Onset HeatMap Clips-Only CNN

March 20, 2025

1 CNN for Onset HeatMap Clip Images (30 secs)

1.1 1 - All the imports

1.2 9 - Limited Genres Easy (hip hop and classical)

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

```
2025-03-20 08:41:39.613557: 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.
```

1.3 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)

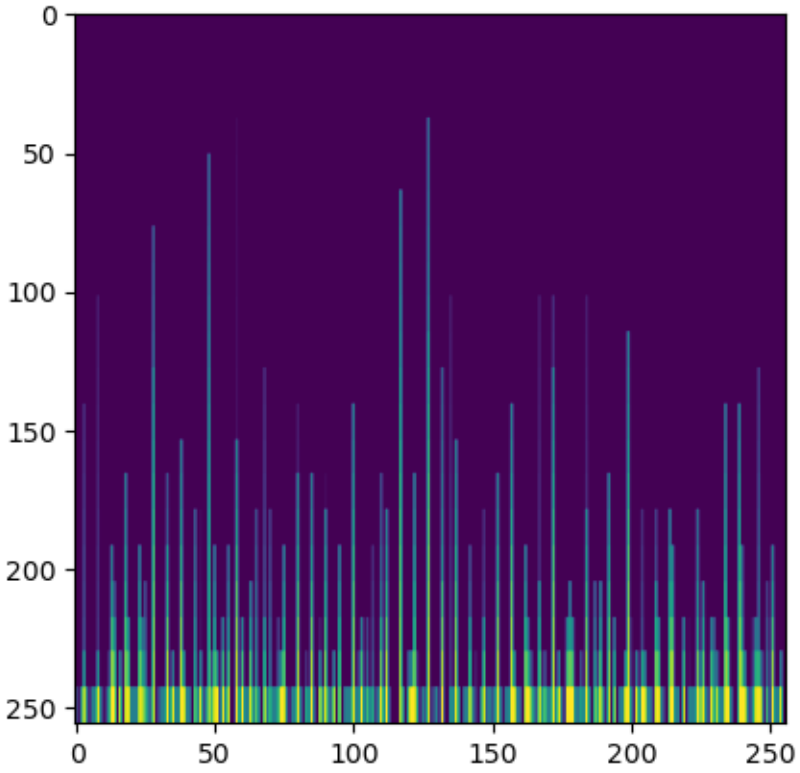
3 4 - Load the images

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



[7]:
```
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
25/25             105s 4s/step -
accuracy: 0.1061 - loss: 5.1916 - val_accuracy: 0.0650 - val_loss: 2.3154 -
learning_rate: 1.0000e-04
Epoch 2/20
25/25             123s 5s/step -
accuracy: 0.1150 - loss: 2.9482 - val_accuracy: 0.0650 - val_loss: 2.3520 -
learning_rate: 1.0000e-04
Epoch 3/20
25/25             121s 5s/step -
accuracy: 0.1412 - loss: 2.3341 - val_accuracy: 0.0650 - val_loss: 2.4464 -
learning_rate: 1.0000e-04
Epoch 4/20
25/25             136s 5s/step -
accuracy: 0.1281 - loss: 2.3173 - val_accuracy: 0.0650 - val_loss: 2.5232 -
learning_rate: 1.0000e-04
Epoch 5/20
25/25             168s 7s/step -
accuracy: 0.1586 - loss: 2.2689 - val_accuracy: 0.0700 - val_loss: 2.6112 -
learning_rate: 5.0000e-05
Epoch 6/20
25/25             226s 8s/step -
accuracy: 0.1396 - loss: 2.2341 - val_accuracy: 0.0700 - val_loss: 2.6679 -
learning_rate: 5.0000e-05
Epoch 7/20
25/25             194s 8s/step -
accuracy: 0.1520 - loss: 2.2193 - val_accuracy: 0.0800 - val_loss: 2.7509 -
learning_rate: 5.0000e-05
Epoch 8/20
25/25             193s 8s/step -
accuracy: 0.1539 - loss: 2.2110 - val_accuracy: 0.0600 - val_loss: 2.8038 -
learning_rate: 2.5000e-05
Epoch 9/20
25/25             199s 8s/step -
accuracy: 0.1673 - loss: 2.2068 - val_accuracy: 0.0650 - val_loss: 2.8834 -
learning_rate: 2.5000e-05
Epoch 10/20
25/25             200s 8s/step -
accuracy: 0.1934 - loss: 2.1725 - val_accuracy: 0.0650 - val_loss: 2.9112 -
learning_rate: 2.5000e-05
Epoch 11/20
25/25             206s 8s/step -
accuracy: 0.1984 - loss: 2.1963 - val_accuracy: 0.0650 - val_loss: 2.8959 -
learning_rate: 1.2500e-05
Epoch 12/20
```

```
25/25             196s 8s/step -
accuracy: 0.2034 - loss: 2.1489 - val_accuracy: 0.0650 - val_loss: 2.8437 -
learning_rate: 1.2500e-05
Epoch 13/20
25/25             198s 8s/step -
accuracy: 0.1745 - loss: 2.1789 - val_accuracy: 0.0650 - val_loss: 2.8210 -
learning_rate: 1.2500e-05
Epoch 14/20
25/25             193s 8s/step -
accuracy: 0.2092 - loss: 2.1111 - val_accuracy: 0.0650 - val_loss: 2.7541 -
learning_rate: 6.2500e-06
Epoch 15/20
25/25             203s 8s/step -
accuracy: 0.2285 - loss: 2.1127 - val_accuracy: 0.0650 - val_loss: 2.6887 -
learning_rate: 6.2500e-06
Epoch 16/20
25/25             253s 8s/step -
accuracy: 0.2495 - loss: 2.0934 - val_accuracy: 0.0650 - val_loss: 2.6372 -
learning_rate: 6.2500e-06
Epoch 17/20
25/25             206s 8s/step -
accuracy: 0.2200 - loss: 2.1068 - val_accuracy: 0.0650 - val_loss: 2.5722 -
learning_rate: 3.1250e-06
Epoch 18/20
25/25             264s 8s/step -
accuracy: 0.2011 - loss: 2.1200 - val_accuracy: 0.0650 - val_loss: 2.5084 -
learning_rate: 3.1250e-06
Epoch 19/20
25/25             202s 8s/step -
accuracy: 0.1937 - loss: 2.1010 - val_accuracy: 0.0750 - val_loss: 2.4556 -
learning_rate: 3.1250e-06
Epoch 20/20
25/25             202s 8s/step -
accuracy: 0.2401 - loss: 2.1081 - val_accuracy: 0.0750 - val_loss: 2.4195 -
learning_rate: 1.5625e-06
```

[8]:
```
<keras.src.callbacks.history.History at 0x7fe454247140>
```

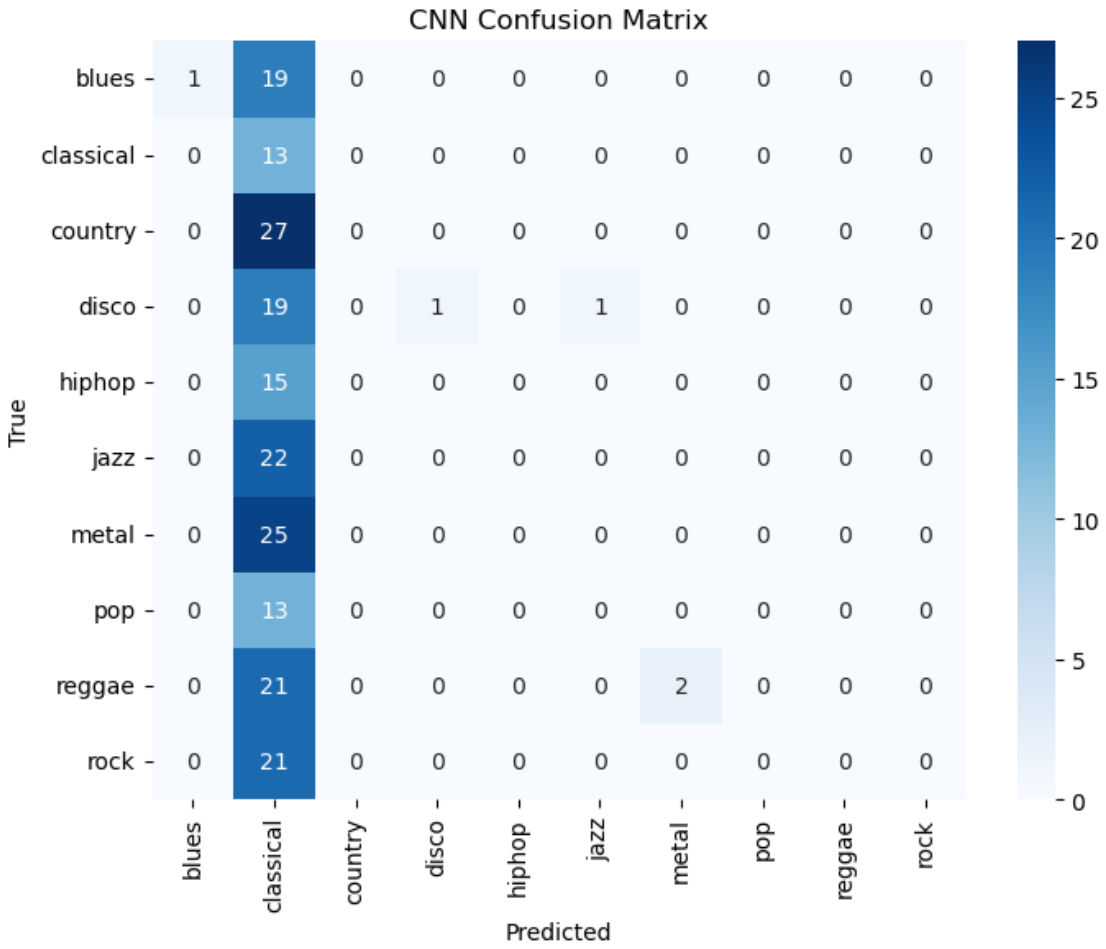
[9]:
```
evaluation = model.evaluate(X_test, y_test)
print(f"Test accuracy: {evaluation[1]:.3f}")
```

```
7/7               8s 1s/step -
accuracy: 0.0848 - loss: 2.3906
Test accuracy: 0.075
```

4 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()
```

```
7/7                 9s 1s/step
```



4.1 9 - Limited Genres Easy (metal and classical)

[11]:
```
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_heatmaps_full')
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])

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

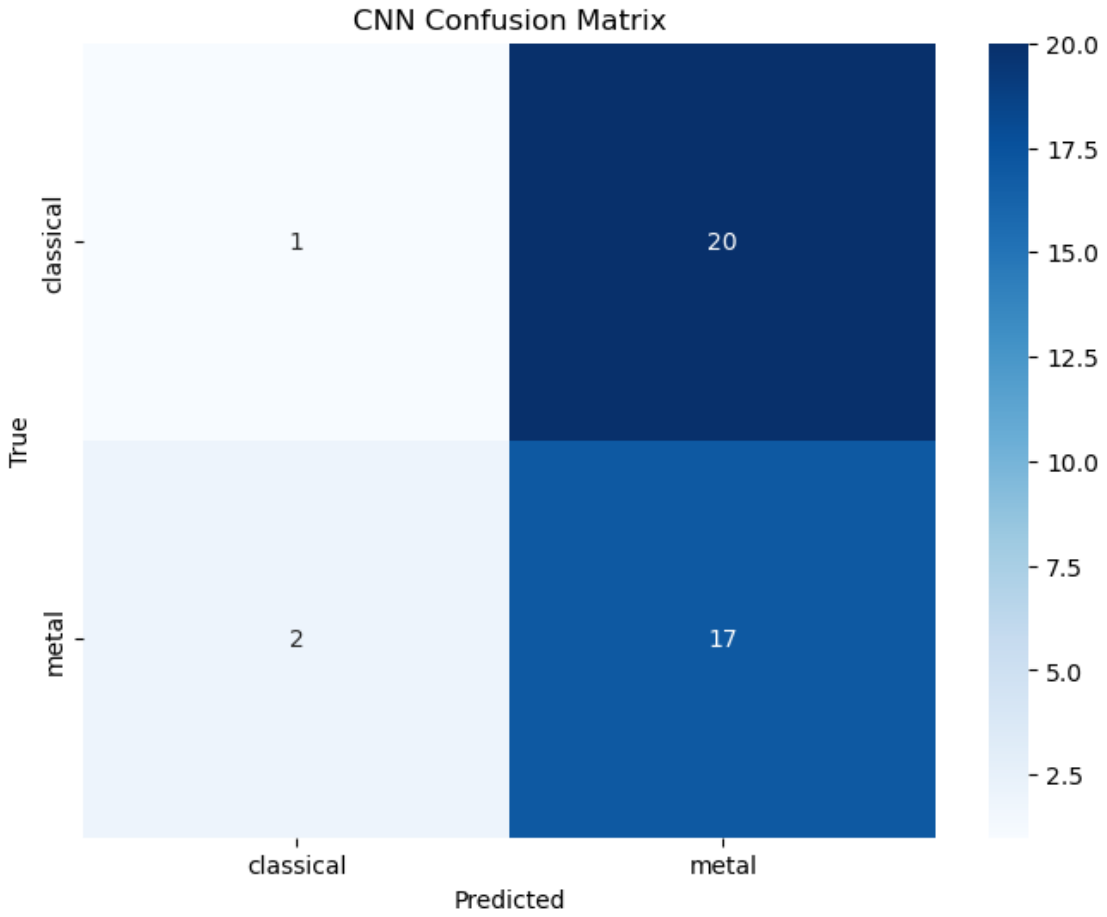
```
Going through classical
Going through metal

/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
5/5            37s 5s/step -
accuracy: 0.3360 - loss: 2.2019 - val_accuracy: 0.5250 - val_loss: 1.7437 -
learning_rate: 1.0000e-04
Epoch 2/20
5/5            46s 6s/step -
accuracy: 0.5345 - loss: 1.5688 - val_accuracy: 0.5250 - val_loss: 0.9345 -
learning_rate: 1.0000e-04
Epoch 3/20
5/5            26s 5s/step -
accuracy: 0.5222 - loss: 0.9817 - val_accuracy: 0.5250 - val_loss: 0.7125 -
learning_rate: 1.0000e-04
Epoch 4/20
5/5            27s 5s/step -
accuracy: 0.6174 - loss: 0.9286 - val_accuracy: 0.4750 - val_loss: 0.7252 -
learning_rate: 1.0000e-04
Epoch 5/20
5/5            41s 6s/step -
accuracy: 0.5018 - loss: 0.9679 - val_accuracy: 0.4750 - val_loss: 0.7334 -
learning_rate: 1.0000e-04
Epoch 6/20
5/5            26s 5s/step -
accuracy: 0.5095 - loss: 0.9446 - val_accuracy: 0.4750 - val_loss: 0.7306 -
learning_rate: 1.0000e-04
Epoch 7/20
5/5            25s 5s/step -
accuracy: 0.5746 - loss: 0.8505 - val_accuracy: 0.4750 - val_loss: 0.7433 -
learning_rate: 5.0000e-05
Epoch 8/20
5/5            44s 6s/step -
accuracy: 0.5622 - loss: 0.8215 - val_accuracy: 0.5000 - val_loss: 0.7451 -
learning_rate: 5.0000e-05
Epoch 9/20
5/5            35s 4s/step -
accuracy: 0.5201 - loss: 0.8696 - val_accuracy: 0.5250 - val_loss: 0.7435 -
learning_rate: 5.0000e-05
Epoch 10/20
5/5            47s 6s/step -
accuracy: 0.4818 - loss: 0.8410 - val_accuracy: 0.5250 - val_loss: 0.7406 -
```

```
learning_rate: 2.5000e-05
Epoch 11/20
5/5             39s 5s/step -
accuracy: 0.5142 - loss: 0.8295 - val_accuracy: 0.5250 - val_loss: 0.7362 -
learning_rate: 2.5000e-05
Epoch 12/20
5/5             28s 5s/step -
accuracy: 0.5217 - loss: 0.8633 - val_accuracy: 0.5250 - val_loss: 0.7315 -
learning_rate: 2.5000e-05
Epoch 13/20
5/5             29s 6s/step -
accuracy: 0.5056 - loss: 0.8586 - val_accuracy: 0.5250 - val_loss: 0.7300 -
learning_rate: 1.2500e-05
Epoch 14/20
5/5             27s 5s/step -
accuracy: 0.5070 - loss: 0.8114 - val_accuracy: 0.5250 - val_loss: 0.7284 -
learning_rate: 1.2500e-05
Epoch 15/20
5/5             42s 6s/step -
accuracy: 0.5667 - loss: 0.8313 - val_accuracy: 0.4750 - val_loss: 0.7272 -
learning_rate: 1.2500e-05
Epoch 16/20
5/5             34s 4s/step -
accuracy: 0.4239 - loss: 0.8820 - val_accuracy: 0.5250 - val_loss: 0.7270 -
learning_rate: 6.2500e-06
Epoch 17/20
5/5             40s 4s/step -
accuracy: 0.4872 - loss: 0.8493 - val_accuracy: 0.3750 - val_loss: 0.7269 -
learning_rate: 6.2500e-06
Epoch 18/20
5/5             20s 4s/step -
accuracy: 0.5150 - loss: 0.7990 - val_accuracy: 0.4250 - val_loss: 0.7264 -
learning_rate: 6.2500e-06
Epoch 19/20
5/5             26s 5s/step -
accuracy: 0.4751 - loss: 0.8763 - val_accuracy: 0.4500 - val_loss: 0.7263 -
learning_rate: 3.1250e-06
Epoch 20/20
5/5             26s 5s/step -
accuracy: 0.5227 - loss: 0.7999 - val_accuracy: 0.4500 - val_loss: 0.7261 -
learning_rate: 3.1250e-06
2/2             2s 517ms/step -
accuracy: 0.4458 - loss: 0.7252
Test accuracy: 0.450
```

4.2 10 - Confusion Matrix Easy (classical and metal)

[12]:
```
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()
```

```
2/2                 3s 816ms/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_heatmaps_full')
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])

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

```
Going through disco
```

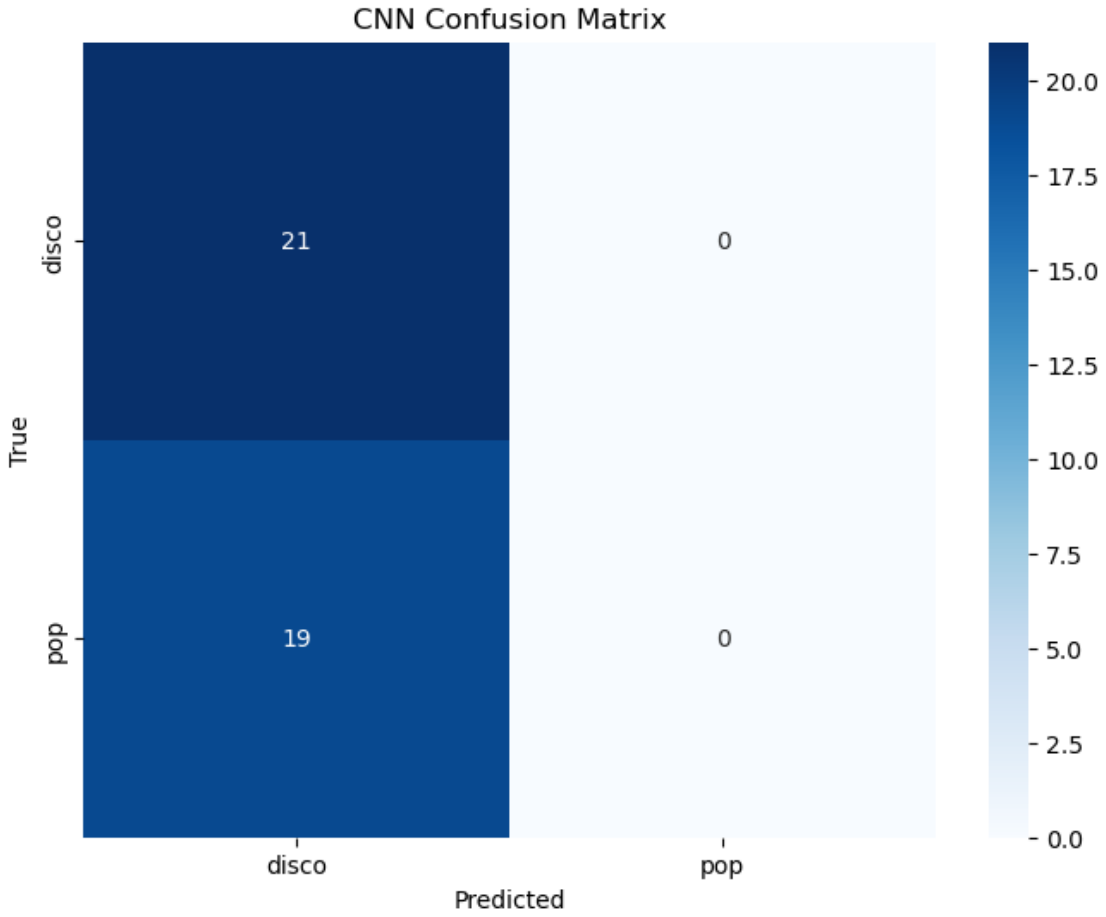
```
Going through pop

/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
5/5             35s 4s/step -
accuracy: 0.3817 - loss: 2.1705 - val_accuracy: 0.5750 - val_loss: 1.5722 -
learning_rate: 1.0000e-04
Epoch 2/20
5/5             44s 5s/step -
accuracy: 0.5223 - loss: 1.4571 - val_accuracy: 0.5000 - val_loss: 0.7991 -
learning_rate: 1.0000e-04
Epoch 3/20
5/5             25s 5s/step -
accuracy: 0.5693 - loss: 0.9315 - val_accuracy: 0.5250 - val_loss: 0.7161 -
learning_rate: 1.0000e-04
Epoch 4/20
5/5             21s 4s/step -
accuracy: 0.5073 - loss: 1.0299 - val_accuracy: 0.5250 - val_loss: 0.7094 -
learning_rate: 1.0000e-04
Epoch 5/20
5/5             26s 5s/step -
accuracy: 0.5982 - loss: 0.7377 - val_accuracy: 0.5250 - val_loss: 0.7178 -
learning_rate: 1.0000e-04
Epoch 6/20
5/5             39s 5s/step -
accuracy: 0.5341 - loss: 0.9096 - val_accuracy: 0.5250 - val_loss: 0.7552 -
learning_rate: 1.0000e-04
Epoch 7/20
5/5             27s 6s/step -
accuracy: 0.5165 - loss: 0.8303 - val_accuracy: 0.5250 - val_loss: 0.7837 -
learning_rate: 1.0000e-04
Epoch 8/20
5/5             35s 4s/step -
accuracy: 0.5103 - loss: 0.8579 - val_accuracy: 0.5250 - val_loss: 0.7827 -
learning_rate: 5.0000e-05
Epoch 9/20
5/5             25s 5s/step -
accuracy: 0.5364 - loss: 0.8132 - val_accuracy: 0.5250 - val_loss: 0.7677 -
learning_rate: 5.0000e-05
Epoch 10/20
5/5             41s 5s/step -
accuracy: 0.4668 - loss: 0.8497 - val_accuracy: 0.5250 - val_loss: 0.7547 -
learning_rate: 5.0000e-05
```

```
Epoch 11/20
5/5             25s 5s/step -
accuracy: 0.5617 - loss: 0.7396 - val_accuracy: 0.5250 - val_loss: 0.7468 -
learning_rate: 2.5000e-05
Epoch 12/20
5/5             41s 5s/step -
accuracy: 0.5841 - loss: 0.7744 - val_accuracy: 0.5250 - val_loss: 0.7380 -
learning_rate: 2.5000e-05
Epoch 13/20
5/5             23s 5s/step -
accuracy: 0.4975 - loss: 0.8441 - val_accuracy: 0.5250 - val_loss: 0.7360 -
learning_rate: 2.5000e-05
Epoch 14/20
5/5             38s 4s/step -
accuracy: 0.4671 - loss: 0.8460 - val_accuracy: 0.5250 - val_loss: 0.7371 -
learning_rate: 1.2500e-05
Epoch 15/20
5/5             24s 5s/step -
accuracy: 0.5212 - loss: 0.8194 - val_accuracy: 0.5250 - val_loss: 0.7373 -
learning_rate: 1.2500e-05
Epoch 16/20
5/5             37s 4s/step -
accuracy: 0.5019 - loss: 0.8355 - val_accuracy: 0.5250 - val_loss: 0.7389 -
learning_rate: 1.2500e-05
Epoch 17/20
5/5             17s 3s/step -
accuracy: 0.4374 - loss: 0.8388 - val_accuracy: 0.5250 - val_loss: 0.7397 -
learning_rate: 6.2500e-06
Epoch 18/20
5/5             25s 5s/step -
accuracy: 0.4596 - loss: 0.8316 - val_accuracy: 0.5250 - val_loss: 0.7411 -
learning_rate: 6.2500e-06
Epoch 19/20
5/5             39s 4s/step -
accuracy: 0.4844 - loss: 0.8266 - val_accuracy: 0.5250 - val_loss: 0.7426 -
learning_rate: 6.2500e-06
Epoch 20/20
5/5             17s 3s/step -
accuracy: 0.4684 - loss: 0.8585 - val_accuracy: 0.5250 - val_loss: 0.7431 -
learning_rate: 3.1250e-06
2/2             1s 277ms/step -
accuracy: 0.5375 - loss: 0.7367
Test accuracy: 0.525
```

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()
```

```
WARNING:tensorflow:5 out of the last 10 calls to <function
TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at
0x7fe3d4775120> triggered tf.function retracing. Tracing is expensive and the
excessive number of tracings could be due to (1) creating @tf.function
repeatedly in a loop, (2) passing tensors with different shapes, (3) passing
Python objects instead of tensors. For (1), please define your @tf.function
outside of the loop. For (2), @tf.function has reduce_retracing=True option that
can avoid unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
1/2             1s
1s/stepWARNING:tensorflow:6 out of the last 11 calls to <function
TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at
0x7fe3d4775120> triggered tf.function retracing. Tracing is expensive and the
excessive number of tracings could be due to (1) creating @tf.function
repeatedly in a loop, (2) passing tensors with different shapes, (3) passing
Python objects instead of tensors. For (1), please define your @tf.function
outside of the loop. For (2), @tf.function has reduce_retracing=True option that
can avoid unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
2/2             2s 818ms/step
```



4.5 13 - Limited Genres Medium (5 random)

[15]:
```
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_heatmaps_full')
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])

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

```
['disco', 'classical', 'metal', 'jazz', 'hiphop']
Going through disco
Going through classical
Going through metal
Going through jazz
Going through hiphop
```

```
/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
13/13             66s 4s/step -
accuracy: 0.1944 - loss: 2.2586 - val_accuracy: 0.1000 - val_loss: 2.0004 -
learning_rate: 1.0000e-04
```

```
Epoch 2/20
13/13             79s 4s/step -
accuracy: 0.2143 - loss: 1.9949 - val_accuracy: 0.1400 - val_loss: 1.8431 -
learning_rate: 1.0000e-04
Epoch 3/20
13/13             52s 4s/step -
accuracy: 0.2080 - loss: 1.9352 - val_accuracy: 0.1400 - val_loss: 1.8102 -
learning_rate: 1.0000e-04
Epoch 4/20
13/13             82s 4s/step -
accuracy: 0.2098 - loss: 1.8371 - val_accuracy: 0.1400 - val_loss: 1.7450 -
learning_rate: 1.0000e-04
Epoch 5/20
13/13             52s 4s/step -
accuracy: 0.2019 - loss: 1.8072 - val_accuracy: 0.1400 - val_loss: 1.7685 -
learning_rate: 1.0000e-04
Epoch 6/20
13/13             82s 4s/step -
accuracy: 0.2087 - loss: 1.7991 - val_accuracy: 0.1000 - val_loss: 1.7316 -
learning_rate: 1.0000e-04
Epoch 7/20
13/13             80s 4s/step -
accuracy: 0.2113 - loss: 1.7729 - val_accuracy: 0.1100 - val_loss: 1.7375 -
learning_rate: 1.0000e-04
Epoch 8/20
13/13             48s 4s/step -
accuracy: 0.1852 - loss: 1.8287 - val_accuracy: 0.1100 - val_loss: 1.7202 -
learning_rate: 1.0000e-04
Epoch 9/20
13/13             41s 3s/step -
accuracy: 0.2274 - loss: 1.7319 - val_accuracy: 0.2300 - val_loss: 1.6942 -
learning_rate: 1.0000e-04
Epoch 10/20
13/13             89s 4s/step -
accuracy: 0.2161 - loss: 1.7645 - val_accuracy: 0.1100 - val_loss: 1.6961 -
learning_rate: 1.0000e-04
Epoch 11/20
13/13             84s 4s/step -
accuracy: 0.1931 - loss: 1.7808 - val_accuracy: 0.1000 - val_loss: 1.6908 -
learning_rate: 1.0000e-04
Epoch 12/20
13/13             80s 4s/step -
accuracy: 0.2211 - loss: 1.7237 - val_accuracy: 0.1000 - val_loss: 1.6996 -
learning_rate: 1.0000e-04
Epoch 13/20
13/13             81s 4s/step -
accuracy: 0.2255 - loss: 1.7607 - val_accuracy: 0.1000 - val_loss: 1.6904 -
learning_rate: 1.0000e-04
```

```
Epoch 14/20
13/13             80s 4s/step -
accuracy: 0.1748 - loss: 1.7509 - val_accuracy: 0.1200 - val_loss: 1.6798 -
learning_rate: 1.0000e-04
Epoch 15/20
13/13             50s 4s/step -
accuracy: 0.2056 - loss: 1.6924 - val_accuracy: 0.1300 - val_loss: 1.6678 -
learning_rate: 1.0000e-04
Epoch 16/20
13/13             42s 3s/step -
accuracy: 0.2514 - loss: 1.7221 - val_accuracy: 0.1200 - val_loss: 1.6769 -
learning_rate: 1.0000e-04
Epoch 17/20
13/13             41s 3s/step -
accuracy: 0.2226 - loss: 1.6488 - val_accuracy: 0.1500 - val_loss: 1.6404 -
learning_rate: 1.0000e-04
Epoch 18/20
13/13             91s 4s/step -
accuracy: 0.2980 - loss: 1.6018 - val_accuracy: 0.2300 - val_loss: 1.6436 -
learning_rate: 1.0000e-04
Epoch 19/20
13/13             75s 3s/step -
accuracy: 0.2511 - loss: 1.6318 - val_accuracy: 0.2300 - val_loss: 1.6199 -
learning_rate: 1.0000e-04
Epoch 20/20
13/13             43s 3s/step -
accuracy: 0.3054 - loss: 1.5997 - val_accuracy: 0.3200 - val_loss: 1.5685 -
learning_rate: 1.0000e-04
4/4             3s 614ms/step -
accuracy: 0.3384 - loss: 1.5679
Test accuracy: 0.320
```

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()
```

```
4/4                 4s 801ms/step
```