Onset Images-Only (3 secs) CNN

March 20, 2025

1 CNN for Onset Images (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
```

```
2025-03-20 08:41:43.911772: 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, BatchNormalization

model = models.Sequential([
```

```
    Conv2D(32, (3, 3), activation='relu', input_shape=(256, 256, 1)),
    BatchNormalization(),
    MaxPooling2D((2, 2)),

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

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

    Conv2D(256, (3, 3), activation='relu'),
    BatchNormalization(),
    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
batch_normalization (BatchNormalization)	(None, 254, 254, 32)	128
max_pooling2d (MaxPooling2D)	(None, 127, 127, 32)	0

conv2d_1 (Conv2D)	(None, 125, 125, 64)	18,496
batch_normalization_1 (BatchNormalization)	(None, 125, 125, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 62, 62, 64)	0
conv2d_2 (Conv2D)	(None, 60, 60, 128)	73,856
batch_normalization_2 (BatchNormalization)	(None, 60, 60, 128)	512
max_pooling2d_2 (MaxPooling2D)	(None, 30, 30, 128)	0
conv2d_3 (Conv2D)	(None, 28, 28, 256)	295,168
batch_normalization_3 (BatchNormalization)	(None, 28, 28, 256)	1,024
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: 26,245,898 (100.12 MB)

Trainable params: 26,244,938 (100.12 MB)

Non-trainable params: 960 (3.75 KB)

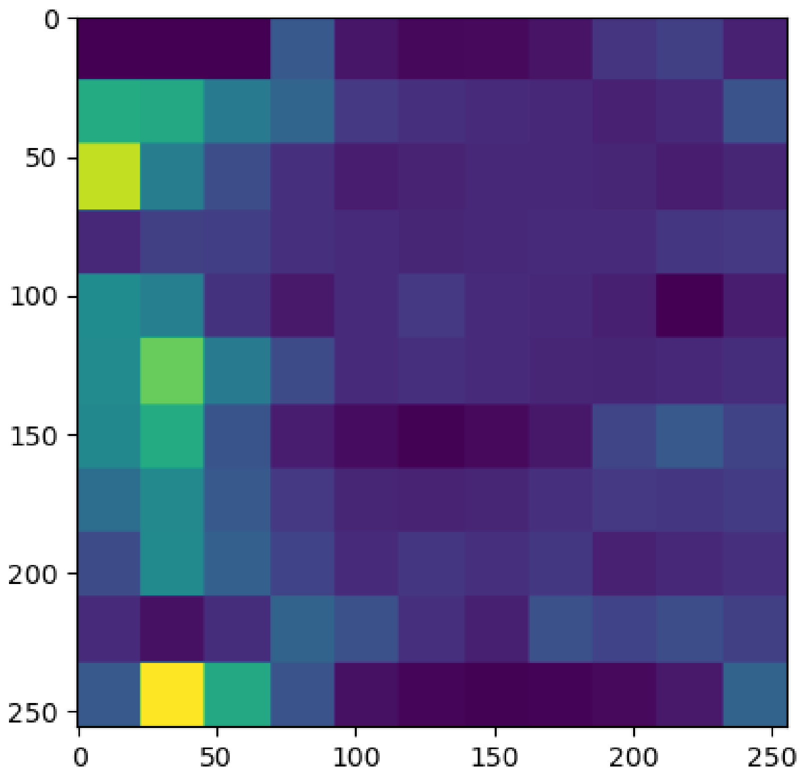
3 4 - Load the images

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

GENRES = ['blues', 'classical', 'country', 'disco', 'hiphop', 'jazz', 'metal',␣
↪'pop', 'reggae', 'rock']
FILE_PATH = os.path.join('Data', 'onset_images (3 secs)')
X = []
y = []

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

# Define the 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

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])

        # Apply the augmentation
        image = augment_image(image)

        image = image.numpy()  # Convert to numpy array for further processing
        X.append(image)
        y.append(GENRE_TO_INDEX[genre])

X = np.array(X)
y = np.array(y)

# Split the data
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()
```



[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)
```

[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              1863s 7s/step -
accuracy: 0.1132 - loss: 3.5405 - val_accuracy: 0.1170 - val_loss: 3.0788 -
learning_rate: 1.0000e-04
Epoch 2/20
250/250              1860s 7s/step -
accuracy: 0.1250 - loss: 2.3005 - val_accuracy: 0.1205 - val_loss: 2.2570 -
learning_rate: 1.0000e-04
Epoch 3/20
250/250              1530s 6s/step -
accuracy: 0.1194 - loss: 2.2837 - val_accuracy: 0.1285 - val_loss: 2.2577 -
learning_rate: 1.0000e-04
Epoch 4/20
250/250              1350s 5s/step -
accuracy: 0.1249 - loss: 2.2645 - val_accuracy: 0.1305 - val_loss: 2.2375 -
learning_rate: 1.0000e-04
Epoch 5/20
250/250              1366s 5s/step -
accuracy: 0.1289 - loss: 2.2525 - val_accuracy: 0.1730 - val_loss: 2.2200 -
learning_rate: 1.0000e-04
Epoch 6/20
250/250              1222s 5s/step -
accuracy: 0.1505 - loss: 2.2529 - val_accuracy: 0.1815 - val_loss: 2.2299 -
learning_rate: 1.0000e-04
Epoch 7/20
250/250              1111s 4s/step -
accuracy: 0.1562 - loss: 2.2266 - val_accuracy: 0.1745 - val_loss: 2.2296 -
learning_rate: 1.0000e-04
Epoch 8/20
250/250              1088s 4s/step -
accuracy: 0.1618 - loss: 2.2299 - val_accuracy: 0.1730 - val_loss: 2.2095 -
learning_rate: 1.0000e-04
Epoch 9/20
250/250              1083s 4s/step -
accuracy: 0.1668 - loss: 2.2180 - val_accuracy: 0.1835 - val_loss: 2.2344 -
learning_rate: 1.0000e-04
Epoch 10/20
250/250              1147s 4s/step -
accuracy: 0.1680 - loss: 2.2001 - val_accuracy: 0.1810 - val_loss: 2.1958 -
learning_rate: 1.0000e-04
Epoch 11/20
250/250              1141s 4s/step -
accuracy: 0.1603 - loss: 2.2082 - val_accuracy: 0.1810 - val_loss: 2.1725 -
learning_rate: 1.0000e-04
Epoch 12/20
```

```
250/250 1107s 4s/step -
accuracy: 0.1694 - loss: 2.1916 - val_accuracy: 0.1845 - val_loss: 2.1874 -
learning_rate: 1.0000e-04
Epoch 13/20
250/250 1098s 4s/step -
accuracy: 0.1678 - loss: 2.1786 - val_accuracy: 0.1910 - val_loss: 2.1795 -
learning_rate: 1.0000e-04
Epoch 14/20
250/250 984s 4s/step -
accuracy: 0.1880 - loss: 2.1694 - val_accuracy: 0.1875 - val_loss: 2.1693 -
learning_rate: 1.0000e-04
Epoch 15/20
250/250 915s 4s/step -
accuracy: 0.1941 - loss: 2.1569 - val_accuracy: 0.1955 - val_loss: 2.1518 -
learning_rate: 1.0000e-04
Epoch 16/20
250/250 914s 4s/step -
accuracy: 0.1906 - loss: 2.1366 - val_accuracy: 0.1910 - val_loss: 2.1657 -
learning_rate: 1.0000e-04
Epoch 17/20
250/250 906s 4s/step -
accuracy: 0.1929 - loss: 2.1492 - val_accuracy: 0.1960 - val_loss: 2.1515 -
learning_rate: 1.0000e-04
Epoch 18/20
250/250 930s 4s/step -
accuracy: 0.1934 - loss: 2.1295 - val_accuracy: 0.1895 - val_loss: 2.1523 -
learning_rate: 1.0000e-04
Epoch 19/20
250/250 938s 4s/step -
accuracy: 0.2128 - loss: 2.1029 - val_accuracy: 0.1995 - val_loss: 2.1331 -
learning_rate: 1.0000e-04
Epoch 20/20
250/250 906s 4s/step -
accuracy: 0.2029 - loss: 2.1045 - val_accuracy: 0.1925 - val_loss: 2.1503 -
learning_rate: 1.0000e-04
```

[8]:
```
<keras.src.callbacks.history.History at 0x7ff098607e30>
```

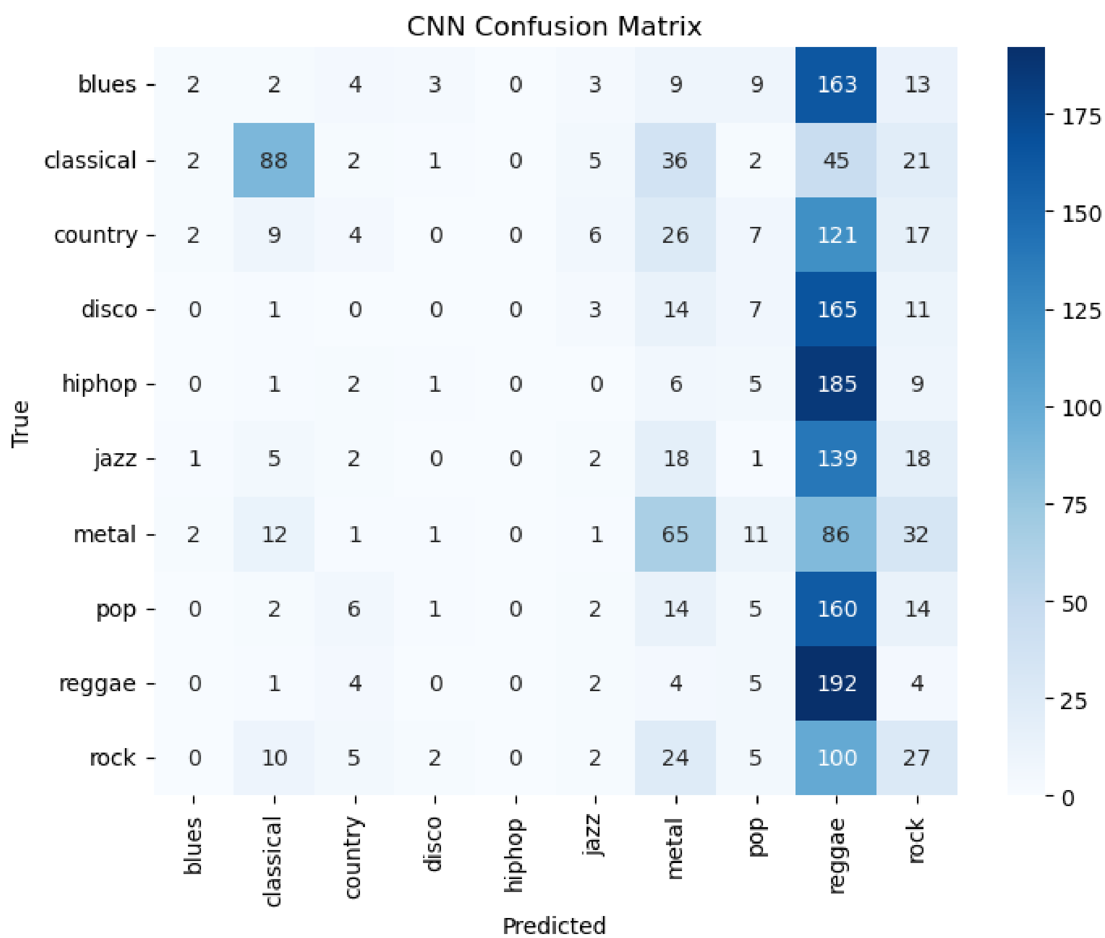
[9]:
```
evaluation = model.evaluate(X_test, y_test)
print(f"Test accuracy: {evaluation[1]:.3f}")
```

```
63/63 24s 383ms/step -
accuracy: 0.1958 - loss: 2.1536
Test accuracy: 0.192
```

4 Apply the confusion matrix after the model

[10]:
```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()
```

```
63/63                 30s 467ms/step
```



4.1 9 - Limited Genres Easy (metal and classical)

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

GENRES = ['classical', 'metal']
FILE_PATH = os.path.join('Data', 'onset_images (3 secs)')
X = []
y = []

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

# Define the 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

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])

        # Apply the augmentation
        image = augment_image(image)

        image = image.numpy()  # Convert to numpy array for further processing
        X.append(image)
        y.append(GENRE_TO_INDEX[genre])

X = np.array(X)
y = np.array(y)

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

```python
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}")
```

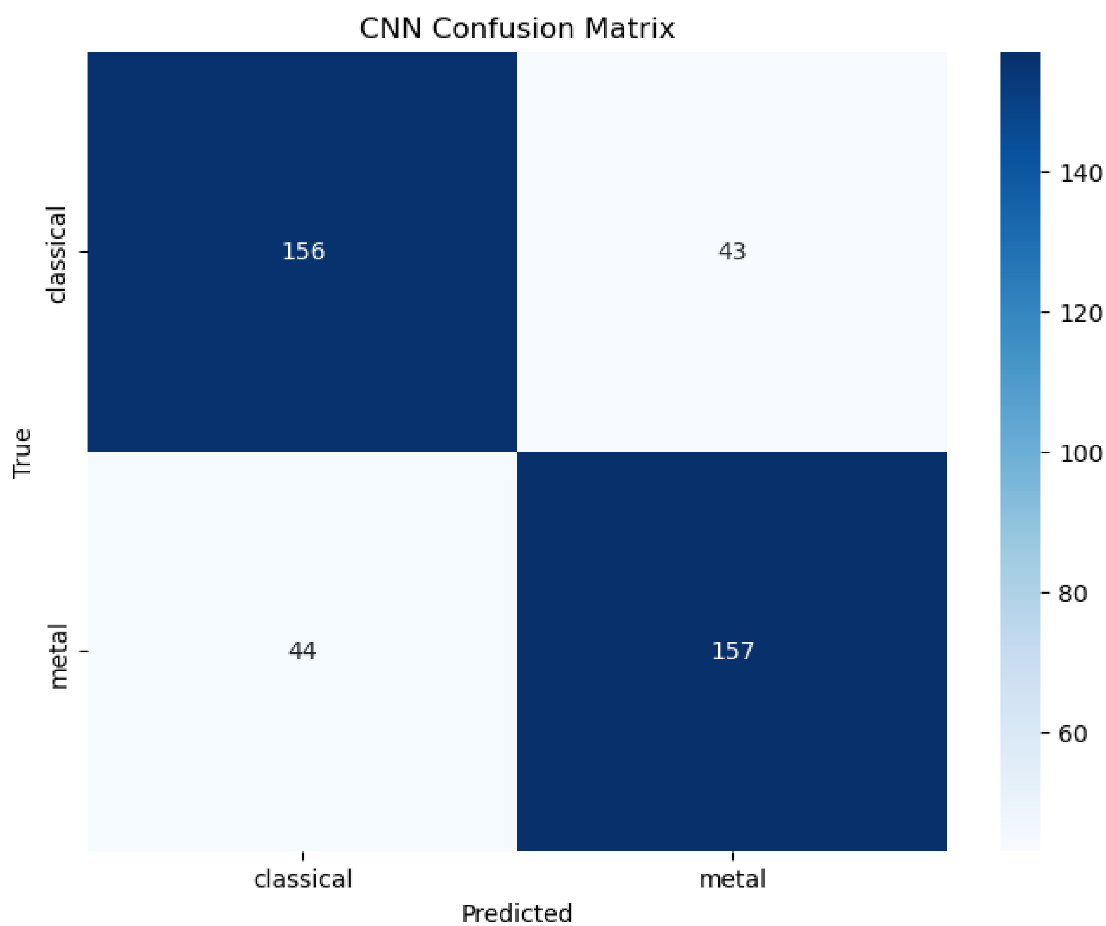
```
Going through classical
Going through metal

/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            100s 2s/step -
accuracy: 0.4175 - loss: 1.6760 - val_accuracy: 0.4975 - val_loss: 0.7377 -
learning_rate: 1.0000e-04
Epoch 2/20
50/50            150s 2s/step -
accuracy: 0.5195 - loss: 0.8581 - val_accuracy: 0.5025 - val_loss: 0.7270 -
learning_rate: 1.0000e-04
Epoch 3/20
50/50            136s 2s/step -
accuracy: 0.5086 - loss: 0.7997 - val_accuracy: 0.5025 - val_loss: 0.7051 -
learning_rate: 1.0000e-04
Epoch 4/20
50/50            107s 2s/step -
accuracy: 0.4783 - loss: 0.7946 - val_accuracy: 0.5025 - val_loss: 0.7003 -
learning_rate: 1.0000e-04
Epoch 5/20
50/50            142s 2s/step -
accuracy: 0.4817 - loss: 0.7887 - val_accuracy: 0.4950 - val_loss: 0.7002 -
learning_rate: 1.0000e-04
Epoch 6/20
50/50            106s 2s/step -
accuracy: 0.4998 - loss: 0.7626 - val_accuracy: 0.5025 - val_loss: 0.6978 -
learning_rate: 1.0000e-04
Epoch 7/20
50/50            106s 2s/step -
accuracy: 0.5261 - loss: 0.7442 - val_accuracy: 0.6450 - val_loss: 0.6816 -
learning_rate: 1.0000e-04
Epoch 8/20
50/50            106s 2s/step -
accuracy: 0.5367 - loss: 0.7270 - val_accuracy: 0.6450 - val_loss: 0.6732 -
learning_rate: 1.0000e-04
Epoch 9/20
50/50            144s 2s/step -
accuracy: 0.4789 - loss: 0.7390 - val_accuracy: 0.6700 - val_loss: 0.6671 -
learning_rate: 1.0000e-04
Epoch 10/20
50/50            98s 2s/step -
accuracy: 0.5619 - loss: 0.7045 - val_accuracy: 0.6525 - val_loss: 0.6538 -
```

```
learning_rate: 1.0000e-04
Epoch 11/20
50/50              138s 2s/step -
accuracy: 0.5493 - loss: 0.7117 - val_accuracy: 0.6975 - val_loss: 0.6456 -
learning_rate: 1.0000e-04
Epoch 12/20
50/50              148s 2s/step -
accuracy: 0.6042 - loss: 0.6687 - val_accuracy: 0.7100 - val_loss: 0.6160 -
learning_rate: 1.0000e-04
Epoch 13/20
50/50              97s 2s/step -
accuracy: 0.5932 - loss: 0.6742 - val_accuracy: 0.7625 - val_loss: 0.5815 -
learning_rate: 1.0000e-04
Epoch 14/20
50/50              128s 2s/step -
accuracy: 0.6479 - loss: 0.6420 - val_accuracy: 0.7375 - val_loss: 0.5887 -
learning_rate: 1.0000e-04
Epoch 15/20
50/50              139s 2s/step -
accuracy: 0.6716 - loss: 0.6262 - val_accuracy: 0.7700 - val_loss: 0.5640 -
learning_rate: 1.0000e-04
Epoch 16/20
50/50              85s 2s/step -
accuracy: 0.6852 - loss: 0.5949 - val_accuracy: 0.7325 - val_loss: 0.5605 -
learning_rate: 1.0000e-04
Epoch 17/20
50/50              73s 1s/step -
accuracy: 0.6984 - loss: 0.5944 - val_accuracy: 0.8075 - val_loss: 0.5401 -
learning_rate: 1.0000e-04
Epoch 18/20
50/50              77s 2s/step -
accuracy: 0.6953 - loss: 0.5889 - val_accuracy: 0.7975 - val_loss: 0.5173 -
learning_rate: 1.0000e-04
Epoch 19/20
50/50              84s 2s/step -
accuracy: 0.7287 - loss: 0.5664 - val_accuracy: 0.7925 - val_loss: 0.5113 -
learning_rate: 1.0000e-04
Epoch 20/20
50/50              85s 2s/step -
accuracy: 0.7460 - loss: 0.5221 - val_accuracy: 0.7825 - val_loss: 0.4950 -
learning_rate: 1.0000e-04
13/13              5s 357ms/step -
accuracy: 0.7693 - loss: 0.4996
Test accuracy: 0.783
```

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              6s 392ms/step
```



4.3 11 - Limited genres Hard (disco and pop)

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

GENRES = ['disco', 'pop']
FILE_PATH = os.path.join('Data', 'onset_images (3 secs)')
X = []
y = []

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

# Define the 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

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])

        # Apply the augmentation
        image = augment_image(image)

        image = image.numpy()  # Convert to numpy array for further processing
        X.append(image)
        y.append(GENRE_TO_INDEX[genre])

X = np.array(X)
y = np.array(y)

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

```python
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}")
```

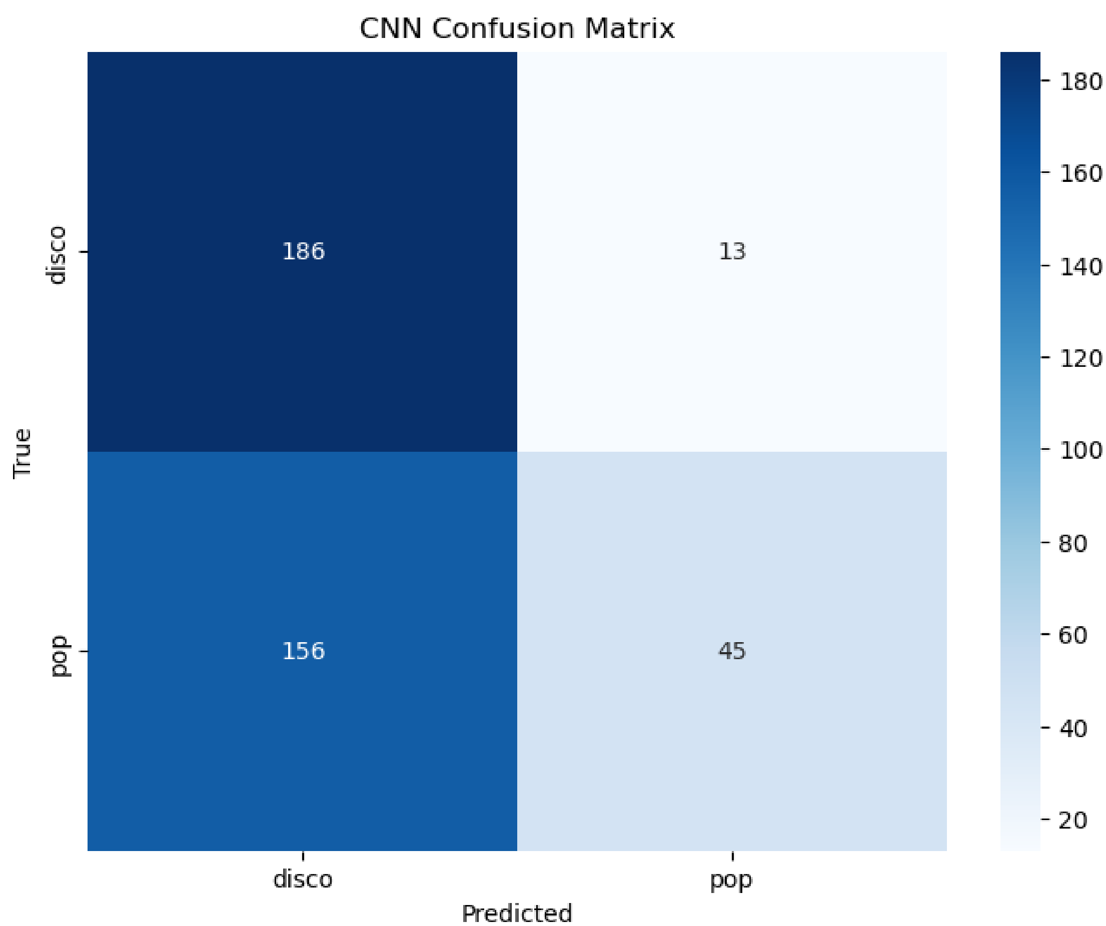
```
Going through disco
Going through pop

/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            92s 2s/step -
accuracy: 0.4711 - loss: 1.5474 - val_accuracy: 0.4975 - val_loss: 0.7273 -
learning_rate: 1.0000e-04
Epoch 2/20
50/50            85s 2s/step -
accuracy: 0.5005 - loss: 0.8964 - val_accuracy: 0.4975 - val_loss: 0.7434 -
learning_rate: 1.0000e-04
Epoch 3/20
50/50            142s 2s/step -
accuracy: 0.5047 - loss: 0.8305 - val_accuracy: 0.4975 - val_loss: 0.7535 -
learning_rate: 1.0000e-04
Epoch 4/20
50/50            76s 2s/step -
accuracy: 0.5056 - loss: 0.7747 - val_accuracy: 0.4975 - val_loss: 0.7063 -
learning_rate: 1.0000e-04
Epoch 5/20
50/50            78s 2s/step -
accuracy: 0.4867 - loss: 0.7940 - val_accuracy: 0.4975 - val_loss: 0.7180 -
learning_rate: 1.0000e-04
Epoch 6/20
50/50            84s 2s/step -
accuracy: 0.4767 - loss: 0.7776 - val_accuracy: 0.5175 - val_loss: 0.7024 -
learning_rate: 1.0000e-04
Epoch 7/20
50/50            84s 2s/step -
accuracy: 0.5087 - loss: 0.7436 - val_accuracy: 0.5250 - val_loss: 0.6969 -
learning_rate: 1.0000e-04
Epoch 8/20
50/50            83s 2s/step -
accuracy: 0.5095 - loss: 0.7388 - val_accuracy: 0.5325 - val_loss: 0.7045 -
learning_rate: 1.0000e-04
Epoch 9/20
50/50            73s 1s/step -
accuracy: 0.5498 - loss: 0.7101 - val_accuracy: 0.5450 - val_loss: 0.6918 -
learning_rate: 1.0000e-04
Epoch 10/20
50/50            81s 2s/step -
accuracy: 0.5490 - loss: 0.7197 - val_accuracy: 0.5925 - val_loss: 0.6827 -
```

```
learning_rate: 1.0000e-04
Epoch 11/20
50/50             71s 1s/step -
accuracy: 0.5592 - loss: 0.7007 - val_accuracy: 0.5750 - val_loss: 0.6790 -
learning_rate: 1.0000e-04
Epoch 12/20
50/50             83s 2s/step -
accuracy: 0.5863 - loss: 0.6899 - val_accuracy: 0.5875 - val_loss: 0.6649 -
learning_rate: 1.0000e-04
Epoch 13/20
50/50             69s 1s/step -
accuracy: 0.5869 - loss: 0.6779 - val_accuracy: 0.5475 - val_loss: 0.6949 -
learning_rate: 1.0000e-04
Epoch 14/20
50/50             86s 2s/step -
accuracy: 0.5788 - loss: 0.6911 - val_accuracy: 0.6125 - val_loss: 0.6633 -
learning_rate: 1.0000e-04
Epoch 15/20
50/50             113s 1s/step -
accuracy: 0.5975 - loss: 0.6617 - val_accuracy: 0.6075 - val_loss: 0.6639 -
learning_rate: 1.0000e-04
Epoch 16/20
50/50             53s 1s/step -
accuracy: 0.6041 - loss: 0.6818 - val_accuracy: 0.6100 - val_loss: 0.6561 -
learning_rate: 1.0000e-04
Epoch 17/20
50/50             66s 1s/step -
accuracy: 0.5724 - loss: 0.6777 - val_accuracy: 0.6300 - val_loss: 0.6569 -
learning_rate: 1.0000e-04
Epoch 18/20
50/50             99s 2s/step -
accuracy: 0.6226 - loss: 0.6694 - val_accuracy: 0.6200 - val_loss: 0.6494 -
learning_rate: 1.0000e-04
Epoch 19/20
50/50             83s 2s/step -
accuracy: 0.6243 - loss: 0.6526 - val_accuracy: 0.6325 - val_loss: 0.6521 -
learning_rate: 1.0000e-04
Epoch 20/20
50/50             71s 1s/step -
accuracy: 0.6321 - loss: 0.6214 - val_accuracy: 0.5775 - val_loss: 0.6825 -
learning_rate: 1.0000e-04
13/13             5s 371ms/step -
accuracy: 0.6019 - loss: 0.6623
Test accuracy: 0.577
```

4.4 12 - Confusion Matrix Hard (disco and pop)

[14]:
```
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 402ms/step
```



4.5 13 - Limited Genres Medium (5 random)

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

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

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

# Define the 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

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])

        # Apply the augmentation
        image = augment_image(image)

        image = image.numpy()  # Convert to numpy array for further processing
        X.append(image)
        y.append(GENRE_TO_INDEX[genre])

X = np.array(X)
y = np.array(y)
```

```
# Split the data
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])
```

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

```
['metal', 'hiphop', 'rock', 'pop', 'reggae']
Going through metal
Going through hiphop
Going through rock
Going through pop
Going through reggae
```

```
/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              217s 2s/step -
accuracy: 0.1711 - loss: 1.9709 - val_accuracy: 0.1910 - val_loss: 1.6577 -
learning_rate: 1.0000e-04
Epoch 2/20
125/125              211s 2s/step -
accuracy: 0.2235 - loss: 1.7180 - val_accuracy: 0.2850 - val_loss: 1.5746 -
learning_rate: 1.0000e-04
Epoch 3/20
125/125              261s 2s/step -
accuracy: 0.2406 - loss: 1.6507 - val_accuracy: 0.2980 - val_loss: 1.5391 -
learning_rate: 1.0000e-04
Epoch 4/20
125/125              212s 2s/step -
accuracy: 0.2706 - loss: 1.6051 - val_accuracy: 0.3070 - val_loss: 1.5561 -
learning_rate: 1.0000e-04
Epoch 5/20
125/125              211s 2s/step -
accuracy: 0.2951 - loss: 1.5643 - val_accuracy: 0.3060 - val_loss: 1.5512 -
learning_rate: 1.0000e-04
Epoch 6/20
125/125              207s 2s/step -
accuracy: 0.3046 - loss: 1.5388 - val_accuracy: 0.3220 - val_loss: 1.4770 -
learning_rate: 1.0000e-04
Epoch 7/20
125/125              267s 2s/step -
accuracy: 0.3286 - loss: 1.4867 - val_accuracy: 0.3390 - val_loss: 1.4539 -
learning_rate: 1.0000e-04
Epoch 8/20
125/125              200s 2s/step -
accuracy: 0.3491 - loss: 1.4441 - val_accuracy: 0.3520 - val_loss: 1.4432 -
```

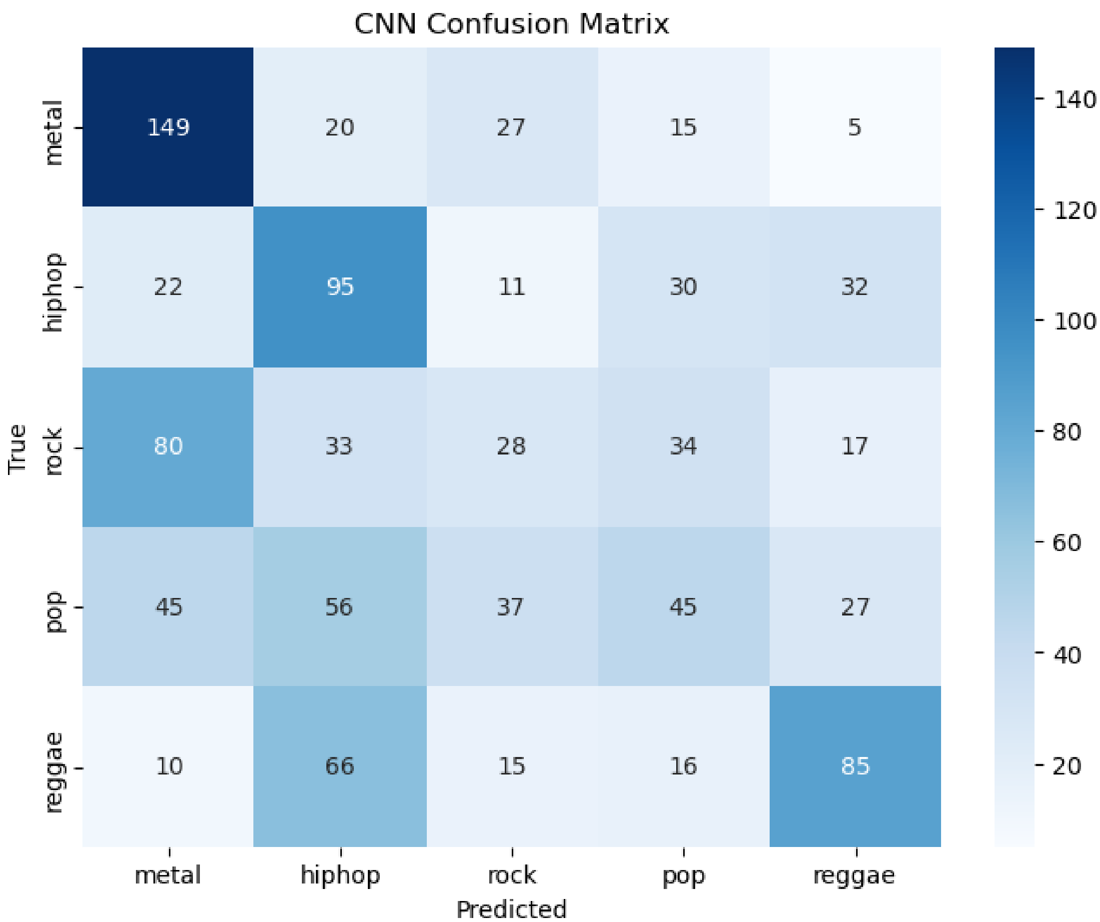
```
learning_rate: 1.0000e-04
Epoch 9/20
125/125             211s 2s/step -
accuracy: 0.3473 - loss: 1.4379 - val_accuracy: 0.3850 - val_loss: 1.4166 -
learning_rate: 1.0000e-04
Epoch 10/20
125/125             209s 2s/step -
accuracy: 0.3772 - loss: 1.4155 - val_accuracy: 0.3580 - val_loss: 1.4156 -
learning_rate: 1.0000e-04
Epoch 11/20
125/125             209s 2s/step -
accuracy: 0.3976 - loss: 1.3669 - val_accuracy: 0.3700 - val_loss: 1.4038 -
learning_rate: 1.0000e-04
Epoch 12/20
125/125             211s 2s/step -
accuracy: 0.4297 - loss: 1.3329 - val_accuracy: 0.3630 - val_loss: 1.4242 -
learning_rate: 1.0000e-04
Epoch 13/20
125/125             198s 2s/step -
accuracy: 0.4218 - loss: 1.3162 - val_accuracy: 0.3870 - val_loss: 1.4086 -
learning_rate: 1.0000e-04
Epoch 14/20
125/125             212s 2s/step -
accuracy: 0.4469 - loss: 1.2922 - val_accuracy: 0.3960 - val_loss: 1.3872 -
learning_rate: 1.0000e-04
Epoch 15/20
125/125             201s 2s/step -
accuracy: 0.4581 - loss: 1.2677 - val_accuracy: 0.3890 - val_loss: 1.3838 -
learning_rate: 1.0000e-04
Epoch 16/20
125/125             211s 2s/step -
accuracy: 0.4746 - loss: 1.2510 - val_accuracy: 0.3990 - val_loss: 1.4205 -
learning_rate: 1.0000e-04
Epoch 17/20
125/125             236s 1s/step -
accuracy: 0.5125 - loss: 1.1667 - val_accuracy: 0.3930 - val_loss: 1.4099 -
learning_rate: 1.0000e-04
Epoch 18/20
125/125             134s 1s/step -
accuracy: 0.5258 - loss: 1.1525 - val_accuracy: 0.4010 - val_loss: 1.4010 -
learning_rate: 1.0000e-04
Epoch 19/20
125/125             135s 1s/step -
accuracy: 0.5778 - loss: 1.0529 - val_accuracy: 0.4060 - val_loss: 1.4144 -
learning_rate: 5.0000e-05
Epoch 20/20
125/125             134s 1s/step -
accuracy: 0.5897 - loss: 1.0158 - val_accuracy: 0.4020 - val_loss: 1.4457 -
```

```
learning_rate: 5.0000e-05
32/32              7s 219ms/step -
accuracy: 0.4182 - loss: 1.4294
Test accuracy: 0.402
```

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              7s 220ms/step
```


CNN Confusion Matrix
metal
hiphop
rock
pop
reggae
True
149
20
27
15
5
22
95
11
30
32
80
33
28
34
17
45
56
37
45
27
10
66
15
16
85
metal
hiphop
rock
pop
reggae
Predicted
140
120
100
80
60
40
20