Mel-Spectrogram-Only (3 secs) CNN

March 21, 2025

1 CNN for Mel-Spectrogram (3 secs)

1.1 1 - All the imports

[1]:
```
import os
import numpy as np
from sklearn.model_selection import train_test_split
import tensorflow as tf
```

```
2025-03-21 00:50:05.068849: I tensorflow/core/platform/cpu_feature_guard.cc:210]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild
TensorFlow with the appropriate compiler flags.
```

1.2 2 - Put the data within the model

[2]:
```
# Import a single image and save it to be read by the model

image = os.path.join('blues.00000.png')

# Load the image
image = tf.io.read_file(image)

# Convert to a numpy array
image = tf.image.decode_png(image, channels=1)
image = tf.image.convert_image_dtype(image, tf.float32)
image = tf.image.resize(image, [256, 256])
image = image.numpy()
```

2 3 - Create the model

[3]:
```
from tensorflow.keras import models
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
  ↪Dropout, Normalization

model = models.Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(256, 256, 1)),
    Normalization(),
    MaxPooling2D((2, 2)),

    Conv2D(64, (3, 3), activation='relu'),
    Normalization(),
    MaxPooling2D((2, 2)),

    Conv2D(128, (3, 3), activation='relu'),
    Normalization(),
    MaxPooling2D((2, 2)),

    Conv2D(256, (3, 3), activation='relu'),
    Normalization(),
    MaxPooling2D((2, 2)),

    Flatten(),

    Dense(512, activation='relu'),
    Dropout(0.5),

    Dense(256, activation='relu'),
    Dropout(0.5),

    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])
```

```
/opt/conda/lib/python3.12/site-
packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

[4]:
```
model.summary()
```

```
Model: "sequential"


 Layer (type)                    Output Shape                  Param #

 conv2d (Conv2D)                 (None, 254, 254, 32)              320

 normalization (Normalization)   (None, 254, 254, 32)               65

 max_pooling2d (MaxPooling2D)    (None, 127, 127, 32)                0
```

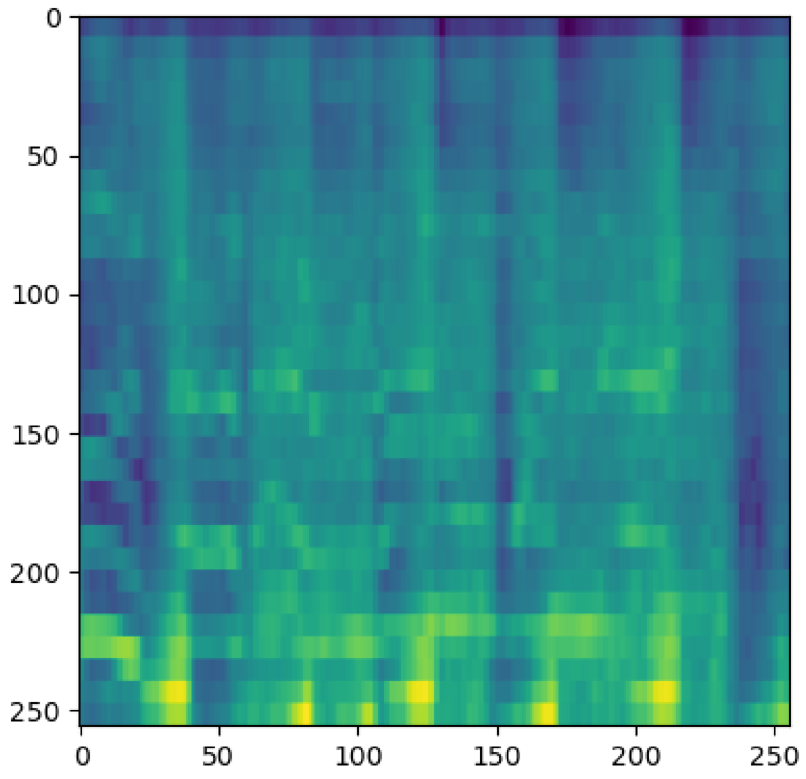
```
conv2d_1 (Conv2D)                 (None, 125, 125, 64)            18,496

normalization_1 (Normalization)   (None, 125, 125, 64)               129

max_pooling2d_1 (MaxPooling2D)    (None, 62, 62, 64)                   0

conv2d_2 (Conv2D)                 (None, 60, 60, 128)             73,856

normalization_2 (Normalization)   (None, 60, 60, 128)                257

max_pooling2d_2 (MaxPooling2D)    (None, 30, 30, 128)                  0

conv2d_3 (Conv2D)                 (None, 28, 28, 256)            295,168

normalization_3 (Normalization)   (None, 28, 28, 256)                513

max_pooling2d_3 (MaxPooling2D)    (None, 14, 14, 256)                  0

flatten (Flatten)                 (None, 50176)                        0

dense (Dense)                     (None, 512)                 25,690,624

dropout (Dropout)                 (None, 512)                          0

dense_1 (Dense)                   (None, 256)                    131,328

dropout_1 (Dropout)               (None, 256)                          0

dense_2 (Dense)                   (None, 128)                     32,896

dense_3 (Dense)                   (None, 10)                       1,290


Total params: 26,244,942 (100.12 MB)

Trainable params: 26,243,978 (100.11 MB)

Non-trainable params: 964 (3.78 KB)
```

3 4 - Load the images

```
[5]: import os
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split

# Augmentation function
def augment_image(image):
    image = tf.image.random_flip_left_right(image)
    image = tf.image.random_brightness(image, max_delta=0.1)
    image = tf.image.random_contrast(image, 0.8, 1.2)
    return image

# Define the genres and file paths
GENRES = ['blues', 'classical', 'country', 'disco', 'hiphop', 'jazz', 'metal',␣
 ↪'pop', 'reggae', 'rock']
FILE_PATH = os.path.join('Data', 'mel_spectrograms (3 secs)',␣
 ↪'mel_spectrogram_32')
X = []
y = []

GENRE_TO_INDEX = {genre: index for index, genre in enumerate(GENRES)}

# Loop through the genres and load the images with augmentation
for genre in GENRES:
    genre_dir = os.path.join(FILE_PATH, genre)
    print(f"Going through {genre}")
    for file in os.listdir(genre_dir):
        image = tf.io.read_file(os.path.join(genre_dir, file))
        image = tf.image.decode_png(image, channels=1)
        image = tf.image.convert_image_dtype(image, tf.float32)
        image = tf.image.resize(image, [256, 256])  # Resize to 256x256
        image = augment_image(image)  # Apply augmentation
        image = image.numpy()  # Convert to numpy array
        X.append(image)
        y.append(GENRE_TO_INDEX[genre])

# Convert lists to numpy arrays
X = np.array(X)
y = np.array(y)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

```
Going through blues
Going through classical
Going through country
Going through disco
Going through hiphop
Going through jazz
Going through metal
Going through pop
Going through reggae
Going through rock
```

[6]:
```python
# Show image as a sanity check
import matplotlib.pyplot as plt
plt.imshow(X_train[22].reshape(256, 256))
plt.show()
```



3.1 5 - Compile the model

[7]:
```python
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau

model.compile(optimizer=Adam(learning_rate=0.0001),␣
 ↪loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

```
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3,␣
  ↪min_lr=1e-6)
```

3.2 6 - Fit the model

```
[8]: model.fit(X_train, y_train, epochs=20, validation_data=(X_test, y_test),␣
       ↪batch_size=32, callbacks=[reduce_lr])
```

```
Epoch 1/20
250/250              830s 3s/step -
accuracy: 0.1181 - loss: 2.2824 - val_accuracy: 0.3290 - val_loss: 1.9061 -
learning_rate: 1.0000e-04
Epoch 2/20
250/250              712s 3s/step -
accuracy: 0.2743 - loss: 1.9318 - val_accuracy: 0.4310 - val_loss: 1.6012 -
learning_rate: 1.0000e-04
Epoch 3/20
250/250              626s 3s/step -
accuracy: 0.3719 - loss: 1.7169 - val_accuracy: 0.4710 - val_loss: 1.4633 -
learning_rate: 1.0000e-04
Epoch 4/20
250/250              714s 3s/step -
accuracy: 0.4479 - loss: 1.5282 - val_accuracy: 0.5485 - val_loss: 1.2960 -
learning_rate: 1.0000e-04
Epoch 5/20
250/250              526s 2s/step -
accuracy: 0.4976 - loss: 1.4125 - val_accuracy: 0.5570 - val_loss: 1.2938 -
learning_rate: 1.0000e-04
Epoch 6/20
250/250              461s 2s/step -
accuracy: 0.5229 - loss: 1.3421 - val_accuracy: 0.6175 - val_loss: 1.1246 -
learning_rate: 1.0000e-04
Epoch 7/20
250/250              483s 2s/step -
accuracy: 0.5596 - loss: 1.2555 - val_accuracy: 0.6045 - val_loss: 1.1433 -
learning_rate: 1.0000e-04
Epoch 8/20
250/250              444s 2s/step -
accuracy: 0.5734 - loss: 1.2133 - val_accuracy: 0.6375 - val_loss: 1.0453 -
learning_rate: 1.0000e-04
Epoch 9/20
250/250              437s 2s/step -
accuracy: 0.6044 - loss: 1.1334 - val_accuracy: 0.6340 - val_loss: 1.0793 -
learning_rate: 1.0000e-04
Epoch 10/20
250/250              425s 2s/step -
accuracy: 0.6247 - loss: 1.1012 - val_accuracy: 0.6705 - val_loss: 0.9742 -
learning_rate: 1.0000e-04
```

```
Epoch 11/20
250/250          466s 2s/step -
accuracy: 0.6575 - loss: 0.9958 - val_accuracy: 0.6665 - val_loss: 0.9736 -
learning_rate: 1.0000e-04
Epoch 12/20
250/250          438s 2s/step -
accuracy: 0.6716 - loss: 0.9424 - val_accuracy: 0.6880 - val_loss: 0.9243 -
learning_rate: 1.0000e-04
Epoch 13/20
250/250          371s 1s/step -
accuracy: 0.7040 - loss: 0.8489 - val_accuracy: 0.6835 - val_loss: 0.9218 -
learning_rate: 1.0000e-04
Epoch 14/20
250/250          343s 1s/step -
accuracy: 0.7258 - loss: 0.7881 - val_accuracy: 0.6875 - val_loss: 0.9188 -
learning_rate: 1.0000e-04
Epoch 15/20
250/250          344s 1s/step -
accuracy: 0.7435 - loss: 0.7375 - val_accuracy: 0.7185 - val_loss: 0.8566 -
learning_rate: 1.0000e-04
Epoch 16/20
250/250          342s 1s/step -
accuracy: 0.7725 - loss: 0.6650 - val_accuracy: 0.7100 - val_loss: 0.8820 -
learning_rate: 1.0000e-04
Epoch 17/20
250/250          345s 1s/step -
accuracy: 0.7985 - loss: 0.5842 - val_accuracy: 0.7145 - val_loss: 0.9150 -
learning_rate: 1.0000e-04
Epoch 18/20
250/250          345s 1s/step -
accuracy: 0.8125 - loss: 0.5475 - val_accuracy: 0.7275 - val_loss: 0.8871 -
learning_rate: 1.0000e-04
Epoch 19/20
250/250          343s 1s/step -
accuracy: 0.8446 - loss: 0.4566 - val_accuracy: 0.7420 - val_loss: 0.8679 -
learning_rate: 5.0000e-05
Epoch 20/20
250/250          345s 1s/step -
accuracy: 0.8661 - loss: 0.4052 - val_accuracy: 0.7325 - val_loss: 0.8763 -
learning_rate: 5.0000e-05
```

```
[8]: <keras.src.callbacks.history.History at 0x7f1878728c50>
```

3.3 7 - Check the accuracy

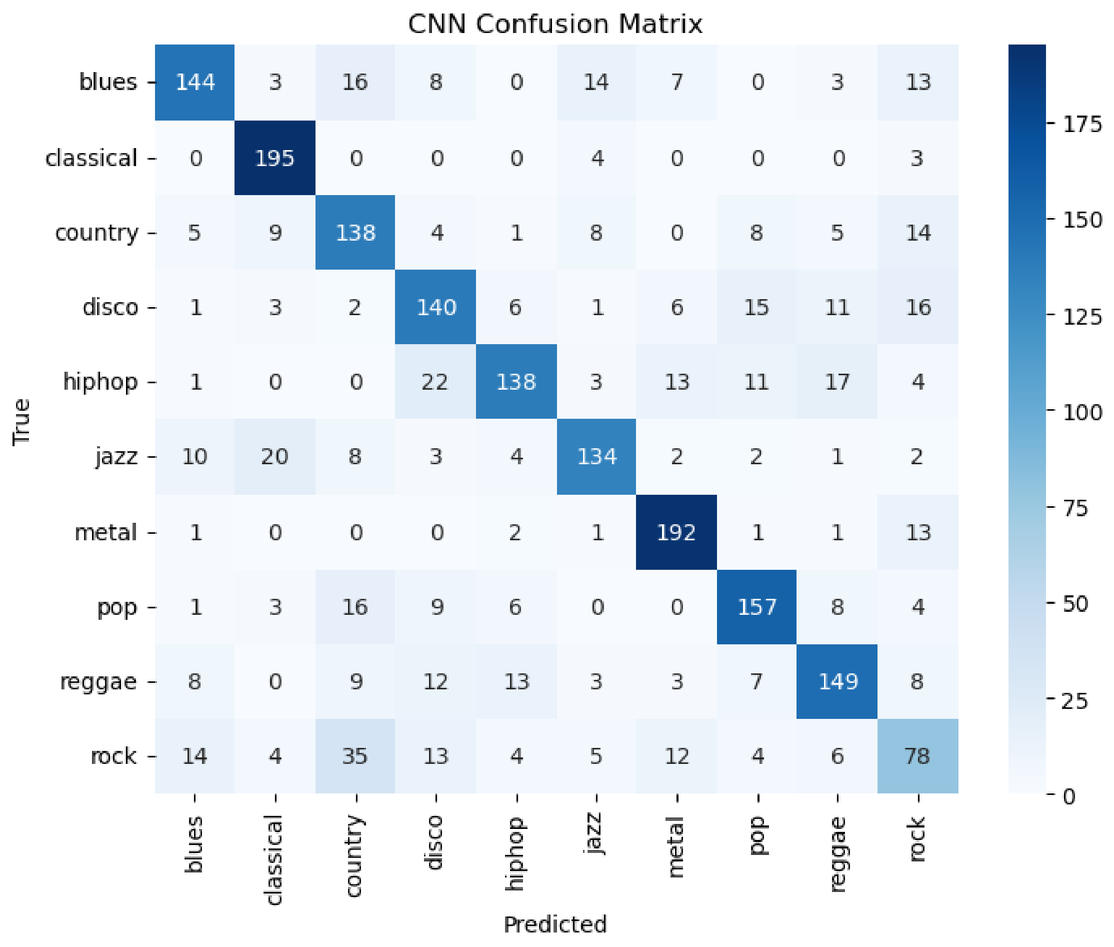
[9]:
```
evaluation = model.evaluate(X_test, y_test)
print(f"Test accuracy: {evaluation[1]:.3f}")
```

```
63/63                19s 298ms/step -
accuracy: 0.7450 - loss: 0.8492
Test accuracy: 0.733
```

4 8 - Apply the confusion matrix after the model

[10]:
```
import seaborn as sns
# from sklearn.metrics import confusion
import numpy as NP
from sklearn.metrics import confusion_matrix

cnn_preds = np.argmax(model.predict(X_test), axis=1)
cnn_cm = confusion_matrix(y_test, cnn_preds)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cnn_cm, annot=True, fmt="d", cmap="Blues", xticklabels=GENRES,␣
  ↪yticklabels=GENRES)
plt.title("CNN Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()
```

```
63/63                19s 298ms/step
```



True \ Predicted	blues	classical	country	disco	hiphop	jazz	metal	pop	reggae	rock
blues	144	3	16	8	0	14	7	0	3	13
classical	0	195	0	0	0	4	0	0	0	3
country	5	9	138	4	1	8	0	8	5	14
disco	1	3	2	140	6	1	6	15	11	16
hiphop	1	0	0	22	138	3	13	11	17	4
jazz	10	20	8	3	4	134	2	2	1	2
metal	1	0	0	0	2	1	192	1	1	13
pop	1	3	16	9	6	0	0	157	8	4
reggae	8	0	9	12	13	3	3	7	149	8
rock	14	4	35	13	4	5	12	4	6	78

4.1 9 - Limited Genres Easy (metal and classical)

[11]:
```python
import os
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split

# Augmentation function
def augment_image(image):
    image = tf.image.random_flip_left_right(image)
    image = tf.image.random_brightness(image, max_delta=0.1)
    image = tf.image.random_contrast(image, 0.8, 1.2)
    return image

# Define the genres and file paths
GENRES = ['classical', 'metal']
```

```python
FILE_PATH = os.path.join('Data', 'mel_spectrograms (3 secs)',␣
↪'mel_spectrogram_32')
X = []
y = []

GENRE_TO_INDEX = {genre: index for index, genre in enumerate(GENRES)}

# Loop through the genres and load the images with augmentation
for genre in GENRES:
    genre_dir = os.path.join(FILE_PATH, genre)
    print(f"Going through {genre}")
    for file in os.listdir(genre_dir):
        image = tf.io.read_file(os.path.join(genre_dir, file))
        image = tf.image.decode_png(image, channels=1)
        image = tf.image.convert_image_dtype(image, tf.float32)
        image = tf.image.resize(image, [256, 256])  # Resize to 256x256
        image = augment_image(image)  # Apply augmentation
        image = image.numpy()  # Convert to numpy array
        X.append(image)
        y.append(GENRE_TO_INDEX[genre])

# Convert lists to numpy arrays
X = np.array(X)
y = np.array(y)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

from tensorflow.keras import models
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
↪Dropout, Normalization

model = models.Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(256, 256, 1)),
    Normalization(),
    MaxPooling2D((2, 2)),

    Conv2D(64, (3, 3), activation='relu'),
    Normalization(),
    MaxPooling2D((2, 2)),

    Conv2D(128, (3, 3), activation='relu'),
    Normalization(),
    MaxPooling2D((2, 2)),

    Conv2D(256, (3, 3), activation='relu'),
    Normalization(),
    MaxPooling2D((2, 2)),

    Flatten(),

    Dense(512, activation='relu'),
    Dropout(0.5),

    Dense(256, activation='relu'),
    Dropout(0.5),

    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])

from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau

model.compile(optimizer=Adam(learning_rate=0.0001),␣
 ↪loss='sparse_categorical_crossentropy', metrics=['accuracy'])
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3,␣
 ↪min_lr=1e-6)

model.fit(X_train, y_train, epochs=20, validation_data=(X_test, y_test),␣
 ↪batch_size=32, callbacks=[reduce_lr])

evaluation = model.evaluate(X_test, y_test)
print(f"Test accuracy: {evaluation[1]:.3f}")
```

```
Going through classical
Going through metal
```

```
/opt/conda/lib/python3.12/site-
packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
Epoch 1/20
50/50                 76s 1s/step -
accuracy: 0.4199 - loss: 1.5496 - val_accuracy: 0.8700 - val_loss: 0.5522 -
learning_rate: 1.0000e-04
Epoch 2/20
50/50                 70s 1s/step -
accuracy: 0.6751 - loss: 0.6745 - val_accuracy: 0.8875 - val_loss: 0.2671 -
learning_rate: 1.0000e-04
Epoch 3/20
```

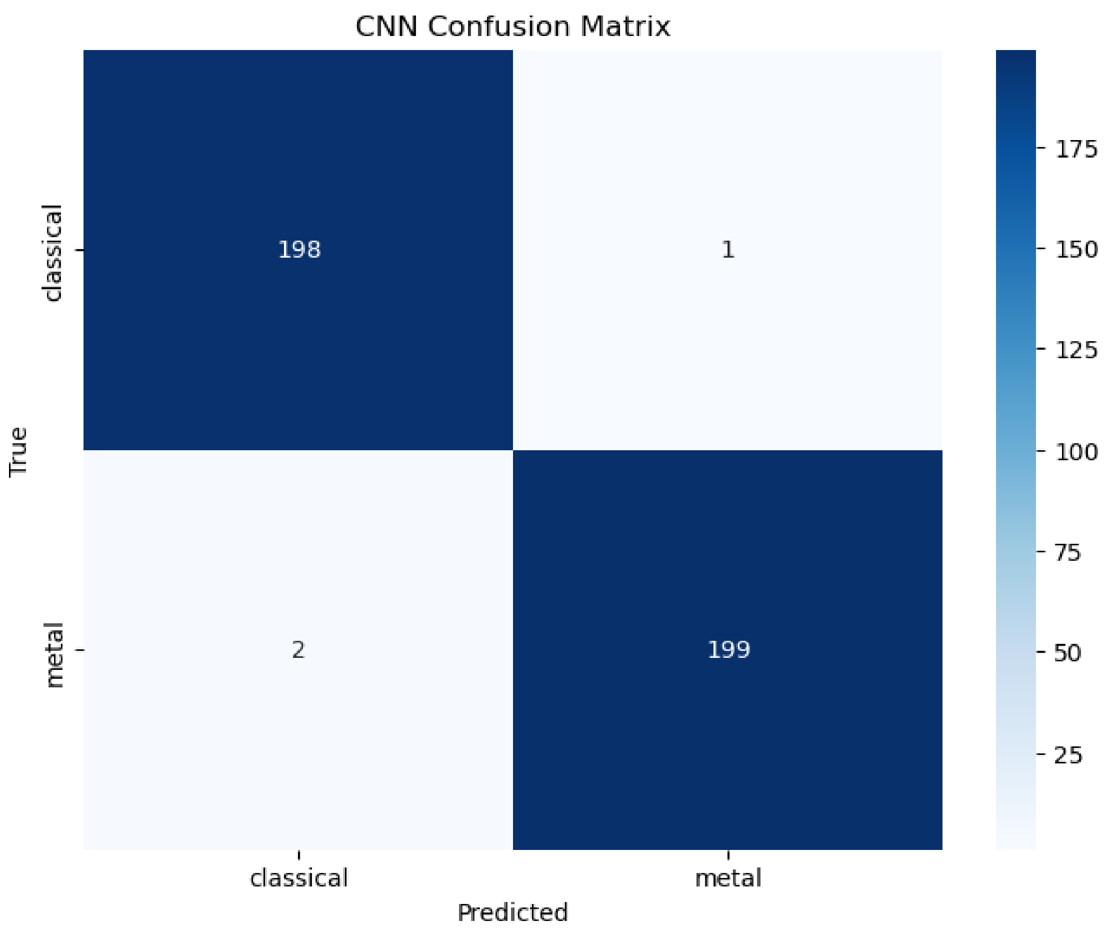
```
50/50           70s 1s/step -
accuracy: 0.8319 - loss: 0.4381 - val_accuracy: 0.9375 - val_loss: 0.1656 -
learning_rate: 1.0000e-04
Epoch 4/20
50/50           70s 1s/step -
accuracy: 0.9056 - loss: 0.2454 - val_accuracy: 0.9750 - val_loss: 0.0729 -
learning_rate: 1.0000e-04
Epoch 5/20
50/50           70s 1s/step -
accuracy: 0.9626 - loss: 0.1381 - val_accuracy: 0.9775 - val_loss: 0.0592 -
learning_rate: 1.0000e-04
Epoch 6/20
50/50           70s 1s/step -
accuracy: 0.9630 - loss: 0.1106 - val_accuracy: 0.9775 - val_loss: 0.0660 -
learning_rate: 1.0000e-04
Epoch 7/20
50/50           69s 1s/step -
accuracy: 0.9744 - loss: 0.0731 - val_accuracy: 0.9600 - val_loss: 0.1002 -
learning_rate: 1.0000e-04
Epoch 8/20
50/50           70s 1s/step -
accuracy: 0.9689 - loss: 0.0846 - val_accuracy: 0.9825 - val_loss: 0.0407 -
learning_rate: 1.0000e-04
Epoch 9/20
50/50           69s 1s/step -
accuracy: 0.9865 - loss: 0.0384 - val_accuracy: 0.9875 - val_loss: 0.0254 -
learning_rate: 1.0000e-04
Epoch 10/20
50/50           69s 1s/step -
accuracy: 0.9907 - loss: 0.0299 - val_accuracy: 0.9900 - val_loss: 0.0280 -
learning_rate: 1.0000e-04
Epoch 11/20
50/50           82s 1s/step -
accuracy: 0.9873 - loss: 0.0519 - val_accuracy: 0.9875 - val_loss: 0.0361 -
learning_rate: 1.0000e-04
Epoch 12/20
50/50           69s 1s/step -
accuracy: 0.9888 - loss: 0.0287 - val_accuracy: 0.9925 - val_loss: 0.0280 -
learning_rate: 1.0000e-04
Epoch 13/20
50/50           68s 1s/step -
accuracy: 0.9929 - loss: 0.0252 - val_accuracy: 0.9925 - val_loss: 0.0275 -
learning_rate: 5.0000e-05
Epoch 14/20
50/50           68s 1s/step -
accuracy: 0.9927 - loss: 0.0200 - val_accuracy: 0.9925 - val_loss: 0.0277 -
learning_rate: 5.0000e-05
Epoch 15/20
```

```
50/50              70s 1s/step -
accuracy: 0.9957 - loss: 0.0198 - val_accuracy: 0.9950 - val_loss: 0.0220 -
learning_rate: 5.0000e-05
Epoch 16/20
50/50              68s 1s/step -
accuracy: 0.9969 - loss: 0.0144 - val_accuracy: 0.9900 - val_loss: 0.0303 -
learning_rate: 5.0000e-05
Epoch 17/20
50/50              68s 1s/step -
accuracy: 0.9907 - loss: 0.0216 - val_accuracy: 0.9925 - val_loss: 0.0346 -
learning_rate: 5.0000e-05
Epoch 18/20
50/50              68s 1s/step -
accuracy: 0.9943 - loss: 0.0191 - val_accuracy: 0.9950 - val_loss: 0.0224 -
learning_rate: 5.0000e-05
Epoch 19/20
50/50              69s 1s/step -
accuracy: 0.9968 - loss: 0.0086 - val_accuracy: 0.9925 - val_loss: 0.0216 -
learning_rate: 2.5000e-05
Epoch 20/20
50/50              69s 1s/step -
accuracy: 0.9972 - loss: 0.0098 - val_accuracy: 0.9925 - val_loss: 0.0304 -
learning_rate: 2.5000e-05
13/13              4s 299ms/step -
accuracy: 0.9948 - loss: 0.0235
Test accuracy: 0.993
```

4.2 10 - Confusion Matrix Easy (classical and metal)

[12]:
```python
import seaborn as sns
# from sklearn.metrics import confusion
import numpy as NP
from sklearn.metrics import confusion_matrix

cnn_preds = np.argmax(model.predict(X_test), axis=1)
cnn_cm = confusion_matrix(y_test, cnn_preds)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cnn_cm, annot=True, fmt="d", cmap="Blues", xticklabels=GENRES,␣
 ↪yticklabels=GENRES)
plt.title("CNN Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()
```

```
13/13              4s 312ms/step
```



4.3 11 - Limited genres Hard (disco and pop)

[13]:
```
import os
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split

# Augmentation function
def augment_image(image):
    image = tf.image.random_flip_left_right(image)
    image = tf.image.random_brightness(image, max_delta=0.1)
    image = tf.image.random_contrast(image, 0.8, 1.2)
    return image

# Define the genres and file paths
GENRES = ['disco', 'pop']
```

```python
FILE_PATH = os.path.join('Data', 'mel_spectrograms (3 secs)',␣
↪'mel_spectrogram_32')
X = []
y = []

GENRE_TO_INDEX = {genre: index for index, genre in enumerate(GENRES)}

# Loop through the genres and load the images with augmentation
for genre in GENRES:
    genre_dir = os.path.join(FILE_PATH, genre)
    print(f"Going through {genre}")
    for file in os.listdir(genre_dir):
        image = tf.io.read_file(os.path.join(genre_dir, file))
        image = tf.image.decode_png(image, channels=1)
        image = tf.image.convert_image_dtype(image, tf.float32)
        image = tf.image.resize(image, [256, 256])  # Resize to 256x256
        image = augment_image(image)  # Apply augmentation
        image = image.numpy()  # Convert to numpy array
        X.append(image)
        y.append(GENRE_TO_INDEX[genre])

# Convert lists to numpy arrays
X = np.array(X)
y = np.array(y)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)

from tensorflow.keras import models
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
↪Dropout, Normalization

model = models.Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(256, 256, 1)),
    Normalization(),
    MaxPooling2D((2, 2)),

    Conv2D(64, (3, 3), activation='relu'),
    Normalization(),
    MaxPooling2D((2, 2)),

    Conv2D(128, (3, 3), activation='relu'),
    Normalization(),
    MaxPooling2D((2, 2)),

    Conv2D(256, (3, 3), activation='relu'),
    Normalization(),
    MaxPooling2D((2, 2)),

    Flatten(),

    Dense(512, activation='relu'),
    Dropout(0.5),

    Dense(256, activation='relu'),
    Dropout(0.5),

    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])

from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau

model.compile(optimizer=Adam(learning_rate=0.0001),␣
 ↪loss='sparse_categorical_crossentropy', metrics=['accuracy'])
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3,␣
 ↪min_lr=1e-6)

model.fit(X_train, y_train, epochs=20, validation_data=(X_test, y_test),␣
 ↪batch_size=32, callbacks=[reduce_lr])

evaluation = model.evaluate(X_test, y_test)
print(f"Test accuracy: {evaluation[1]:.3f}")
```

```
Going through disco
Going through pop
```

```
/opt/conda/lib/python3.12/site-
packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
Epoch 1/20
50/50                 75s 1s/step -
accuracy: 0.4177 - loss: 1.5275 - val_accuracy: 0.5025 - val_loss: 0.7135 -
learning_rate: 1.0000e-04
Epoch 2/20
50/50                 71s 1s/step -
accuracy: 0.5026 - loss: 0.8607 - val_accuracy: 0.5025 - val_loss: 0.8029 -
learning_rate: 1.0000e-04
Epoch 3/20
```

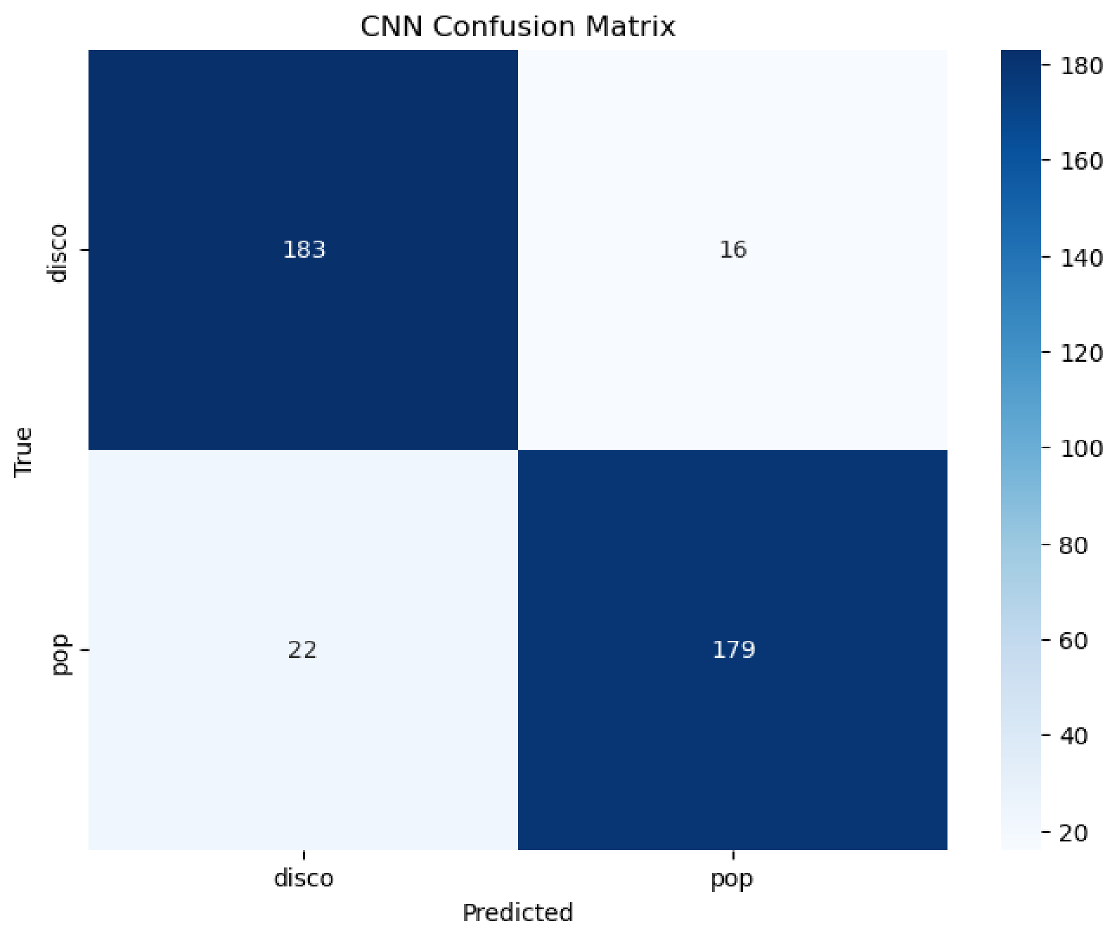
```
50/50             71s 1s/step -
accuracy: 0.5607 - loss: 0.7479 - val_accuracy: 0.7225 - val_loss: 0.5487 -
learning_rate: 1.0000e-04
Epoch 4/20
50/50             70s 1s/step -
accuracy: 0.6589 - loss: 0.6701 - val_accuracy: 0.7500 - val_loss: 0.5436 -
learning_rate: 1.0000e-04
Epoch 5/20
50/50             70s 1s/step -
accuracy: 0.7177 - loss: 0.5688 - val_accuracy: 0.8175 - val_loss: 0.3993 -
learning_rate: 1.0000e-04
Epoch 6/20
50/50             70s 1s/step -
accuracy: 0.7772 - loss: 0.4741 - val_accuracy: 0.8375 - val_loss: 0.3298 -
learning_rate: 1.0000e-04
Epoch 7/20
50/50             69s 1s/step -
accuracy: 0.7930 - loss: 0.4201 - val_accuracy: 0.8275 - val_loss: 0.3146 -
learning_rate: 1.0000e-04
Epoch 8/20
50/50             69s 1s/step -
accuracy: 0.8362 - loss: 0.3568 - val_accuracy: 0.8550 - val_loss: 0.2882 -
learning_rate: 1.0000e-04
Epoch 9/20
50/50             69s 1s/step -
accuracy: 0.8348 - loss: 0.3579 - val_accuracy: 0.8525 - val_loss: 0.2892 -
learning_rate: 1.0000e-04
Epoch 10/20
50/50             68s 1s/step -
accuracy: 0.8586 - loss: 0.3028 - val_accuracy: 0.9000 - val_loss: 0.2413 -
learning_rate: 1.0000e-04
Epoch 11/20
50/50             68s 1s/step -
accuracy: 0.8636 - loss: 0.2856 - val_accuracy: 0.8900 - val_loss: 0.2489 -
learning_rate: 1.0000e-04
Epoch 12/20
50/50             70s 1s/step -
accuracy: 0.8699 - loss: 0.2868 - val_accuracy: 0.8925 - val_loss: 0.2449 -
learning_rate: 1.0000e-04
Epoch 13/20
50/50             69s 1s/step -
accuracy: 0.8836 - loss: 0.2921 - val_accuracy: 0.8925 - val_loss: 0.2369 -
learning_rate: 1.0000e-04
Epoch 14/20
50/50             68s 1s/step -
accuracy: 0.8798 - loss: 0.2897 - val_accuracy: 0.9000 - val_loss: 0.2240 -
learning_rate: 1.0000e-04
Epoch 15/20
```

```
50/50             68s 1s/step -
accuracy: 0.8868 - loss: 0.2604 - val_accuracy: 0.8825 - val_loss: 0.2319 -
learning_rate: 1.0000e-04
Epoch 16/20
50/50             68s 1s/step -
accuracy: 0.8904 - loss: 0.2518 - val_accuracy: 0.8875 - val_loss: 0.2423 -
learning_rate: 1.0000e-04
Epoch 17/20
50/50             69s 1s/step -
accuracy: 0.8943 - loss: 0.2336 - val_accuracy: 0.9025 - val_loss: 0.2257 -
learning_rate: 1.0000e-04
Epoch 18/20
50/50             69s 1s/step -
accuracy: 0.9102 - loss: 0.2245 - val_accuracy: 0.9000 - val_loss: 0.2218 -
learning_rate: 5.0000e-05
Epoch 19/20
50/50             68s 1s/step -
accuracy: 0.9113 - loss: 0.2051 - val_accuracy: 0.9000 - val_loss: 0.2298 -
learning_rate: 5.0000e-05
Epoch 20/20
50/50             69s 1s/step -
accuracy: 0.8944 - loss: 0.2437 - val_accuracy: 0.9050 - val_loss: 0.2207 -
learning_rate: 5.0000e-05
13/13             4s 294ms/step -
accuracy: 0.9166 - loss: 0.2189
Test accuracy: 0.905
```

4.4 12 - Confusion Matrix Hard (disco and pop)

[14]:
```python
import seaborn as sns
# from sklearn.metrics import confusion
import numpy as NP
from sklearn.metrics import confusion_matrix

cnn_preds = np.argmax(model.predict(X_test), axis=1)
cnn_cm = confusion_matrix(y_test, cnn_preds)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cnn_cm, annot=True, fmt="d", cmap="Blues", xticklabels=GENRES,␣
 ↪yticklabels=GENRES)
plt.title("CNN Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()
```

```
13/13             5s 304ms/step
```



4.5 13 - Limited Genres Medium (5 random)

[15]:
```python
import os
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
import random

# Augmentation function
def augment_image(image):
    image = tf.image.random_flip_left_right(image)
    image = tf.image.random_brightness(image, max_delta=0.1)
    image = tf.image.random_contrast(image, 0.8, 1.2)
    return image

# Define the genres and file paths
```

```
GENRES = ['blues', 'classical', 'country', 'disco', 'hiphop', 'jazz', 'metal',␣
 ↪'pop', 'reggae', 'rock']
GENRES = random.sample(GENRES, 5)
print(GENRES)
FILE_PATH = os.path.join('Data', 'mel_spectrograms (3 secs)',␣
 ↪'mel_spectrogram_32')
X = []
y = []

GENRE_TO_INDEX = {genre: index for index, genre in enumerate(GENRES)}

# Loop through the genres and load the images with augmentation
for genre in GENRES:
    genre_dir = os.path.join(FILE_PATH, genre)
    print(f"Going through {genre}")
    for file in os.listdir(genre_dir):
        image = tf.io.read_file(os.path.join(genre_dir, file))
        image = tf.image.decode_png(image, channels=1)
        image = tf.image.convert_image_dtype(image, tf.float32)
        image = tf.image.resize(image, [256, 256])  # Resize to 256x256
        image = augment_image(image)  # Apply augmentation
        image = image.numpy()  # Convert to numpy array
        X.append(image)
        y.append(GENRE_TO_INDEX[genre])

# Convert lists to numpy arrays
X = np.array(X)
y = np.array(y)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

from tensorflow.keras import models
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
 ↪Dropout, Normalization

model = models.Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(256, 256, 1)),
    Normalization(),
    MaxPooling2D((2, 2)),

    Conv2D(64, (3, 3), activation='relu'),
    Normalization(),
    MaxPooling2D((2, 2)),

    Conv2D(128, (3, 3), activation='relu'),
    Normalization(),
    MaxPooling2D((2, 2)),

    Conv2D(256, (3, 3), activation='relu'),
    Normalization(),
    MaxPooling2D((2, 2)),

    Flatten(),

    Dense(512, activation='relu'),
    Dropout(0.5),

    Dense(256, activation='relu'),
    Dropout(0.5),

    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])

from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau

model.compile(optimizer=Adam(learning_rate=0.0001),␣
 ↪loss='sparse_categorical_crossentropy', metrics=['accuracy'])
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3,␣
 ↪min_lr=1e-6)

model.fit(X_train, y_train, epochs=20, validation_data=(X_test, y_test),␣
 ↪batch_size=32, callbacks=[reduce_lr])

evaluation = model.evaluate(X_test, y_test)
print(f"Test accuracy: {evaluation[1]:.3f}")
```

```
['blues', 'rock', 'hiphop', 'country', 'disco']
Going through blues
Going through rock
Going through hiphop
Going through country
Going through disco
```

```
/opt/conda/lib/python3.12/site-
packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
Epoch 1/20
```

```
125/125            177s 1s/step -
accuracy: 0.1964 - loss: 1.9536 - val_accuracy: 0.1900 - val_loss: 1.6794 -
learning_rate: 1.0000e-04
Epoch 2/20
125/125            172s 1s/step -
accuracy: 0.2076 - loss: 1.7389 - val_accuracy: 0.3200 - val_loss: 1.5350 -
learning_rate: 1.0000e-04
Epoch 3/20
125/125            172s 1s/step -
accuracy: 0.2943 - loss: 1.5734 - val_accuracy: 0.4180 - val_loss: 1.3831 -
learning_rate: 1.0000e-04
Epoch 4/20
125/125            174s 1s/step -
accuracy: 0.3721 - loss: 1.4144 - val_accuracy: 0.4070 - val_loss: 1.3806 -
learning_rate: 1.0000e-04
Epoch 5/20
125/125            171s 1s/step -
accuracy: 0.4331 - loss: 1.3202 - val_accuracy: 0.4960 - val_loss: 1.2106 -
learning_rate: 1.0000e-04
Epoch 6/20
125/125            171s 1s/step -
accuracy: 0.4720 - loss: 1.2352 - val_accuracy: 0.5560 - val_loss: 1.1300 -
learning_rate: 1.0000e-04
Epoch 7/20
125/125            203s 1s/step -
accuracy: 0.5007 - loss: 1.2035 - val_accuracy: 0.5650 - val_loss: 1.0782 -
learning_rate: 1.0000e-04
Epoch 8/20
125/125            173s 1s/step -
accuracy: 0.5319 - loss: 1.1484 - val_accuracy: 0.5960 - val_loss: 1.0219 -
learning_rate: 1.0000e-04
Epoch 9/20
125/125            173s 1s/step -
accuracy: 0.5356 - loss: 1.1285 - val_accuracy: 0.5480 - val_loss: 1.1151 -
learning_rate: 1.0000e-04
Epoch 10/20
125/125            171s 1s/step -
accuracy: 0.5588 - loss: 1.1055 - val_accuracy: 0.5790 - val_loss: 1.0657 -
learning_rate: 1.0000e-04
Epoch 11/20
125/125            172s 1s/step -
accuracy: 0.5677 - loss: 1.0449 - val_accuracy: 0.6220 - val_loss: 0.9840 -
learning_rate: 1.0000e-04
Epoch 12/20
125/125            171s 1s/step -
accuracy: 0.5929 - loss: 0.9995 - val_accuracy: 0.6310 - val_loss: 0.9409 -
learning_rate: 1.0000e-04
Epoch 13/20
```

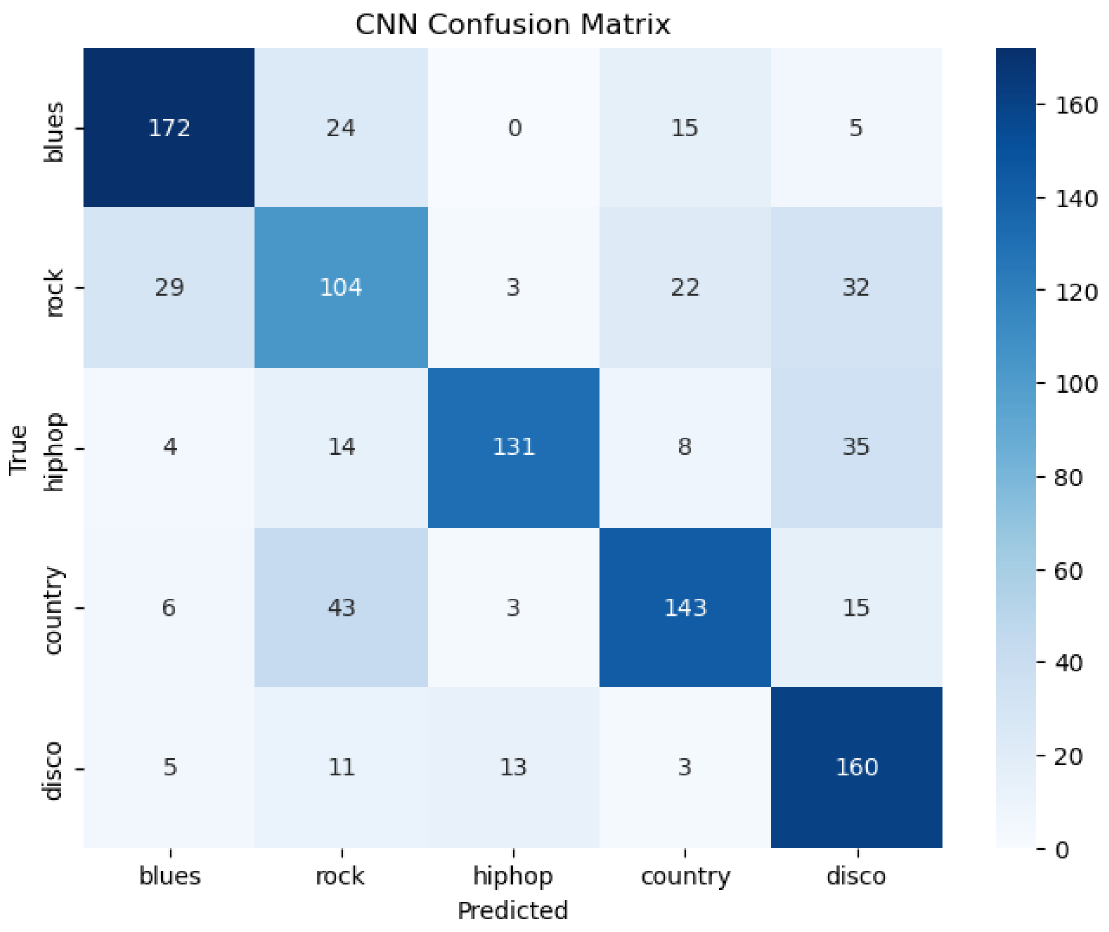
```
125/125              170s 1s/step -
accuracy: 0.6281 - loss: 0.9427 - val_accuracy: 0.6520 - val_loss: 0.9057 -
learning_rate: 1.0000e-04
Epoch 14/20
125/125              175s 1s/step -
accuracy: 0.6364 - loss: 0.9229 - val_accuracy: 0.6890 - val_loss: 0.8632 -
learning_rate: 1.0000e-04
Epoch 15/20
125/125              155s 1s/step -
accuracy: 0.6582 - loss: 0.8890 - val_accuracy: 0.6780 - val_loss: 0.8709 -
learning_rate: 1.0000e-04
Epoch 16/20
125/125              174s 1s/step -
accuracy: 0.6770 - loss: 0.8359 - val_accuracy: 0.6310 - val_loss: 0.9327 -
learning_rate: 1.0000e-04
Epoch 17/20
125/125              175s 1s/step -
accuracy: 0.6904 - loss: 0.8073 - val_accuracy: 0.6960 - val_loss: 0.7926 -
learning_rate: 1.0000e-04
Epoch 18/20
125/125              173s 1s/step -
accuracy: 0.7221 - loss: 0.7387 - val_accuracy: 0.7070 - val_loss: 0.7769 -
learning_rate: 1.0000e-04
Epoch 19/20
125/125              207s 1s/step -
accuracy: 0.7309 - loss: 0.6846 - val_accuracy: 0.6960 - val_loss: 0.8257 -
learning_rate: 1.0000e-04
Epoch 20/20
125/125              175s 1s/step -
accuracy: 0.7481 - loss: 0.6468 - val_accuracy: 0.7100 - val_loss: 0.7609 -
learning_rate: 1.0000e-04
32/32              10s 306ms/step -
accuracy: 0.7336 - loss: 0.7391
Test accuracy: 0.710
```

4.6 14 - Confusion Matrix Medium (5 random)

[16]:
```python
import seaborn as sns
# from sklearn.metrics import confusion
import numpy as NP
from sklearn.metrics import confusion_matrix

cnn_preds = np.argmax(model.predict(X_test), axis=1)
cnn_cm = confusion_matrix(y_test, cnn_preds)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
```

```
sns.heatmap(cnn_cm, annot=True, fmt="d", cmap="Blues", xticklabels=GENRES,␣
  ↪yticklabels=GENRES)
plt.title("CNN Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()
```

```
32/32                 10s 312ms/step
```


CNN Confusion Matrix
True
Predicted
blues
rock
hiphop
country
disco