Problem Code:

202425ODD-UCS749-SESS-LE1-0911

Problem Title:

Recognise My Voice Commands

Name – Tanisha Maheshwary

Group – 4CO28

Roll no – 102153037

INDEX

INTRODUCTION:

This project begins with a concise summary of a research paper relevant to voice command recognition. Following this, the given dataset is analyzed, visualized, and used to train a baseline voice command classifier. The project then explores model accuracy and fine-tuning the classifier to perform on personalized voice data, followed by implementing checksum verification for asset integrity.

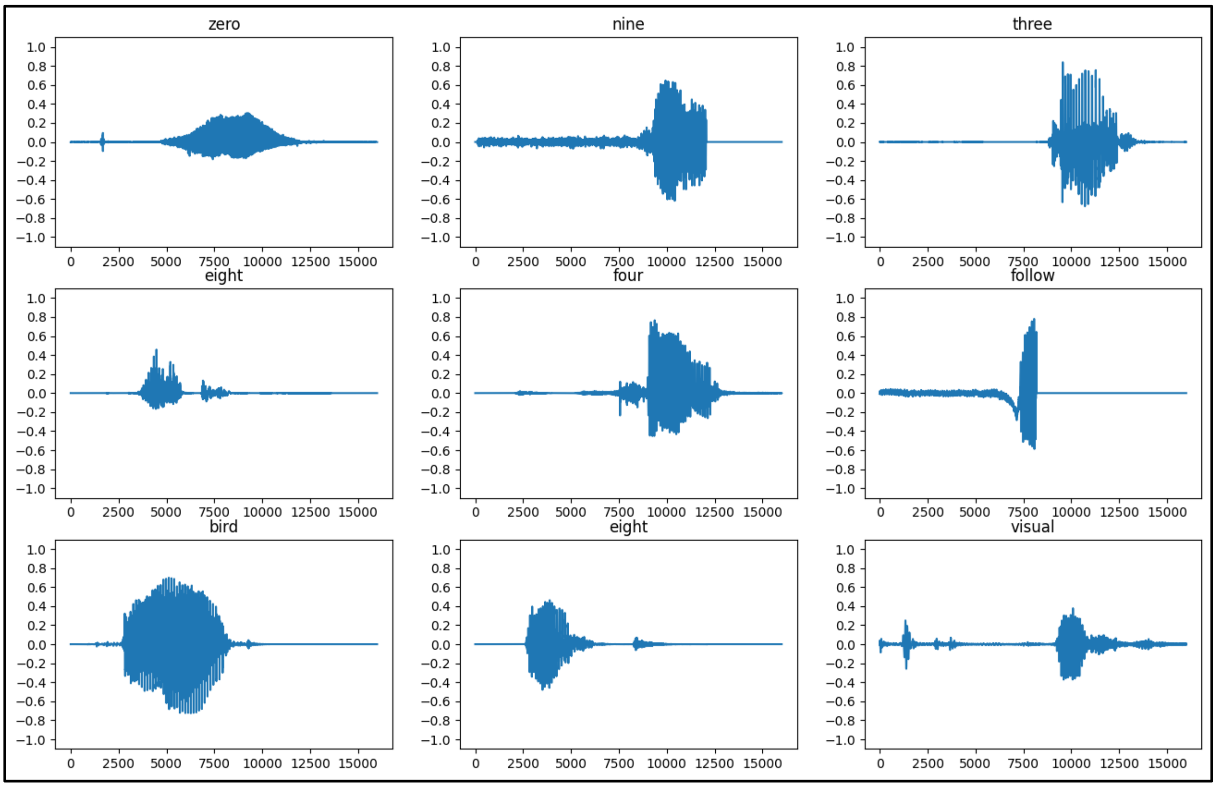
DRIVE LINK FOR COLAB NOTEBOOK:

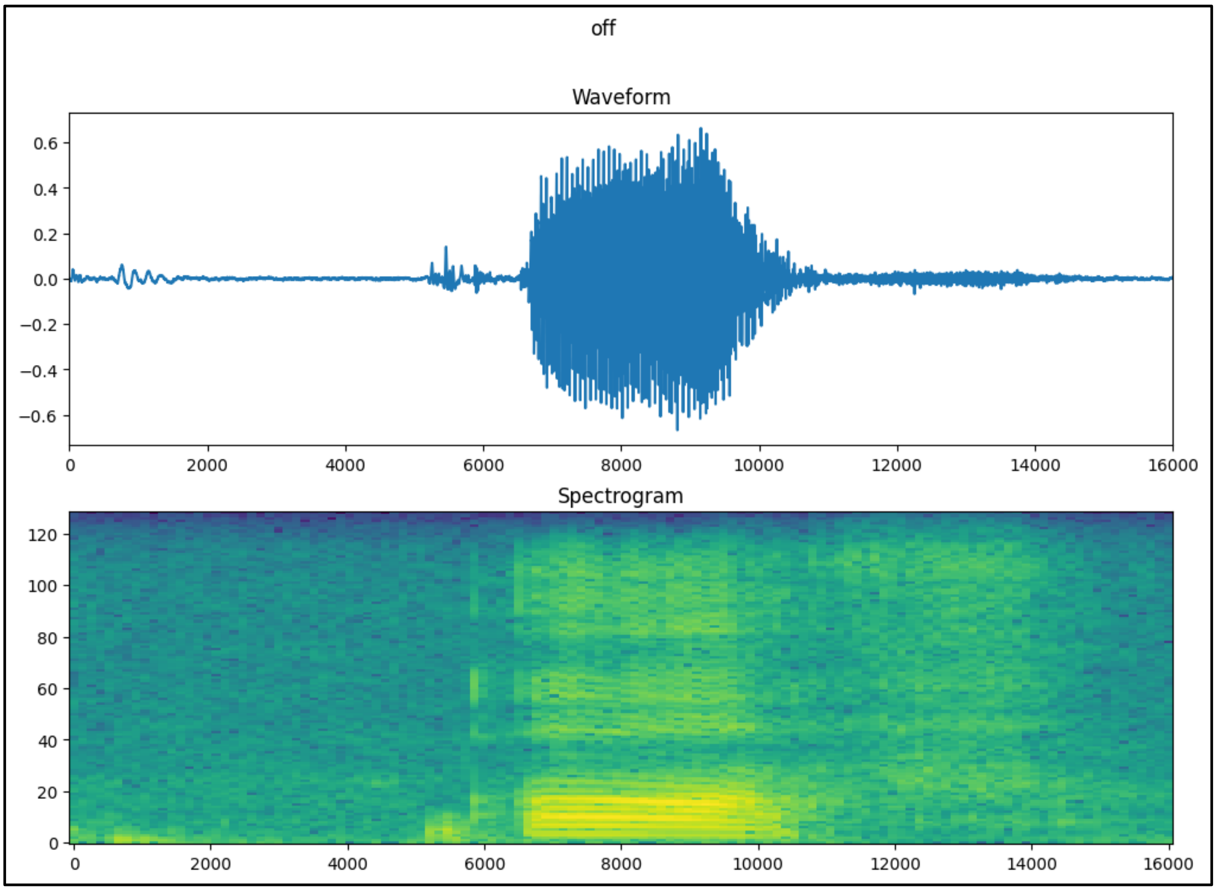
https://drive.google.com/drive/folders/1FVtGT7I6ByjjR4tECTlZECQzm8Pw6Qx-?usp=sharing

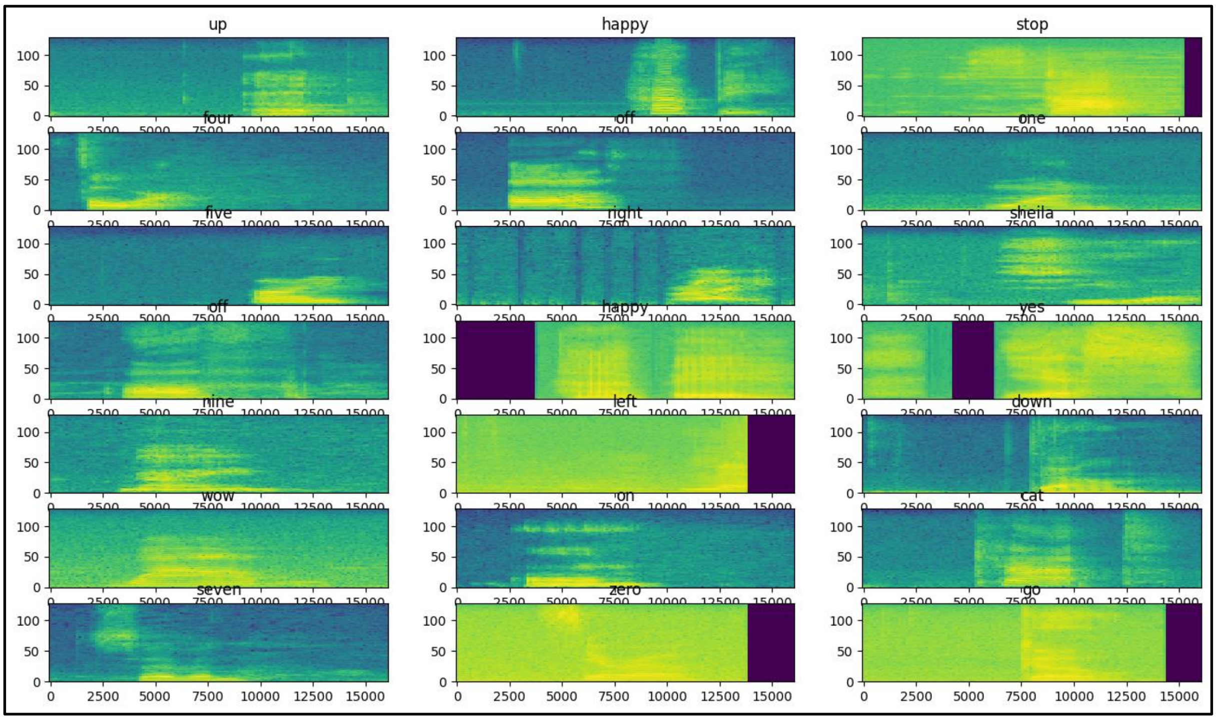
DESCRIPTION OF PAPER IN 50 WORDS:

The paper introduces the "Speech Commands" dataset, which is designed for improving keyword spotting and speech recognition. It contains 65,000 audio recordings from over 2,600 speakers, covering 12 different keywords. The dataset includes diverse speech samples from various environments and accents, making it useful for training models to recognize specific words in real-world conditions. It is openly available for research, helping researchers build and test their speech recognition systems more effectively.

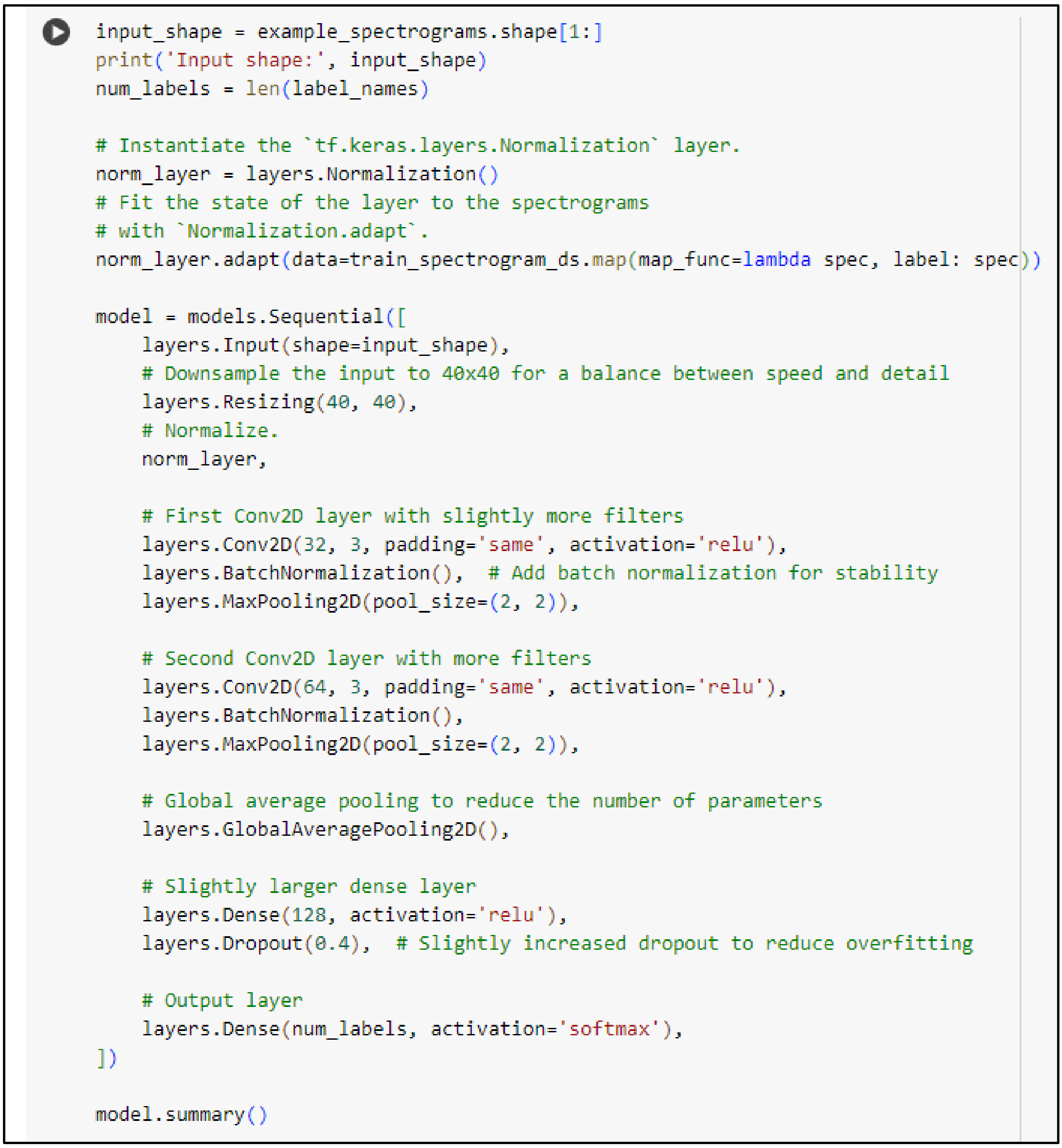
VISUALIZATIONS:



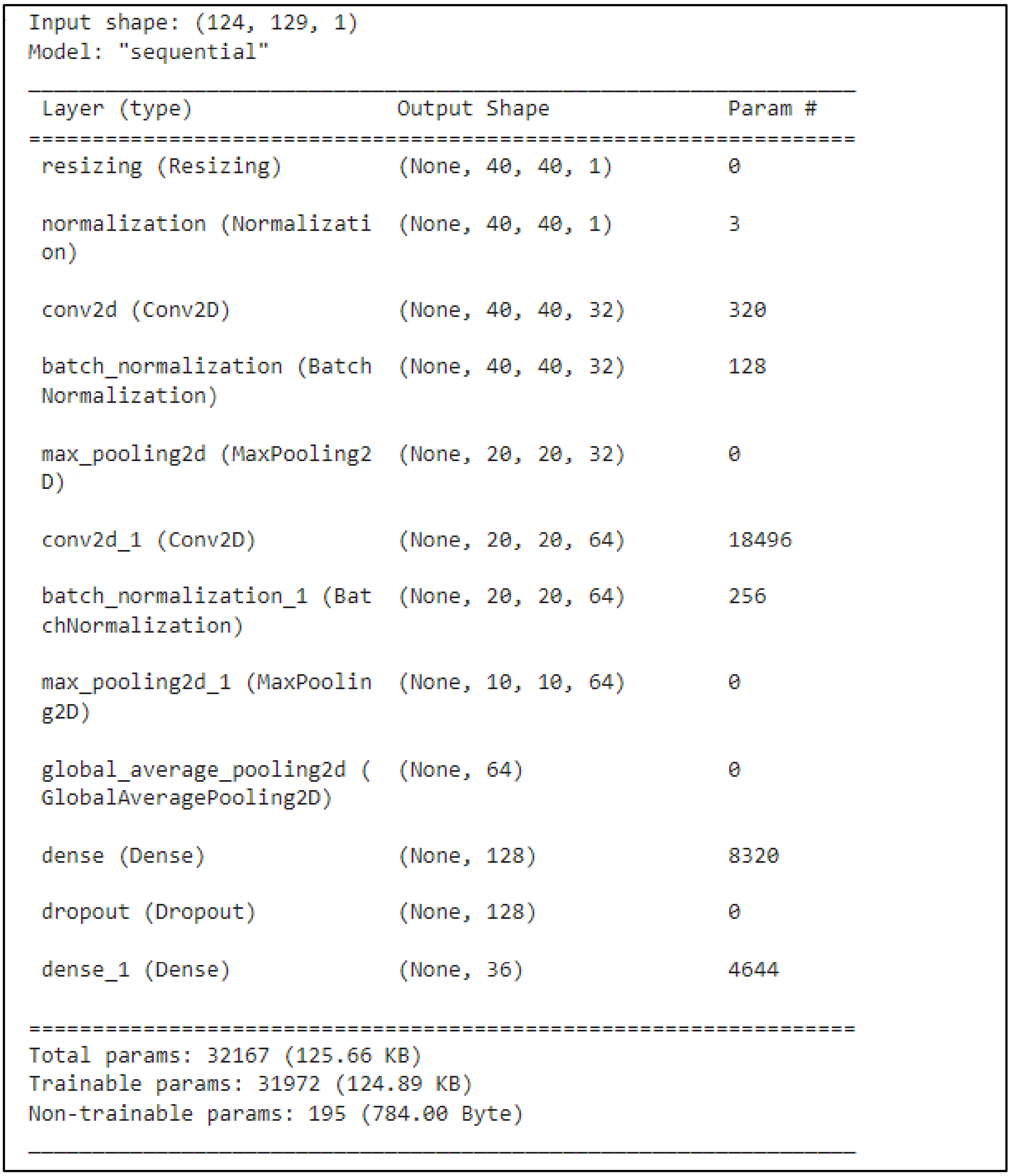



up
happy
stop
four
off
one
five
right
sheila
off
happy
yes
nine
left
down
wow
on
cat
seven
zero
go
0
50
100
0
2500
5000
7500
10000
12500
15000

MODEL:

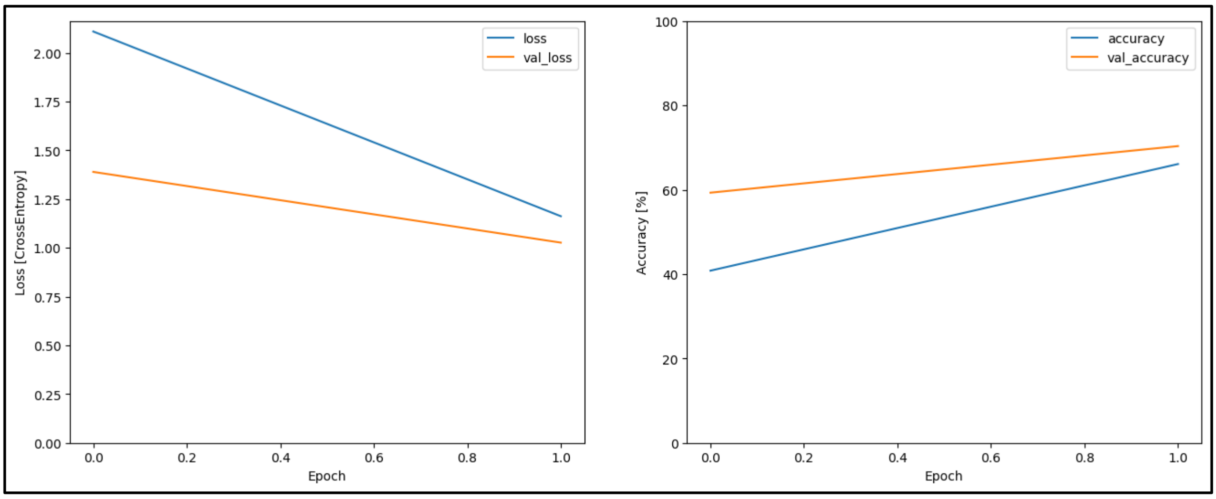


```
input_shape = example_spectrograms.shape[1:]
print('Input shape:', input_shape)
num_labels = len(label_names)

# Instantiate the `tf.keras.layers.Normalization` layer.
norm_layer = layers.Normalization()
# Fit the state of the layer to the spectrograms
# with `Normalization.adapt`.
norm_layer.adapt(data=train_spectrogram_ds.map(map_func=lambda spec, label: spec))

model = models.Sequential([
    layers.Input(shape=input_shape),
    # Downsample the input to 40x40 for a balance between speed and detail
    layers.Resizing(40, 40),
    # Normalize.
    norm_layer,

    # First Conv2D layer with slightly more filters
    layers.Conv2D(32, 3, padding='same', activation='relu'),
    layers.BatchNormalization(),  # Add batch normalization for stability
    layers.MaxPooling2D(pool_size=(2, 2)),

    # Second Conv2D layer with more filters
    layers.Conv2D(64, 3, padding='same', activation='relu'),
    layers.BatchNormalization(),
    layers.MaxPooling2D(pool_size=(2, 2)),

    # Global average pooling to reduce the number of parameters
    layers.GlobalAveragePooling2D(),

    # Slightly larger dense layer
    layers.Dense(128, activation='relu'),
    layers.Dropout(0.4),  # Slightly increased dropout to reduce overfitting

    # Output layer
    layers.Dense(num_labels, activation='softmax'),
])

model.summary()
```

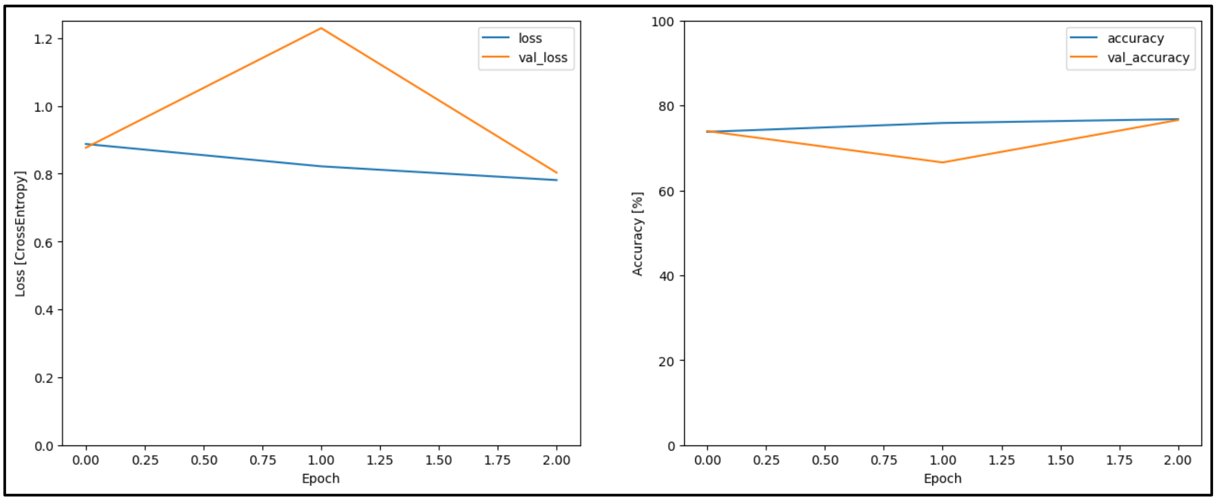


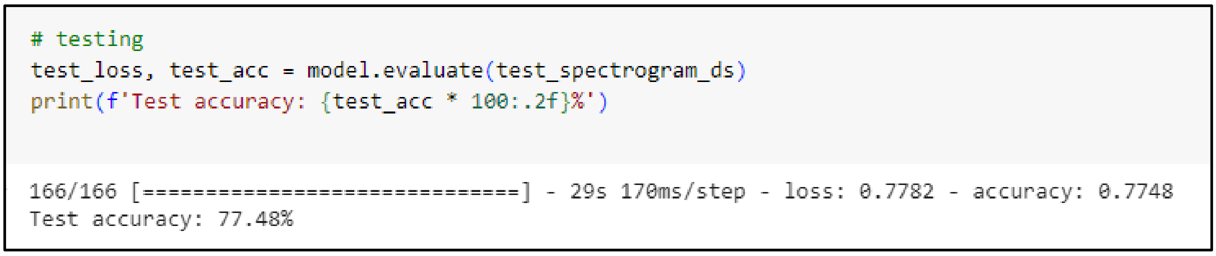
```
Input shape: (124, 129, 1)
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 resizing (Resizing)         (None, 40, 40, 1)         0         
                                                                 
 normalization (Normalizati  (None, 40, 40, 1)         3         
 on)                                                             
                                                                 
 conv2d (Conv2D)             (None, 40, 40, 32)        320       
                                                                 
 batch_normalization (Batch  (None, 40, 40, 32)        128       
 Normalization)                                                  
                                                                 
 max_pooling2d (MaxPooling2  (None, 20, 20, 32)        0         
 D)                                                              
                                                                 
 conv2d_1 (Conv2D)           (None, 20, 20, 64)        18496     
                                                                 
 batch_normalization_1 (Bat  (None, 20, 20, 64)        256       
 chNormalization)                                                
                                                                 
 max_pooling2d_1 (MaxPoolin  (None, 10, 10, 64)        0         
 g2D)                                                            
                                                                 
 global_average_pooling2d (  (None, 64)                0         
 GlobalAveragePooling2D)                                         
                                                                 
 dense (Dense)               (None, 128)               8320      
                                                                 
 dropout (Dropout)           (None, 128)               0         
                                                                 
 dense_1 (Dense)             (None, 36)                4644      
                                                                 
=================================================================
Total params: 32167 (125.66 KB)
Trainable params: 31972 (124.89 KB)
Non-trainable params: 195 (784.00 Byte)
_________________________________________________________________
```

MODEL ACCURACY:



FINE TUNING ACCURACY:

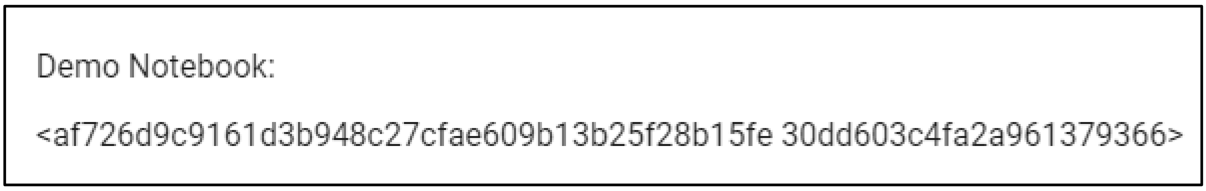




```
# testing
test_loss, test_acc = model.evaluate(test_spectrogram_ds)
print(f'Test accuracy: {test_acc * 100:.2f}%')
```

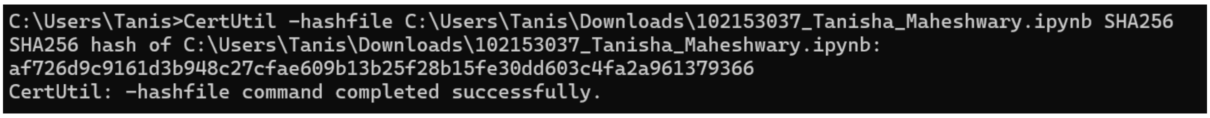
```
166/166 [==============================] - 29s 170ms/step - loss: 0.7782 - accuracy: 0.7748
Test accuracy: 77.48%
```

CHECKSUM:



Demo Notebook:

<af726d9c9161d3b948c27cfae609b13b25f28b15fe 30dd603c4fa2a961379366>



```
C:\Users\Tanis>CertUtil -hashfile C:\Users\Tanis\Downloads\102153037_Tanisha_Maheshwary.ipynb SHA256
SHA256 hash of C:\Users\Tanis\Downloads\102153037_Tanisha_Maheshwary.ipynb:
af726d9c9161d3b948c27cfae609b13b25f28b15fe30dd603c4fa2a961379366
CertUtil: -hashfile command completed successfully.
```