Music Genre Classification Using Spectrograms and Deep Learning

Configuration Manual
Data Analytics

Kenechukwu Otito Ajufo
Student ID: x19190174

School of Computing
National College of Ireland

Supervisor: Dr. Christian Horn

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Kenechukwu Otito Ajufo
Student ID:	x19190174
Programme:	Data Analytics
Year:	2020
Module:	Configuration Manual
Supervisor:	Dr. Christian Horn
Submission Due Date:	17/12/2020
Project Title:	Music Genre Classification Using Spectrograms and Deep Learning
Word Count:	8271
Page Count:	29

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	KOAjufo
Date:	16th December 2020

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Music Genre Classification Using Spectrograms and Deep Learning

Kenechukwu Otito Ajufo
x19190174

Abstract

This configuration manual provides step by step instructions on how to best replicate this research project. It begins with the overview of the hardware and software requirements used in running the research experiment. The code snippet of the audio processing and implementation of models are also included in the document.

1 Hardware and Software Requirements

1.1 Hardware Requirements

- Processor : AMD Ryzen 5 3500U with Radeon Vega Mobile Gfx @ 2.10 GHz
- Installed Memory (RAM): 8.00 GB
- Storage : 512 GB SSHD
- Operating System : Windows 10, 64-bit

1.2 Software Requirements

- Anaconda: This is an open-source package manager developed for Data Analysis, Data Science & Machine Learning.
- Jupyter Notebook: Python with the Jupyter Notebook is used for processing the audio data and implementation of the baseline and binary classification models.
- Google Colaboratory: The Hyperparameter Optimization experiments are performed on Google Colaboratory.
- Python Libraries: Keras, TensorFlow, NumPy, Librosa, Pandas, Matplotlib, Seaborn, Sklearn

2 Getting Started

After downloading the iPython Notebooks, the following steps help you run the experiments on your local machine.

2.1 Installation

To reproduce the results from the experiments on your local machine, you need to install the dependencies.

1. Anaconda: Please, install the suitable version for your machine. This is available online at https://www.anaconda.com/products/individual

Alternatively, with a Google account, you can head to https://colab.research.google.com/, select the Upload tab, choose the project iPython Notebooks, upload, and run the experiments.

2.2 Set Up

For the purpose of running the experiments on your local machine, the environment in Anaconda has to be set up properly. The following describe the steps to set up Anaconda and Google Colaboratory on your local machine. It also includes how to install the Python libraries that are used in the experiments.

2.2.1 Getting TensorFlow on Anaconda

- After Anaconda is installed, Open the Anaconda Navigator, Click on the Anaconda Command Prompt to run it.
- In the Anaconda Command Prompt, type **conda create -n tensorflow python=3.7**, this creates a new environment in Anaconda called TensorFlow.
- In the same window type **activate tensorflow** This activates the newly created environment.
- Run **conda install pandas matplotlib jupyter notebook scipy scikit-learn numpy librosa seaborn** to install all the python libraries used in the experiment.
- Follow the prompts. Be patient, the download may take a while.
- To install TensorFlow use the command **pip install tensorflow**
- To install Keras Tuner type **pip install -U keras-tuner**
- Finally, launch Jupyter Notebook by typing **jupyter notebook** in the same Anaconda Command Prompt window.
- The Jupyter Notebook application will launch on your default browser, please, navigate to where the Jupyter Notebooks are stored on your local machine. Click on them to open.
- Click on the first cell and hit the Run button.

2.2.2 Getting Started on Google Colab

- After opening https://colab.research.google.com/ and uploading the iPython Notebooks. Please add a new code cell above the Import Libraries cell.
- Install the libraries for the project using **!pip install pandas matplotlib scipy scikit-learn numpy librosa seaborn tensorflow keras-tuner**

- When this is done, click on the cell below, and run them one after the other.

3 Project Development

3.1 Dataset

The GTZAN dataset[1] is used in this research. It contains 1000 audio files. Each file is 30 seconds long, the dataset is spread across 10 genres with equal number of songs in each genre. This dataset is first used in the paper by (Tzanetakis and Cook; 2002). A supplementary dataset is obtained from Kaggle[2]

3.2 Exploratory Data Analysis

With the intention of understanding the dataset, the Wave plots of the first song in each genre are compute. The plots reveal the disparities of each genre.

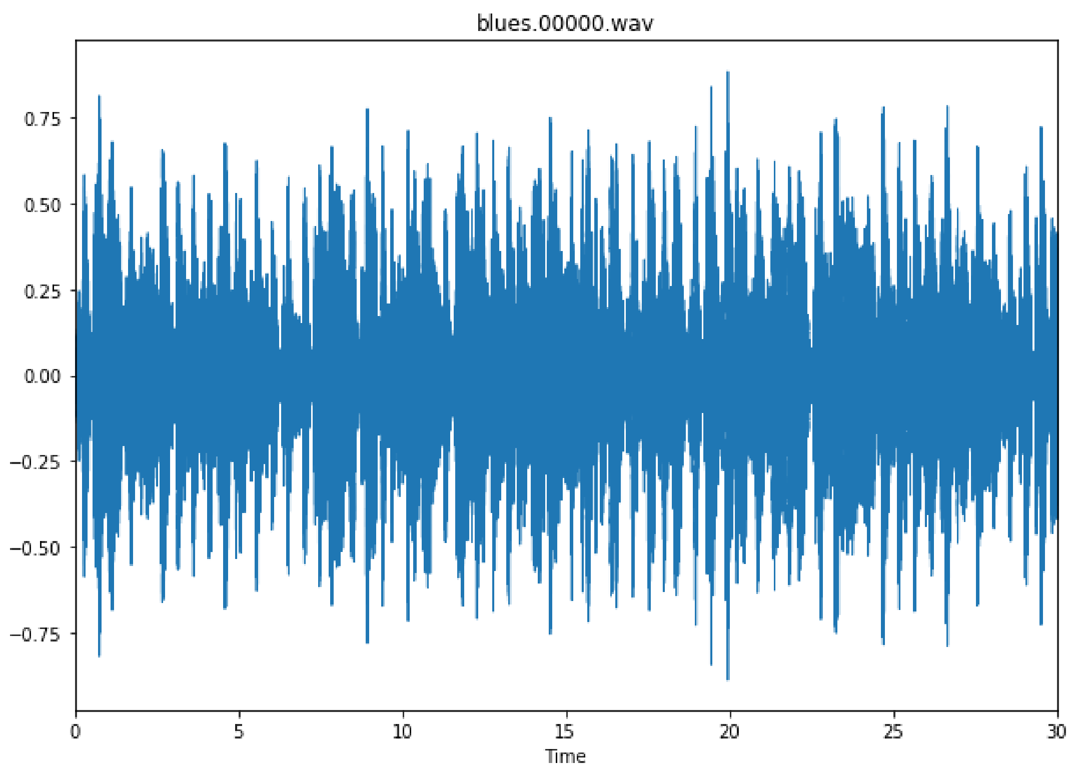
3.2.1 Plot Waveplot Function takes in a genre as an argument and plots the wave form for the first song in the genre class

```
[8]: def plot_waveplot(genre):

         # Loading in the audio file
         y, sr = librosa.core.load(f'D:/Data/genres_original/{genre}/{genre}.
     ↪00000.wav')

         # Computing and Plotting the waveplot
         plt.figure(figsize=(10,7))
         plt.title('{}.00000.wav'.format(genre))
         librosa.display.waveplot(y, sr=sr)
         plt.show()
```

3.2.2 Plot Waveplot for Blues

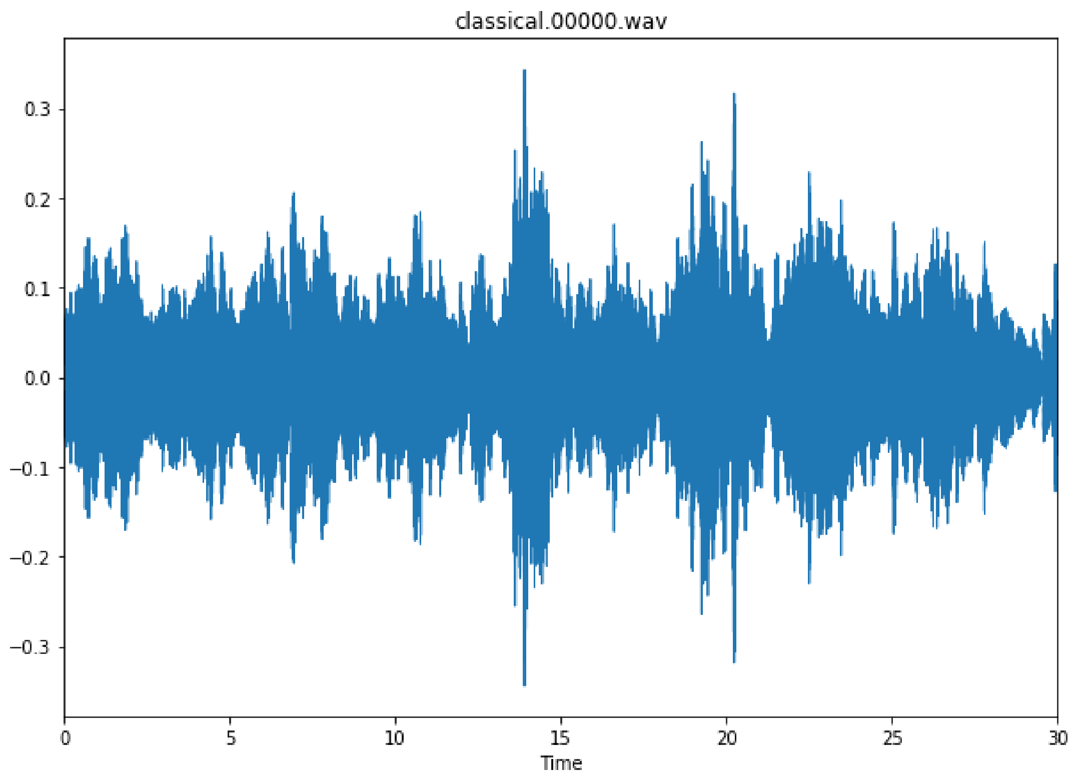
```
[9]: plot_waveplot('blues')
```

[1]http://marsyas.info/downloads/datasets.html
[2]https://www.kaggle.com/andradaolteanu/gtzan-dataset-music-genre-classification



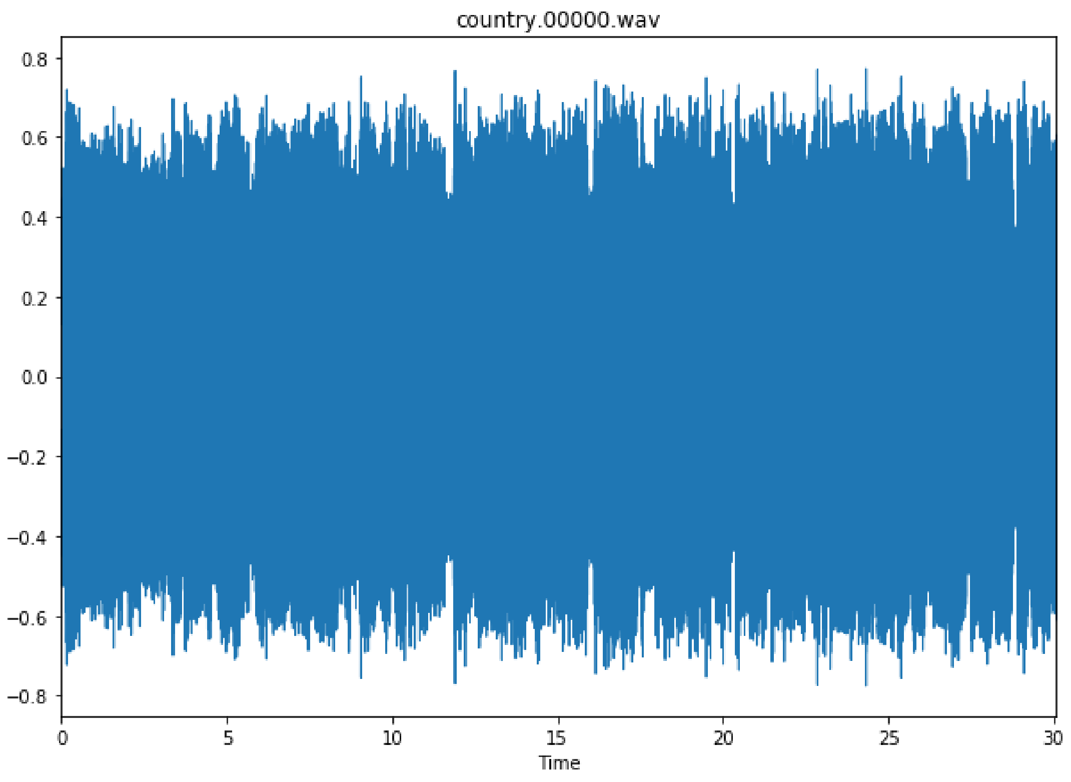
3.2.3 Plot Waveplot for Classical

```
[10]: plot_waveplot('classical')
```



3.2.4 Plot Waveplot for Country

[11]:
```
plot_waveplot('country')
```

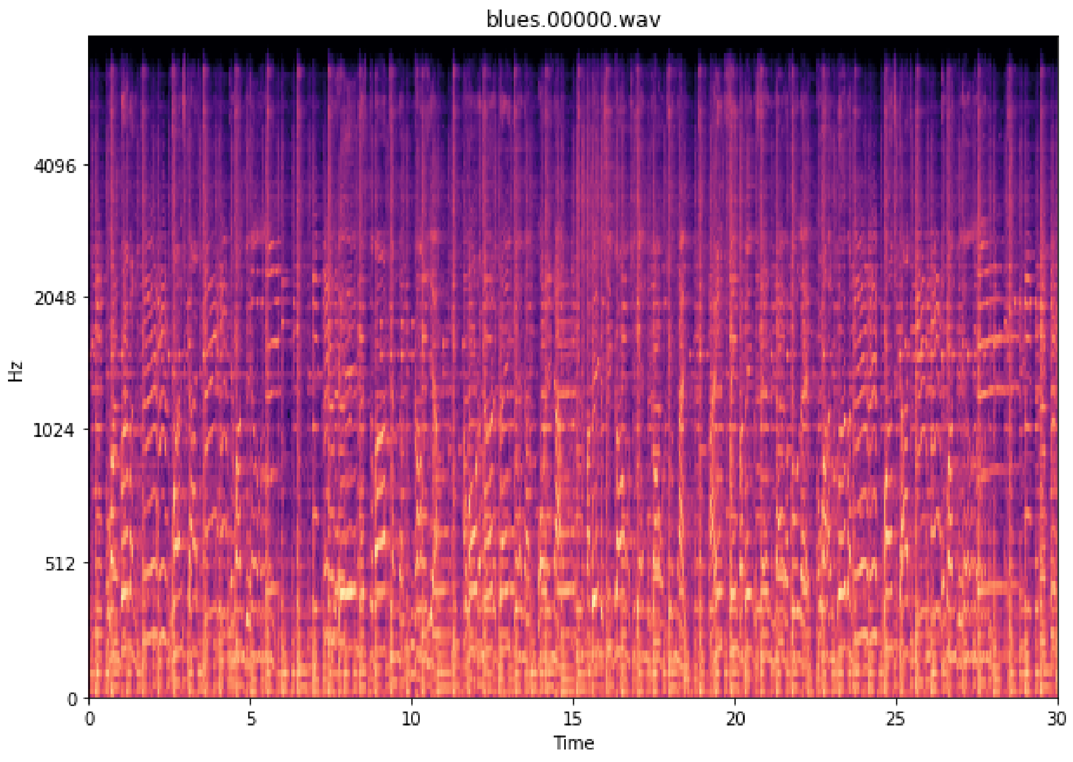


3.2.5 Plot Spectrogram Function takes in a genre as an argument and plots the mel spectrogram for the first song in the genre class

[19]:
```
def plot_spectrogram(genre):

    # Loading in the audio file
    y, sr = librosa.core.load(f'D:/Data/genres_original/{genre}/{genre}.
 ↪00000.wav')

    # Computing the spectrogram and transforming it to the decibal␣
 ↪scale
    spect = librosa.feature.melspectrogram(y=y, sr=sr)
    spect = librosa.power_to_db(spect, ref=np.max) # Converting to␣
 ↪decibels

    # Plotting the transformed spectrogram
    plt.figure(figsize=(10,7))
    librosa.display.specshow(spect, y_axis='mel', fmax=8000,␣
↪x_axis='time')
    plt.title('{}.00000.wav'.format(genre))
    plt.show()
```

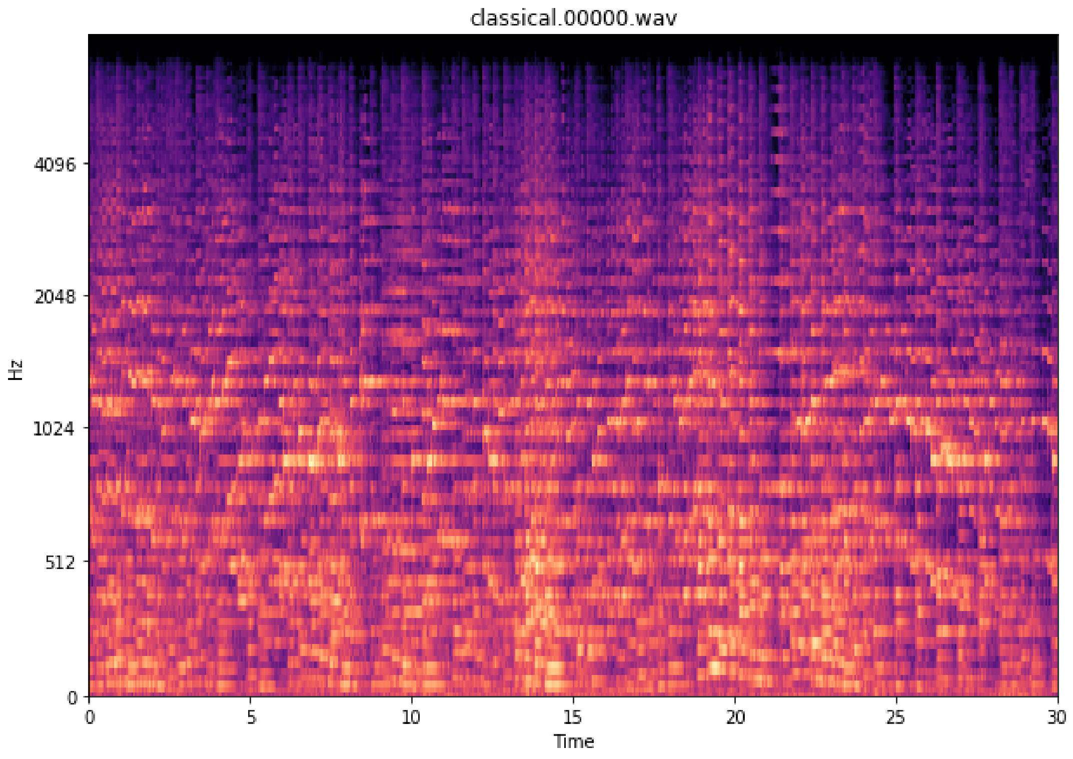
Spectrogram for Blues Genre

[20]:
```
plot_spectrogram('blues')
```



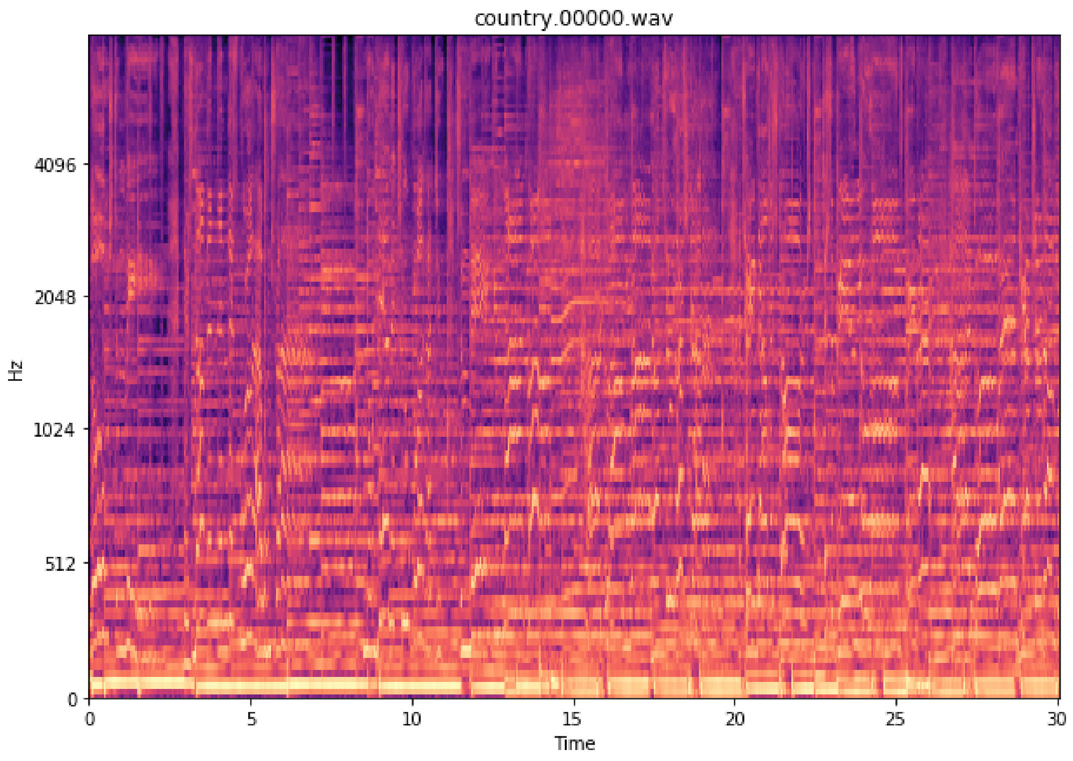
Spectrogram for Classical Genre

[21]:
```
plot_spectrogram('classical')
```



Spectrogram for Country Genre

[22]:
```
plot_spectrogram('country')
```



3.3 Data Preparation

3.3.1 Audio Processing

The audio data is processed using Librosa[3]. In this stage, the spectrogram images are generated for each song in the data. The Spectrogram images are produced at the sampling rate of 22050Hz, Fast Fourier transform (FFT) window length of 2048 and a hop length of 1024. The function to generate the Spectrograms is shown below.

3.3.2 generate_mel_spectrogram() function takes in the path containing the audio files and produces the spectrogram for the audio files, reshapes them so that they are all the same size, and saves them as a numpy array.

3.3.3 The function also creates a list of genre labels and maps them to numeric values.

Returns:

X, a Numpy array containing spectrogram data from the audio files in the directory

y is also a Numpy array of the genre labels

```
[ ]: def generate_mel_spectrogram(directory):

         # Creating empty lists for mel spectrograms and labels
         labels = []
         mel_specs = []


         # Looping through each file in the directory
         for file in os.scandir(directory):

             # Loading in the audio file
             y, sample_rate = librosa.core.load(file)
             h_length=1024
             fft=2048
             ref=np.max
             # Extracting the label and adding it to the list
             label = str(file).split('.')[0][11:]
             labels.append(label)
             # Computing the mel spectrograms
             spectogram = librosa.feature.melspectrogram(y=y,␣
      ↪sr=sample_rate, n_fft=fft, hop_length=h_length)
             spectogram = librosa.power_to_db(spectogram, ref=ref)

             # Adapting the size to be 128 x 660
```

[3] https://librosa.org/doc/latest/index.html

```
        if spectogram.shape[1] != 660:
            spectogram.resize(128,660, refcheck=False)

        # Adding the mel spectrogram to the list
        mel_specs.append(spectogram)

    # Converting the list or arrays to an array
    X = np.array(mel_specs)

    # Converting labels to numeric values
    labels = pd.Series(labels)
    label_dictionary = {
        'blues': 0,
        'classical': 1,
        'country': 2,
        'disco': 3,
        'hiphop': 4,
        'jazz': 5,
        'metal': 6,
        'pop': 7,
        'reggae': 8,
        'rock': 9
    }
    y = labels.map(label_dictionary).values

    # Returning the mel spectrograms and labels
    return X, y
```

The NumPy Arrays containing the mel spectrograms and labels are saved to the local machine.

3.3.4 Save Numpy NdArray to Local Machine

[ ]:
```
from numpy import save

save('mel_specs.npy', X)
```

[ ]:
```
from numpy import save

save('labels.npy', y)
```

3.3.5 Train/Test Split

The dataset is split into two sets. 80 percent for training and 20 percent for testing and validation. To ensure the results are reproducible random state is set to 42, the labels are also stratified. The code snippet for the Train/Test Split is shown below.

Train test split

```
[4]: X_train, X_test, y_train, y_test = train_test_split(X, y,␣
       ↪random_state=42, stratify=y, test_size=0.2)
```

3.4 Data Mining Algorithms

3.4.1 Baseline CNN Model

The baseline CNN model consists of two 2D convolutional layers, with each having a 2D Maxpooling layer. This is followed by a Flatten layer and a fully connected dense layer. The output layer is a Dense layer with 10 neurons and a Softmax activation function. The model architecture remain the same in the binary classification, the only change is made to the neurons in the output layer. The code snippet used to construct the baseline CNN model is shown below.

3.4.2 Define the Convolutional Neural Network Model

```
[9]: model = Sequential(name='CNN')
     model.add(Conv2D(32, (3, 3), activation='relu', input_shape=(128, 660,␣
       ↪1)))
     model.add(MaxPooling2D(pool_size=(2,4)))
     model.add(Conv2D(64, (3, 3), activation='relu'))
     model.add(MaxPooling2D(pool_size=(2,4)))
     model.add(Flatten())
     model.add(Dense(64, activation='relu'))
     model.add(Dense(10, activation='softmax'))
```

3.4.3 Compile the CNN Model

```
[10]: model.compile(loss='categorical_crossentropy', optimizer='adam',␣
        ↪metrics=['accuracy'])
```

3.4.4 CNN Model Summary

```
[11]: model.summary()
```

```
Model: "CNN"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 126, 658, 32)      320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 63, 164, 32)       0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 61, 162, 64)       18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 30, 40, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 76800)             0

_________________________________________________________________
dense (Dense)                (None, 64)                4915264

_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 4,934,730
Trainable params: 4,934,730
Non-trainable params: 0

_________________________________________________________________
```

3.4.5 Baseline CRNN Model

The baseline CRNN model has two units. In the Convolutional Unit, there are four 2D convolutional layers, with each have BatchNormalization, 2D Maxpooling and Dropout Layers. The last layer of the Convolutional Unit is reshaped and used as input to the two Gated Recurrent Unit layers. The output layer is a Dense layer with 10 neurons and a Softmax activation function. This model structure is inspired by CRNN model structure in the article by (Nasrullah and Zhao; 2019). In the binary classification experiment, the number of neurons in the output is reduced to two. The code snippet used to create the baseline CRNN model is shown below.

3.4.6 Define the Convolutional Recurrent Neural Network Model

[10]:
```
model = Sequential(name='CRNN')
model.add(Conv2D(64, (3, 3), activation='elu', input_shape=(128, 660,␣
 ↪1)))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(Dropout(0.1))
model.add(Conv2D(128, (3, 3), activation='elu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,4)))
model.add(Dropout(0.1))
model.add(Conv2D(128, (3, 3), activation='elu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,4)))
model.add(Dropout(0.1))
model.add(Conv2D(128, (3, 3), activation='elu'))
model.add(BatchNormalization())
model.add(MaxPooling2D(pool_size=(2,4)))
model.add(Dropout(0.1))
resize_shape = model.output_shape[2] * model.output_shape[3]
model.add(Reshape((model.output_shape[1], resize_shape)))
model.add(GRU(64, return_sequences=True))
model.add(GRU(64, return_sequences=False))
model.add(Dense(10, activation='softmax'))
```

3.4.7 Compile the CRNN Model

[11]:
```
model.compile(loss='categorical_crossentropy', optimizer='adam',␣
  ↪metrics=['accuracy'])
```

3.4.8 CRNN Model Summary

[12]:
```
model.summary()
```

```
Model: "CRNN"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 126, 658, 64)      640
_________________________________________________________________
batch_normalization (BatchNo (None, 126, 658, 64)      256
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 63, 329, 64)       0
_________________________________________________________________
dropout (Dropout)            (None, 63, 329, 64)       0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 61, 327, 128)      73856
_________________________________________________________________
batch_normalization_1 (Batch (None, 61, 327, 128)      512
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 30, 81, 128)       0
_________________________________________________________________
dropout_1 (Dropout)          (None, 30, 81, 128)       0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 28, 79, 128)       147584
_________________________________________________________________
batch_normalization_2 (Batch (None, 28, 79, 128)       512
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 14, 19, 128)       0
_________________________________________________________________
dropout_2 (Dropout)          (None, 14, 19, 128)       0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 12, 17, 128)       147584
_________________________________________________________________
batch_normalization_3 (Batch (None, 12, 17, 128)       512
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 6, 4, 128)         0
_________________________________________________________________
dropout_3 (Dropout)          (None, 6, 4, 128)         0
_________________________________________________________________
reshape (Reshape)            (None, 6, 512)            0
_________________________________________________________________
gru (GRU)                    (None, 6, 64)             110784
```

```
_________________________________________________________________
gru_1 (GRU)                  (None, 64)                24768
_________________________________________________________________
dense (Dense)                (None, 10)                650
=================================================================
Total params: 507,658
Trainable params: 506,762
Non-trainable params: 896
_________________________________________________________________
```

3.5 Hyperparameter Optimization

The Baseline Models are optimized in three experiments, first, the units in each layers are tuned, the second experiment explored tuning the activation functions in each layer. In the final experiment the units and activation layers are tuned. In all three experiments, the objective is to maximize validation accuracy. The hyperparameter optimization technique used is Random Search, this is implemented with the Keras Tuner[4] library. The code snippets below show the third hyperparameter optimization experiment for the two baseline models.

3.5.1 CNN Model

3.5.2 Define the Keras Tuner Hypermodels (CNN Model)

[ ]:
```python
from tensorflow import keras
from tensorflow.keras import layers
from kerastuner.tuners import RandomSearch
from kerastuner import HyperModel

class CNNHyperModel(HyperModel):
    def __init__(self, input_shape, num_classes):
        self.input_shape = input_shape
        self.num_classes = num_classes

    def build(self, hp):
        model = keras.Sequential()
        model.add(
            Conv2D(
                filters=hp.Int(name='units_1',
            min_value=16, max_value=128, step=16),
                kernel_size=(3,3),
                activation=hp.Choice(
                'Conv1_activation',
                values=['relu', 'tanh', 'elu', 'sigmoid'],
                default='relu'
                ),
                input_shape=self.input_shape
```

[4]https://keras-team.github.io/keras-tuner/

```
            )
        )
        model.add(MaxPooling2D(pool_size=(2,4)))
        model.add(
            Conv2D(
                filters=hp.Int(name='units_2',
            min_value=16, max_value=128, step=16),
                kernel_size=(3,3),
                activation=hp.Choice(
                    'Conv2_activation',
                    values=['relu', 'tanh', 'elu', 'sigmoid'],
                    default='relu'
                ),
                input_shape=self.input_shape
            )
        )
        model.add(MaxPooling2D(pool_size=(2,4)))
        model.add(Flatten())
        model.add(Dense(units=hp.Int(name='dense_units',
            min_value=16, max_value=128, step=16), activation=hp.Choice(
                    'dense_activation',
                    values=['relu', 'tanh', 'elu', 'sigmoid'],
                    default='relu'
                )))
        model.add(Dense(self.num_classes, activation='softmax'))

        model.compile(
            optimizer=keras.optimizers.Adam(
                hp.Float(
                    'learning_rate',
                    min_value=1e-4,
                    max_value=1e-2,
                    sampling='LOG',
                    default=1e-3
                )
            ),
            loss='categorical_crossentropy',
            metrics=['accuracy']
        )
        return model
```

3.5.3 Setting up Random Search Optimization for the CNN Model

[ ]:
```
NUM_CLASSES = 10
INPUT_SHAPE = (128, 660, 1)
SEED = 2
HYPERBAND_MAX_EPOCHS = 25
```

```
MAX_TRIALS = 20
EXECUTION_PER_TRIAL = 2

hypermodel = CNNHyperModel(input_shape=INPUT_SHAPE,␣
 ↪num_classes=NUM_CLASSES)

tuner = RandomSearch(
    hypermodel,
    objective='val_accuracy',
    seed=SEED,
    max_trials=MAX_TRIALS,
    executions_per_trial=EXECUTION_PER_TRIAL,
    directory='random_search',
    project_name='research_project'
)
```

CNN Model Search Space Summary

[ ]:
```
tuner.search_space_summary()
```

```
Search space summary
Default search space size: 7
units_1 (Int)
{'default': None, 'conditions': [], 'min_value': 16, 'max_value': 128,␣
 ↪'step':
16, 'sampling': None}
Conv1_activation (Choice)
{'default': 'relu', 'conditions': [], 'values': ['relu', 'tanh', 'elu',
'sigmoid'], 'ordered': False}
units_2 (Int)
{'default': None, 'conditions': [], 'min_value': 16, 'max_value': 128,␣
 ↪'step':
16, 'sampling': None}
Conv2_activation (Choice)
{'default': 'relu', 'conditions': [], 'values': ['relu', 'tanh', 'elu',
'sigmoid'], 'ordered': False}
dense_units (Int)
{'default': None, 'conditions': [], 'min_value': 16, 'max_value': 128,␣
 ↪'step':
16, 'sampling': None}
dense_activation (Choice)
{'default': 'relu', 'conditions': [], 'values': ['relu', 'tanh', 'elu',
'sigmoid'], 'ordered': False}
learning_rate (Float)
{'default': 0.001, 'conditions': [], 'min_value': 0.0001, 'max_value': 0.
 ↪01,
'step': None, 'sampling': 'log'}
```

3.5.4 CNN Units & Activation Functions Hyperparameter Tuning

[ ]:
```
N_EPOCH_SEARCH = 25

tuner.search(X_train, y_train, epochs=N_EPOCH_SEARCH,␣
 ↪validation_split=0.1)
```

```
Trial 20 Complete [00h 01m 06s]
val_accuracy: 0.08749999850988388

Best val_accuracy So Far: 0.65625
Total elapsed time: 00h 20m 29s
INFO:tensorflow:Oracle triggered exit
```

3.5.5 Show a summary of the Random Search for the CNN Model

[ ]:
```
tuner.results_summary()
```

```
Results summary
Results in random_search/research_project
Showing 10 best trials
Objective(name='val_accuracy', direction='max')
Trial summary
Hyperparameters:
units_1: 128
Conv1_activation: tanh
units_2: 48
Conv2_activation: elu
dense_units: 112
dense_activation: elu
learning_rate: 0.00028406107009737317
Score: 0.65625
Trial summary
Hyperparameters:
units_1: 32
Conv1_activation: tanh
units_2: 16
Conv2_activation: elu
dense_units: 112
dense_activation: relu
learning_rate: 0.0001227867741533623
Score: 0.6375000178813934
Trial summary
Hyperparameters:
units_1: 64
Conv1_activation: relu
units_2: 32
Conv2_activation: sigmoid
```

```
dense_units: 96
dense_activation: relu
learning_rate: 0.00044412650444693207
Score: 0.637499988079071
Trial summary
Hyperparameters:
units_1: 48
Conv1_activation: tanh
units_2: 48
Conv2_activation: sigmoid
dense_units: 112
dense_activation: sigmoid
learning_rate: 0.00011510936029076842
Score: 0.2812500074505806
Trial summary
Hyperparameters:
units_1: 64
Conv1_activation: sigmoid
units_2: 32
Conv2_activation: relu
dense_units: 112
dense_activation: tanh
learning_rate: 0.008249331566936491
Score: 0.13750000298023224
Trial summary
Hyperparameters:
units_1: 112
Conv1_activation: tanh
units_2: 64
Conv2_activation: relu
dense_units: 32
dense_activation: relu
learning_rate: 0.0006951671294413008
Score: 0.13124999776482582
Trial summary
Hyperparameters:
units_1: 48
Conv1_activation: elu
units_2: 112
Conv2_activation: tanh
dense_units: 96
dense_activation: elu
learning_rate: 0.0010712131812413617
Score: 0.125
Trial summary
Hyperparameters:
units_1: 128
Conv1_activation: elu
```

```
units_2: 80
Conv2_activation: elu
dense_units: 64
dense_activation: sigmoid
learning_rate: 0.0009294745699713808
Score: 0.11875000223517418
Trial summary
Hyperparameters:
units_1: 80
Conv1_activation: relu
units_2: 32
Conv2_activation: tanh
dense_units: 80
dense_activation: elu
learning_rate: 0.0004543609940965399
Score: 0.11250000074505806
Trial summary
Hyperparameters:
units_1: 80
Conv1_activation: tanh
units_2: 48
Conv2_activation: sigmoid
dense_units: 112
dense_activation: sigmoid
learning_rate: 0.0008960175671873151
Score: 0.11250000074505806
```

3.5.6 CRNN Model

3.5.7 Define the Keras Tuner Hypermodels (CRNN Model)

[ ]:

```
from tensorflow import keras
from tensorflow.keras import layers
from kerastuner.tuners import RandomSearch
from kerastuner import HyperModel

class CRNNHyperModel(HyperModel):
    def __init__(self, input_shape, num_classes):
        self.input_shape = input_shape
        self.num_classes = num_classes

    def build(self, hp):
        model = keras.Sequential()
        model.add(
            Conv2D(
                filters=hp.Int(name='conv_units_1',
            min_value=16, max_value=128, step=16),
                kernel_size=(3,3),
                activation=hp.Choice(
                    'Conv1_activation',
                    values=['relu', 'tanh', 'elu', 'sigmoid'],
                    default='elu'
                ),
                input_shape=self.input_shape
        ))
        model.add(BatchNormalization())
        model.add(MaxPooling2D(pool_size=(2,2)))
        model.add(Dropout(0.1))
        model.add(
            Conv2D(
                filters=hp.Int(name='conv_units_2', min_value=16,␣
↪max_value=128, step=16),
                kernel_size=(3,3),
                activation=hp.Choice(
                    'Conv2_activation',
                    values=['relu', 'tanh', 'elu', 'sigmoid'],
                    default='elu'
                ),
                input_shape=self.input_shape
            )
        )
        model.add(BatchNormalization())
        model.add(MaxPooling2D(pool_size=(2,2)))
        model.add(Dropout(0.1))
        model.add(
            Conv2D(
                filters=hp.Int(name='conv_units_3', min_value=16,␣
↪max_value=128, step=16),
                kernel_size=(3,3),
                activation=hp.Choice(
                    'Conv3_activation',
                    values=['relu', 'tanh', 'elu', 'sigmoid'],
                    default='elu'
                ),
                input_shape=self.input_shape
            )
        )
        model.add(BatchNormalization())
        model.add(MaxPooling2D(pool_size=(2,2)))
        model.add(Dropout(0.1))
        model.add(
            Conv2D(
                filters=hp.Int(name='conv_units_4', min_value=16,␣
↪max_value=128, step=16),
                kernel_size=(3,3),
                activation=hp.Choice(
                    'Conv4_activation',
                    values=['relu', 'tanh', 'elu', 'sigmoid'],
                    default='elu'
                ),
                input_shape=self.input_shape
            )
        )
        model.add(BatchNormalization())
        model.add(MaxPooling2D(pool_size=(2,2)))
        model.add(Dropout(0.1))
        resize_shape = model.output_shape[2] * model.output_shape[3]
        model.add(Reshape((model.output_shape[1], resize_shape)))
        model.add(GRU(units=hp.Int(name='gru_units_1', min_value=16,␣
↪max_value=128, step=16), return_sequences=True, activation=hp.Choice(
                    'GRU1_activation',
                    values=['relu', 'tanh', 'elu', 'sigmoid'],
                    default='relu'
                )))
        model.add(GRU(units=hp.Int(name='gru_units_2', min_value=16,␣
↪max_value=128, step=16), return_sequences=False, activation=hp.
↪Choice(
                    'GRU2_activation',
                    values=['relu', 'tanh', 'elu', 'sigmoid'],
                    default='relu'
                )))
        model.add(Dense(self.num_classes, activation='softmax'))

        model.compile(
            optimizer=keras.optimizers.Adam(
                hp.Float(
                    'learning_rate',
                    min_value=1e-4,
                    max_value=1e-2,
                    sampling='LOG',
                    default=1e-3
                )
            ),
            loss='categorical_crossentropy',
            metrics=['accuracy']
        )
        return model
```

3.5.8 Setting up Random Search Optimization for the CRNN Model

```
[ ]: NUM_CLASSES = 10
     INPUT_SHAPE = (128, 660, 1)
     SEED = 2
     HYPERBAND_MAX_EPOCHS = 25
     MAX_TRIALS = 20
     EXECUTION_PER_TRIAL = 2

     hypermodel = CRNNHyperModel(input_shape=INPUT_SHAPE,␣
      ↪num_classes=NUM_CLASSES)

     tuner = RandomSearch(
         hypermodel,
         objective='val_accuracy',
         seed=SEED,
         max_trials=MAX_TRIALS,
         executions_per_trial=EXECUTION_PER_TRIAL,
         directory='random_search',
         project_name='research_project'
     )
```

CRNN Search Space Summary

```
[ ]: tuner.search_space_summary()
```

```
Search space summary
Default search space size: 13
conv_units_1 (Int)
{'default': None, 'conditions': [], 'min_value': 16, 'max_value': 128,␣
 ↪'step':
16, 'sampling': None}
Conv1_activation (Choice)
{'default': 'elu', 'conditions': [], 'values': ['relu', 'tanh', 'elu',
'sigmoid'], 'ordered': False}
conv_units_2 (Int)
{'default': None, 'conditions': [], 'min_value': 16, 'max_value': 128,␣
 ↪'step':
16, 'sampling': None}
Conv2_activation (Choice)
{'default': 'elu', 'conditions': [], 'values': ['relu', 'tanh', 'elu',
'sigmoid'], 'ordered': False}
conv_units_3 (Int)
{'default': None, 'conditions': [], 'min_value': 16, 'max_value': 128,␣
 ↪'step':
16, 'sampling': None}
Conv3_activation (Choice)
{'default': 'elu', 'conditions': [], 'values': ['relu', 'tanh', 'elu',
'sigmoid'], 'ordered': False}
conv_units_4 (Int)
{'default': None, 'conditions': [], 'min_value': 16, 'max_value': 128,␣
 ↪'step':
16, 'sampling': None}
Conv4_activation (Choice)
{'default': 'elu', 'conditions': [], 'values': ['relu', 'tanh', 'elu',
'sigmoid'], 'ordered': False}
gru_units_1 (Int)
{'default': None, 'conditions': [], 'min_value': 16, 'max_value': 128,␣
 ↪'step':
16, 'sampling': None}
GRU1_activation (Choice)
{'default': 'relu', 'conditions': [], 'values': ['relu', 'tanh', 'elu',
'sigmoid'], 'ordered': False}
gru_units_2 (Int)
{'default': None, 'conditions': [], 'min_value': 16, 'max_value': 128,␣
 ↪'step':
16, 'sampling': None}
GRU2_activation (Choice)
{'default': 'relu', 'conditions': [], 'values': ['relu', 'tanh', 'elu',
'sigmoid'], 'ordered': False}
learning_rate (Float)
{'default': 0.001, 'conditions': [], 'min_value': 0.0001, 'max_value': 0.
 ↪01,
'step': None, 'sampling': 'log'}
```

3.5.9 CRNN Units & Activation Functions Hyperparameter tuning

[ ]:
```
N_EPOCH_SEARCH = 25

tuner.search(X_train, y_train, epochs=N_EPOCH_SEARCH,␣
 ↪validation_split=0.1)
```

```
Trial 20 Complete [00h 06m 01s]
val_accuracy: 0.574999988079071

Best val_accuracy So Far: 0.643750011920929
Total elapsed time: 02h 48m 04s
INFO:tensorflow:Oracle triggered exit
```

3.5.10 Show a summary of the Random Search for the CRNN Model

[ ]:
```
tuner.results_summary()
```

```
Results summary
Results in random_search/research_project
Showing 10 best trials
```

```
Objective(name='val_accuracy', direction='max')
Trial summary
Hyperparameters:
conv_units_1: 48
Conv1_activation: relu
conv_units_2: 96
Conv2_activation: sigmoid
conv_units_3: 64
Conv3_activation: tanh
conv_units_4: 80
Conv4_activation: tanh
gru_units_1: 112
GRU1_activation: elu
gru_units_2: 96
GRU2_activation: elu
learning_rate: 0.0004938970115853091
Score: 0.643750011920929
Trial summary
Hyperparameters:
conv_units_1: 64
Conv1_activation: sigmoid
conv_units_2: 64
Conv2_activation: elu
conv_units_3: 80
Conv3_activation: relu
conv_units_4: 32
Conv4_activation: relu
gru_units_1: 96
GRU1_activation: relu
gru_units_2: 80
GRU2_activation: tanh
learning_rate: 0.0007880149255876653
Score: 0.612500011920929
Trial summary
Hyperparameters:
conv_units_1: 32
Conv1_activation: relu
conv_units_2: 80
Conv2_activation: relu
conv_units_3: 32
Conv3_activation: tanh
conv_units_4: 80
Conv4_activation: elu
gru_units_1: 48
GRU1_activation: relu
gru_units_2: 80
GRU2_activation: tanh
learning_rate: 0.0008264439766792489
```

```
Score: 0.59375
Trial summary
Hyperparameters:
conv_units_1: 80
Conv1_activation: relu
conv_units_2: 48
Conv2_activation: relu
conv_units_3: 16
Conv3_activation: sigmoid
conv_units_4: 32
Conv4_activation: sigmoid
gru_units_1: 64
GRU1_activation: relu
gru_units_2: 80
GRU2_activation: elu
learning_rate: 0.001300398010371077
Score: 0.581250011920929
Trial summary
Hyperparameters:
conv_units_1: 48
Conv1_activation: sigmoid
conv_units_2: 64
Conv2_activation: relu
conv_units_3: 80
Conv3_activation: elu
conv_units_4: 16
Conv4_activation: sigmoid
gru_units_1: 48
GRU1_activation: relu
gru_units_2: 16
GRU2_activation: tanh
learning_rate: 0.0007358058711559769
Score: 0.574999988079071
Trial summary
Hyperparameters:
conv_units_1: 112
Conv1_activation: relu
conv_units_2: 128
Conv2_activation: relu
conv_units_3: 80
Conv3_activation: tanh
conv_units_4: 48
Conv4_activation: sigmoid
gru_units_1: 112
GRU1_activation: sigmoid
gru_units_2: 64
GRU2_activation: sigmoid
learning_rate: 0.0001385536126653817
```

```
Score: 0.5687499940395355
Trial summary
Hyperparameters:
conv_units_1: 96
Conv1_activation: elu
conv_units_2: 48
Conv2_activation: relu
conv_units_3: 64
Conv3_activation: relu
conv_units_4: 128
Conv4_activation: relu
gru_units_1: 112
GRU1_activation: tanh
gru_units_2: 80
GRU2_activation: sigmoid
learning_rate: 0.00016439503999433
Score: 0.5562500059604645
Trial summary
Hyperparameters:
conv_units_1: 128
Conv1_activation: tanh
conv_units_2: 48
Conv2_activation: elu
conv_units_3: 112
Conv3_activation: elu
conv_units_4: 48
Conv4_activation: sigmoid
gru_units_1: 80
GRU1_activation: sigmoid
gru_units_2: 64
GRU2_activation: elu
learning_rate: 0.00016355245798526342
Score: 0.550000011920929
Trial summary
Hyperparameters:
conv_units_1: 48
Conv1_activation: sigmoid
conv_units_2: 32
Conv2_activation: sigmoid
conv_units_3: 96
Conv3_activation: tanh
conv_units_4: 112
Conv4_activation: tanh
gru_units_1: 64
GRU1_activation: relu
gru_units_2: 32
GRU2_activation: relu
learning_rate: 0.0006951671294413008
```

```
Score: 0.5312500149011612
Trial summary
Hyperparameters:
conv_units_1: 48
Conv1_activation: relu
conv_units_2: 48
Conv2_activation: elu
conv_units_3: 112
Conv3_activation: sigmoid
conv_units_4: 32
Conv4_activation: tanh
gru_units_1: 64
GRU1_activation: sigmoid
gru_units_2: 112
GRU2_activation: sigmoid
learning_rate: 0.001989751755243559
Score: 0.5
```

4 Evaluation

4.1 Baseline Models Confusion Matrices

4.1.1 Baseline CNN Confusion Matrix

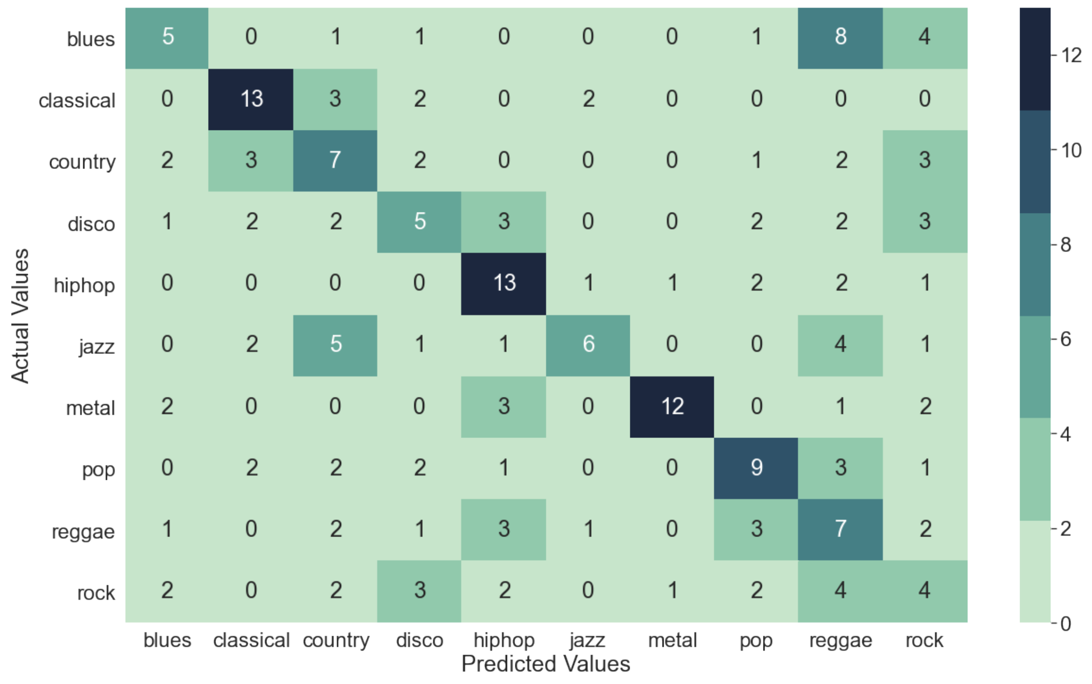
4.1.2 CNN Confusion Matrix

[18]:
```
conf_matrix = confusion_matrix(np.argmax(y_test, 1), np.
 ↪argmax(predictions, 1))
conf_matrix
```

[18]:
```
array([[ 5,  0,  1,  1,  0,  0,  0,  1,  8,  4],
       [ 0, 13,  3,  2,  0,  2,  0,  0,  0,  0],
       [ 2,  3,  7,  2,  0,  0,  0,  1,  2,  3],
       [ 1,  2,  2,  5,  3,  0,  0,  2,  2,  3],
       [ 0,  0,  0,  0, 13,  1,  1,  2,  2,  1],
       [ 0,  2,  5,  1,  1,  6,  0,  0,  4,  1],
       [ 2,  0,  0,  0,  3,  0, 12,  0,  1,  2],
       [ 0,  2,  2,  2,  1,  0,  0,  9,  3,  1],
       [ 1,  0,  2,  1,  3,  1,  0,  3,  7,  2],
       [ 2,  0,  2,  3,  2,  0,  1,  2,  4,  4]], dtype=int64)
```

Creating a heatmap for the CNN confusion matrix for display

[23]:
```
plt.figure(figsize= (20,12))
sns.set(font_scale = 2);
ax = sns.heatmap(confusion_df, annot=True, cmap=sns.
 ↪cubehelix_palette(rot=-.4));
ax.set(xlabel='Predicted Values', ylabel='Actual Values');
```



4.1.3 Baseline CRNN Confusion Matrix

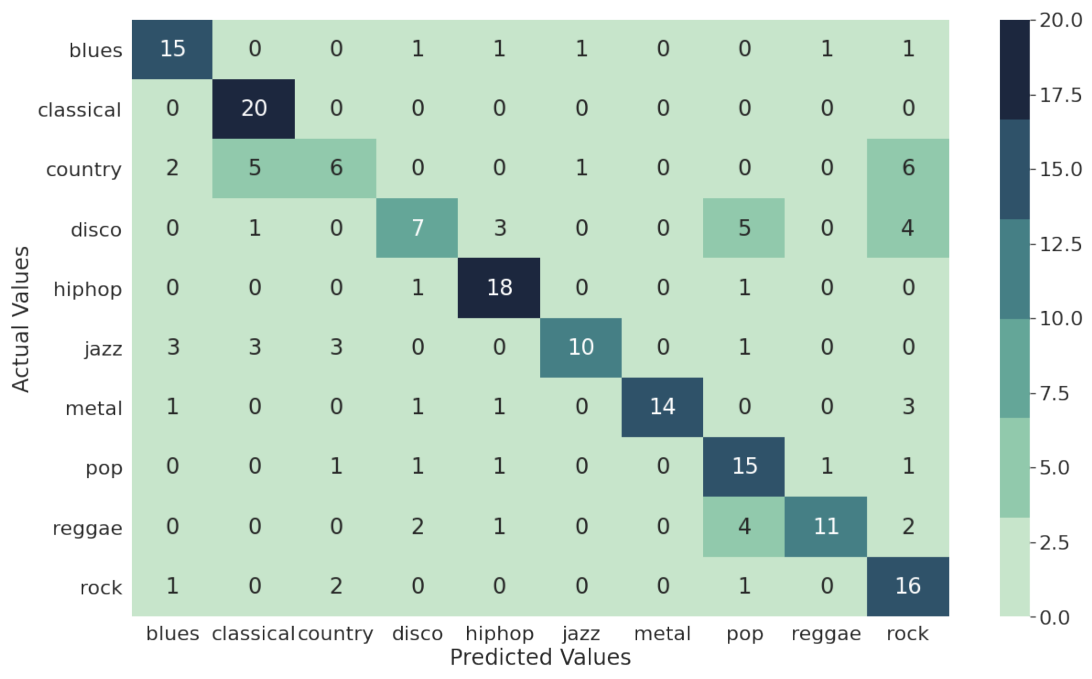
4.1.4 CRNN Confusion Matrix

```
[19]: conf_matrix = confusion_matrix(np.argmax(y_test, 1), np.
        ↪argmax(predictions, 1))
      conf_matrix
```

```
[19]: array([[15,  0,  0,  1,  1,  1,  0,  0,  1,  1],
             [ 0, 20,  0,  0,  0,  0,  0,  0,  0,  0],
             [ 2,  5,  6,  0,  0,  1,  0,  0,  0,  6],
             [ 0,  1,  0,  7,  3,  0,  0,  5,  0,  4],
             [ 0,  0,  0,  1, 18,  0,  0,  1,  0,  0],
             [ 3,  3,  3,  0,  0, 10,  0,  1,  0,  0],
             [ 1,  0,  0,  1,  1,  0, 14,  0,  0,  3],
             [ 0,  0,  1,  1,  1,  0,  0, 15,  1,  1],
             [ 0,  0,  0,  2,  1,  0,  0,  4, 11,  2],
             [ 1,  0,  2,  0,  0,  0,  0,  1,  0, 16]])
```

Creating a heatmap for the CRNN confusion matrix for display

```
[24]: plt.figure(figsize= (20,12))
      sns.set(font_scale = 2);
      ax = sns.heatmap(confusion_df, annot=True, cmap=sns.
        ↪cubehelix_palette(rot=-.4));
      ax.set(xlabel='Predicted Values', ylabel='Actual Values');
```



	blues	classical	country	disco	hiphop	jazz	metal	pop	reggae	rock
blues	15	0	0	1	1	1	0	0	1	1
classical	0	20	0	0	0	0	0	0	0	0
country	2	5	6	0	0	1	0	0	0	6
disco	0	1	0	7	3	0	0	5	0	4
hiphop	0	0	0	1	18	0	0	1	0	0
jazz	3	3	3	0	0	10	0	1	0	0
metal	1	0	0	1	1	0	14	0	0	3
pop	0	0	1	1	1	0	0	15	1	1
reggae	0	0	0	2	1	0	0	4	11	2
rock	1	0	2	0	0	0	0	1	0	16

4.2 Baseline Models Classification Report

4.2.1 Baseline CNN Classification Report

[24]:
```
from sklearn.metrics import classification_report

print(classification_report(np.argmax(y_test, 1),  np.
 ↪argmax(predictions, 1)))
```

```
              precision    recall  f1-score   support

           0       0.38      0.25      0.30        20
           1       0.59      0.65      0.62        20
           2       0.29      0.35      0.32        20
           3       0.29      0.25      0.27        20
           4       0.50      0.65      0.57        20
           5       0.60      0.30      0.40        20
           6       0.86      0.60      0.71        20
           7       0.45      0.45      0.45        20
           8       0.21      0.35      0.26        20
           9       0.19      0.20      0.20        20

    accuracy                           0.41       200
   macro avg       0.44      0.40      0.41       200
weighted avg       0.44      0.41      0.41       200
```

4.2.2 Baseline CRNN Classification Report

```
[25]: from sklearn.metrics import classification_report

      print(classification_report(np.argmax(y_test, 1),  np.
        ↪argmax(predictions, 1)))
```

```
              precision    recall  f1-score   support

           0       0.68      0.75      0.71        20
           1       0.69      1.00      0.82        20
           2       0.50      0.30      0.37        20
           3       0.54      0.35      0.42        20
           4       0.72      0.90      0.80        20
           5       0.83      0.50      0.62        20
           6       1.00      0.70      0.82        20
           7       0.56      0.75      0.64        20
           8       0.85      0.55      0.67        20
           9       0.48      0.80      0.60        20

    accuracy                           0.66       200
   macro avg       0.68      0.66      0.65       200
weighted avg       0.68      0.66      0.65       200
```

References

Nasrullah, Z. and Zhao, Y. (2019). Music Artist Classification with Convolutional Recurrent Neural Networks, *2019 International Joint Conference on Neural Networks (IJCNN)*, IEEE, pp. 1–8. DOI:https://doi.org/10.1109/IJCNN.2019.8851988.

Tzanetakis, G. and Cook, P. (2002). Musical genre classification of audio signals, *IEEE Transactions on speech and audio processing* **10**(5): 293–302. DOI:https://doi.org/10.1109/TSA.2002.800560.