```
In [3]: !pip install -U -q tensorflow tensorflow_datasets
        #!apt install --allow-change-held-packages libcudnn8=8.1.0.77-1+cuda11.2
```

Import the necessary packages

```
In [4]: import os
        import pathlib

        import matplotlib.pyplot as plt
        import numpy as np
        import seaborn as sns
        from numpy import random
        import shutil

        import tensorflow as tf

        from tensorflow.keras import layers
        from tensorflow.keras import models
```

```
WARNING:tensorflow:From C:\Users\mfarag\AppData\Roaming\Python\Python39\site-packages\ke
ras\src\losses.py:2976: The name tf.losses.sparse_softmax_cross_entropy is deprecated. P
lease use tf.compat.v1.losses.sparse_softmax_cross_entropy instead.
```

Download the speech commands dataset

```
In [7]: DATASET_PATH = 'data/'

        data_dir = pathlib.Path(DATASET_PATH)
        if not data_dir.exists():
            tf.keras.utils.get_file(
              'speech_commands.zip',
              origin='http://download.tensorflow.org/data/speech_commands_v0.02.tar.gz',
              extract=True,
              cache_dir='.', cache_subdir='data')
```

Get different class labels

```
In [8]: commands = np.array(tf.io.gfile.listdir(str('data')))
        commands = commands[commands != 'README.md']
        print('Commands:', commands)
```

```
Commands: ['.DS_Store' 'backward' 'bed' 'bird' 'cat' 'dog' 'down' 'eight' 'five'
 'follow' 'forward' 'four' 'go' 'happy' 'house' 'learn' 'left' 'LICENSE'
 'marvin' 'mydataset' 'nine' 'no' 'off' 'on' 'one' 'right' 'seven'
 'sheila' 'six' 'speech_commands.zip' 'stop' 'testing_list.txt' 'three'
 'tree' 'two' 'up' 'validation_list.txt' 'visual' 'wow' 'yes' 'zero'
 '_background_noise_']
```

Create the directories for on, off, others and silent data

```
In [ ]: try:
            os.mkdir('data/mydataset')
        except Exception as e:
            print("Error creating folder. Error details {}".format(e))
```

In [13]:
```python
import shutil, errno

def copy_data(src, dst):
    try:
        shutil.copytree(src, dst)
    except OSError as exc: # python >2.5
        if exc.errno in (errno.ENOTDIR, errno.EINVAL):
            shutil.copy(src, dst)
        else: raise
```

In [15]:
```python
copy_data('data/on','data/mydataset/on')
copy_data('data/off','data/mydataset/off')
```

In [11]:
```python
# You may use the following code instead of calling the copy_data() function above if yo
#!cp -r 'data/on' 'data/mydataset'
#!cp -r 'data/off' 'data/mydataset'
```

```
'cp' is not recognized as an internal or external command,
operable program or batch file.
'cp' is not recognized as an internal or external command,
operable program or batch file.
```

In [16]:
```python
os.mkdir('data/mydataset/silent')
os.mkdir('data/mydataset/others')
```

In [17]:
```python
other_labels = ["yes", "no", "up", "down", "left", "right","bed", "bird", "cat", "dog",
sample_per_label = 150


for label in other_labels:
    path = 'data/'+label
    f = os.listdir(path)
    num_files = len(f)
    file_indx = np.arange(num_files)
    random.shuffle(file_indx)
    for i in range(sample_per_label):
        index = file_indx[i]
        file_name = f[index]
        source = path + '/' + file_name
        destination = 'data/mydataset/others/' + file_name
        # copy only files
        if os.path.isfile(source):
            shutil.copy(source, destination)
```

Create silent data

In [18]:
```python
import scipy
from scipy.io.wavfile import write

fs = 16000
num_files = 2400

for i in range(num_files):
    sample = np.zeros(fs)
    filename = str(i*100)+'silent.wav'
    sample = sample + 0.1*random.randn(fs)
    scipy.io.wavfile.write('data/mydataset/silent/'+filename, fs, sample.astype(np.int16
```

Split the data into training and validation set

In [22]:
```
train_ds, val_ds = tf.keras.utils.audio_dataset_from_directory(
    directory='data/mydataset',
    labels='inferred',
    batch_size=64,
    class_names=None,
    validation_split=0.2,
    seed=0,
    output_sequence_length=16000,
    subset='both')

label_names = np.array(train_ds.class_names)
print()
print('label names:', label_names)
```

```
Found 11943 files belonging to 4 classes.
Using 9555 files for training.
Using 2388 files for validation.

label names: ['off' 'on' 'others' 'silent']
```

Remove the extra audio channel

In [23]:
```
train_ds.element_spec

def squeeze(audio, labels):
    audio = tf.squeeze(audio, axis=-1)
    return audio, labels

train_ds = train_ds.map(squeeze, tf.data.AUTOTUNE)
val_ds = val_ds.map(squeeze, tf.data.AUTOTUNE)
```

In [24]:
```
test_ds = val_ds.shard(num_shards=2, index=0)
val_ds = val_ds.shard(num_shards=2, index=1)
```

In [25]:
```
for sample_audio, sample_label in train_ds.take(1):
  print(sample_audio.shape)
  print(sample_label.shape)
```

```
(64, 16000)
(64,)
```

Function to compute audio spectrogram

In [26]:
```
def get_spectrogram(waveform):
    # Convert the waveform to a spectrogram via STFT.
    spectrogram = tf.signal.stft(waveform, frame_length=127, frame_step=64)
    # Obtain the absolute magnitude of the STFT.
    spectrogram = tf.abs(spectrogram)
    # Add a third dimension as channel, so that the spectrogram can be used
    # as image-like input data with convolution layers (which expect
    # shape (`batch_size`, `height`, `width`, `channels`).
    spectrogram = spectrogram[..., tf.newaxis]
    return spectrogram
```

Audio playback

In [27]:
```
from IPython import display

for i in range(5):
    indx = random.randint(0, sample_audio.shape[0]-1)
    label = label_names[sample_label[indx]]
    waveform = sample_audio[indx]
    #spectrogram = get_spectrogram(waveform)

    print('Label:', label)
    print('Audio shape:', waveform.shape)
    #print('Spectrogram shape:', spectrogram.shape)
    print('Audio playback')
    display.display(display.Audio(waveform, rate=16000))
```

```
Label: on
Audio shape: (16000,)
Audio playback
```

0:00 / 0:01

```
Label: on
Audio shape: (16000,)
Audio playback
```

0:00 / 0:01

```
Label: silent
Audio shape: (16000,)
Audio playback
```

```
C:\ProgramData\Anaconda3\lib\site-packages\IPython\lib\display.py:187: RuntimeWarning: i
nvalid value encountered in divide
  scaled = data / normalization_factor * 32767
```

0:00 / 0:01

```
Label: others
Audio shape: (16000,)
Audio playback
```

0:00 / 0:01

```
Label: silent
Audio shape: (16000,)
Audio playback
```

0:00 / 0:01

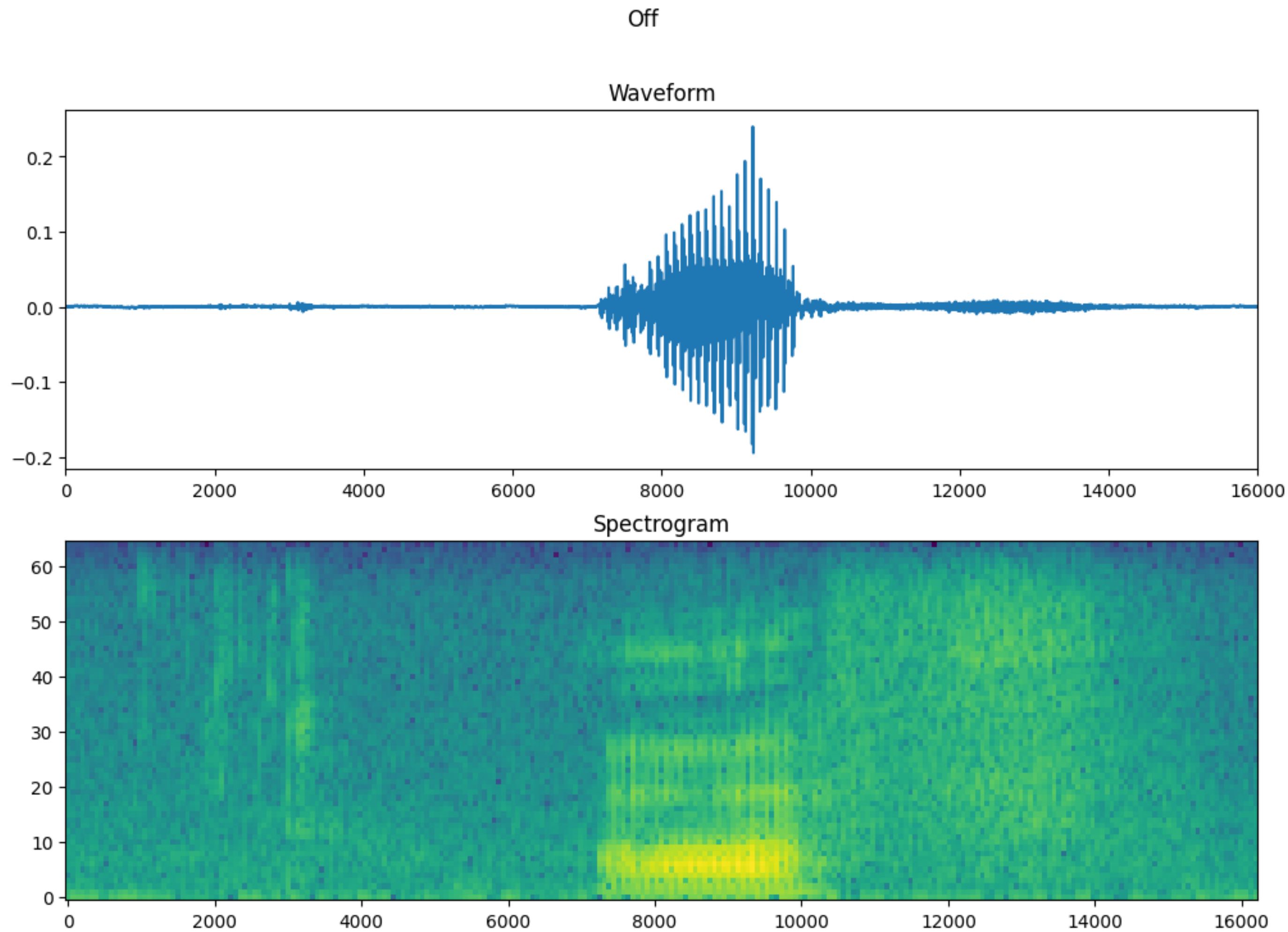
Spectrogram plot function

In [28]:
```
def plot_spectrogram(spectrogram, ax):
    if len(spectrogram.shape) > 2:
        assert len(spectrogram.shape) == 3
        spectrogram = np.squeeze(spectrogram, axis=-1)
    # Convert the frequencies to log scale and transpose, so that the time is
    # represented on the x-axis (columns).
    # Add an epsilon to avoid taking a log of zero.
    log_spec = np.log(spectrogram.T + np.finfo(float).eps)
    height = log_spec.shape[0]
    width = log_spec.shape[1]
    X = np.linspace(0, np.size(spectrogram), num=width, dtype=int)
    Y = range(height)
    ax.pcolormesh(X, Y, log_spec)
```

Plotting the spectrogram

In [43]:
```
indx = random.randint(0, sample_audio.shape[0]-1)
label = label_names[sample_label[indx]]
waveform = sample_audio[indx]
spectrogram = get_spectrogram(waveform)
fig, axes = plt.subplots(2, figsize=(12, 8))
timescale = np.arange(waveform.shape[0])
axes[0].plot(timescale, waveform.numpy())
axes[0].set_title('Waveform')
axes[0].set_xlim([0, 16000])

plot_spectrogram(spectrogram.numpy(), axes[1])
axes[1].set_title('Spectrogram')
plt.suptitle(label.title())
plt.show()
```




```
In [44]: print('Spectrogram shape:', spectrogram.shape)
```

```
Spectrogram shape: (249, 65, 1)
```

```
In [45]: <p style="page-break-after:always;"></p>def make_spectrogram_ds(ds):
             return ds.map(
                 map_func=lambda audio,label: (get_spectrogram(audio), label),
                 num_parallel_calls=tf.data.AUTOTUNE)
```

Convert into TensorFlow equivalent dataset

In [46]:
```
train_spectrogram_ds = make_spectrogram_ds(train_ds)
val_spectrogram_ds = make_spectrogram_ds(val_ds)
test_spectrogram_ds = make_spectrogram_ds(test_ds)
```

In [47]:
```
train_spectrogram_ds = train_spectrogram_ds.cache().shuffle(10000).prefetch(tf.data.AUTO
val_spectrogram_ds = val_spectrogram_ds.cache().prefetch(tf.data.AUTOTUNE)
test_spectrogram_ds = test_spectrogram_ds.cache().prefetch(tf.data.AUTOTUNE)
```

Define the CNN model

In [48]:
```
from tensorflow.python.util.nest import flatten
from tensorflow.python.ops.gen_nn_ops import Conv2D
for example_spectrograms, example_spect_labels in train_spectrogram_ds.take(1):
    input_shape = example_spectrograms.shape[1:]

num_classes = 4
model = models.Sequential()
model.add(layers.Conv2D(8, (8, 8), strides=(2, 2), padding='SAME', activation='relu', in
model.add(layers.Flatten())
model.add(layers.Dropout(0.1))
model.add(layers.Dense(num_classes, activation = 'softmax'))

model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_1 (Conv2D)           (None, 125, 33, 8)        520

 flatten_1 (Flatten)         (None, 33000)             0

 dropout_1 (Dropout)         (None, 33000)             0

 dense_1 (Dense)             (None, 4)                 132004

=================================================================
Total params: 132524 (517.67 KB)
Trainable params: 132524 (517.67 KB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

Train the model

In [49]:
```
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=False),
              metrics=['accuracy'])

model.fit(train_spectrogram_ds, validation_data=val_spectrogram_ds, epochs=100)
```

```
Epoch 1/100
150/150 [==============================] - 5s 34ms/step - loss: 0.7019 - accuracy: 0.737
2 - val_loss: 0.5655 - val_accuracy: 0.7867
Epoch 2/100
150/150 [==============================] - 3s 20ms/step - loss: 0.5059 - accuracy: 0.806
5 - val_loss: 0.5406 - val_accuracy: 0.8072
Epoch 3/100
150/150 [==============================] - 4s 24ms/step - loss: 0.4285 - accuracy: 0.840
9 - val_loss: 0.4994 - val_accuracy: 0.8225
Epoch 4/100
150/150 [==============================] - 4s 27ms/step - loss: 0.3749 - accuracy: 0.855
5 - val_loss: 0.5082 - val_accuracy: 0.8140
Epoch 5/100
150/150 [==============================] - 4s 26ms/step - loss: 0.3234 - accuracy: 0.881
1 - val_loss: 0.5113 - val_accuracy: 0.8336
Epoch 6/100
150/150 [==============================] - 4s 26ms/step - loss: 0.2908 - accuracy: 0.894
5 - val_loss: 0.5295 - val_accuracy: 0.8396
Epoch 7/100
150/150 [==============================] - 4s 26ms/step - loss: 0.2565 - accuracy: 0.907
0 - val_loss: 0.5359 - val_accuracy: 0.8473
Epoch 8/100
150/150 [==============================] - 4s 26ms/step - loss: 0.2249 - accuracy: 0.917
2 - val_loss: 0.5522 - val_accuracy: 0.8439
Epoch 9/100
150/150 [==============================] - 4s 25ms/step - loss: 0.2197 - accuracy: 0.922
0 - val_loss: 0.5789 - val_accuracy: 0.8311
Epoch 10/100
150/150 [==============================] - 4s 25ms/step - loss: 0.1916 - accuracy: 0.928
5 - val_loss: 0.5794 - val_accuracy: 0.8370
Epoch 11/100
150/150 [==============================] - 4s 26ms/step - loss: 0.1686 - accuracy: 0.941
0 - val_loss: 0.5989 - val_accuracy: 0.8447
Epoch 12/100
150/150 [==============================] - 4s 25ms/step - loss: 0.1597 - accuracy: 0.944
4 - val_loss: 0.6178 - val_accuracy: 0.8464
Epoch 13/100
150/150 [==============================] - 4s 26ms/step - loss: 0.1625 - accuracy: 0.943
0 - val_loss: 0.6485 - val_accuracy: 0.8507
Epoch 14/100
150/150 [==============================] - 4s 25ms/step - loss: 0.1360 - accuracy: 0.954
9 - val_loss: 0.6332 - val_accuracy: 0.8498
Epoch 15/100
150/150 [==============================] - 4s 26ms/step - loss: 0.1259 - accuracy: 0.956
9 - val_loss: 0.6698 - val_accuracy: 0.8549
Epoch 16/100
150/150 [==============================] - 4s 26ms/step - loss: 0.1337 - accuracy: 0.952
5 - val_loss: 0.6200 - val_accuracy: 0.8567
Epoch 17/100
150/150 [==============================] - 4s 25ms/step - loss: 0.1212 - accuracy: 0.961
1 - val_loss: 0.6871 - val_accuracy: 0.8575
Epoch 18/100
150/150 [==============================] - 4s 25ms/step - loss: 0.1081 - accuracy: 0.963
2 - val_loss: 0.7310 - val_accuracy: 0.8549
Epoch 19/100
150/150 [==============================] - 4s 26ms/step - loss: 0.1061 - accuracy: 0.963
9 - val_loss: 0.6513 - val_accuracy: 0.8712
```

```
Epoch 20/100
150/150 [==============================] - 4s 26ms/step - loss: 0.0944 - accuracy: 0.968
3 - val_loss: 0.7277 - val_accuracy: 0.8575
Epoch 21/100
150/150 [==============================] - 4s 25ms/step - loss: 0.0926 - accuracy: 0.968
8 - val_loss: 0.7337 - val_accuracy: 0.8549
Epoch 22/100
150/150 [==============================] - 4s 25ms/step - loss: 0.0927 - accuracy: 0.969
1 - val_loss: 0.7629 - val_accuracy: 0.8464
Epoch 23/100
150/150 [==============================] - 4s 25ms/step - loss: 0.0833 - accuracy: 0.973
4 - val_loss: 0.8181 - val_accuracy: 0.8456
Epoch 24/100
150/150 [==============================] - 4s 25ms/step - loss: 0.0769 - accuracy: 0.976
3 - val_loss: 0.7973 - val_accuracy: 0.8584
Epoch 25/100
150/150 [==============================] - 4s 25ms/step - loss: 0.0782 - accuracy: 0.974
0 - val_loss: 0.8597 - val_accuracy: 0.8490
Epoch 26/100
150/150 [==============================] - 4s 26ms/step - loss: 0.0679 - accuracy: 0.979
2 - val_loss: 0.7724 - val_accuracy: 0.8584
Epoch 27/100
150/150 [==============================] - 4s 25ms/step - loss: 0.0753 - accuracy: 0.976
2 - val_loss: 0.7937 - val_accuracy: 0.8524
Epoch 28/100
150/150 [==============================] - 4s 26ms/step - loss: 0.0774 - accuracy: 0.974
4 - val_loss: 0.8365 - val_accuracy: 0.8626
Epoch 29/100
150/150 [==============================] - 4s 25ms/step - loss: 0.0696 - accuracy: 0.977
6 - val_loss: 0.8783 - val_accuracy: 0.8490
Epoch 30/100
150/150 [==============================] - 4s 25ms/step - loss: 0.0640 - accuracy: 0.981
8 - val_loss: 0.8795 - val_accuracy: 0.8524
Epoch 31/100
150/150 [==============================] - 4s 25ms/step - loss: 0.0687 - accuracy: 0.978
1 - val_loss: 0.9203 - val_accuracy: 0.8464
Epoch 32/100
150/150 [==============================] - 4s 26ms/step - loss: 0.0692 - accuracy: 0.977
7 - val_loss: 0.9106 - val_accuracy: 0.8481
Epoch 33/100
150/150 [==============================] - 4s 26ms/step - loss: 0.0630 - accuracy: 0.981
5 - val_loss: 0.9537 - val_accuracy: 0.8294
Epoch 34/100
150/150 [==============================] - 4s 26ms/step - loss: 0.0703 - accuracy: 0.979
4 - val_loss: 0.9356 - val_accuracy: 0.8481
Epoch 35/100
150/150 [==============================] - 4s 27ms/step - loss: 0.0538 - accuracy: 0.983
3 - val_loss: 1.2497 - val_accuracy: 0.8123
Epoch 36/100
150/150 [==============================] - 4s 26ms/step - loss: 0.0591 - accuracy: 0.981
4 - val_loss: 1.0972 - val_accuracy: 0.8507
Epoch 37/100
150/150 [==============================] - 4s 25ms/step - loss: 0.0499 - accuracy: 0.984
9 - val_loss: 1.0532 - val_accuracy: 0.8524
Epoch 38/100
150/150 [==============================] - 4s 25ms/step - loss: 0.0513 - accuracy: 0.984
2 - val_loss: 1.0435 - val_accuracy: 0.8584
Epoch 39/100
150/150 [==============================] - 4s 26ms/step - loss: 0.0429 - accuracy: 0.988
3 - val_loss: 1.0482 - val_accuracy: 0.8584
Epoch 40/100
150/150 [==============================] - 4s 25ms/step - loss: 0.0454 - accuracy: 0.986
7 - val_loss: 1.0639 - val_accuracy: 0.8524
Epoch 41/100
150/150 [==============================] - 4s 26ms/step - loss: 0.0424 - accuracy: 0.988
0 - val_loss: 1.0509 - val_accuracy: 0.8515
```

```
Epoch 42/100
150/150 [==============================] - 4s 25ms/step - loss: 0.0516 - accuracy: 0.984
7 - val_loss: 1.1303 - val_accuracy: 0.8447
Epoch 43/100
150/150 [==============================] - 4s 25ms/step - loss: 0.0497 - accuracy: 0.985
5 - val_loss: 1.0628 - val_accuracy: 0.8558
Epoch 44/100
150/150 [==============================] - 4s 26ms/step - loss: 0.0416 - accuracy: 0.989
3 - val_loss: 1.1114 - val_accuracy: 0.8498
Epoch 45/100
150/150 [==============================] - 4s 26ms/step - loss: 0.0340 - accuracy: 0.990
7 - val_loss: 1.2002 - val_accuracy: 0.8515
Epoch 46/100
150/150 [==============================] - 4s 26ms/step - loss: 0.0438 - accuracy: 0.987
4 - val_loss: 1.1257 - val_accuracy: 0.8541
Epoch 47/100
150/150 [==============================] - 4s 25ms/step - loss: 0.0361 - accuracy: 0.990
7 - val_loss: 1.1307 - val_accuracy: 0.8643
Epoch 48/100
150/150 [==============================] - 4s 26ms/step - loss: 0.0435 - accuracy: 0.987
9 - val_loss: 1.1535 - val_accuracy: 0.8532
Epoch 49/100
150/150 [==============================] - 4s 26ms/step - loss: 0.0336 - accuracy: 0.991
1 - val_loss: 1.2894 - val_accuracy: 0.8353
Epoch 50/100
150/150 [==============================] - 4s 26ms/step - loss: 0.0348 - accuracy: 0.989
6 - val_loss: 1.1907 - val_accuracy: 0.8524
Epoch 51/100
150/150 [==============================] - 4s 26ms/step - loss: 0.0339 - accuracy: 0.990
7 - val_loss: 1.2835 - val_accuracy: 0.8584
Epoch 52/100
150/150 [==============================] - 4s 25ms/step - loss: 0.0333 - accuracy: 0.991
3 - val_loss: 1.3025 - val_accuracy: 0.8507
Epoch 53/100
150/150 [==============================] - 4s 25ms/step - loss: 0.0346 - accuracy: 0.990
9 - val_loss: 1.2703 - val_accuracy: 0.8541
Epoch 54/100
150/150 [==============================] - 4s 26ms/step - loss: 0.0534 - accuracy: 0.987
3 - val_loss: 1.1766 - val_accuracy: 0.8490
Epoch 55/100
150/150 [==============================] - 4s 25ms/step - loss: 0.0371 - accuracy: 0.988
8 - val_loss: 1.2618 - val_accuracy: 0.8541
Epoch 56/100
150/150 [==============================] - 4s 25ms/step - loss: 0.0277 - accuracy: 0.990
8 - val_loss: 1.2664 - val_accuracy: 0.8515
Epoch 57/100
150/150 [==============================] - 4s 26ms/step - loss: 0.0351 - accuracy: 0.988
7 - val_loss: 1.3095 - val_accuracy: 0.8481
Epoch 58/100
150/150 [==============================] - 4s 25ms/step - loss: 0.0312 - accuracy: 0.990
9 - val_loss: 1.3152 - val_accuracy: 0.8507
Epoch 59/100
150/150 [==============================] - 4s 25ms/step - loss: 0.0323 - accuracy: 0.991
1 - val_loss: 1.2770 - val_accuracy: 0.8626
Epoch 60/100
150/150 [==============================] - 4s 25ms/step - loss: 0.0246 - accuracy: 0.993
7 - val_loss: 1.4742 - val_accuracy: 0.8567
Epoch 61/100
150/150 [==============================] - 4s 27ms/step - loss: 0.0239 - accuracy: 0.993
6 - val_loss: 1.3405 - val_accuracy: 0.8567
Epoch 62/100
150/150 [==============================] - 4s 28ms/step - loss: 0.0356 - accuracy: 0.989
8 - val_loss: 1.3289 - val_accuracy: 0.8490
Epoch 63/100
150/150 [==============================] - 4s 26ms/step - loss: 0.0287 - accuracy: 0.991
8 - val_loss: 1.4465 - val_accuracy: 0.8618
```

```
Epoch 64/100
150/150 [==============================] - 4s 25ms/step - loss: 0.0212 - accuracy: 0.995
3 - val_loss: 1.3015 - val_accuracy: 0.8532
Epoch 65/100
150/150 [==============================] - 4s 25ms/step - loss: 0.0241 - accuracy: 0.993
3 - val_loss: 1.4121 - val_accuracy: 0.8456
Epoch 66/100
150/150 [==============================] - 4s 26ms/step - loss: 0.0317 - accuracy: 0.991
6 - val_loss: 1.3014 - val_accuracy: 0.8524
Epoch 67/100
150/150 [==============================] - 4s 26ms/step - loss: 0.0192 - accuracy: 0.995
5 - val_loss: 1.4520 - val_accuracy: 0.8473
Epoch 68/100
150/150 [==============================] - 4s 25ms/step - loss: 0.0298 - accuracy: 0.991
8 - val_loss: 1.3982 - val_accuracy: 0.8498
Epoch 69/100
150/150 [==============================] - 4s 25ms/step - loss: 0.0377 - accuracy: 0.988
9 - val_loss: 1.5203 - val_accuracy: 0.8422
Epoch 70/100
150/150 [==============================] - 4s 25ms/step - loss: 0.0308 - accuracy: 0.990
5 - val_loss: 1.3974 - val_accuracy: 0.8524
Epoch 71/100
150/150 [==============================] - 4s 24ms/step - loss: 0.0347 - accuracy: 0.990
4 - val_loss: 1.6131 - val_accuracy: 0.8447
Epoch 72/100
150/150 [==============================] - 4s 24ms/step - loss: 0.0367 - accuracy: 0.990
0 - val_loss: 1.5021 - val_accuracy: 0.8524
Epoch 73/100
150/150 [==============================] - 4s 24ms/step - loss: 0.0296 - accuracy: 0.990
8 - val_loss: 1.5733 - val_accuracy: 0.8439
Epoch 74/100
150/150 [==============================] - 4s 24ms/step - loss: 0.0197 - accuracy: 0.994
2 - val_loss: 1.4788 - val_accuracy: 0.8541
Epoch 75/100
150/150 [==============================] - 4s 24ms/step - loss: 0.0238 - accuracy: 0.993
7 - val_loss: 1.4350 - val_accuracy: 0.8439
Epoch 76/100
150/150 [==============================] - 4s 25ms/step - loss: 0.0236 - accuracy: 0.993
3 - val_loss: 1.4575 - val_accuracy: 0.8490
Epoch 77/100
150/150 [==============================] - 4s 25ms/step - loss: 0.0182 - accuracy: 0.995
3 - val_loss: 1.5390 - val_accuracy: 0.8524
Epoch 78/100
150/150 [==============================] - 4s 25ms/step - loss: 0.0179 - accuracy: 0.995
5 - val_loss: 1.5860 - val_accuracy: 0.8498
Epoch 79/100
150/150 [==============================] - 4s 26ms/step - loss: 0.0280 - accuracy: 0.991
8 - val_loss: 1.4369 - val_accuracy: 0.8549
Epoch 80/100
150/150 [==============================] - 4s 25ms/step - loss: 0.0272 - accuracy: 0.991
9 - val_loss: 1.4235 - val_accuracy: 0.8507
Epoch 81/100
150/150 [==============================] - 4s 24ms/step - loss: 0.0229 - accuracy: 0.993
2 - val_loss: 1.4645 - val_accuracy: 0.8456
Epoch 82/100
150/150 [==============================] - 4s 26ms/step - loss: 0.0235 - accuracy: 0.992
8 - val_loss: 1.5766 - val_accuracy: 0.8430
Epoch 83/100
150/150 [==============================] - 4s 26ms/step - loss: 0.0239 - accuracy: 0.992
6 - val_loss: 1.5001 - val_accuracy: 0.8498
Epoch 84/100
150/150 [==============================] - 4s 25ms/step - loss: 0.0205 - accuracy: 0.995
2 - val_loss: 1.6584 - val_accuracy: 0.8387
Epoch 85/100
150/150 [==============================] - 4s 24ms/step - loss: 0.0302 - accuracy: 0.990
7 - val_loss: 1.4316 - val_accuracy: 0.8515
```

```
Epoch 86/100
150/150 [==============================] - 4s 26ms/step - loss: 0.0196 - accuracy: 0.994
9 - val_loss: 1.5240 - val_accuracy: 0.8498
Epoch 87/100
150/150 [==============================] - 4s 25ms/step - loss: 0.0162 - accuracy: 0.995
8 - val_loss: 1.4775 - val_accuracy: 0.8422
Epoch 88/100
150/150 [==============================] - 4s 27ms/step - loss: 0.0260 - accuracy: 0.993
5 - val_loss: 1.5533 - val_accuracy: 0.8456
Epoch 89/100
150/150 [==============================] - 4s 25ms/step - loss: 0.0141 - accuracy: 0.995
9 - val_loss: 1.4910 - val_accuracy: 0.8567
Epoch 90/100
150/150 [==============================] - 4s 27ms/step - loss: 0.0154 - accuracy: 0.996
0 - val_loss: 1.5386 - val_accuracy: 0.8584
Epoch 91/100
150/150 [==============================] - 4s 26ms/step - loss: 0.0186 - accuracy: 0.994
6 - val_loss: 1.6028 - val_accuracy: 0.8387
Epoch 92/100
150/150 [==============================] - 4s 25ms/step - loss: 0.0284 - accuracy: 0.992
3 - val_loss: 1.6403 - val_accuracy: 0.8515
Epoch 93/100
150/150 [==============================] - 4s 25ms/step - loss: 0.0175 - accuracy: 0.996
3 - val_loss: 1.5072 - val_accuracy: 0.8481
Epoch 94/100
150/150 [==============================] - 4s 25ms/step - loss: 0.0138 - accuracy: 0.996
2 - val_loss: 1.5584 - val_accuracy: 0.8490
Epoch 95/100
150/150 [==============================] - 4s 25ms/step - loss: 0.0236 - accuracy: 0.993
1 - val_loss: 1.7911 - val_accuracy: 0.8422
Epoch 96/100
150/150 [==============================] - 4s 25ms/step - loss: 0.0213 - accuracy: 0.994
5 - val_loss: 1.6426 - val_accuracy: 0.8370
Epoch 97/100
150/150 [==============================] - 4s 24ms/step - loss: 0.0174 - accuracy: 0.996
2 - val_loss: 1.6368 - val_accuracy: 0.8541
Epoch 98/100
150/150 [==============================] - 4s 26ms/step - loss: 0.0132 - accuracy: 0.996
8 - val_loss: 1.6944 - val_accuracy: 0.8490
Epoch 99/100
150/150 [==============================] - 4s 25ms/step - loss: 0.0208 - accuracy: 0.994
3 - val_loss: 1.6616 - val_accuracy: 0.8498
Epoch 100/100
150/150 [==============================] - 4s 26ms/step - loss: 0.0136 - accuracy: 0.996
0 - val_loss: 1.6619 - val_accuracy: 0.8532
```

Out[49]:

```
<keras.src.callbacks.History at 0x1fcacfdd430>
```

Evaluate on test set

In [50]:

```python
model.evaluate(test_spectrogram_ds)
model.save('model.h5')
```

```
19/19 [==============================] - 1s 35ms/step - loss: 1.6347 - accuracy: 0.8495
C:\Users\mfarag\AppData\Roaming\Python\Python39\site-packages\keras\src\engine\training.
py:3103: UserWarning: You are saving your model as an HDF5 file via `model.save()`. This
file format is considered legacy. We recommend using instead the native Keras format, e.
g. `model.save('my_model.keras')`.
  saving_api.save_model(
```

Convert into equivalent TfLite model

In [51]:
```
model = tf.keras.models.load_model('model.h5')
converter = tf.lite.TFLiteConverter.from_keras_model(model)

converter.optimizations = [tf.lite.Optimize.DEFAULT]
tflite_quant_model = converter.convert()

with open('model.tflite', 'wb') as f:
    f.write(tflite_quant_model)
tflite_models_dir = pathlib.Path('/content/tflite_models/')
tflite_models_dir.mkdir(exist_ok=True, parents=True)
tflite_model_file = tflite_models_dir/"model.tflite"
tflite_model_file.write_bytes(tflite_quant_model)
```

```
INFO:tensorflow:Assets written to: C:\Users\mfarag\AppData\Local\Temp\tmpa09v_8u_\assets
```


```
INFO:tensorflow:Assets written to: C:\Users\mfarag\AppData\Local\Temp\tmpa09v_8u_\assets
```


Out[51]:
```
136320
```

Test the TFLite model

In [53]:
```
tflite_model_file = 'model.tflite'
# Initialize the TFLite interpreter
interpreter = tf.lite.Interpreter(model_path=tflite_model_file)
interpreter.allocate_tensors()
input_info = interpreter.get_input_details()[0]
input_index = input_info['index']
scale, offset = input_info['quantization']

input_index = interpreter.get_input_details()[0]['index']
output_index = interpreter.get_output_details()[0]['index']

total_count = 0.0
accurate_count = 0.0

for x, y_true in test_spectrogram_ds:
    input_shape = x.shape[1:]

    for count in range(x.shape[0]):
        temp = x[count,:]
        temp = tf.reshape(temp,(1, temp.shape[0], temp.shape[1], temp.shape[2]))

        interpreter.set_tensor(input_index, temp)
        interpreter.invoke()
        prediction = interpreter.get_tensor(output_index)[0]
        prediction = np.argmax(prediction)

        if prediction == y_true[count].numpy():
            accurate_count += 1

        total_count += 1

accuracy = accurate_count/total_count
print('Accuracy : ', accuracy)
```

```
Accuracy :  0.8536184210526315
```