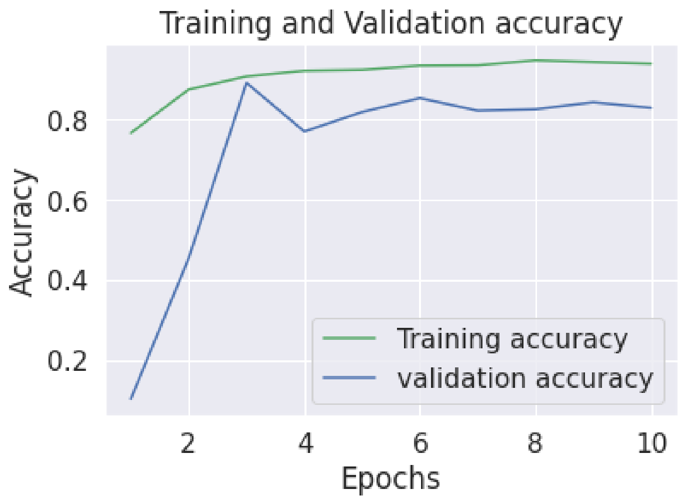


Fig 30: Training and Validation accuracy.

After initial epochs, the model stopped learning and showed an accuracy of around 80%. For training the model, Adam was chosen as the optimizer since it was best suited for this complex network and yielded a good result for the task. Use of Callbacks was also made so that our Model would train, store data and stop automatically. This has proved to be a very useful tool especially in a model with as many layers as ResNet-50. The only two callbacks used here were ModelCheckpoint -to save the model after every epoch and EarlyStopping- so that it would store the best and most accurate result after model stopped learning. Lastly, validation loss has been monitored for the purpose of plotting the model.

```
plt.plot(epochs,loss_train, label='training_loss')
plt.plot(epochs,loss_val, 'r', label='val_loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

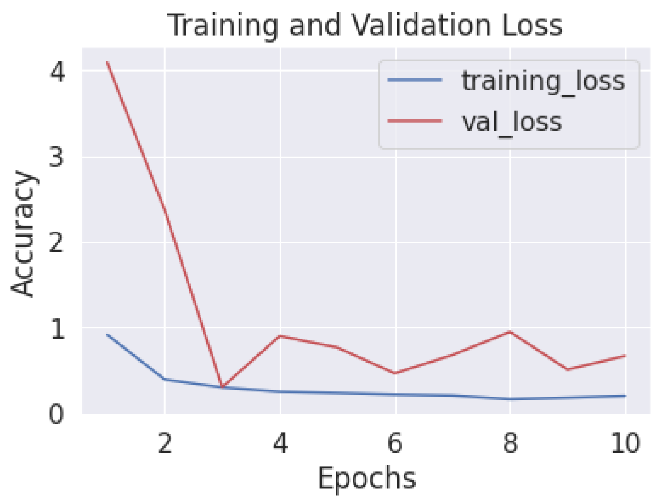


Fig 31: Training and Validation Loss.

Model shows global minima of loss at third epoch.

```
[109] print("Loss on Test Data: ", predict_evaluate[0])

      Loss on Test Data:  0.7181047201156616
```

```
[110] print('Accuracy on Test Data ', predict_evaluate[1]*100)

      Accuracy on Test Data  81.49999976158142
```

Fig 32: Testing loss and accuracy.

The final Confusion matrix was found to be as follows:

```
plt.show()
```

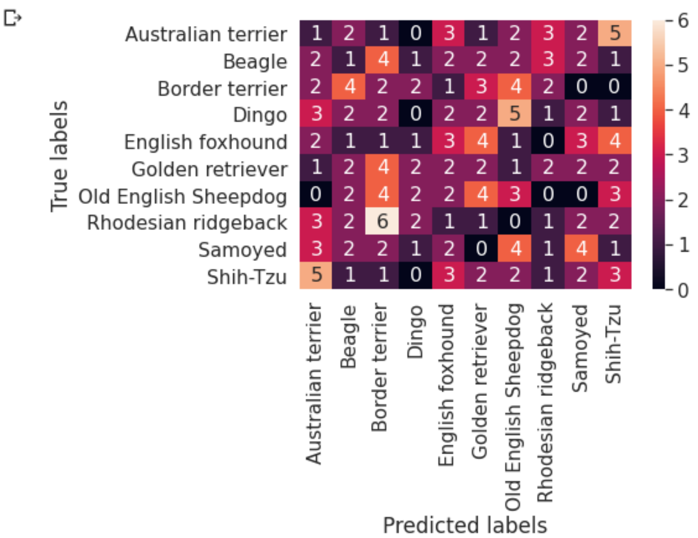


Fig 33: Confusion Matrix.

The FLOP count of the network is as follows:

```
[ ]  print("Network FLOPS: ",get_flops(model_path))

     Network FLOPS:  98247242
```

Concluding Remarks:

The ImageWoof dataset was very useful in leveraging our pre-trained network. The advantage of using pre-trained CNN is that we get a more accurate result because we are working on and improving a model which has already been built. Fine tuning gives us the advantage of being able to modify the weights of the network according to how we want it to work for our dataset. The network used here, that is, Resnet-50 has huge number of parameters and training such a network would take up a lot of data and time. Hence, fine-tuning a large network such as this is always a good option. We can see that the network performed well and gave quite a good accuracy.

Solution 2c:

Aim: To illustrate the performance of the network developed in 2b by applying it to previously unseen images of the Dog breed classes that we have acquired ourself.

The previous developed model in 2b was loaded and used. Previously unseen images were uploaded and the output came out very accurate. Summary of the model has been shown below:

```
models.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
model_3 (Functional)         (None, 10)                24641930
=================================================================
Total params: 24,641,930
Trainable params: 16,060,810
Non-trainable params: 8,581,120
_________________________________________________________________
```

Fig 34: Model Summary.

Output of the Code is as follows: Names of the Dog breeds have also been printed below the images.

Shih-Tzu

Golden retriever

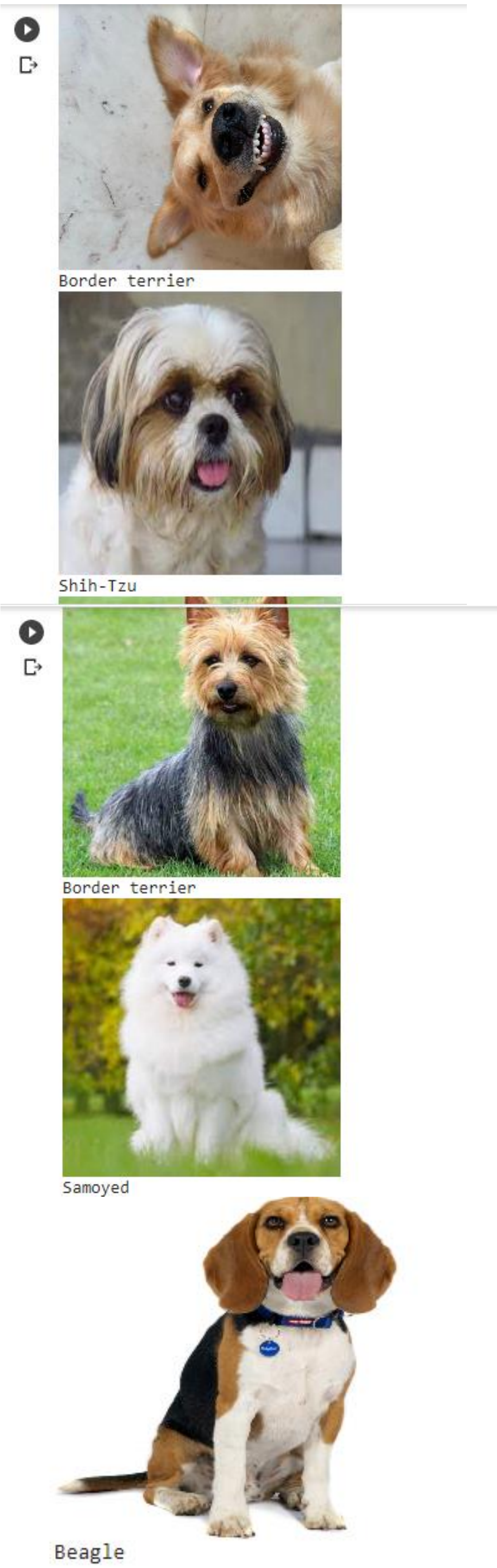


Fig 35: Output.

Concluding Remarks:

It can be seen clearly that the Model performed well for the set of previously unseen images. With the test accuracy of 81.5%, It proved to be well efficient in its functioning, Except for a wrong prediction for the third image which was a Golden Retriever. Perhaps such an image of a smiling dog had not been available in the training dataset and as a result gave a wrong prediction here. Overall, it gave correct predictions for the rest of the test images which were given.

Solution 3a:

Aim: To implement the original UNet image segmentation architecture on the given Oxford-IIIT Dataset.

The main purpose of Segmentation is to be able to mask the image. Here, the model was designed from scratch by using combinations of two convolution layers and then a maxpooling layer after every two layers. The network was composed of an encoder, a bridge and a decoder. At the Encoder, down sampling was done. All features are mapped to a single output vector in this step. The bridge is connecting the Encoder and Decoder and the Decoder was used to Up sample the feature Maps to get back the original size of the input image. Concatenation was done with the skip connection (The feature map from the encoder). This helps to give localized information which makes the segmentation possible.

IMPLEMENTATION:

Before creating the Model, required libraries have been imported. Height and Width of input were set to 256,256. Next, the paths for Training, Validation and Test data were given and then the Training Data was split into 75% Training and 25% Validation Data. This ratio was chosen keeping in mind the importance of having more training data compared to validation data as more learning would be done which would hence lead to a more accurate result.
Then, calling_images and calling_mask functions were defined to pre-process the image data and mask data respectively. Training and Validation data was Pre-Processed and then the Model was designed. The Model was compiled using the Adam optimizer which minimized the loss function. The output of the model has three classes that is, foreground, border and background. The model summary was printed and the diagram of the same has been stored in the drive under folder 'thirdques'with file name 'thirdques_a'. The Model diagram has also been stored in the same folder. The Model gave a Validation Loss of 0.441% and a Validation Accuracy of 82.687%. The graphs of the same have been plotted below:

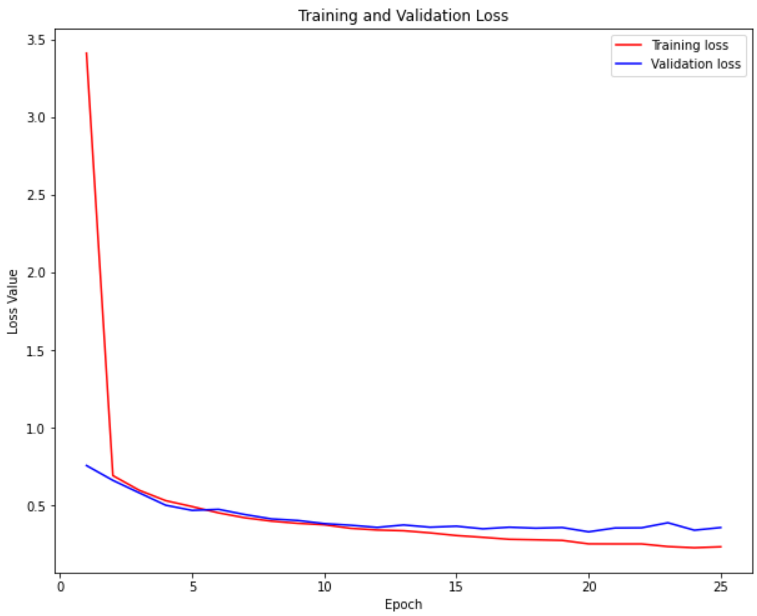


Fig 36: Training and Validation Loss.

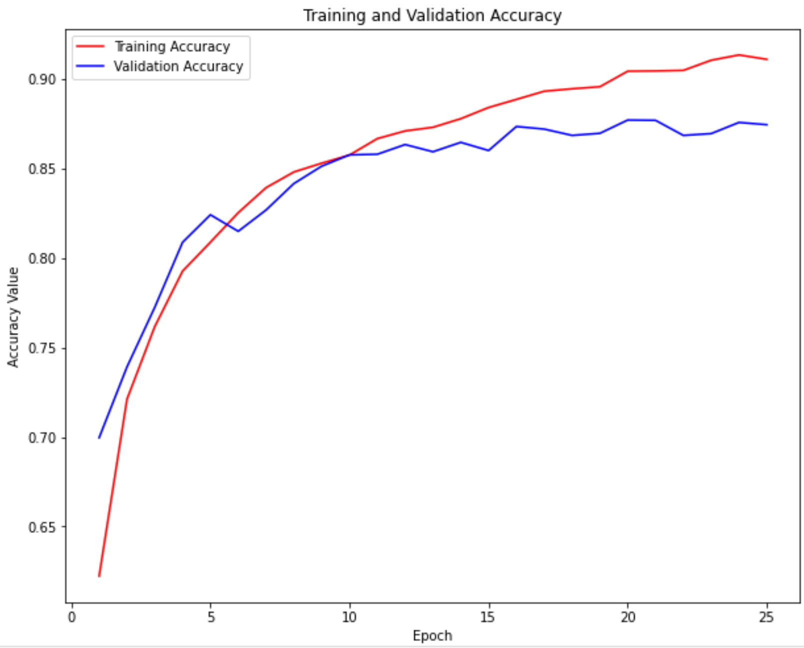


Fig 36: Training and Validation Accuracy.

The Test Accuracy was found to be 88% and Loss was 0.321% :

```
[ ] print("Test Accuracy: " ,Y_predict[1]*100)

    Test Accuracy:  88.01537752151489
```

```
[ ] print("Test Loss: " ,Y_predict[0])

    Test Loss:  0.32192081212997437
```

Fig 37: Testing Accuracy and Loss.

The Segmentation task output came out as follows:

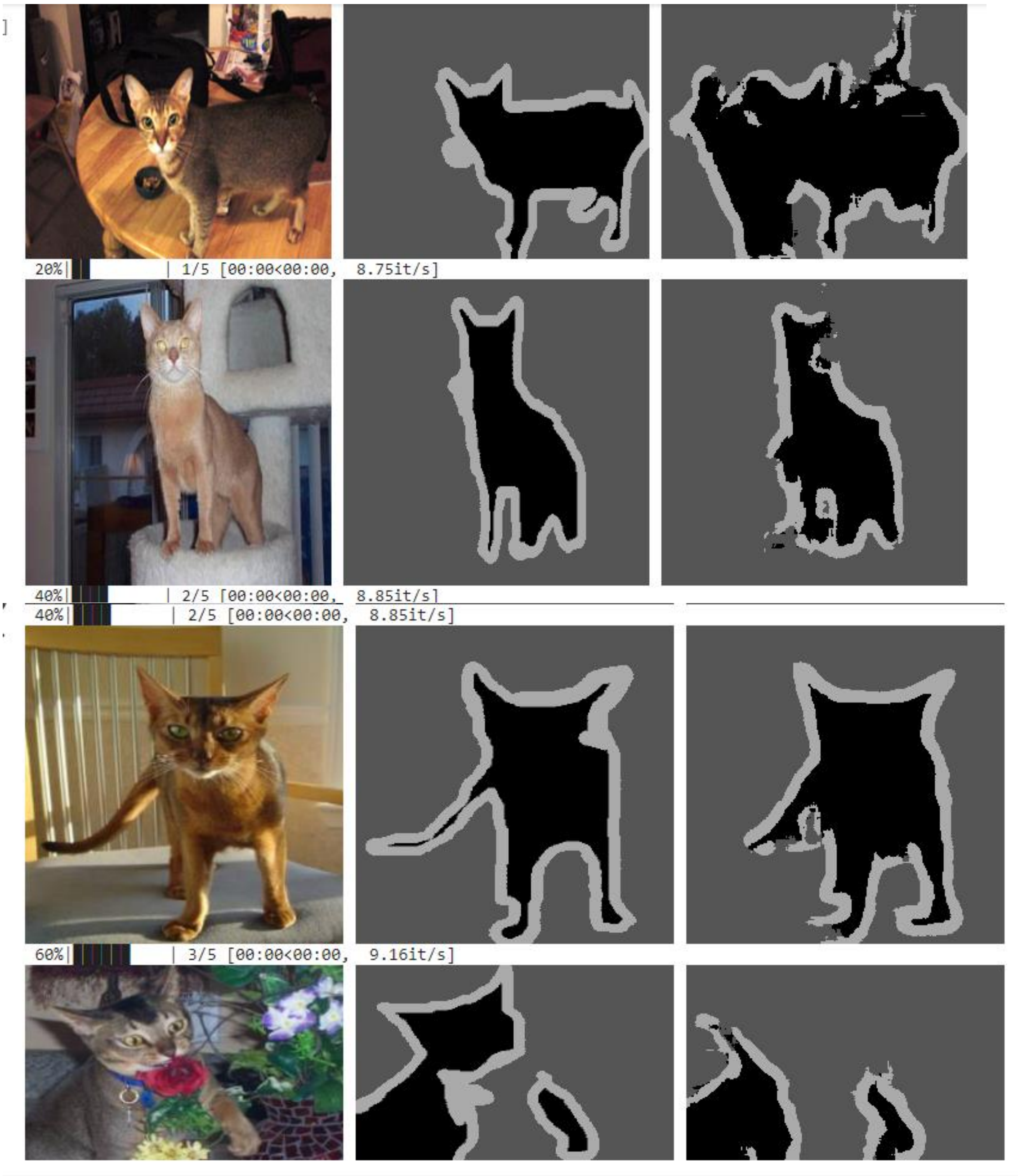


Fig 38: Segmentation Output.

The Flop count was 123658520 per second:

```
[ ]  print("Network FLOPS: ",get_flops(model_path))

     Network FLOPS:  123658520
```

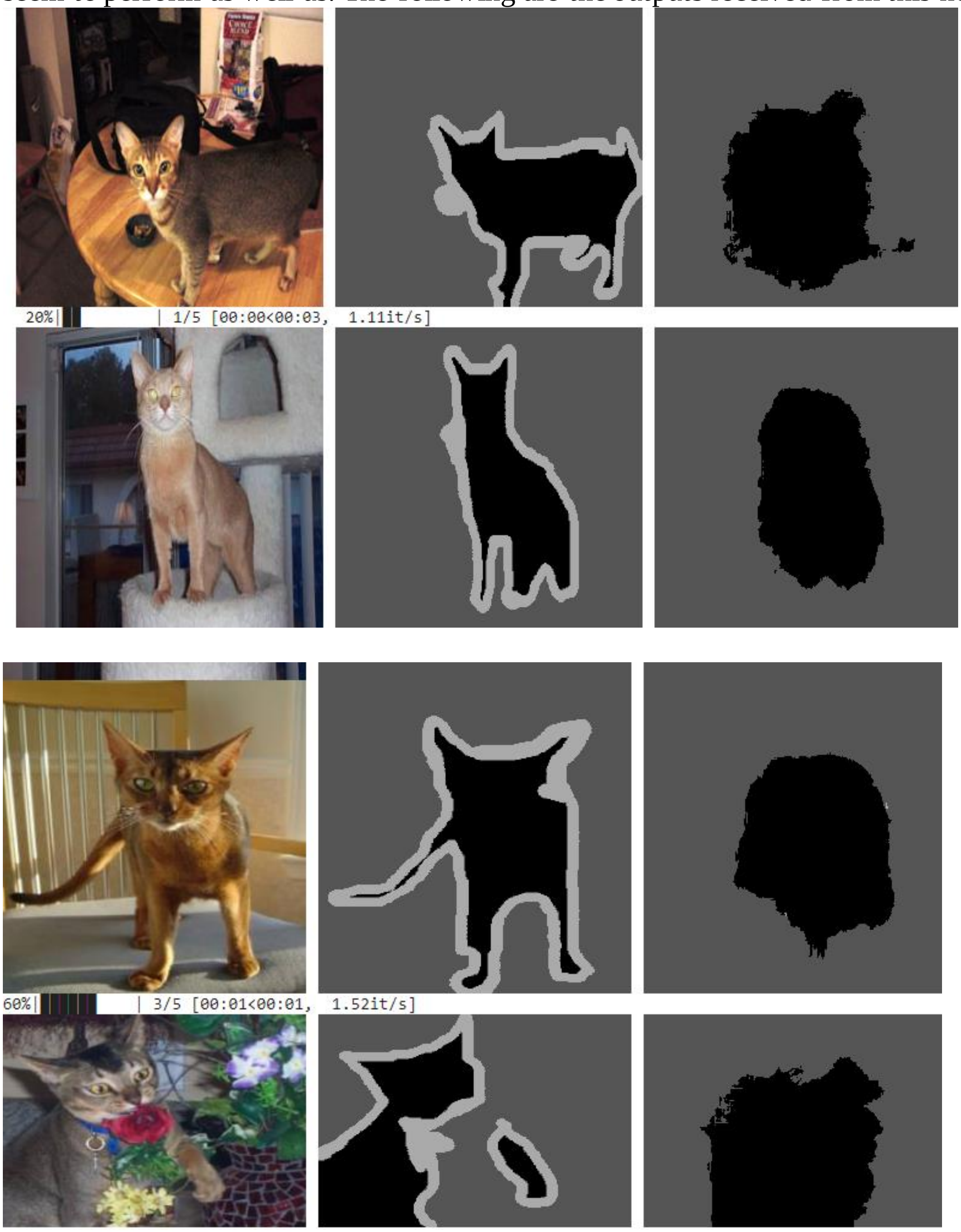
Concluding Remarks:

The Model was built and Segmentation was done. With accuracy of 80s, it proved to be a good performer and gave a well predicted image. As no Fully connected layers were used here, it performed fast. We thus created an end to end fully convolutional network and were able to perform the segmentation task on the provided dataset.

Solution 3b.

Aim: To improve Previous Network's (developed in 3a) Performance.

Augmentation was implemented as an attempt to improve the previous network's performance but it did not seem to perform as well as. The following are the outputs received from this network:



<u>Fig 39: Outputs.</u>

Concluding Remarks:

By adding augmentation such as flipping the images randomly, the model did not seem to improve. This proves that the UNet model performs better without any additional implementations.

Solution 3c.

Aim: To generate predictions for all images in the test set and unseen ‘in the wild’ images.

As the Network in 3b did not show any improvement as compared to the one in 3a, Testing the unseen images has been done on the network developed in 3a. It gave the following outputs:

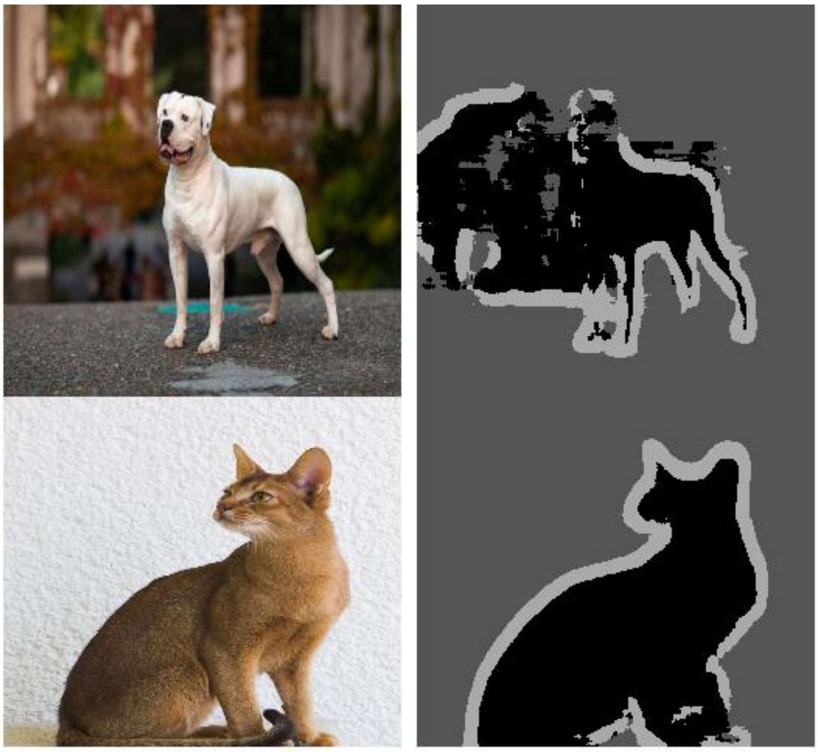


Fig 40: Final test output.

Concluding Remarks:

The UNet model was built from scratch and it was observed that it had performed well for completing its task here, that is to segment the input images. As seen from the above output, it gave quite a good output for the unseen RGB images which were fed as inputs.

Appendix:

Solution 1a Code:

```
# -*- coding: utf-8 -*-
"""firstques.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1l1Gl0o_UgpqqC6gCpjQg5JjM7OLnAoJk
"""

import keras,os
from keras.models import Sequential
from time import time
from keras.layers import Dense, Conv2D, MaxPool2D , Flatten
from keras.preprocessing.image  import ImageDataGenerator
import numpy as np
from keras.callbacks import ModelCheckpoint, EarlyStopping
```

```
from google.colab import drive
drive.mount('/content/drive')

train_data = ImageDataGenerator(rescale=1./255)
training_data =
train_data.flow_from_directory(directory="/content/drive/MyDrive/imagenette_4class/train",target_size=(2
24,224),shuffle=True)

valid_data = ImageDataGenerator(rescale=1./255)
validation_data=
valid_data.flow_from_directory(directory="/content/drive/MyDrive/imagenette_4class/validation",target_si
ze=(224,224), shuffle=True)

test_data = ImageDataGenerator(rescale=1./255)
testing_data=
test_data.flow_from_directory(directory="/content/drive/MyDrive/imagenette_4class/test",target_size=(224,
224), shuffle= True)

model = Sequential()
model.add(Conv2D(input_shape=(224,224,3),kernel_initializer='normal',filters=32,kernel_size=(3,3),paddin
g="same",dilation_rate=1,strides=(1,1), activation="relu"))
model.add(Conv2D(filters=32,kernel_size=(3,3),padding="same",dilation_rate=1, strides=(1,1),
activation="relu"))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Conv2D(filters=64,kernel_size=(3,3),padding="same",dilation_rate=1, strides=(1,1),
activation="relu"))
model.add(Conv2D(filters=64,kernel_size=(3,3),padding="same",dilation_rate=1, strides=(1,1),
activation="relu"))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Flatten())
model.add(Dense(units=512,activation="relu"))
model.add(Dense(units=4,activation="softmax"))

from keras.optimizers import SGD
learning_rate = 0.01
sgd = SGD(lr=learning_rate)
model.compile(optimizer=sgd, loss= 'categorical_crossentropy', metrics=['accuracy'])
model.summary()

from keras.utils.vis_utils import plot_model
plot_model(model, to_file='/content/drive/MyDrive/Model/Vgg16_model_plot.png', show_shapes=True,
show_layer_names=True)

callbacks = [ModelCheckpoint(filepath='/content/drive/MyDrive/Model/VGG16_model.hdf5',
monitor='val_loss', mode='min',save_best_only=True,verbose=1),
            EarlyStopping(monitor='val_loss',patience=7, mode='min')]

batch_size = 32

t0 = time()
history =
model.fit(training_data,steps_per_epoch=4800//batch_size,validation_data=validation_data,validation_steps
=400//batch_size,epochs=100, callbacks=callbacks)
```

```
print('model took', int(time() - t0), 's')

import matplotlib.pyplot as plt
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title("Validation accuracy and Accuracy")
plt.ylabel("Valid Accuracy")
plt.xlabel("Epoch")
plt.legend(["Accuracy","Validation Accuracy"])
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title("Validation Loss and loss")
plt.ylabel("Valid Loss")
plt.xlabel("Epoch")
plt.legend(["Loss","Validation Loss"])
plt.show()

print(history.history.keys())

print("Validation Loss:",history.history['val_loss'][6])

print("Validation Accuracy:",(history.history['val_accuracy'][6])*100)

test_loss, test_acc = model.evaluate(testing_data)
print('Test Accuracy: ', (test_acc)*100)
print('Test Loss: ' , test_loss)

accuracy = model.predict(testing_data)

predicted_classes = np.argmax(accuracy,axis=1)

true_classes = testing_data.classes

class_label = list(testing_data.class_indices.keys())

true_classes

import sklearn.metrics as metrics
report = metrics.classification_report(true_classes,  predicted_classes, target_names=class_label)
print(report)

cm = metrics.confusion_matrix(testing_data.classes, predicted_classes)

import seaborn as sn
import pandas as pd

df_cm = pd.DataFrame(cm, columns=np.unique(class_label),  index = np.unique(class_label))

sn.set(font_scale=1.4) # for label size
sn.heatmap(df_cm, annot=True, annot_kws={"size": 16}) # font size
plt.xlabel("Predicted labels")
```

```
plt.ylabel("True  labels")
plt.show()

import tensorflow as tf

def get_flops(model_path):
    session = tf.compat.v1.Session()
    graph = tf.compat.v1.get_default_graph()


    with graph.as_default():
        with session.as_default():
            model = tf.keras.models.load_model(model_h5_path)

            run_meta = tf.compat.v1.RunMetadata()
            opts = tf.compat.v1.profiler.ProfileOptionBuilder.float_operation()

            flops = tf.compat.v1.profiler.profile(graph=graph,
                                                  run_meta=run_meta, cmd='op', options=opts)

            return flops.total_float_ops

model_path = "/content/drive/MyDrive/Model/VGG16_model.hdf5"
tf.compat.v1.reset_default_graph()

print("Network FLOPS: ",get_flops(model_path))
```

Soultion 1b Code:

```
# -*- coding: utf-8 -*-
"""firstquesb.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1wwAxp9WPLKhQhnmEGcrfi56SZ5PyWRjP
"""

import keras,os
from keras.models import Sequential
from time import time
from keras.layers import Dense, Conv2D, MaxPool2D ,BatchNormalization, Flatten
from keras.preprocessing.image  import ImageDataGenerator
from keras.layers import Dropout
from keras import regularizers
import numpy as np
from keras.callbacks import ModelCheckpoint, EarlyStopping

from google.colab import drive
drive.mount('/content/drive')

train_data = ImageDataGenerator(rescale=1./255,
```

```
                              width_shift_range=[-150,150],
                              height_shift_range=0.25,
                              horizontal_flip=True,
                              rotation_range=0.7,
                              brightness_range=[0.2,1.0],
                              zoom_range=[0.5,1.0])
training_data =
train_data.flow_from_directory(directory="/content/drive/MyDrive/imagenette_4class/train",target_size=(2
24,224),shuffle=True)

valid_data = ImageDataGenerator(rescale=1./255)
validation_data=
valid_data.flow_from_directory(directory="/content/drive/MyDrive/imagenette_4class/validation",target_si
ze=(224,224),shuffle=True)

test_data = ImageDataGenerator(rescale=1./255)
testing_data=
test_data.flow_from_directory(directory="/content/drive/MyDrive/imagenette_4class/test",target_size=(224,
224),shuffle=True)

"""1.Dropout with Augmentation"""

model = Sequential()
model.add(Conv2D(input_shape=(224,224,3),kernel_initializer='normal',filters=32,kernel_size=(3,3),paddin
g="same",dilation_rate=1, strides=(1,1), activation="relu"))
model.add(Conv2D(filters=32,kernel_size=(3,3),padding="same",dilation_rate=1, strides=(1,1),
activation="relu"))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Conv2D(filters=64,kernel_size=(3,3),padding="same",dilation_rate=1, strides=(1,1),
activation="relu"))
model.add(Conv2D(filters=64,kernel_size=(3,3),padding="same",dilation_rate=1, strides=(1,1),
activation="relu"))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Flatten())
model.add(Dropout(0.05))
model.add(Dense(units=512, activation="relu"))
model.add(Dense(units=4,activation="softmax"))

#kernel_regularizer=keras.regularizers.l1_l2(l1=0.1, l2=0.01),

from keras.optimizers import SGD
learning_rate = 0.01
sgd = SGD(lr=learning_rate)
model.compile(optimizer=sgd, loss= 'categorical_crossentropy', metrics=['accuracy'])
model.summary()

from keras.utils.vis_utils import plot_model
plot_model(model, to_file='/content/drive/MyDrive/Model/Vgg16_model_2_dopoutwithaug_plot.png',
show_shapes=True, show_layer_names=True)

callbacks = [ModelCheckpoint(filepath='/content/drive/MyDrive/Model/VGG16_model_2_dropout.hdf5',
monitor='val_loss', mode='min',save_best_only=True,verbose=1),
          EarlyStopping(monitor='val_loss',patience=7, mode='min')]
```

```
batch_size = 32

t0 = time()
history =
model.fit(training_data,steps_per_epoch=4800//batch_size,validation_data=validation_data,validation_steps
=400//batch_size,epochs=100, callbacks=callbacks)
print('model took', int(time() - t0), 's')

import matplotlib.pyplot as plt
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title("Validation accuracy and Accuracy")
plt.ylabel("Valid Accuracy")
plt.xlabel("Epoch")
plt.legend(["Accuracy","Validation Accuracy"])
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title("Validation Loss and loss")
plt.ylabel("Valid Loss")
plt.xlabel("Epoch")
plt.legend(["Loss","Validation Loss"])
plt.show()

print(history.history.keys())

print("Validation Loss:",history.history['val_loss'][6])

print("Validation Accuracy:",(history.history['val_accuracy'][6])*100)

test_loss, test_acc = model.evaluate(testing_data)
print('Test Accuracy: ', (test_acc)*100)
print('Test Loss: ' , test_loss)

accuracy = model.predict(testing_data)

predicted_classes = np.argmax(accuracy,axis=1)

true_classes = testing_data.classes

class_label = list(testing_data.class_indices.keys())

true_classes

import sklearn.metrics as metrics
report = metrics.classification_report(true_classes,  predicted_classes, target_names=class_label)
print(report)

cm = metrics.confusion_matrix(testing_data.classes, predicted_classes)

import seaborn as sn
```

```
import pandas as pd

df_cm = pd.DataFrame(cm, columns=np.unique(class_label), index = np.unique(class_label))

sn.set(font_scale=1.4) # for label size
sn.heatmap(df_cm, annot=True, annot_kws={"size": 16}) # font size
plt.xlabel("Predicted labels")
plt.ylabel("True labels")
plt.show()

"""2.Network with BatchNoramlization and Augmentation"""

model = Sequential()
model.add(Conv2D(input_shape=(224,224,3),kernel_initializer='normal',filters=32,kernel_size=(3,3),paddin
g="same",dilation_rate=1, strides=(1,1), activation="relu"))
model.add(Conv2D(filters=32,kernel_size=(3,3),padding="same",dilation_rate=1, strides=(1,1),
activation="relu"))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Conv2D(filters=64,kernel_size=(3,3),padding="same",dilation_rate=1, strides=(1,1),
activation="relu"))
model.add(Conv2D(filters=64,kernel_size=(3,3),padding="same",dilation_rate=1, strides=(1,1),
activation="relu"))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Flatten())
model.add(BatchNormalization())
model.add(Dense(units=512, activation="relu"))
model.add(Dense(units=4,activation="softmax"))

from keras.optimizers import SGD
learning_rate = 0.01
sgd = SGD(lr=learning_rate)
model.compile(optimizer=sgd, loss= 'categorical_crossentropy', metrics=['accuracy'])
model.summary()

from keras.utils.vis_utils import plot_model
plot_model(model, to_file='/content/drive/MyDrive/Model/Vgg16_model_2_batchNoraml_plot.png',
show_shapes=True, show_layer_names=True)

callbacks = [ModelCheckpoint(filepath='/content/drive/MyDrive/Model/VGG16_model_2_Batch.hdf5',
monitor='val_loss', mode='min',save_best_only=True,verbose=1),
            EarlyStopping(monitor='val_loss',patience=7, mode='min')]

batch_size = 32

t0 = time()
history =
model.fit(training_data,steps_per_epoch=4800//batch_size,validation_data=validation_data,validation_steps
=400//batch_size,epochs=100, callbacks=callbacks)
print('model took', int(time() - t0), 's')

import matplotlib.pyplot as plt
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
```

```
plt.title("Validation accuracy and Accuracy")
plt.ylabel("Valid Accuracy")
plt.xlabel("Epoch")
plt.legend(["Accuracy","Validation Accuracy"])
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title("Validation Loss and loss")
plt.ylabel("Valid Loss")
plt.xlabel("Epoch")
plt.legend(["Loss","Validation Loss"])
plt.show()

print(history.history.keys())
predicted_classes = np.argmax(accuracy,axis=1)

print("Validation Loss:",history.history['val_loss'][6])

print("Validation Accuracy:",(history.history['val_accuracy'][6])*100)

test_loss, test_acc = model.evaluate(testing_data)
print('Test Accuracy: ', (test_acc)*100)
print('Test Loss: ' , test_loss)

true_classes = testing_data.classes
class_label = list(testing_data.class_indices.keys())

import sklearn.metrics as metrics
report = metrics.classification_report(true_classes, predicted_classes, target_names=class_label)
print(report)

cm = metrics.confusion_matrix(testing_data.classes, predicted_classes)

df_cm = pd.DataFrame(cm, columns=np.unique(class_label), index = np.unique(class_label))

sn.set(font_scale=1.4) # for label size
sn.heatmap(df_cm, annot=True, annot_kws={"size": 16}) # font size
plt.xlabel("Predicted labels")
plt.ylabel("True labels")
plt.show()

"""3.Network with Augmentation alone"""

model = Sequential()
model.add(Conv2D(input_shape=(224,224,3),kernel_initializer='normal',filters=32,kernel_size=(3,3),paddin
g="same",dilation_rate=1, strides=(1,1), activation="relu"))
model.add(Conv2D(filters=32,kernel_size=(3,3),padding="same",dilation_rate=1, strides=(1,1),
activation="relu"))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Conv2D(filters=64,kernel_size=(3,3),padding="same",dilation_rate=1, strides=(1,1),
activation="relu"))
```

```
model.add(Conv2D(filters=64,kernel_size=(3,3),padding="same",dilation_rate=1, strides=(1,1), activation="relu"))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Flatten())
model.add(Dense(units=512, activation="relu"))
model.add(Dense(units=4,activation="softmax"))

from keras.optimizers import SGD
learning_rate = 0.01
sgd = SGD(lr=learning_rate)
model.compile(optimizer=sgd, loss= 'categorical_crossentropy', metrics=['accuracy'])
model.summary()

from keras.utils.vis_utils import plot_model
plot_model(model, to_file='/content/drive/MyDrive/Model/Vgg16_model_2_Aug_plot.png', show_shapes=True, show_layer_names=True)

callbacks = [ModelCheckpoint(filepath='/content/drive/MyDrive/Model/VGG16_model_2_Aug.hdf5', monitor='val_loss', mode='min',save_best_only=True,verbose=1),
        EarlyStopping(monitor='val_loss',patience=7, mode='min')]

t0 = time()
history =
model.fit(training_data,steps_per_epoch=4800//batch_size,validation_data=validation_data,validation_steps=400//batch_size,epochs=100, callbacks=callbacks)
print('model took', int(time() - t0), 's')

import matplotlib.pyplot as plt
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title("Validation accuracy and Accuracy")
plt.ylabel("Valid Accuracy")
plt.xlabel("Epoch")
plt.legend(["Accuracy","Validation Accuracy"])
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title("Validation Loss and loss")
plt.ylabel("Valid Loss")
plt.xlabel("Epoch")
plt.legend(["Loss","Validation Loss"])
plt.show()

print("Validation Loss:",history.history['val_loss'][6])

print("Validation Accuracy:",(history.history['val_accuracy'][6])*100)

test_loss, test_acc = model.evaluate(testing_data)
print('Test Accuracy: ', (test_acc)*100)
print('Test Loss: ' , test_loss)

accuracy = model.predict(testing_data)
```

```
predicted_classes = np.argmax(accuracy,axis=1)

true_classes = testing_data.classes
class_label = list(testing_data.class_indices.keys())

import sklearn.metrics as metrics
report = metrics.classification_report(true_classes, predicted_classes, target_names=class_label)
print(report)

cm = metrics.confusion_matrix(testing_data.classes, predicted_classes)

df_cm = pd.DataFrame(cm, columns=np.unique(class_label), index = np.unique(class_label))

sn.set(font_scale=1.4) # for label size
sn.heatmap(df_cm, annot=True, annot_kws={"size": 16}) # font size
plt.xlabel("Predicted labels")
plt.ylabel("True labels")
plt.show()

"""4.Network with Regularization and Augmentation"""

model = Sequential()
model.add(Conv2D(input_shape=(224,224,3),kernel_initializer='normal',filters=32,kernel_size=(3,3),paddin
g="same",dilation_rate=1, strides=(1,1), activation="relu"))
model.add(Conv2D(filters=32,kernel_size=(3,3),padding="same",dilation_rate=1, strides=(1,1),
activation="relu"))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Conv2D(filters=64,kernel_size=(3,3),padding="same",dilation_rate=1, strides=(1,1),
activation="relu"))
model.add(Conv2D(filters=64,kernel_size=(3,3),padding="same",dilation_rate=1, strides=(1,1),
activation="relu"))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Flatten())
model.add(Dense(units=512, activation="relu", kernel_regularizer=keras.regularizers.l1_l2(l1=0.1,
l2=0.01)))
model.add(Dense(units=4,activation="softmax"))

from keras.optimizers import SGD
learning_rate = 0.01
sgd = SGD(lr=learning_rate)
model.compile(optimizer=sgd, loss= 'categorical_crossentropy', metrics=['accuracy'])
model.summary()

from keras.utils.vis_utils import plot_model
plot_model(model, to_file='/content/drive/MyDrive/Model/Vgg16_model_2_regularization_plot.png',
show_shapes=True, show_layer_names=True)

callbacks =
[ModelCheckpoint(filepath='/content/drive/MyDrive/Model/VGG16_model_2_regularization.hdf5',
monitor='val_loss', mode='min',save_best_only=True,verbose=1),
        EarlyStopping(monitor='val_loss',patience=7, mode='min')]
```

```
batch_size=32

t0 = time()
history =
model.fit(training_data,steps_per_epoch=4800//batch_size,validation_data=validation_data,validation_steps
=400//batch_size,epochs=100, callbacks=callbacks)
print('model took', int(time() - t0), 's')

import matplotlib.pyplot as plt
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title("Validation accuracy and Accuracy")
plt.ylabel("Valid Accuracy")
plt.xlabel("Epoch")
plt.legend(["Accuracy","Validation Accuracy"])
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title("Validation Loss and loss")
plt.ylabel("Valid Loss")
plt.xlabel("Epoch")
plt.legend(["Loss","Validation Loss"])
plt.show()

print("Validation Loss:",history.history['val_loss'][6])

print("Validation Accuracy:",(history.history['val_accuracy'][6])*100)

test_loss, test_acc = model.evaluate(testing_data)
print('Test Accuracy: ', (test_acc)*100)
print('Test Loss: ' , test_loss)

accuracy = model.predict(testing_data)

predicted_classes = np.argmax(accuracy,axis=1)

true_classes = testing_data.classes
class_label = list(testing_data.class_indices.keys())

import sklearn.metrics as metrics
report = metrics.classification_report(true_classes, predicted_classes, target_names=class_label)
print(report)

cm = metrics.confusion_matrix(testing_data.classes, predicted_classes)

import pandas as pd
df_cm = pd.DataFrame(cm, columns=np.unique(class_label), index = np.unique(class_label))

import seaborn as sn
sn.set(font_scale=1.4) # for label size
sn.heatmap(df_cm, annot=True, annot_kws={"size": 16}) # font size
plt.xlabel("Predicted labels")
```

```
plt.ylabel("True labels")
plt.show()

"""5.Network without EarlyStopping and Augmentation

"""

model = Sequential()
model.add(Conv2D(input_shape=(224,224,3),kernel_initializer='normal',filters=32,kernel_size=(3,3),paddin
g="same",dilation_rate=1, strides=(1,1), activation="relu"))
model.add(Conv2D(filters=32,kernel_size=(3,3),padding="same",dilation_rate=1, strides=(1,1),
activation="relu"))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Conv2D(filters=64,kernel_size=(3,3),padding="same",dilation_rate=1, strides=(1,1),
activation="relu"))
model.add(Conv2D(filters=64,kernel_size=(3,3),padding="same",dilation_rate=1, strides=(1,1),
activation="relu"))
model.add(MaxPool2D(pool_size=(2,2)))
model.add(Flatten())
model.add(Dense(units=512, activation="relu"))
#model.add(BatchNormalization())
#model.add(Dropout(0.1))
model.add(Dense(units=4,activation="softmax"))

from keras.optimizers import Adam
opt = Adam(lr=0.01)
model.compile(optimizer=opt, loss= 'categorical_crossentropy', metrics=['accuracy'])
model.summary()

from keras.utils.vis_utils import plot_model
plot_model(model,
to_file='/content/drive/MyDrive/Model/Vgg16_model_2_without_earlystopping_plot.png',
show_shapes=True, show_layer_names=True)

callbacks =
[ModelCheckpoint(filepath='/content/drive/MyDrive/Model/VGG16_model_2_without_earlystopping.hdf5',
monitor='val_loss', mode='min',save_best_only=True,verbose=1)]

batch_size=32
t0 = time()
history =
model.fit(training_data,steps_per_epoch=4800//batch_size,validation_data=validation_data,validation_steps
=400//batch_size,epochs=10, callbacks=callbacks)
print('model took', int(time() - t0), 's')

import matplotlib.pyplot as plt
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title("Validation accuracy and Accuracy")
plt.ylabel("Valid Accuracy")
plt.xlabel("Epoch")
plt.legend(["Accuracy","Validation Accuracy"])
plt.show()
```

```
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title("Validation Loss and loss")
plt.ylabel("Valid Loss")
plt.xlabel("Epoch")
plt.legend(["Loss","Validation Loss"])
plt.show()

print("Validation Loss:",history.history['val_loss'][6])

print("Validation Accuracy:",(history.history['val_accuracy'][6])*100)

test_loss, test_acc = model.evaluate(testing_data)
print('Test Accuracy: ', (test_acc)*100)
print('Test Loss: ' , test_loss)

accuracy = model.predict(testing_data)

predicted_classes = np.argmax(accuracy,axis=1)

true_classes = testing_data.classes
class_label = list(testing_data.class_indices.keys())

import sklearn.metrics as metrics
report = metrics.classification_report(true_classes, predicted_classes, target_names=class_label)
print(report)

cm = metrics.confusion_matrix(testing_data.classes, predicted_classes)

import pandas as pd
df_cm = pd.DataFrame(cm, columns=np.unique(class_label), index = np.unique(class_label))

import seaborn as sn
sn.set(font_scale=1.4) # for label size
sn.heatmap(df_cm, annot=True, annot_kws={"size": 16}) # font size
plt.xlabel("Predicted labels")
plt.ylabel("True labels")
plt.show()
```

Solution 2a Code:

```
# -*- coding: utf-8 -*-
"""secondques(a).ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/11WOIB3btNA2xznKqt7GUjEoFfhKjY-ua
"""

!pip install split-folders

import splitfolders
```

```
splitfolders.ratio("/content/drive/MyDrive/imagewoof-320/val",output
="/content/drive/MyDrive/SplitData",seed=48,ratio=(0.6,0.4))

from google.colab import drive
drive.mount('/content/drive')
```

Solution 2b Code:

```
# -*- coding: utf-8 -*-
"""secondques(b).ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1vEkBxl8H6sxty6Arfxtl1ObrU3sskPXW
"""

from google.colab import drive
drive.mount('/content/drive')

import keras,os
from keras.models import Sequential
from time import time
from keras.models import Model
from keras.optimizers import Adam
from keras.layers import Dense, Conv2D, MaxPool2D , Flatten, Dropout
from keras.layers import BatchNormalization
from keras.callbacks import ModelCheckpoint, EarlyStopping, ReduceLROnPlateau
from keras.applications.resnet50 import preprocess_input, decode_predictions
from keras.preprocessing.image  import ImageDataGenerator
from keras.layers import Input
from keras.applications.resnet50 import ResNet50, preprocess_input, decode_predictions
import numpy as np
from tensorflow.keras.layers import AveragePooling2D

breed_name = {
    'Shih-Tzu': 0,
    'Rhodesian ridgeback': 1,
    'Beagle': 2,
    'English foxhound': 3,
    'Australian terrier': 4,
    'Border terrier': 5,
    'Golden retriever': 6,
    'Old English Sheepdog' :7,
    'Samoyed': 8,
    'Dingo': 9
}

train_data = ImageDataGenerator(rescale=1./255)
training_data = train_data.flow_from_directory(directory="/content/drive/MyDrive/imagewoof-
320/train",target_size=(224,224), shuffle=True)
```

```
valid_data = ImageDataGenerator(rescale=1./255)
validation_data = valid_data.flow_from_directory(directory="/content/drive/MyDrive/SplitData/val",
target_size=(224,224), shuffle=True)

test_data = ImageDataGenerator(rescale=1./255)
testing_data = test_data.flow_from_directory(directory="/content/drive/MyDrive/SplitData/test",
target_size=(224,224), shuffle=True)

training_data.class_indices = breed_name
validation_data.class_indices = breed_name
testing_data.class_indices = breed_name

base_model = ResNet50(weights="imagenet", include_top=False,input_tensor=Input(shape=(224, 224, 3)))

for layer in base_model.layers[:142]:
  layer.trainable=False
for layer in base_model.layers[142:]:
  layer.trainable=True
for layer in base_model.layers[:]:
  if isinstance(layer, BatchNormalization):
    layer.trainable=True

base_model.summary()

for i, layer in enumerate(base_model.layers):
  print(i, layer.name, layer.trainable)

head_model = base_model.output
head_model = AveragePooling2D(pool_size=(7, 7))(head_model)
head_model = Flatten(name="flatten")(head_model)
head_model = Dense(512, activation="relu")(head_model)
head_model = Dropout(0.3)(head_model)
head_model = Dense(10, activation="softmax")(head_model)

from keras.models import load_model
model= load_model('/content/drive/MyDrive/secondquesbmodel/resnet50_2b.hdf5')
from keras.utils.vis_utils import plot_model
plot_model(model, to_file='/content/drive/MyDrive/secondquesbmodel/resnet50_model_plot.png',
show_shapes=True, show_layer_names=True)

model = Model(inputs=base_model.input, outputs=head_model)

opt = Adam(lr=0.01)
model.compile(loss="categorical_crossentropy", optimizer=opt,metrics=["accuracy"])

batch_size=32

callbacks= [ModelCheckpoint(filepath='/content/drive/MyDrive/secondquesbmodel/resnet50_2b.hdf5',
monitor='val_loss', mode="min",save_best_only=True,verbose=1),
        EarlyStopping(monitor='val_loss', mode='min',patience=7, verbose=1, restore_best_weights=True)]

t = time()
```

```
History=
model.fit(training_data,steps_per_epoch=12455//batch_size,validation_data=validation_data,validation_step
s=300//batch_size,epochs=100, callbacks=callbacks)
print('model took', int(time() - t), 's')

import matplotlib.pyplot as plt

accuracy_train = History.history['accuracy']
accuracy_val = History.history['val_accuracy']
epochs = range(1,11)
plt.plot(epochs, accuracy_train, 'g', label='Training accuracy')
plt.plot(epochs, accuracy_val, 'b', label='validation accuracy')
plt.title('Training and Validation accuracy')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

loss_train = History.history['loss']
loss_val = History.history['val_loss']
epochs = range(1,11)
plt.plot(epochs,loss_train, label='training_loss')
plt.plot(epochs,loss_val, 'r', label='val_loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.legend()
plt.show()

import seaborn as sn

from keras.models import load_model

model = load_model('/content/drive/MyDrive/secondquesbmodel/resnet50_2b.hdf5')

predict = model.predict(testing_data)

predict_evaluate = model.evaluate(testing_data)

print("Loss on Test Data: ", predict_evaluate[0])

print('Accuracy on Test Data ', predict_evaluate[1]*100)

predicted_classes = np.argmax(predict, axis=1)
true_classes = testing_data.classes
class_labels = list(testing_data.class_indices.keys())

import sklearn.metrics as metrics

report = metrics.classification_report(true_classes, predicted_classes, target_names=class_labels)
print(report)

import pandas as pd
```

```
cm = metrics.confusion_matrix(testing_data.classes, predicted_classes)

df_cm = pd.DataFrame(cm, columns=np.unique(class_labels), index = np.unique(class_labels))

sn.set(font_scale=1.4)
sn.heatmap(df_cm, annot=True, annot_kws={"size": 16})
plt.xlabel("Predicted labels")
plt.ylabel("True labels")
plt.show()

import tensorflow as tf

def get_flops(model_path):
    session = tf.compat.v1.Session()
    graph = tf.compat.v1.get_default_graph()


    with graph.as_default():
        with session.as_default():
            model = tf.keras.models.load_model(model_path)

            run_meta = tf.compat.v1.RunMetadata()
            opts = tf.compat.v1.profiler.ProfileOptionBuilder.float_operation()

            flops = tf.compat.v1.profiler.profile(graph=graph,
                                                  run_meta=run_meta, cmd='op', options=opts)

            return flops.total_float_ops

model_path = "/content/drive/MyDrive/secondquesbmodel/resnet50_2b.hdf5"
tf.compat.v1.reset_default_graph()

print("Network FLOPS: ",get_flops(model_path))
```

Solution 2c Code:

```
# -*- coding: utf-8 -*-
"""secondques_c.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1wo3oJzB3oF0JmaitxNe9tiGla8hX3DlV
"""

from google.colab import drive
drive.mount('/content/drive')

from keras.models import load_model
import cv2
import numpy as np
from keras.optimizers import Adam
```

```
import os

model = load_model('/content/drive/MyDrive/secondquesbmodel/resnet50_2b.hdf5')

from keras.models import Sequential

models = Sequential()

breed_name = {
    'Shih-Tzu': 0,
    'Rhodesian ridgeback': 1,
    'Beagle': 2,
    'English foxhound': 3,
    'Australian terrier': 4,
    'Border terrier': 5,
    'Golden retriever': 6,
    'Old English Sheepdog' :7,
    'Samoyed': 8,
    'Dingo': 9
}

models.add(model)

opt = Adam(lr=0.01)

models.compile(loss="categorical_crossentropy", optimizer="adam",metrics=["accuracy"])

models.summary()

from google.colab.patches import cv2_imshow

mydir = Path("path/to/my/dir")
for file in mydir.glob('*.mp4'):
    print(file.name)
    # do your stuff

from PIL import Image
from pathlib import Path
import numpy as np
from skimage import transform
def load(filename):
   np_image = Image.open(filename)
   np_image = np.array(np_image).astype('float32')/255
   np_image = transform.resize(np_image, (224, 224, 3))
   np_image = np.expand_dims(np_image, axis=0)
   return np_image

test_directory = Path("/content/drive/MyDrive/dogbreeds/")
for image_name in test_directory.glob('Dogs/*.jpg'):
  #print(image_name)
  image = cv2.imread(str(image_name))
  image = cv2.resize(image,(224,224))
  #print(image.shape)
```

```
  cv2_imshow(image)
  image = load(image_name)
  prediction = model.predict(image)
  image_classes  = prediction.argmax(axis=-1)
  image_classes  = np.int(image_classes)
  for dog_name, dog_number in breed_name.items():
    if image_classes == dog_number:
      print(dog_name)
```

Solution 3a and c Code:

```
# -*- coding: utf-8 -*-
"""thirdques_a,c.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1j46E3XEAScPo40AXVNDnthv4f61RJ0X_
"""

from google.colab import drive
drive.mount('/content/drive')

!tar -xvf  '/content/drive/MyDrive/oxford-iiit-pet.tgz' -C '/content/'

import os
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import tensorflow as tf
import cv2

from keras.models import Input, Model
from keras.layers import Conv2D, MaxPooling2D, Dropout,UpSampling2D, concatenate,
Conv2DTranspose
from keras.optimizers import Adam
from keras.callbacks import ModelCheckpoint, LearningRateScheduler, EarlyStopping
from keras import backend as keras
from keras.preprocessing.image  import ImageDataGenerator

Height = 256
Width = 256

data_path = '/content/oxford-iiit-pet'

train_valid_path = os.path.join(data_path, "annotations/trainval.txt")
test_path = os.path.join(data_path, "annotations/test.txt")

train_valid_df = pd.read_csv(train_valid_path, sep=" ", header= None)
train_valid_names = train_valid_df[0].values

train_images = [os.path.join(data_path, f"images/{name}.jpg")  for name in train_valid_names]

train_masks = [os.path.join(data_path, f"annotations/trimaps/{name}.png") for name in train_valid_names]
```

```
# Splitting data
train_x, valid_x = train_test_split(train_images,  test_size = 0.25, random_state = 43)
train_y, valid_y = train_test_split(train_masks,  test_size = 0.25, random_state = 43)

def calling_images(x):
  x = cv2.imread(x,  cv2.IMREAD_COLOR)
  x = cv2.resize(x,  (Width, Height))
  x = x / 255.0
  x = x.astype(np.float32)
  return x

def calling_mask(y):
  y = cv2.imread(y,  cv2.IMREAD_GRAYSCALE)
  y = cv2.resize(y,  (Width, Height))
  y = y -1
  y = y.astype(np.int32)
  return y

def preprocessing_data(x,y):
  def encode_decode(x,y):
    x= x.decode()
    y = y.decode()

    image  = calling_images(x)
    mask  = calling_mask(y)

    return image,  mask
  image,  mask  = tf.numpy_function(encode_decode, [x, y], [tf.float32, tf.int32])
  mask  = tf.one_hot(mask, 3, dtype=tf.int32)
  image.set_shape([Height,  Width, 3])
  mask.set_shape([Height,  Width, 3])

  return image,  mask

# PreProcessing  the training_data.
training_data = tf.data.Dataset.from_tensor_slices((train_x,  train_y))
training_data = training_data.shuffle(buffer_size=3000)
training_data = training_data.map(preprocessing_data)
training_data = training_data.batch(8)
training_data = training_data.repeat()
training_data = training_data.prefetch(2)

# Preprocessing  the validation data.
validation_data = tf.data.Dataset.from_tensor_slices((valid_x,  valid_y))
validation_data = validation_data.shuffle(buffer_size=3000)
validation_data = validation_data.map(preprocessing_data)
validation_data = validation_data.batch(8)
validation_data = validation_data.repeat()
validation_data = validation_data.prefetch(2)

input_size  = (256,256,3)
```

```
# Creating Unet Model
input_layer = Input(input_size)
conv_layer_1 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(input_layer)
conv_layer_1 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(conv_layer_1)
pooling_layer_1 = MaxPooling2D(pool_size=(2, 2))(conv_layer_1)
conv_layer_2 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(pooling_layer_1)
conv_layer_2 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(conv_layer_2)
pooling_layer_2 = MaxPooling2D(pool_size=(2, 2))(conv_layer_2)
conv_layer_3 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(pooling_layer_2)
conv_layer_3 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(conv_layer_3)
pooling_layer_3 = MaxPooling2D(pool_size=(2, 2))(conv_layer_3)
conv_layer_4 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(pooling_layer_3)
conv_layer_4 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(conv_layer_4)
dropout_layer_4 = Dropout(0.5)(conv_layer_4)
pooling_layer_4 = MaxPooling2D(pool_size=(2, 2))(dropout_layer_4)
conv_layer_5 = Conv2D(1024, 3, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(pooling_layer_4)
conv_layer_5 = Conv2D(1024, 3, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(conv_layer_5)
dropout_layer_5 = Dropout(0.5)(conv_layer_5)
UpSampling_layer_6 = Conv2D(512, 2, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(UpSampling2D(size = (2,2))(dropout_layer_5))
merge_layer_6 = concatenate([dropout_layer_4,UpSampling_layer_6], axis = 3 )
conv_layer_6 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(merge_layer_6)
conv_layer_6 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(conv_layer_6)
UpSampling_layer_7 = Conv2D(256, 2, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(UpSampling2D(size = (2,2))(conv_layer_6))
merge_layer_7 = concatenate([conv_layer_3,UpSampling_layer_7], axis = 3)
conv_layer_7 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(merge_layer_7)
conv_layer_7 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(conv_layer_7)

UpSampling_layer_8 = Conv2D(128, 2, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(UpSampling2D(size = (2,2))(conv_layer_7))
merge_layer_8 = concatenate([conv_layer_2,UpSampling_layer_8], axis = 3)
conv_layer_8 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(merge_layer_8)
conv_layer_8 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(conv_layer_8)

UpSampling_layer_9 = Conv2D(64, 2, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(UpSampling2D(size = (2,2))(conv_layer_8))
```

```
merge_layer_9 = concatenate([conv_layer_1,UpSampling_layer_9], axis = 3)
conv_layer_9 = Conv2D(3, 1, activation = 'softmax', padding = 'same', kernel_initializer =
'he_normal')(merge_layer_9)

model = Model(input_layer,conv_layer_9)

model.compile(optimizer='adam', loss=tf.keras.losses.CategoricalCrossentropy(), metrics=["accuracy"])

model.summary()

from keras.utils.vis_utils import plot_model
plot_model(model, to_file='/content/drive/MyDrive/thirdques/unet_model_plot.png', show_shapes=True,
show_layer_names=True)

callbacks = [
	ModelCheckpoint("/content/drive/MyDrive/thirdques/thirdques_a.hdf5", verbose=1,
save_best_only=True),
	EarlyStopping(monitor="val_loss", patience=5, verbose=1)
]

from time import time
t = time()
history = model.fit(training_data, steps_per_epoch=2944 //8, validation_data=validation_data,
validation_steps=736//8, epochs=100, callbacks=callbacks)
print('model took', int(time() - t), 's')

import matplotlib.pyplot as plt

#Visualising loss
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(1,26)

plt.figure(figsize = (10, 8))
plt.plot(epochs, loss, 'r', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss Value')
plt.legend()
plt.show()

# Visualising Accuracy.
accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']

epochs = range(1,26)

plt.figure(figsize = (10, 8))
plt.plot(epochs, accuracy, 'r', label='Training Accuracy')
plt.plot(epochs, val_accuracy, 'b', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
```

```
plt.xlabel('Epoch')
plt.ylabel('Accuracy Value')
plt.legend()
plt.show()

from keras.models import load_model

model = load_model('/content/drive/MyDrive/thirdques/thirdques_a.hdf5')

# Loading Test data.
# Loading the dataset.
test_path = os.path.join(data_path, "annotations/test.txt")

# Calling the images and mask data.
test_df = pd.read_csv(test_path, sep=" ", header=None)
test_names = test_df[0].values
test_images = [os.path.join(data_path, f"images/{test_name}.jpg") for test_name in test_names]
test_masks = [os.path.join(data_path, f"annotations/trimaps/{test_mask}.png") for test_mask in test_names]

test_data = tf.data.Dataset.from_tensor_slices((test_images, test_masks))
test_data = test_data.map(preprocessing_data)
test_data = test_data.batch(8)

Y_predict = model.evaluate(test_data)

print("Test Accuracy: " ,Y_predict[1]*100)

print("Test Loss: " ,Y_predict[0])

from tqdm import tqdm

from google.colab.patches import cv2_imshow

# Loading Test data.
# Loading the dataset.
test_path = os.path.join(data_path, "annotations/test.txt")

# Calling the images and mask data.
test_df = pd.read_csv(test_path, sep=" ", header=None)
test_names = test_df[0].values
test_images = [os.path.join(data_path, f"images/{test_name}.jpg") for test_name in test_names]
test_masks = [os.path.join(data_path, f"annotations/trimaps/{test_mask}.png") for test_mask in test_names]

for test_image, test_mask in tqdm(zip(test_images, test_masks), total=5):
    test_name = test_image.split("/")[-1]
    test_image = cv2.imread(test_image, cv2.IMREAD_COLOR)
    test_image = cv2.resize(test_image, (Width, Height))
    test_image = test_image / 255.0
    test_image = test_image.astype(np.float32)

    ## Read mask
    test_mask = cv2.imread(test_mask, cv2.IMREAD_GRAYSCALE)
    test_mask = cv2.resize(test_mask, (Width, Height))
```

```
    test_mask = test_mask - 1
    test_mask = np.expand_dims(test_mask, axis=-1)
    test_mask = test_mask * (255/3)
    test_mask = test_mask.astype(np.int32)
    test_mask = np.concatenate([test_mask, test_mask, test_mask], axis=2)

    ## Prediction
    predicition = model.predict(np.expand_dims(test_image, axis=0))[0]
    predicition = np.argmax(predicition, axis=-1)
    predicition = np.expand_dims(predicition, axis=-1)
    predicition = predicition * (255/3)
    predicition = predicition.astype(np.int32)
    predicition = np.concatenate([predicition, predicition, predicition], axis=2)

    test_image = test_image * 255.0
    test_image = test_image.astype(np.int32)

    h, w, _ = test_image.shape
    line = np.ones((h, 10, 3)) * 255

    final_image = np.concatenate([test_image, line, test_mask, line, predicition], axis=1)
    cv2_imshow(final_image)

print(history.history.keys())

print("Validation Loss:",history.history['val_loss'][6])

print("Validation Accuracy:",(history.history['val_accuracy'][6])*100)

import tensorflow as tf

def get_flops(model_path):
  session = tf.compat.v1.Session()
  graph = tf.compat.v1.get_default_graph()


  with graph.as_default():
    with session.as_default():
      model = tf.keras.models.load_model(model_path)

      run_meta = tf.compat.v1.RunMetadata()
      opts = tf.compat.v1.profiler.ProfileOptionBuilder.float_operation()

      flops = tf.compat.v1.profiler.profile(graph=graph,
                                            run_meta=run_meta, cmd='op', options=opts)

      return flops.total_float_ops

model_path = "/content/drive/MyDrive/thirdques/thirdques_a.hdf5"
tf.compat.v1.reset_default_graph()

print("Network FLOPS: ",get_flops(model_path))
```

```
"""**TESTING ON UNSEEN 'IN THE WILD IMAGES (QUESTION 3C)'**"""

from pathlib import Path

unseen_data_path = Path("/content/drive/MyDrive/dogbreeds/segm/")
for test_unseen_image in unseen_data_path.glob('*.jpg'):
    #print(test_unseen_image)
    #test_unseen_image = test_unseen_image.split("/")[-1]
    test_unseen_image = cv2.imread(str(test_unseen_image), cv2.IMREAD_COLOR)
    test_unseen_image = cv2.resize(test_unseen_image, (Width, Height))
    test_unseen_image = test_unseen_image / 255.0
    test_unseen_image = test_unseen_image.astype(np.float32)

    ## Prediction
    predicition = model.predict(np.expand_dims(test_unseen_image, axis=0))[0]
    predicition = np.argmax(predicition, axis=-1)
    predicition = np.expand_dims(predicition, axis=-1)
    predicition = predicition * (255/3)
    predicition = predicition.astype(np.int32)
    predicition = np.concatenate([predicition, predicition, predicition], axis=2)

    test_unseen_image = test_unseen_image * 255.0
    test_unseen_image = test_unseen_image.astype(np.int32)

    h, w, _ = test_unseen_image.shape
    line = np.ones((h, 10, 3)) * 255

    final_image = np.concatenate([test_unseen_image, line, predicition], axis=1)
    cv2_imshow(final_image)
```

Solution 3b Code:

```
"""thirdques_b.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1MfHPZZC0hLfmuq1SJjAnn3gPr-NtdwlH
"""

from google.colab import drive
drive.mount('/content/drive')

!tar -xvf  '/content/drive/MyDrive/oxford-iiit-pet.tgz' -C '/content/'

import os
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
import tensorflow as tf
import cv2

from keras.models import Input, Model
```

```
from keras.layers import Conv2D, MaxPooling2D, Dropout,UpSampling2D, concatenate,
Conv2DTranspose
from keras.optimizers import Adam
from keras.callbacks import ModelCheckpoint, LearningRateScheduler, EarlyStopping
from keras import backend as keras
from keras.preprocessing.image  import ImageDataGenerator

Height = 256
Width = 256

data_path = '/content/oxford-iiit-pet'

train_valid_path = os.path.join(data_path, "annotations/trainval.txt")
test_path = os.path.join(data_path, "annotations/test.txt")

train_valid_df = pd.read_csv(train_valid_path, sep=" ", header= None)
train_valid_names = train_valid_df[0].values

train_images = [os.path.join(data_path, f"images/{name}.jpg") for name in train_valid_names]

train_masks = [os.path.join(data_path, f"annotations/trimaps/{name}.png") for name in train_valid_names]

# Splitting data
train_x, valid_x = train_test_split(train_images,  test_size = 0.25, random_state = 43)
train_y, valid_y = train_test_split(train_masks,  test_size = 0.25, random_state = 43)

def calling_images(x):
  x = cv2.imread(x, cv2.IMREAD_COLOR)
  x = cv2.resize(x,  (Width, Height))
  x = x / 255.0
  x = x.astype(np.float32)
  return x

def calling_mask(y):
  y = cv2.imread(y, cv2.IMREAD_GRAYSCALE)
  y = cv2.resize(y,  (Width, Height))
  y = y -1
  y = y.astype(np.int32)
  return y

def preprocessing_data(x,y):
  def encode_decode(x,y):
    x= x.decode()
    y = y.decode()

image = calling_images(x)
    mask = calling_mask(y)

return image, mask
  image, mask = tf.numpy_function(encode_decode, [x, y], [tf.float32, tf.int32])
  mask = tf.one_hot(mask, 3, dtype=tf.int32)
  image.set_shape([Height,  Width, 3])
  mask.set_shape([Height,  Width, 3])
```

```
return image, mask

# PreProcessing the training_data.
training_data = tf.data.Dataset.from_tensor_slices((train_x, train_y))
training_data = training_data.shuffle(buffer_size=3000)
training_data = training_data.map(preprocessing_data)
training_data = training_data.batch(8)
training_data = training_data.repeat()
training_data = training_data.prefetch(2)

# Preprocessing the validation data.
validation_data = tf.data.Dataset.from_tensor_slices((valid_x, valid_y))
validation_data = validation_data.shuffle(buffer_size=3000)
validation_data = validation_data.map(preprocessing_data)
validation_data = validation_data.batch(8)
validation_data = validation_data.repeat()
validation_data = validation_data.prefetch(2)

input_size = (256,256,3)

# Creating Unet Model
input_layer = Input(input_size)
conv_layer_1 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(input_layer)
conv_layer_1 = Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(conv_layer_1)
pooling_layer_1 = MaxPooling2D(pool_size=(2, 2))(conv_layer_1)
conv_layer_2 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(pooling_layer_1)
conv_layer_2 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(conv_layer_2)
pooling_layer_2 = MaxPooling2D(pool_size=(2, 2))(conv_layer_2)
conv_layer_3 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(pooling_layer_2)
conv_layer_3 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(conv_layer_3)
pooling_layer_3 = MaxPooling2D(pool_size=(2, 2))(conv_layer_3)
conv_layer_4 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(pooling_layer_3)
conv_layer_4 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(conv_layer_4)
dropout_layer_4 = Dropout(0.5)(conv_layer_4)
pooling_layer_4 = MaxPooling2D(pool_size=(2, 2))(dropout_layer_4)
conv_layer_5 = Conv2D(1024, 3, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(pooling_layer_4)
conv_layer_5 = Conv2D(1024, 3, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(conv_layer_5)
dropout_layer_5 = Dropout(0.5)(conv_layer_5)
UpSampling_layer_6 = Conv2D(512, 2, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(UpSampling2D(size = (2,2))(dropout_layer_5))
merge_layer_6 = concatenate([dropout_layer_4,UpSampling_layer_6], axis = 3 )
```

```
conv_layer_6 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(merge_layer_6)
conv_layer_6 = Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(conv_layer_6)
UpSampling_layer_7 = Conv2D(256, 2, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(UpSampling2D(size = (2,2))(conv_layer_6))
merge_layer_7 = concatenate([conv_layer_3,UpSampling_layer_7], axis = 3)
conv_layer_7 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(merge_layer_7)
conv_layer_7 = Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(conv_layer_7)

UpSampling_layer_8 = Conv2D(128, 2, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(UpSampling2D(size = (2,2))(conv_layer_7))
merge_layer_8 = concatenate([conv_layer_2,UpSampling_layer_8], axis = 3)
conv_layer_8 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(merge_layer_8)
conv_layer_8 = Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(conv_layer_8)

UpSampling_layer_9 = Conv2D(64, 2, activation = 'relu', padding = 'same', kernel_initializer =
'he_normal')(UpSampling2D(size = (2,2))(conv_layer_8))
merge_layer_9 = concatenate([conv_layer_1,UpSampling_layer_9], axis = 3)
conv_layer_9 = Conv2D(3, 1, activation = 'softmax', padding = 'same', kernel_initializer =
'he_normal')(merge_layer_9)

model = Model(input_layer,conv_layer_9)

model.compile(optimizer='adam', loss=tf.keras.losses.CategoricalCrossentropy(), metrics=["accuracy"])

model.summary()

from keras.utils.vis_utils import plot_model
plot_model(model, to_file='/content/drive/MyDrive/thirdques/unet_model_b_plot.png', show_shapes=True,
show_layer_names=True)

callbacks = [
	ModelCheckpoint("/content/drive/MyDrive/thirdques/thirdques_b.hdf5", verbose=1,
save_best_only=True),
	EarlyStopping(monitor="val_loss", patience=5, verbose=1)
]

from time import time
t = time()
history = model.fit(training_data, steps_per_epoch=2944 //8, validation_data=validation_data,
validation_steps=736//8, epochs=100, callbacks=callbacks)
print('model took', int(time() - t), 's')

import matplotlib.pyplot as plt

#Visualising loss
loss = history.history['loss']
val_loss = history.history['val_loss']
```

```
epochs = range(1,26)

plt.figure(figsize = (10, 8))
plt.plot(epochs, loss, 'r', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and Validation Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss Value')
plt.legend()
plt.show()

# Visualising Accuracy.
accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']

epochs = range(1,26)

plt.figure(figsize = (10, 8))
plt.plot(epochs, accuracy, 'r', label='Training Accuracy')
plt.plot(epochs, val_accuracy, 'b', label='Validation Accuracy')
plt.title('Training and Validation Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy Value')
plt.legend()
plt.show()

from tqdm import tqdm

from google.colab.patches import cv2_imshow

# Loading Test data.
# Loading the dataset.
test_path = os.path.join(data_path, "annotations/test.txt")

# Calling the images and mask data.
test_df = pd.read_csv(test_path, sep=" ", header=None)
test_names = test_df[0].values
test_images = [os.path.join(data_path, f"images/{test_name}.jpg") for test_name in test_names]
test_masks = [os.path.join(data_path, f"annotations/trimaps/{test_mask}.png") for test_mask in test_names]

for test_image, test_mask in tqdm(zip(test_images, test_masks), total=5):
    test_name = test_image.split("/")[-1]
    test_image = cv2.imread(test_image, cv2.IMREAD_COLOR)
    test_image = cv2.resize(test_image, (Width, Height))
    test_image = test_image / 255.0
    test_image = test_image.astype(np.float32)

    ## Read mask
    test_mask = cv2.imread(test_mask, cv2.IMREAD_GRAYSCALE)
    test_mask = cv2.resize(test_mask, (Width, Height))
    test_mask = test_mask - 1
    test_mask = np.expand_dims(test_mask, axis=-1)
```

```
    test_mask = test_mask * (255/3)
    test_mask = test_mask.astype(np.int32)
    test_mask = np.concatenate([test_mask, test_mask, test_mask], axis=2)

    ## Prediction
    predicition = model.predict(np.expand_dims(test_image, axis=0))[0]
    predicition = np.argmax(predicition, axis=-1)
    predicition = np.expand_dims(predicition, axis=-1)
    predicition = predicition * (255/3)
    predicition = predicition.astype(np.int32)
    predicition = np.concatenate([predicition, predicition, predicition], axis=2)

    test_image = test_image * 255.0
    test_image = test_image.astype(np.int32)

    h, w, _ = test_image.shape
    line = np.ones((h, 10, 3)) * 255

    final_image = np.concatenate([test_image, line, test_mask, line, predicition], axis=1)
    cv2_imshow(final_image)

print(history.history.keys())

print("Validation Loss:",history.history['val_loss'][6])

print("Validation Accuracy:",(history.history['val_accuracy'][6])*100)
```

Links for all the Code files:
1a. https://colab.research.google.com/drive/1l1Gl0o_UgpqqC6gCpjQg5JjM7OLnAoJk?usp=sharing

1b. https://colab.research.google.com/drive/1wwAxp9WPLKhQhnmEGcrfi56SZ5PyWRjP?usp=sharing

2a. https://colab.research.google.com/drive/11WOIB3btNA2xznKqt7GUjEoFfhKjY-ua?usp=sharing

2b. https://colab.research.google.com/drive/1vEkBxl8H6sxty6Arfxtl1ObrU3sskPXW?usp=sharing

2c. https://colab.research.google.com/drive/1wo3oJzB3oF0JmaitxNe9tiGla8hX3DlV?usp=sharing

3a and c. https://colab.research.google.com/drive/1j46E3XEAScPo40AXVNDnthv4f61RJ0X_?usp=sharing
3b. https://colab.research.google.com/drive/1MfHPZZC0hLfmuq1SJjAnn3gPr-NtdwlH?usp=sharing

Links for all the Diagram files:

Solution 1:
https://drive.google.com/drive/folders/1MHNWubQB8g9WQQgjoaBtS69iVpOwVSlR?usp=sharing

Solution 2:
https://drive.google.com/drive/folders/1qX9NeGYBslAkApEr64t5EEl5WxmQTVJd?usp=sharing

Solution 3:

https://drive.google.com/drive/folders/1Kb-Vj_79EQxgGYzLetaNsHL4JatxczUl?usp=sharing

Splitted Data in Question 2. : https://drive.google.com/drive/folders/1WoigM81N-S0XuX8v8zYIgZoiiJ08KN8n?usp=sharing

Unseen Images for question 2 and 3:
https://drive.google.com/drive/folders/1oAkB_xaFq5oVSTvwfnwVkVaQ7QGh6QmY?usp=sharing