DEEP LEARNING

THE

MUSIC

GENRE CLASSIFICATION

PRESENTATION

CREATED BY WUT YEE WIN

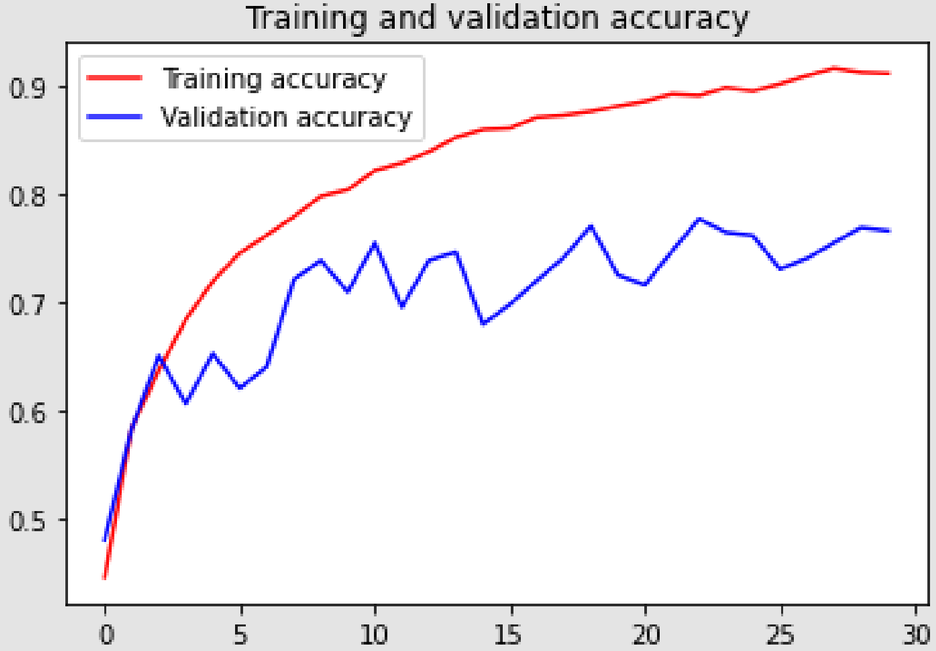
ID - 6540064

WHY IS MUSIC?

- Universal Language
- Personalized Music Recommendation System: Spotify, YouTube

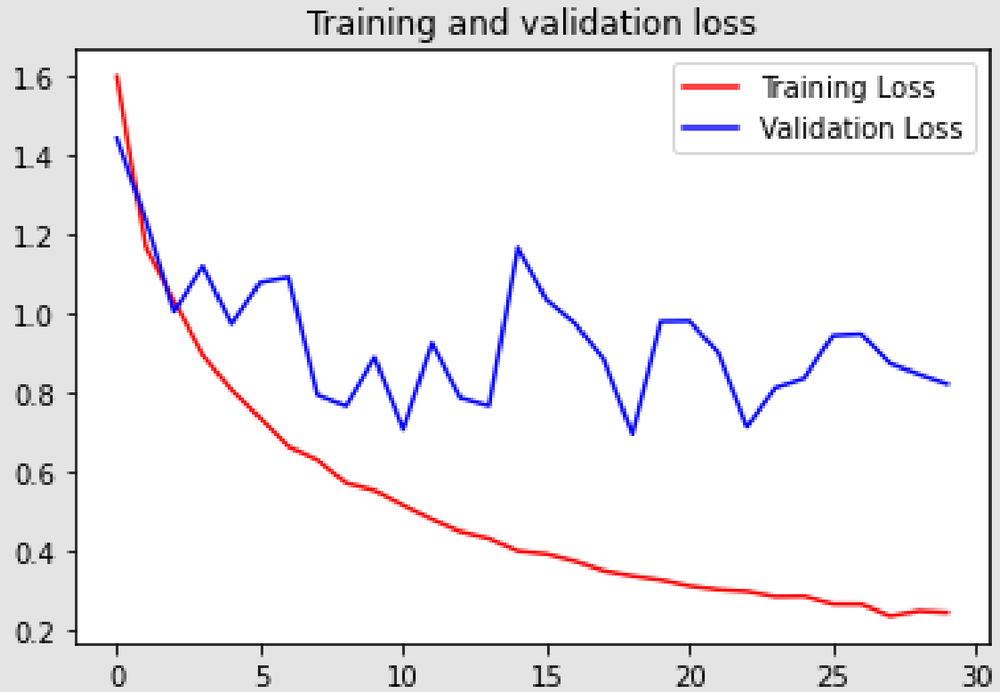
BASELINE MODEL

- Val - Accuracy : 75%
- Classify 30 sec audio files by genre using TensorFlow and Librosa
- Use GTZAN Dataset



TECHNIQUES

- Use MFCCs
- CNN model
- DropOut, BatchNormalization
- Optimization - RMSprop



RMSPROP

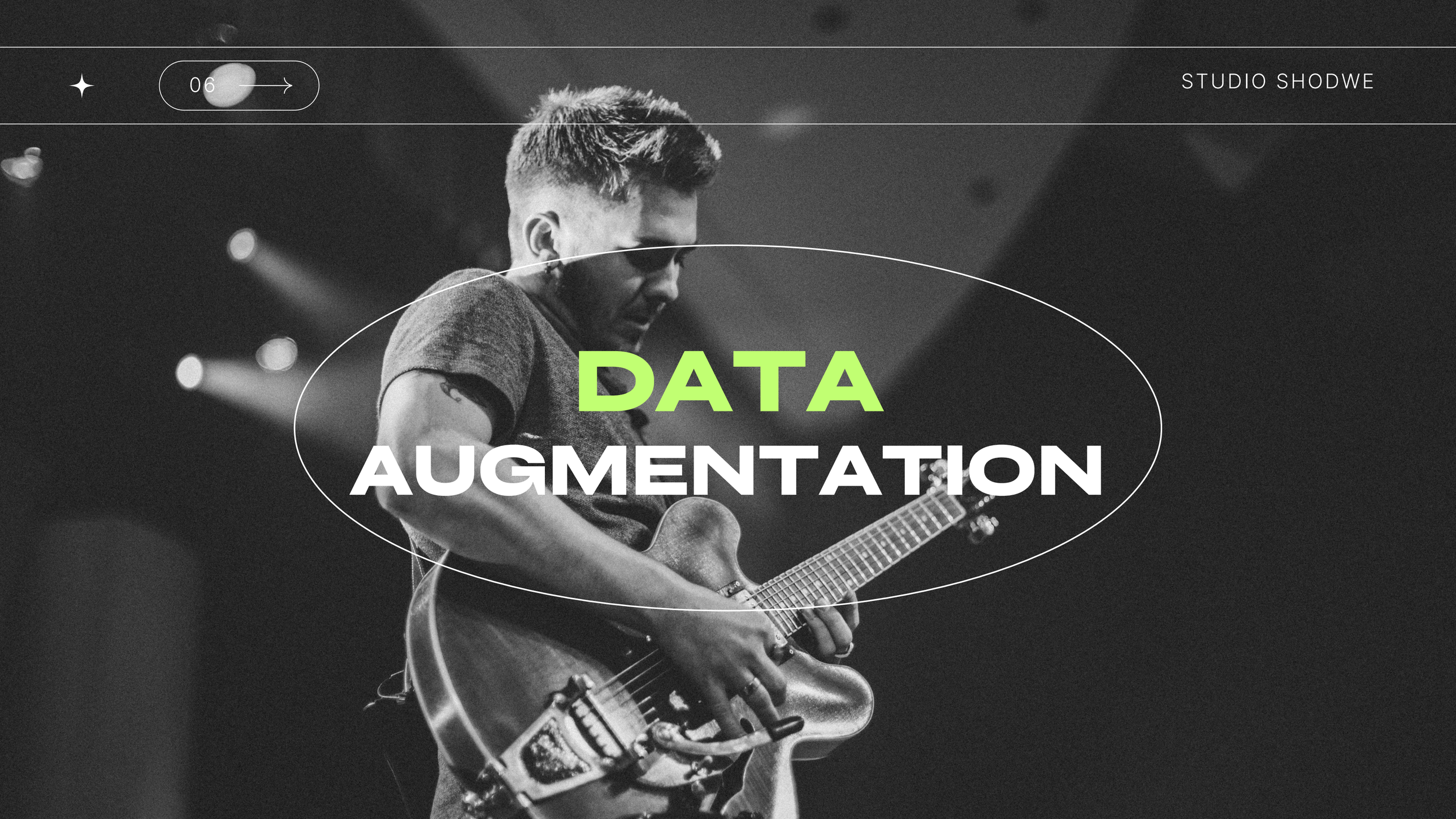
- Particularly effective for tasks involving non-stationary data
- Lack momentum
- Does not always smooth the learning trajectory over time

ADAM

- Combine the strengths of both RMSprop and momentum-based optimizers.
- Adapt the learning rate but also to accelerate convergence through momentum

WHY SWITCH?

- RMSprop can be sensitive to hyperparameters like the learning rate, which often requires fine-tuning to get optimal performance
- While RMSprop worked decently in earlier training, Adam significantly improved the convergence speed and model accuracy



DATA AUGMENTATION

UNSEEN DATA

- Don’t have enough training data
- Increase the diversity of our training dataset by applying various transformations to the existing data
- Pitch Shifting, Add Noise, Change Volume and Time Shifting

```python
def preprocess_data(source_path, json_path, num_mfcc=13, n_fft=2048,
hop_length=512, num_segments=5,augment=True):
    data = {
        "mapping":[],
        "labels": [],
        "mfcc": [],
        }

    samples_per_segment = int(SAMPLES_PER_TRACK/num_segments)

    num_mfcc_vectors_per_segment = math.ceil(samples_per_segment/hop_length
)

    for i, (dirpath, dirnames, filenames) in enumerate(os.walk(source_path
)):
      for file in filenames:
        if os.path.join(dirpath,file) != '/content/drive/MyDrive/Data
/genres_original/jazz/jazz.00054.wav':
          file_path = os.path.join(dirpath,file)
          signal,sample_rate = librosa.load(file_path,sr=SAMPLE_RATE)

          if augment:
            signal = augment_audio(signal,sample_rate)

          for d in range(num_segments):
            start = samples_per_segment * d
            finish = start + samples_per_segment

            mfcc = librosa.feature.mfcc(y=signal[start:finish],sr
=sample_rate,n_mfcc=num_mfcc,n_fft=n_fft,hop_length=hop_length)
            mfcc = mfcc.T

            if len(mfcc) == num_mfcc_vectors_per_segment:
              data["mfcc"].append(mfcc.tolist())
              data["labels"].append(i-1)

    with open(json_path, 'w') as f:
        json.dump(data, f)
    f.close()
```

```python
def augment_audio(audio_data, sample_rate):
    if np.random.rand() < 0.5:
        audio_data = pitch_shift(audio_data, sample_rate, n_steps=np.random
.randint(-5, 5))
    if np.random.rand() < 0.5:
        audio_data = add_noise(audio_data, noise_factor=np.random.uniform(0
.001, 0.01))
    if np.random.rand() < 0.5:
        audio_data = change_volume(audio_data, gain_factor=np.random
.uniform(0.7, 1.3))
    if np.random.rand() < 0.5:
        audio_data = time_shift(audio_data, shift_max=0.2)
    return audio_data
```

```python
def time_shift(audio_data, shift_max=0.2):
  shift = np.random.randint(len(audio_data) * shift_max)
  return np.roll(audio_data, shift)

def change_volume(audio_data, gain_factor=1.5):
  return audio_data * gain_factor

def add_noise(audio_data, noise_factor=0.005):
  noise = np.random.randn(len(audio_data))
  augmented_audio = audio_data + noise_factor * noise
  return augmented_audio

def pitch_shift(audio_data, sample_rate, n_steps=2):
  return librosa.effects.pitch_shift(audio_data, sr=sample_rate, n_steps
=n_steps)
```

PREVENT OVERFITTING

- Add Dropout Layers
- Add Regularization Methods

HYPERTUNING

- Model depend on the right set of hyperparameters.
- Improve model performance
- Balance regularization
- RandomSearch tests a range of possible values and automatically identifies which hyperparameter settings perform the best.

```python
def design_model(input_shape):

    # Let's design the model architecture.
    model = tf.keras.models.Sequential([

        tf.keras.layers.Conv2D(32, (3,3), activation='relu', input_shape
=input_shape),
        tf.keras.layers.MaxPooling2D((3,3), strides=(2,2), padding='same'),
        tf.keras.layers.BatchNormalization(),

        tf.keras.layers.Conv2D(32, (3,3), activation='relu'),
        tf.keras.layers.MaxPooling2D((3,3), strides=(2,2), padding='same'),
        tf.keras.layers.BatchNormalization(),

        tf.keras.layers.Conv2D(32, (2,2), activation='relu'),
        tf.keras.layers.MaxPooling2D((3,3), strides=(2,2), padding='same'),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Dropout(0.3),

        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(64, activation='relu'),
        tf.keras.layers.Dense(len(np.unique(targets)), activation='softmax'
)
    ])

    return model


model.compile(optimizer = tf.keras.optimizers.RMSprop(lr=0.001),
                    loss='sparse_categorical_crossentropy',
                    metrics = ['acc']
                    )
```

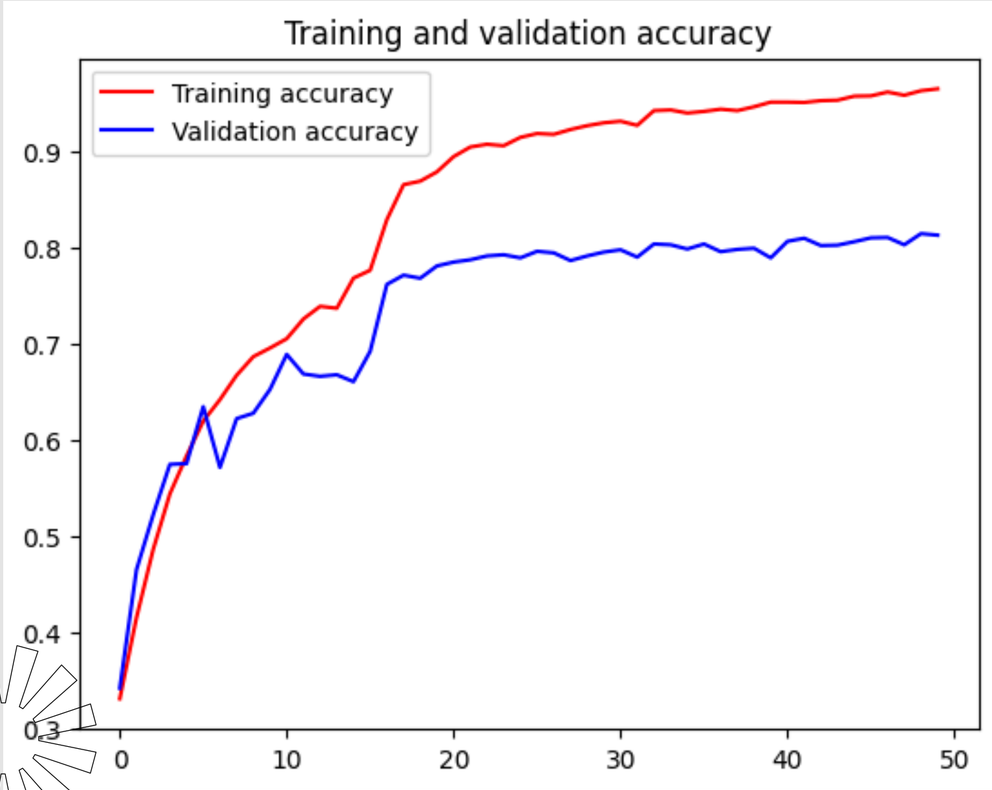
BASELINE MODEL

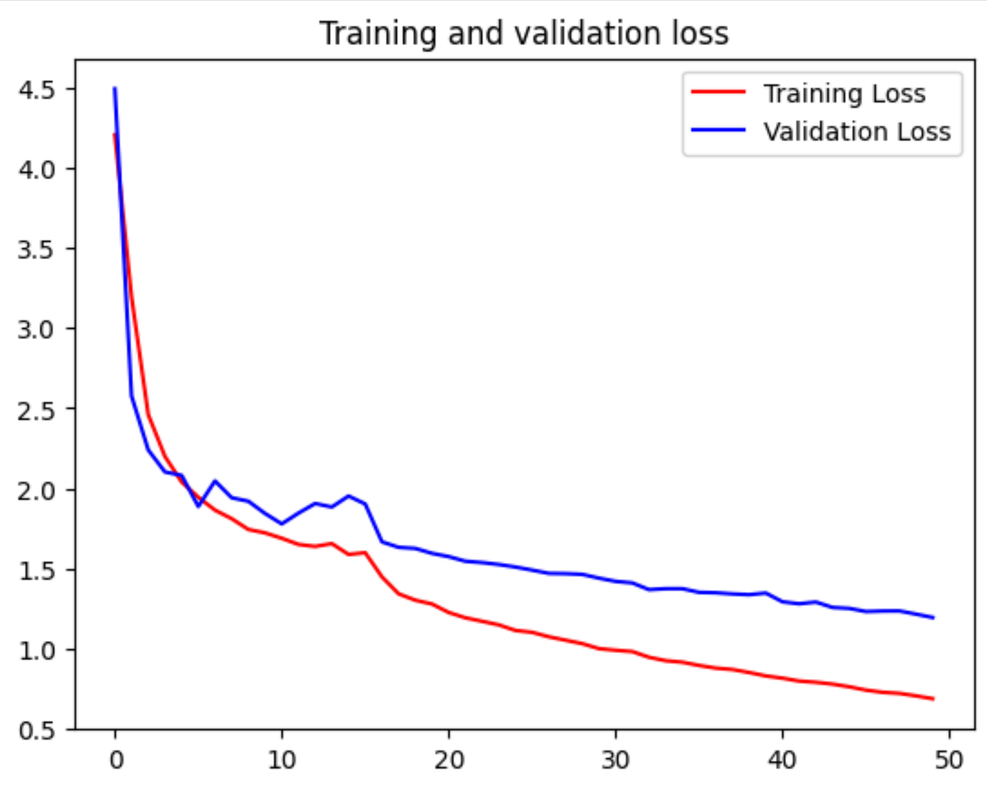
```python
def design_model2(input_shape):

    model = tf.keras.models.Sequential([

        tf.keras.layers.Conv2D(128, (3,3), activation='relu', input_shape
=input_shape, kernel_regularizer=tf.keras.regularizers.l2(0.0005)),
        tf.keras.layers.MaxPooling2D((3,3), strides=(2,2), padding='same'),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Dropout(0.2),  # reduced dropout rate for early
layers

        tf.keras.layers.Conv2D(256, (3,3), activation='relu',
kernel_regularizer=tf.keras.regularizers.l2(0.0005)),
        tf.keras.layers.MaxPooling2D((3,3), strides=(2,2), padding='same'),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Dropout(0.3),  # adjusted dropout

        tf.keras.layers.Conv2D(256, (2,2), activation='relu',
kernel_regularizer=tf.keras.regularizers.l2(0.0005)),
        tf.keras.layers.MaxPooling2D((2,2), strides=(2,2), padding='same'),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Dropout(0.4),

        tf.keras.layers.Flatten(),
        tf.keras.layers.Dense(512, activation='relu', kernel_regularizer=tf
.keras.regularizers.l2(0.0005)),
        tf.keras.layers.Dropout(0.5),

        tf.keras.layers.Dense(10, activation='softmax')
    ])

    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=0.001),
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    return model

# Set up callbacks
early_stopping = EarlyStopping(monitor='val_loss', patience=10,
restore_best_weights=True)
model_checkpoint = ModelCheckpoint('best_model.keras', save_best_only=True,
monitor='val_acc', mode='max')
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.1, patience=5,
min_lr=1e-6)

# Train the model
history = model.fit(Xtrain, ytrain, validation_data=(Xval, yval), epochs=50
, batch_size=32,
                    callbacks=[early_stopping, model_checkpoint, reduce_lr]
)
```

MODIFIED MODEL

RESULTS

- Val_Accuracy : around 80%





WEBSITE

- Two options to get an input
  - Upload file
  - Insert link
- Generate Top 5 Prediction

Music Genre Classification

Choose an input method:

Upload File

Choose an audio file

Drag and drop file here

Limit 200MB per file

Browse files

Music Genre Classification

Choose an input method:

Insert YouTube Link

Upload File

Insert YouTube Link

Enter Spotify URL

Music Genre Classification

Choose an input method:

Insert YouTube Link

Enter YouTube URL:

https://www.youtube.com/watch?v=3f2g4RMfhS0

The insert song link: AC/DC_AC/DC - Let There Be Rock (Official Video)

Top 5 Predicted Genres

classical: 46.66%

country: 19.77%

hiphop: 14.55%

rock: 7.65%

pop: 6.97%

RESOURCE LINKS

ORIGINAL MODEL

- https://www.kaggle.com/code/marchenrysaintfelix/music-genre-cnn-classifier-with-75-val-acc

PROJECT LINK

- https://github.com/AeliaWin/DeepLearning

THANK YOU