MUSIC GENRE CLASSIFICATION

GROUP 10

Team: Mengyao Song & Steven Wang

PROJECT OBJECTIVE

01

- Data description
- How to handle audio data in python

02

Develop deep learning models to be used for music classification

03

Tune and compare models to achieve the best scores to be used for prediction

1.41 GB

- Data
  - genres_original
  - images_original
  - features_30_sec.csv
  - features_3_sec.csv

```
for col in df.columns:
    print(col)
```

```
filename
length
chroma_stft_mean
chroma_stft_var
rms_mean
rms_var
spectral_centroid_mean
spectral_centroid_var
spectral_bandwidth_mean
spectral_bandwidth_var
rolloff_mean
rolloff_var
zero_crossing_rate_mean
zero_crossing_rate_var
harmony_mean
harmony_var
perceptr_mean
perceptr_var
tempo
mfcc1_mean
mfcc1_var
mfcc2_mean
mfcc2_var
mfcc3_mean
mfcc3_var
mfcc4_mean
mfcc4_var
mfcc5_mean
```

```
df30 = pd.read_csv('features_30_sec.csv')
df30.head()
```

	filename	length	chroma_stft_mean	chroma_stft_var	rms_mean	rms_var	spectral_centroid_mean	spectral_centroid_var
0	blues.00000.wav	661794	0.350088	0.088757	0.130228	0.002827	1784.165850	129774.064525
1	blues.00001.wav	661794	0.340914	0.094980	0.095948	0.002373	1530.176679	375850.073649
2	blues.00002.wav	661794	0.363637	0.085275	0.175570	0.002746	1552.811865	156467.643368
3	blues.00003.wav	661794	0.404785	0.093999	0.141093	0.006346	1070.106615	184355.942417
4	blues.00004.wav	661794	0.308526	0.087841	0.091529	0.002303	1835.004266	343399.939274

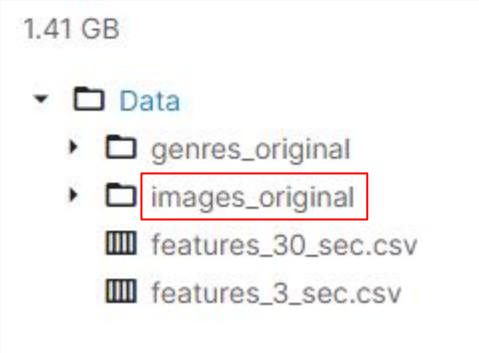
5 rows × 60 columns

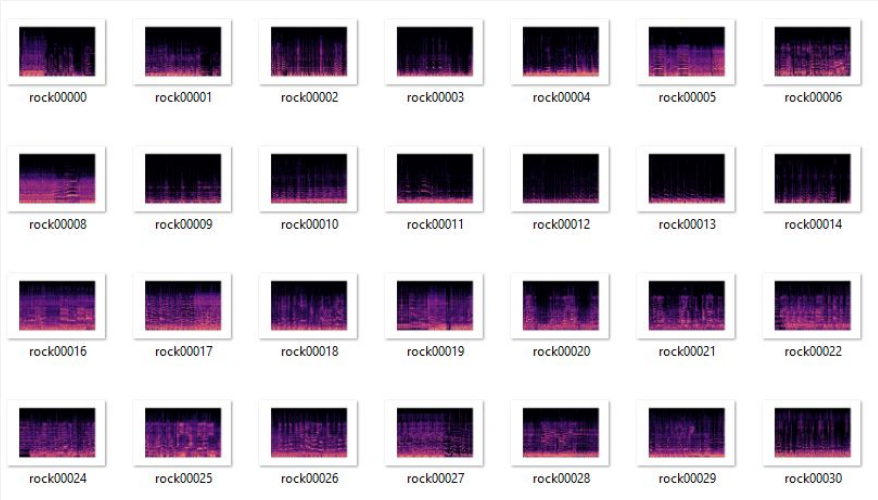
```
print("Dataset has",df.shape)
print("Count of Samples by Genre")
df.label.value_counts().reset_index()
```

```
Dataset has (9990, 60)
Count of Samples by Genre
```

	index	label
0	blues	1000
1	jazz	1000
2	metal	1000
3	pop	1000
4	reggae	1000
5	disco	999
6	classical	998
7	hiphop	998
8	rock	998
9	country	997

Mel Spectrograms





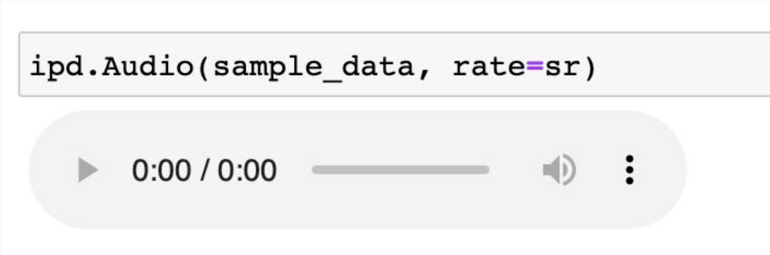
READING & UNDERSTANDING AUDIO DATA

1. Librosa

- a Python module to analyze audio signals in general but geared more towards music.

2. IPython.display.Audio

- lets you play audio directly in your notebook.

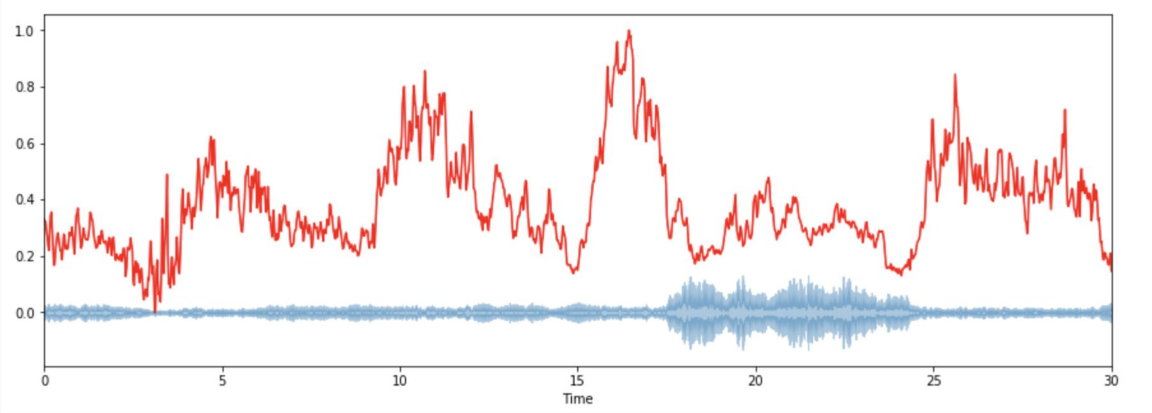


FEATURE EXTRACTION FROM AUDIO SIGNAL

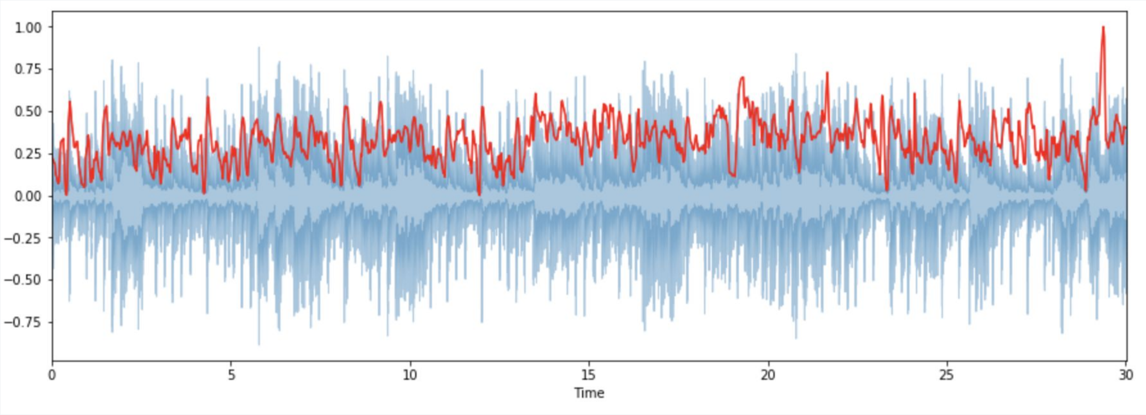
1. Spectral Centroid
2. Spectral Rolloff
3. Spectral Bandwidth
4. Zero-Crossing Rate
5. Mel-Frequency Cepstral Coefficients(MFCCs)
6. Chroma feature

I. SPECTRAL CENTROID

where the " center of mass" for a sound is located.



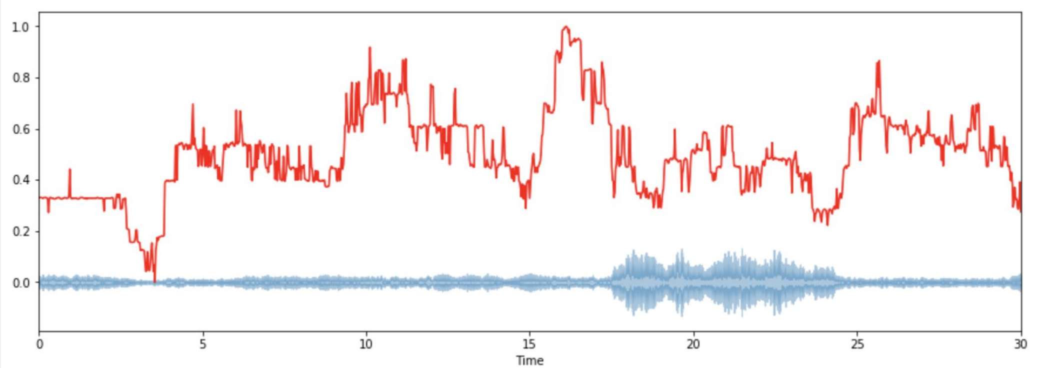
CLASSIC



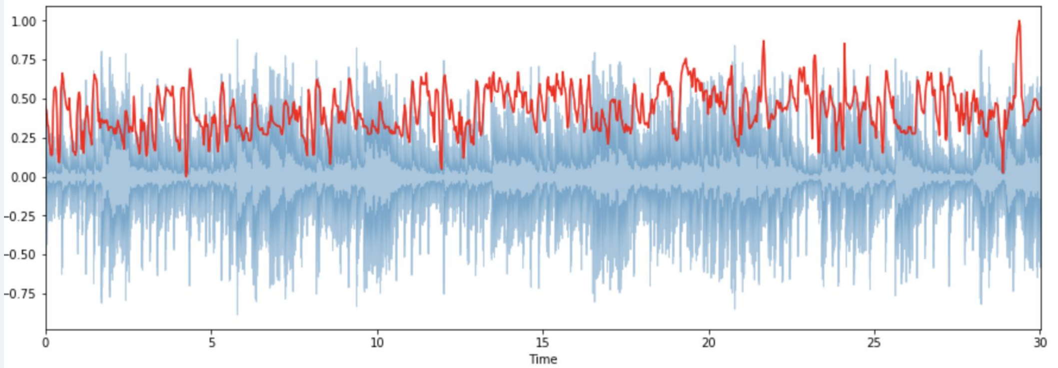
BLUES

2. SPECTRAL ROLLOFF

A measure of the shape of the signal
Represents the frequency at which high frequencies decline to 0

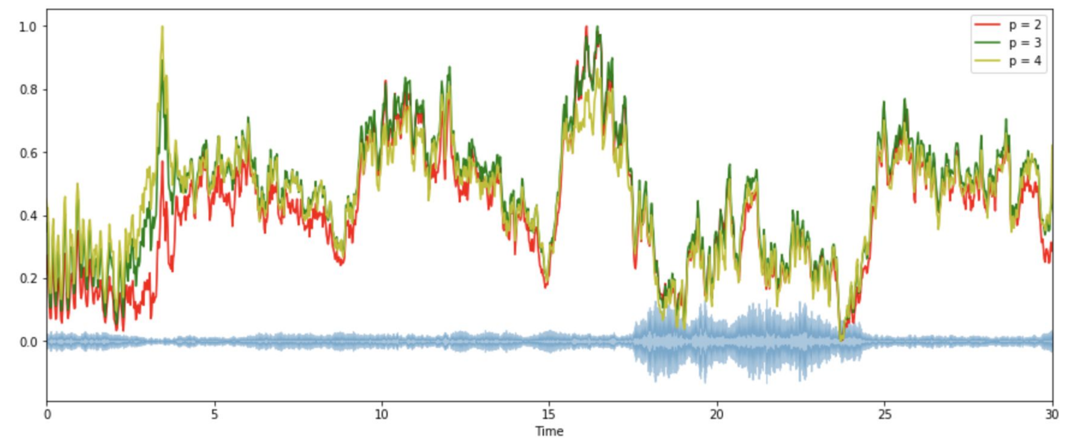


CLASSIC

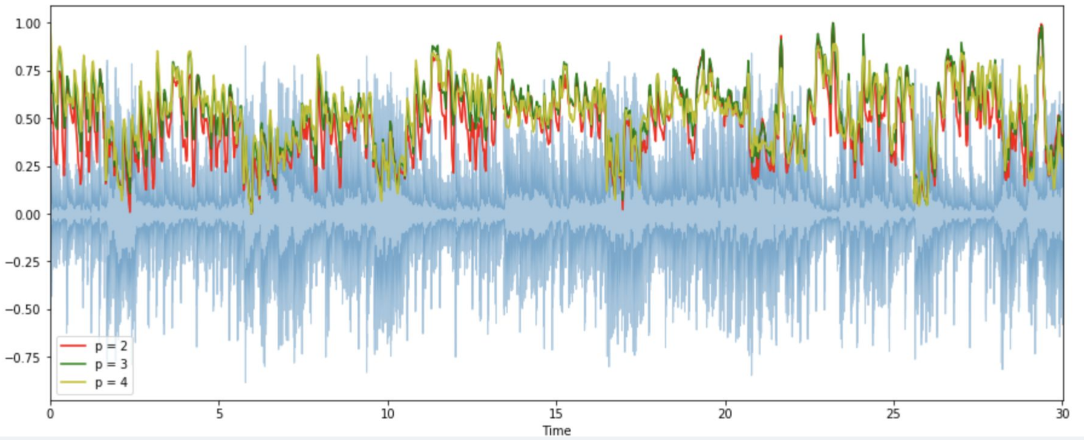


BLUES

3. SPECTRAL BANDWIDTH



CLASSIC



BLUES

4. ZERO-CROSSING RATE

measuring the smoothness of a signal

higher values for highly percussive sounds like those in metal and rock.

```
n0 = 9000
n1 = 9100
zero_crossings = librosa.zero_crossings(sample_data[n0:n1], pad=False)
print(sum(zero_crossings))
```

10

CLASSIC

```
n0 = 9000
n1 = 9100
zero_crossings = librosa.zero_crossings(sample_data2[n0:n1], pad=False)
print(sum(zero_crossings))
```

0

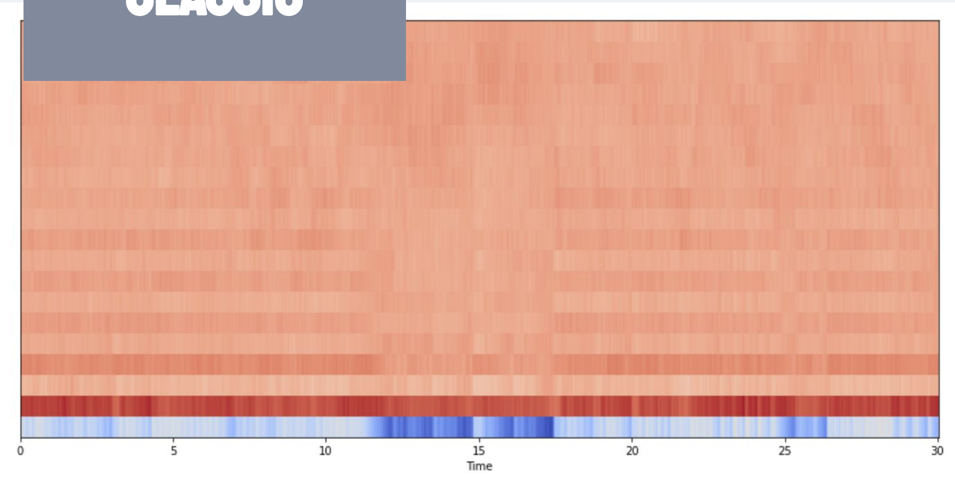
BLUES

5. MEL-FREQUENCY CEPSTRAL COEFFICIENTS (MFCCS)

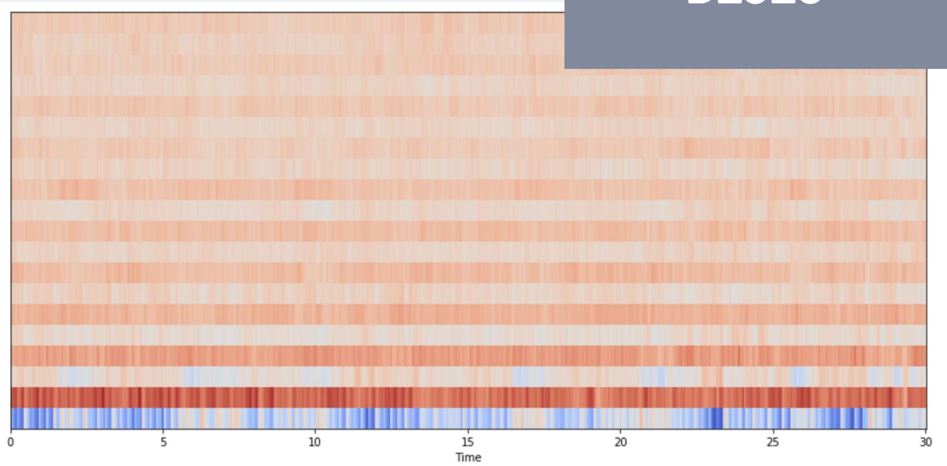
Describe the overall shape of a spectral envelope

models the characteristics of the human voice

CLASSIC

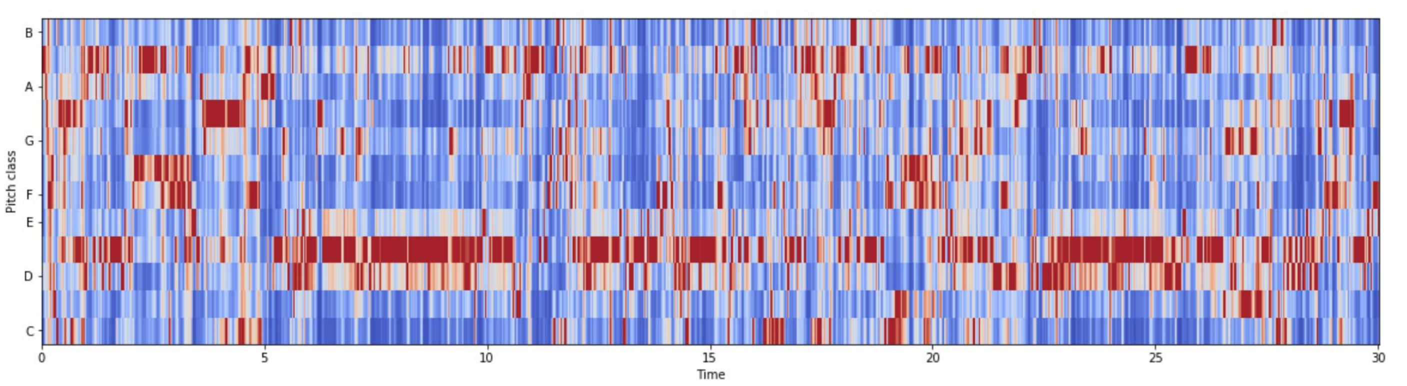


BLUES

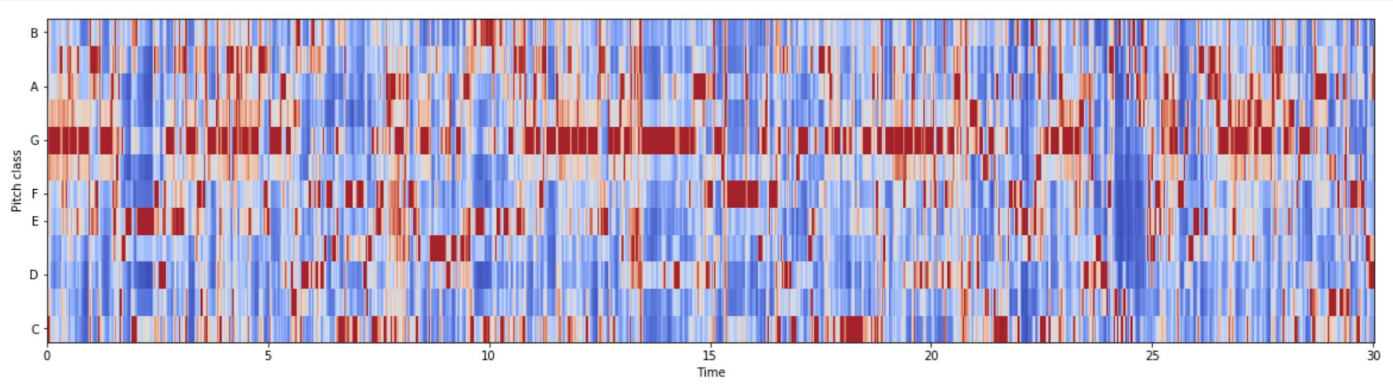


6. CHROMA FEATURE

a robust way to describe a similarity measure between music pieces.



CLASSIC



BLUES

MODELING

NEURAL NETWORK

CONVOLUTIONAL NEURAL NETWORK

EFFICIENTNET (CNN)

DATA PREPARATION

Define inputs and targets and encode target labels

```
X = df.drop(['label','filename'],axis=1)
y = df['label']
```

```
df['label'].unique()
```

```
['blues', 'classical', 'country', 'disco', 'hiphop', 'jazz', 'metal', 'pop', 'reggae', 'rock']
```

```
label_encoder = preprocessing.LabelEncoder()
df['label'] = label_encoder.fit_transform(df['label'])
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

Normalize Data

```
#Normalize data using minmaxscaler
cols = X.columns
min_max_scaler = preprocessing.MinMaxScaler()
np_scaled = min_max_scaler.fit_transform(X)
```

Split data into train & test

```
#Split data into train and test. 80-20 train test split
input_train, input_test, target_train, target_test = train_test_split(X, y, test_size=0.2)
print(input_train.shape, target_train.shape)
```

```
(7992, 58) (7992,)
```

BUILDING THE NN MODEL

```python
model = Sequential()

model.add(Flatten(input_shape=(58,)))
model.add(Dense(512, activation='relu', kernel_regularizer = keras.regularizers.l2(0.001)))
model.add(Dropout(0.3))
model.add(Dense(256, activation='relu', kernel_regularizer = keras.regularizers.l2(0.003)))
model.add(Dropout(0.3))
model.add(Dense(64, activation='relu', kernel_regularizer = keras.regularizers.l2(0.01)))
model.add(Dropout(0.3))
model.add(Dense(32, activation='relu'))
model.add(Dense(10, activation='softmax'))
model.summary()
```

Layer (type)	Output Shape	Param #
flatten_3 (Flatten)	(None, 58)	0
dense_13 (Dense)	(None, 512)	30208
dropout_7 (Dropout)	(None, 512)	0
dense_14 (Dense)	(None, 256)	131328
dropout_8 (Dropout)	(None, 256)	0
dense_15 (Dense)	(None, 64)	16448
dropout_9 (Dropout)	(None, 64)	0
dense_16 (Dense)	(None, 32)	2080
dense_17 (Dense)	(None, 10)	330

Total params: 180,394
Trainable params: 180,394
Non-trainable params: 0

TRAINING THE MODEL

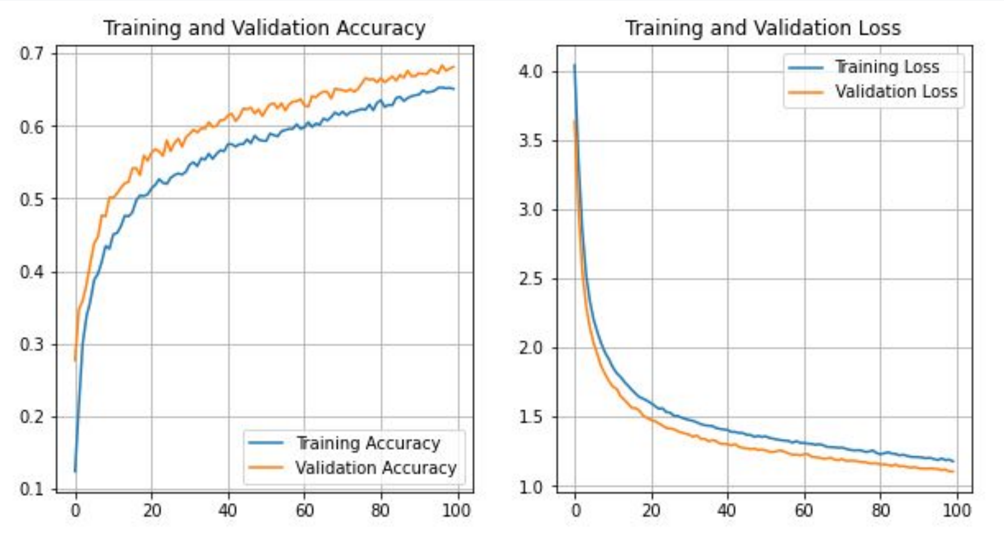
```
early_stopping= EarlyStopping(monitor='val_loss',mode='min',verbose=1,patience=5)
check_pointer = ModelCheckpoint(filepath = 'clf-resnet-checkpoint.hdf5',verbose=1,save_best_only=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss',mode='min',verbose=1,patience=5,min_delta = 0.0001,factor=0.2)
callbacks = [check_pointer,early_stopping,reduce_lr]
```

```
adam = keras.optimizers.Adam(lr=1e-4)
model.compile(optimizer=adam,
             loss="sparse_categorical_crossentropy",
             metrics=["accuracy"])
```

```
history = model.fit(input_train, target_train,
                 validation_data = (input_test,target_test),
                 epochs = 100,
                 batch_size = 32, callbacks = [check_pointer,early_stopping])
```

```
192/219 [=========================>....] - ETA: 0s - loss: 1.2160 - accuracy: 0.6501
Epoch 00097: val_loss improved from 1.14631 to 1.13899, saving model to clf-resnet-checkpoint.hdf5
219/219 [==============================] - 1s 2ms/step - loss: 1.2132 - accuracy: 0.6502 - val_loss: 1.1390 - val_accuracy:
0.6687
Epoch 98/100
196/219 [=========================>....] - ETA: 0s - loss: 1.2127 - accuracy: 0.6488
Epoch 00098: val_loss improved from 1.13899 to 1.13547, saving model to clf-resnet-checkpoint.hdf5
219/219 [==============================] - 1s 3ms/step - loss: 1.2139 - accuracy: 0.6489 - val_loss: 1.1355 - val_accuracy:
0.6750
Epoch 99/100
212/219 [============================>.] - ETA: 0s - loss: 1.2058 - accuracy: 0.6549
Epoch 00099: val_loss improved from 1.13547 to 1.13504, saving model to clf-resnet-checkpoint.hdf5
219/219 [==============================] - 1s 2ms/step - loss: 1.2052 - accuracy: 0.6542 - val_loss: 1.1350 - val_accuracy:
0.6693
Epoch 100/100
184/219 [========================>.....] - ETA: 0s - loss: 1.2114 - accuracy: 0.6503
Epoch 00100: val_loss improved from 1.13504 to 1.12828, saving model to clf-resnet-checkpoint.hdf5
219/219 [==============================] - 1s 3ms/step - loss: 1.2082 - accuracy: 0.6527 - val_loss: 1.1283 - val_accuracy:
0.6770
```

NN - TESTING & EVALUATION




```
test_error, test_accuracy = model.evaluate(input_test, target_test, verbose=1)
print(f"Test loss: {test_error}")
print(f"testing accuracy: {test_accuracy}")
```

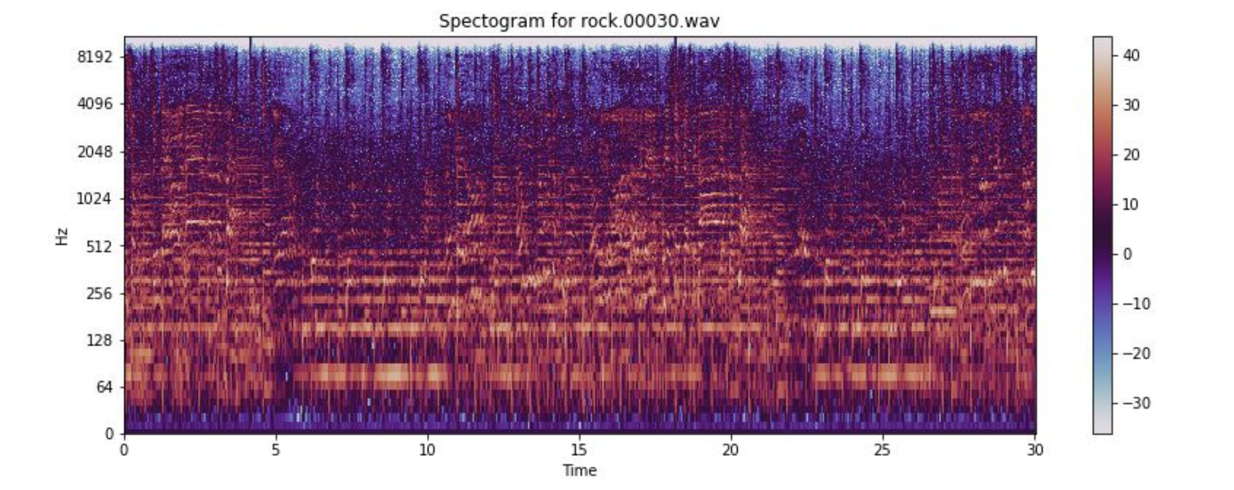
```
94/94 [==============================] - 0s 621us/step - loss: 1.1061 - accuracy: 0.6813
Test loss: 1.106074571609497
testing accuracy: 0.6813480257987976
```

WHY USE CNN?

- Able to process images and audio spectrograms for 2D arrays. Data transformed into a visual representation - **Spectrogram & Mel-Spectrogram**
- CNN detect auditory events in time-frequency reputation by “seeing” them
- Detect patterns and distortion to identify relevant features of different musical genres
- Filters can identify patterns/shapes for specific genres at different frequencies of a spectogram

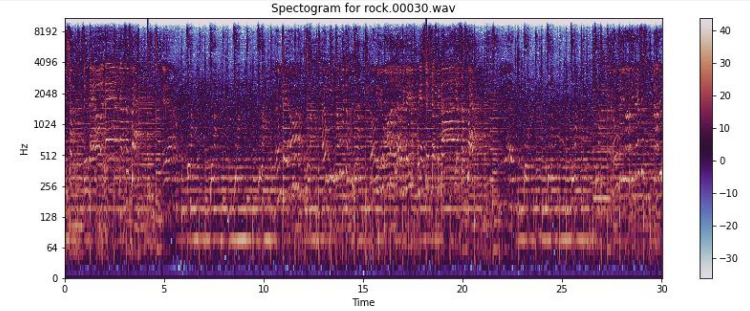
SPECTROGRAM

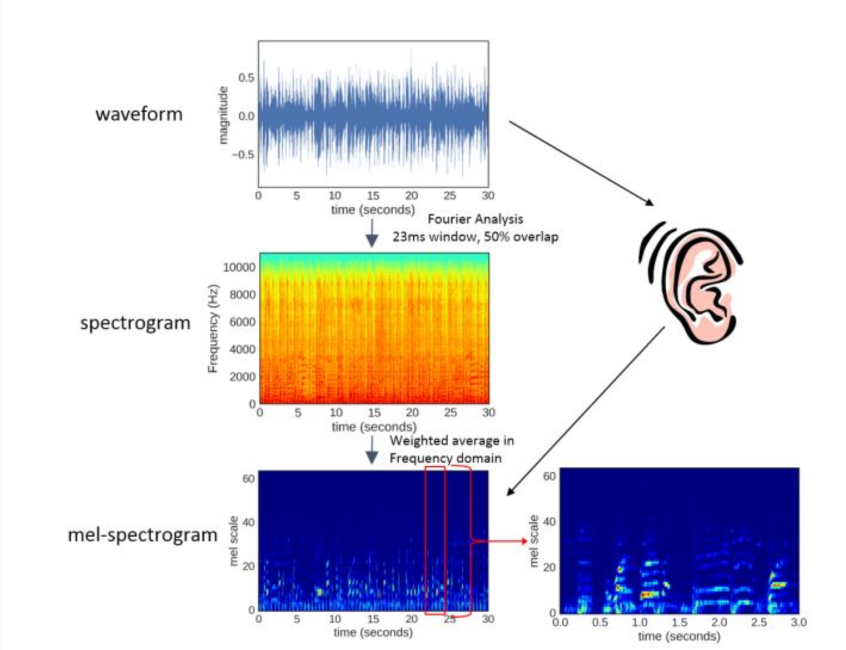
- A spectrogram is a visual way of representing the signal strength, or “loudness”, of a signal over time at various frequencies present in a particular waveform. Not only can one see whether there is more or less energy at, for example, 2 Hz vs 10 Hz, but one can also see how energy levels vary over time.
- A spectrogram is usually depicted as a heat map, i.e., as an image with the intensity shown by varying the color or brightness.



MEL-SPECTROGRAM

In Short, it is just a spectrogram converted into the Mel Scale. The Mel scale mimics how the human ear works. However, although humans can detect differences between a low and high frequency, we have problems detecting them on a linear scale.





CNN

```
BATCH_SIZE=8
TARGET_SIZE=224 # Based on EfficientNet
NUM_CLASSES=10
```

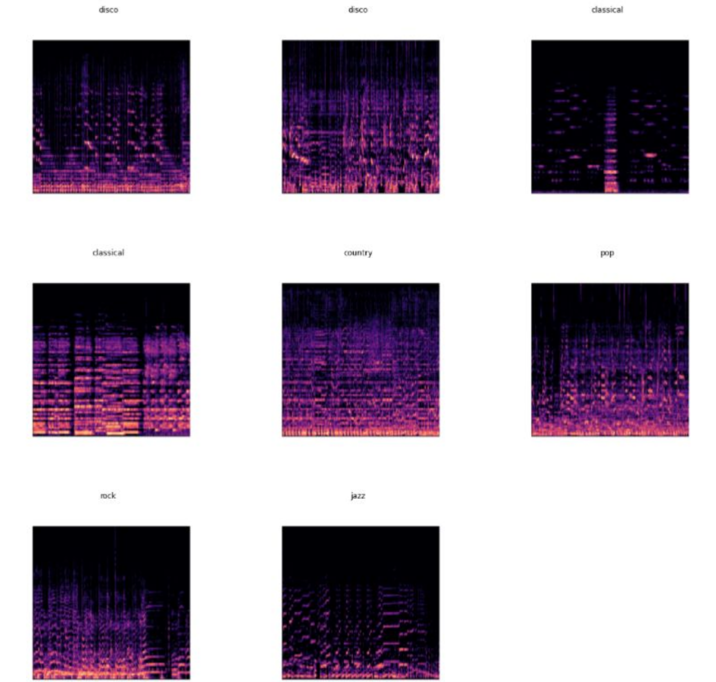
```
train_ds = image_dataset_from_directory(
  img_data,
  validation_split=0.2,
  subset="training",
  seed=123,
  image_size=(TARGET_SIZE, TARGET_SIZE),
  batch_size=BATCH_SIZE)
```

```
Found 999 files belonging to 10 classes.
Using 800 files for training.
```

```
test_ds = image_dataset_from_directory(
  img_data,
  validation_split=0.2,
  subset="testing",
  seed=123,
  image_size=(TARGET_SIZE, TARGET_SIZE),
  batch_size=BATCH_SIZE)
```

```
Found 999 files belonging to 10 classes.
Using 199 files for testing.
```

```
plt.figure(figsize=(20, 20))
for images, labels in train_ds.take(1):
    for i in range(8):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i].numpy().astype("uint8"))
        plt.title(class_names[labels[i]])
        plt.axis("off")
```



BUILDING THE CNN MODEL

```
model = Sequential([
  layers.experimental.preprocessing\
    .Rescaling(1./255, input_shape=(TARGET_SIZE, TARGET_SIZE, 3)),
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Dropout(0.2),
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(NUM_CLASSES)
])
model.summary()
```

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 rescaling_4 (Rescaling)     (None, 224, 224, 3)       0

 conv2d_9 (Conv2D)           (None, 224, 224, 16)      448

 max_pooling2d_9 (MaxPooling  (None, 112, 112, 16)     0
 2D)

 conv2d_10 (Conv2D)          (None, 112, 112, 32)      4640

 max_pooling2d_10 (MaxPoolin  (None, 56, 56, 32)       0
 g2D)

 conv2d_11 (Conv2D)          (None, 56, 56, 64)        18496

 max_pooling2d_11 (MaxPoolin  (None, 28, 28, 64)       0
 g2D)

 dropout_12 (Dropout)        (None, 28, 28, 64)        0

 flatten_6 (Flatten)         (None, 50176)             0

 dense_22 (Dense)            (None, 128)               6422656

 dense_23 (Dense)            (None, 10)                1290

=================================================================
Total params: 6,447,530
Trainable params: 6,447,530
Non-trainable params: 0
_________________________________________________________________
```

CNN

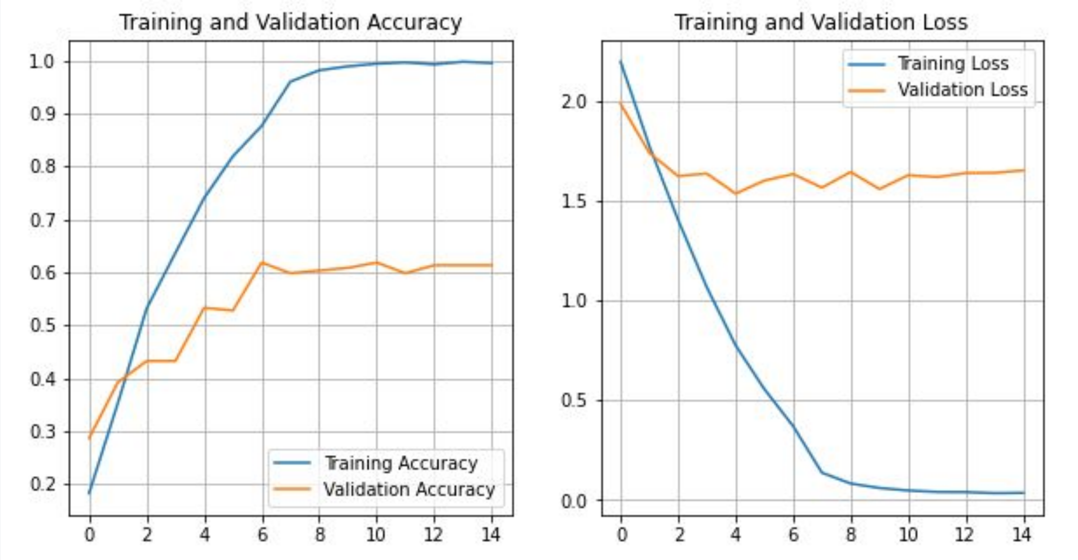
```
model_save = tf.keras.callbacks.ModelCheckpoint('./best_weights.h5',
                                                save_best_only = True,
                                                save_weights_only = True,
                                                monitor = 'val_loss',
                                                mode = 'min', verbose = 1)
early_stop = tf.keras.callbacks.EarlyStopping(monitor = 'val_loss', min_delta = 0.001,
                                             patience = 10, mode = 'min', verbose = 1,
                                             restore_best_weights = True)
reduce_lr = tf.keras.callbacks.ReduceLROnPlateau(monitor = 'val_loss', factor = 0.3,
                                                 patience = 2, min_delta = 0.001,
                                                 mode = 'min', verbose = 1)
```

```
model.compile(optimizer=Adam(lr = 0.001),
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])
```

```
epochs = 30
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs,
  callbacks=[model_save, early_stop, reduce_lr],
  verbose=2
)
```

```
Epoch 00016: val_loss did not improve from 1.44483

Epoch 00016: ReduceLROnPlateau reducing learning rate to 2.429999949526973e-06.
100/100 - 18s - loss: 0.0683 - accuracy: 0.9912 - val_loss: 1.6049 - val_accuracy: 0.6181 - lr: 8.1000e-06 - 18s/epoch - 177
ms/step
Epoch 17/30

Epoch 00017: val_loss did not improve from 1.44483
100/100 - 17s - loss: 0.0656 - accuracy: 0.9925 - val_loss: 1.6056 - val_accuracy: 0.6181 - lr: 2.4300e-06 - 17s/epoch - 172
ms/step
Epoch 18/30

Epoch 00018: val_loss did not improve from 1.44483
Restoring model weights from the end of the best epoch: 8.

Epoch 00018: ReduceLROnPlateau reducing learning rate to 7.289999985005124e-07.
100/100 - 17s - loss: 0.0633 - accuracy: 0.9925 - val_loss: 1.6091 - val_accuracy: 0.6181 - lr: 2.4300e-06 - 17s/epoch - 171
ms/step
Epoch 00018: early stopping
```

CNN - TESTING & EVALUATION



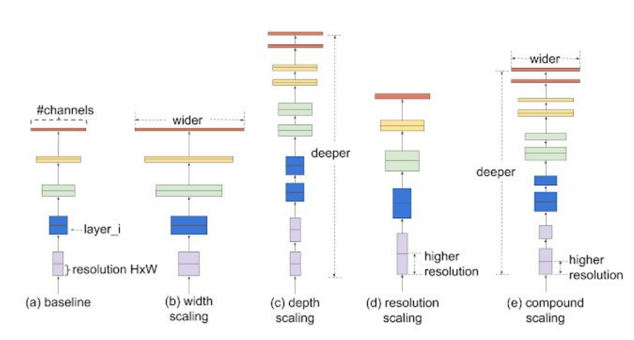

```
scores = model.evaluate(val_ds, class_names)
print('Loss: %.4f' %scores[0])
print('Accuracy: %.4f' %scores[1])
```

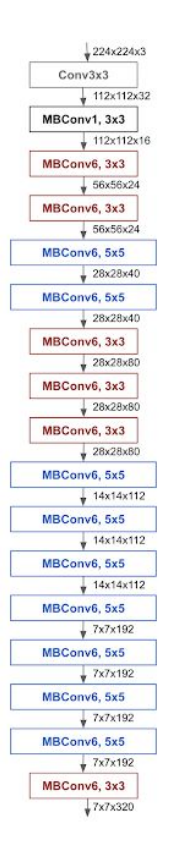
```
25/25 [==============================] - 1s 25ms/step
Loss: 1.5347
Accuracy: 0.6131
```

EFFICIENTNET

A CNN architecture and scaling method that uniformly scales depth/width/resolution dimensions using a *compound coefficient*.

Based on intuition that a bigger image needs more layers to increase the receptive field to capture patterns of that image.





THE EFFICIENTNET MODEL

```
def efficientnet_model():
    cnn_base = EfficientNetB0(include_top = False,
                              weights = "imagenet", drop_connect_rate=0.6,
                               input_shape = (TARGET_SIZE, TARGET_SIZE, 3))
    model = cnn_base.output
    model = layers.GlobalAveragePooling2D()(model)
    model = layers.Dense(NUM_CLASSES, activation = "softmax")(model)
    model = models.Model(cnn_base.input, model)

    model.compile(optimizer = Adam(lr = 0.001),
                  loss = "sparse_categorical_crossentropy",
                  metrics = ["accuracy"])
    return model
effmodel = efficientnet_model()
effmodel.summary()
```

MODEL SUMMARY...

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	[(None, 224, 224, 3)]	0	[]
rescaling_1 (Rescaling)	(None, 224, 224, 3)	0	['input_1[0][0]']
normalization (Normalization)	(None, 224, 224, 3)	7	['rescaling_1[0][0]']
stem_conv_pad (ZeroPadding2D)	(None, 225, 225, 3)	0	['normalization[0][0]']
stem_conv (Conv2D)	(None, 112, 112, 32)	864	['stem_conv_pad[0][0]']
stem_bn (BatchNormalization)	(None, 112, 112, 32)	128	['stem_conv[0][0]']
stem_activation (Activation)	(None, 112, 112, 32)	0	['stem_bn[0][0]']
block1a_dwconv (DepthwiseConv2D)	(None, 112, 112, 32)	288	['stem_activation[0][0]']
block1a_bn (BatchNormalization)	(None, 112, 112, 32)	128	['block1a_dwconv[0][0]']
block1a_activation (Activation)	(None, 112, 112, 32)	0	['block1a_bn[0][0]']
block1a_se_squeeze (GlobalAveragePooling2D)	(None, 32)	0	['block1a_activation[0][0]']
block1a_se_reshape (Reshape)	(None, 1, 1, 32)	0	['block1a_se_squeeze[0][0]']
block1a_se_reduce (Conv2D)	(None, 1, 1, 8)	264	['block1a_se_reshape[0][0]']
block1a_se_expand (Conv2D)	(None, 1, 1, 32)	288	['block1a_se_reduce[0][0]']
block1a_se_excite (Multiply)	(None, 112, 112, 32)	0	['block1a_activation[0][0]', 'block1a_se_expand[0][0]']
block1a_project_conv (Conv2D)	(None, 112, 112, 16)	512	['block1a_se_excite[0][0]']
block1a_project_bn (BatchNormalization)	(None, 112, 112, 16)	64	['block1a_project_conv[0][0]']
block2a_expand_conv (Conv2D)	(None, 112, 112, 96)	1536	['block1a_project_bn[0][0]']
block2a_expand_bn (BatchNormalization)	(None, 112, 112, 96)	384	['block2a_expand_conv[0][0]']
block2a_expand_activation (Activation)	(None, 112, 112, 96)	0	['block2a_expand_bn[0][0]']
block2a_dwconv_pad (ZeroPadding2D)	(None, 113, 113, 96)	0	['block2a_expand_activation[0][0]']
block2a_dwconv (DepthwiseConv2D)	(None, 56, 56, 96)	864	['block2a_dwconv_pad[0][0]']
block2a_bn (BatchNormalization)	(None, 56, 56, 96)	384	['block2a_dwconv[0][0]']
block2a_activation (Activation)	(None, 56, 56, 96)	0	['block2a_bn[0][0]']
block2a_se_squeeze (GlobalAveragePooling2D)	(None, 96)	0	['block2a_activation[0][0]']
block2a_se_reshape (Reshape)	(None, 1, 1, 96)	0	['block2a_se_squeeze[0][0]']
block2a_se_reduce (Conv2D)	(None, 1, 1, 4)	388	['block2a_se_reshape[0][0]']
block2a_se_expand (Conv2D)	(None, 1, 1, 96)	480	['block2a_se_reduce[0][0]']
block2a_se_excite (Multiply)	(None, 56, 56, 96)	0	['block2a_activation[0][0]', 'block2a_se_expand[0][0]']
block2a_project_conv (Conv2D)	(None, 56, 56, 24)	2304	['block2a_se_excite[0][0]']
block2a_project_bn (BatchNormalization)	(None, 56, 56, 24)	96	['block2a_project_conv[0][0]']
block2b_expand_conv (Conv2D)	(None, 56, 56, 144)	3456	['block2a_project_bn[0][0]']
block2b_expand_bn (BatchNormalization)	(None, 56, 56, 144)	576	['block2b_expand_conv[0][0]']
block2b_expand_activation (Activation)	(None, 56, 56, 144)	0	['block2b_expand_bn[0][0]']
block2b_dwconv (DepthwiseConv2D)	(None, 56, 56, 144)	1296	['block2b_expand_activation[0][0]']
block2b_bn (BatchNormalization)	(None, 56, 56, 144)	576	['block2b_dwconv[0][0]']
block2b_activation (Activation)	(None, 56, 56, 144)	0	['block2b_bn[0][0]']
block2b_se_squeeze (GlobalAveragePooling2D)	(None, 144)	0	['block2b_activation[0][0]']
block2b_se_reshape (Reshape)	(None, 1, 1, 144)	0	['block2b_se_squeeze[0][0]']
block2b_se_reduce (Conv2D)	(None, 1, 1, 6)	870	['block2b_se_reshape[0][0]']
block2b_se_expand (Conv2D)	(None, 1, 1, 144)	1008	['block2b_se_reduce[0][0]']
block2b_se_excite (Multiply)	(None, 56, 56, 144)	0	['block2b_activation[0][0]', 'block2b_se_expand[0][0]']
block2b_project_conv (Conv2D)	(None, 56, 56, 24)	3456	['block2b_se_excite[0][0]']
block2b_project_bn (BatchNormalization)	(None, 56, 56, 24)	96	['block2b_project_conv[0][0]']
block2b_drop (Dropout)	(None, 56, 56, 24)	0	['block2b_project_bn[0][0]']
block2b_add (Add)	(None, 56, 56, 24)	0	['block2b_drop[0][0]', 'block2a_project_bn[0][0]']
block3a_expand_conv (Conv2D)	(None, 56, 56, 144)	3456	['block2b_add[0][0]']
block3a_expand_bn (BatchNormalization)	(None, 56, 56, 144)	576	['block3a_expand_conv[0][0]']
block3a_expand_activation (Activation)	(None, 56, 56, 144)	0	['block3a_expand_bn[0][0]']
block3a_dwconv_pad (ZeroPadding2D)	(None, 59, 59, 144)	0	['block3a_expand_activation[0][0]']
block3a_dwconv (DepthwiseConv2D)	(None, 28, 28, 144)	3600	['block3a_dwconv_pad[0][0]']
block3a_bn (BatchNormalization)	(None, 28, 28, 144)	576	['block3a_dwconv[0][0]']
block3a_activation (Activation)	(None, 28, 28, 144)	0	['block3a_bn[0][0]']
block3a_se_squeeze (GlobalAveragePooling2D)	(None, 144)	0	['block3a_activation[0][0]']
block3a_se_reshape (Reshape)	(None, 1, 1, 144)	0	['block3a_se_squeeze[0][0]']
block3a_se_reduce (Conv2D)	(None, 1, 1, 6)	870	['block3a_se_reshape[0][0]']
block3a_se_expand (Conv2D)	(None, 1, 1, 144)	1008	['block3a_se_reduce[0][0]']
block3a_se_excite (Multiply)	(None, 28, 28, 144)	0	['block3a_activation[0][0]', 'block3a_se_expand[0][0]']
block3a_project_conv (Conv2D)	(None, 28, 28, 40)	5760	['block3a_se_excite[0][0]']
block3a_project_bn (BatchNormalization)	(None, 28, 28, 40)	160	['block3a_project_conv[0][0]']
block3b_expand_conv (Conv2D)	(None, 28, 28, 240)	9600	['block3a_project_bn[0][0]']
block3b_expand_bn (BatchNormalization)	(None, 28, 28, 240)	960	['block3b_expand_conv[0][0]']
block3b_expand_activation (Activation)	(None, 28, 28, 240)	0	['block3b_expand_bn[0][0]']
block3b_dwconv (DepthwiseConv2D)	(None, 28, 28, 240)	6000	['block3b_expand_activation[0][0]']
block3b_bn (BatchNormalization)	(None, 28, 28, 240)	960	['block3b_dwconv[0][0]']
block3b_activation (Activation)	(None, 28, 28, 240)	0	['block3b_bn[0][0]']
block3b_se_squeeze (GlobalAveragePooling2D)	(None, 240)	0	['block3b_activation[0][0]']
block3b_se_reshape (Reshape)	(None, 1, 1, 240)	0	['block3b_se_squeeze[0][0]']
block3b_se_reduce (Conv2D)	(None, 1, 1, 10)	2410	['block3b_se_reshape[0][0]']
block3b_se_expand (Conv2D)	(None, 1, 1, 240)	2640	['block3b_se_reduce[0][0]']
block3b_se_excite (Multiply)	(None, 28, 28, 240)	0	['block3b_activation[0][0]', 'block3b_se_expand[0][0]']
block3b_project_conv (Conv2D)	(None, 28, 28, 40)	9600	['block3b_se_excite[0][0]']
block3b_project_bn (BatchNormalization)	(None, 28, 28, 40)	160	['block3b_project_conv[0][0]']
block3b_drop (Dropout)	(None, 28, 28, 40)	0	['block3b_project_bn[0][0]']
block3b_add (Add)	(None, 28, 28, 40)	0	['block3b_drop[0][0]', 'block3a_project_bn[0][0]']
block4a_expand_conv (Conv2D)	(None, 28, 28, 240)	9600	['block3b_add[0][0]']
block4a_expand_bn (BatchNormalization)	(None, 28, 28, 240)	960	['block4a_expand_conv[0][0]']
block4a_expand_activation (Activation)	(None, 28, 28, 240)	0	['block4a_expand_bn[0][0]']
block4a_dwconv_pad (ZeroPadding2D)	(None, 29, 29, 240)	0	['block4a_expand_activation[0][0]']

+4 images...

```
Total params: 4,062,381
Trainable params: 4,020,358
Non-trainable params: 42,023
```

EFFICIENTNET CONT'D

```
epochs = 30
history = effmodel.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs,
  callbacks=[model_save, early_stop, reduce_lr],
  verbose=2
)
```

```
Epoch 00025: val_loss did not improve from 0.78520

Epoch 00025: ReduceLROnPlateau reducing learning rate to 2.1870000637136398e-07.
100/100 - 100s - loss: 0.0339 - accuracy: 0.9912 - val_loss: 0.8080 - val_accuracy: 0.7940 - lr: 7.2900e-07 - 100s/epoch - 1
s/step
Epoch 26/30

Epoch 00026: val_loss did not improve from 0.78520
100/100 - 102s - loss: 0.0431 - accuracy: 0.9887 - val_loss: 0.8063 - val_accuracy: 0.7940 - lr: 2.1870e-07 - 102s/epoch - 1
s/step
Epoch 27/30

Epoch 00027: val_loss did not improve from 0.78520
Restoring model weights from the end of the best epoch: 17.

Epoch 00027: ReduceLROnPlateau reducing learning rate to 6.561000276406048e-08.
100/100 - 96s - loss: 0.0295 - accuracy: 0.9962 - val_loss: 0.8084 - val_accuracy: 0.7940 - lr: 2.1870e-07 - 96s/epoch - 964m
s/step
Epoch 00027: early stopping
```

EFFICIENTNET - TESTING & EVALUATION


```
scores = effmodel.evaluate(test_ds, class_names)
print('Loss: %.4f' %scores[0])
print('Accuracy: %.4f' %scores[1])
```

```
25/25 [==============================] - 5s 189ms/step
Loss: 0.8303
Accuracy: 0.7852
```

SUMMARY

MODEL	TEST LOSS	TEST ACCURACY
NEURAL NETWORK	1.1060	0.6813
CONVOLUTIONAL NEURAL NETWORK	1.5347	0.6131
EFFICIENTNET	0.8303	0.7852

- Sound can represented as various audio signals such as frequency, bandwidth, decibel, etc. These features can be converted into numerical or visual representation to build Models.

- The baseline neural network model performed ok with a 68% accuracy. It required minimal data transformation.

- EfficientNet model had the best scores at 0.83 loss and 79% accuracy. However, it requires a lot of layers to build model and computing power to run the model.

THANK

YOU