Chromagram-Only CNN

March 21, 2025

1 CNN for Chromagram (30 secs)

1.1 1 - All the imports

[1]:

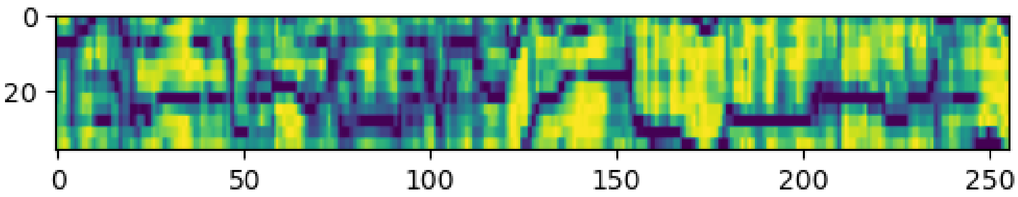
```python
import os
import numpy as np
from sklearn.model_selection import train_test_split
import tensorflow as tf
import matplotlib.pyplot as plt
```

```
2025-03-21 01:32:34.527929: 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]:

```python
# 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, [36, 256])
image = image.numpy()

plt.imshow(image.reshape(36, 256))
plt.show()
```



2 3 - Create the model

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

     model = models.Sequential([
         Conv2D(36, (3, 3), activation='relu', input_shape=(36, 256, 1)),
         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, 34, 254, 36)               360
```

```
max_pooling2d (MaxPooling2D)      (None, 17, 127, 36)              0

flatten (Flatten)                 (None, 77724)                    0

dense (Dense)                     (None, 512)             39,795,200

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: 39,961,074 (152.44 MB)

Trainable params: 39,961,074 (152.44 MB)

Non-trainable params: 0 (0.00 B)
```

3 4 - Load the images

[5]:

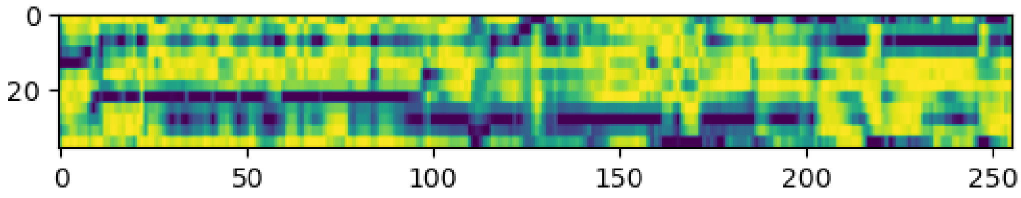
```python
# Load the images from Data/spectrograms/spectrograms_256

# Data/spectrograms/spectrogram_256

GENRES = ['blues', 'classical', 'country', 'disco', 'hiphop', 'jazz', 'metal',␣
 ↪'pop', 'reggae', 'rock']

# GENRES = ["classical", "hiphop", "jazz", "metal", "pop", "rock"]
FILE_PATH = os.path.join('Data', 'chromagrams', 'chromagram_36')
X = []
y = []

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

for genre in GENRES:
    genre_dir = os.path.join(FILE_PATH, genre)
    print(f"Going through {genre}")
    for file in os.listdir(genre_dir):
        try:
            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, [36, 256])
            image = image.numpy()
            X.append(image)
            y.append(GENRE_TO_INDEX[genre])
        except:
            pass

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

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
```

```
2025-03-21 01:33:01.244268: W tensorflow/core/framework/op_kernel.cc:1841]
OP_REQUIRES failed at whole_file_read_ops.cc:116 : FAILED_PRECONDITION:
Data/chromagrams/chromagram_36/country/.ipynb_checkpoints; Is a directory
2025-03-21 01:33:01.248067: I tensorflow/core/framework/local_rendezvous.cc:405]
Local rendezvous is aborting with status: FAILED_PRECONDITION:
Data/chromagrams/chromagram_36/country/.ipynb_checkpoints; Is a directory
```

```
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(36, 256))
plt.show()
```



[7]:

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

[8]:

```
model.fit(X_train, y_train, epochs=20, validation_data=(X_test, y_test),␣
 ↪batch_size=32)
```

```
Epoch 1/20
25/25             20s 605ms/step -
accuracy: 0.0887 - loss: 3.8282 - val_accuracy: 0.1050 - val_loss: 2.3086
Epoch 2/20
25/25             18s 516ms/step -
accuracy: 0.1087 - loss: 2.3252 - val_accuracy: 0.0750 - val_loss: 2.3093
Epoch 3/20
25/25             13s 525ms/step -
accuracy: 0.0837 - loss: 2.3255 - val_accuracy: 0.1050 - val_loss: 2.3070
Epoch 4/20
25/25             19s 478ms/step -
accuracy: 0.1156 - loss: 2.3119 - val_accuracy: 0.0850 - val_loss: 2.3056
Epoch 5/20
25/25             13s 502ms/step -
accuracy: 0.1209 - loss: 2.2963 - val_accuracy: 0.1100 - val_loss: 2.3061
Epoch 6/20
25/25             13s 504ms/step -
accuracy: 0.1141 - loss: 2.2985 - val_accuracy: 0.0700 - val_loss: 2.3078
Epoch 7/20
25/25             21s 510ms/step -
accuracy: 0.1010 - loss: 2.2972 - val_accuracy: 0.1700 - val_loss: 2.3010
Epoch 8/20
25/25             13s 520ms/step -
accuracy: 0.1410 - loss: 2.2775 - val_accuracy: 0.0750 - val_loss: 2.3012
Epoch 9/20
25/25             12s 485ms/step -
accuracy: 0.1610 - loss: 2.2549 - val_accuracy: 0.1850 - val_loss: 2.2285
Epoch 10/20
25/25             12s 476ms/step -
accuracy: 0.2412 - loss: 2.1462 - val_accuracy: 0.1700 - val_loss: 2.0936
Epoch 11/20
25/25             12s 480ms/step -
accuracy: 0.2819 - loss: 1.9968 - val_accuracy: 0.2700 - val_loss: 2.0023
Epoch 12/20
25/25             13s 514ms/step -
accuracy: 0.4096 - loss: 1.7138 - val_accuracy: 0.2650 - val_loss: 1.9469
Epoch 13/20
25/25             11s 444ms/step -
accuracy: 0.4739 - loss: 1.4965 - val_accuracy: 0.2400 - val_loss: 2.0079
Epoch 14/20
```

```
25/25               15s 614ms/step -
accuracy: 0.5945 - loss: 1.2534 - val_accuracy: 0.2650 - val_loss: 2.1258
Epoch 15/20
25/25               16s 633ms/step -
accuracy: 0.6283 - loss: 1.0291 - val_accuracy: 0.2800 - val_loss: 2.0408
Epoch 16/20
25/25               21s 643ms/step -
accuracy: 0.7686 - loss: 0.7454 - val_accuracy: 0.2450 - val_loss: 2.1968
Epoch 17/20
25/25               20s 622ms/step -
accuracy: 0.8107 - loss: 0.5754 - val_accuracy: 0.2400 - val_loss: 2.2549
Epoch 18/20
25/25               21s 634ms/step -
accuracy: 0.8501 - loss: 0.4792 - val_accuracy: 0.2550 - val_loss: 2.6634
Epoch 19/20
25/25               16s 623ms/step -
accuracy: 0.8884 - loss: 0.3575 - val_accuracy: 0.2550 - val_loss: 2.7958
Epoch 20/20
25/25               16s 620ms/step -
accuracy: 0.8950 - loss: 0.3184 - val_accuracy: 0.2950 - val_loss: 2.7490
```

```
[8]: <keras.src.callbacks.history.History at 0x7f9104637ce0>
```

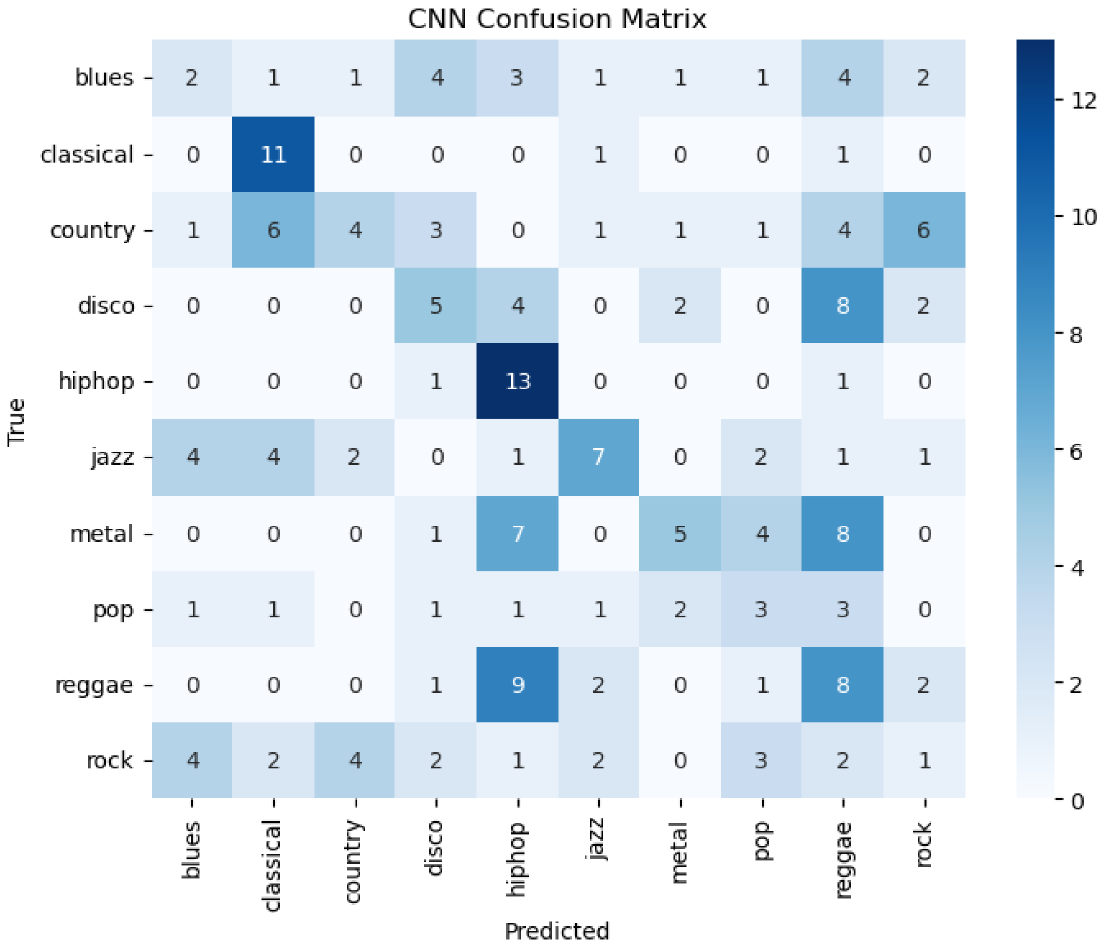
```
[9]: evaluation = model.evaluate(X_test, y_test)
     print(f"Test accuracy: {evaluation[1]:.3f}")
```

```
7/7              1s 80ms/step -
accuracy: 0.3138 - loss: 2.7832
Test accuracy: 0.295
```

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                1s 91ms/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', 'chromagrams', 'chromagram_36')
X = []
y = []

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

```python
# 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):
        try:
            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, [36, 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])
        except:
            continue

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(36, (3, 3), activation='relu', input_shape=(36, 256, 1)),
    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               8s 673ms/step -
accuracy: 0.2683 - loss: 1.9328 - val_accuracy: 0.5250 - val_loss: 0.7993 -
learning_rate: 1.0000e-04
Epoch 2/20
5/5               3s 666ms/step -
accuracy: 0.4878 - loss: 1.2077 - val_accuracy: 0.5250 - val_loss: 0.7842 -
learning_rate: 1.0000e-04
Epoch 3/20
5/5               5s 661ms/step -
accuracy: 0.4925 - loss: 1.0271 - val_accuracy: 0.4750 - val_loss: 0.7034 -
learning_rate: 1.0000e-04
Epoch 4/20
5/5               3s 652ms/step -
accuracy: 0.4592 - loss: 1.1818 - val_accuracy: 0.6750 - val_loss: 0.6938 -
learning_rate: 1.0000e-04
Epoch 5/20
```

```
5/5             3s 662ms/step -
accuracy: 0.4625 - loss: 1.2373 - val_accuracy: 0.5250 - val_loss: 0.7381 -
learning_rate: 1.0000e-04
Epoch 6/20
5/5             3s 649ms/step -
accuracy: 0.4926 - loss: 1.0458 - val_accuracy: 0.5250 - val_loss: 0.7436 -
learning_rate: 1.0000e-04
Epoch 7/20
5/5             3s 632ms/step -
accuracy: 0.5002 - loss: 1.0295 - val_accuracy: 0.5250 - val_loss: 0.7158 -
learning_rate: 1.0000e-04
Epoch 8/20
5/5             3s 644ms/step -
accuracy: 0.4497 - loss: 1.0293 - val_accuracy: 0.5250 - val_loss: 0.7159 -
learning_rate: 5.0000e-05
Epoch 9/20
5/5             3s 642ms/step -
accuracy: 0.4530 - loss: 0.9808 - val_accuracy: 0.5250 - val_loss: 0.7171 -
learning_rate: 5.0000e-05
Epoch 10/20
5/5             3s 646ms/step -
accuracy: 0.5206 - loss: 0.8870 - val_accuracy: 0.5250 - val_loss: 0.7166 -
learning_rate: 5.0000e-05
Epoch 11/20
5/5             3s 617ms/step -
accuracy: 0.5816 - loss: 0.8258 - val_accuracy: 0.5250 - val_loss: 0.7192 -
learning_rate: 2.5000e-05
Epoch 12/20
5/5             5s 666ms/step -
accuracy: 0.5034 - loss: 0.9085 - val_accuracy: 0.5250 - val_loss: 0.7221 -
learning_rate: 2.5000e-05
Epoch 13/20
5/5             3s 635ms/step -
accuracy: 0.5711 - loss: 0.7923 - val_accuracy: 0.5250 - val_loss: 0.7233 -
learning_rate: 2.5000e-05
Epoch 14/20
5/5             5s 713ms/step -
accuracy: 0.5043 - loss: 0.8864 - val_accuracy: 0.5250 - val_loss: 0.7208 -
learning_rate: 1.2500e-05
Epoch 15/20
5/5             3s 660ms/step -
accuracy: 0.5952 - loss: 0.7662 - val_accuracy: 0.5250 - val_loss: 0.7198 -
learning_rate: 1.2500e-05
Epoch 16/20
5/5             5s 658ms/step -
accuracy: 0.5133 - loss: 0.8671 - val_accuracy: 0.5250 - val_loss: 0.7186 -
learning_rate: 1.2500e-05
Epoch 17/20
```

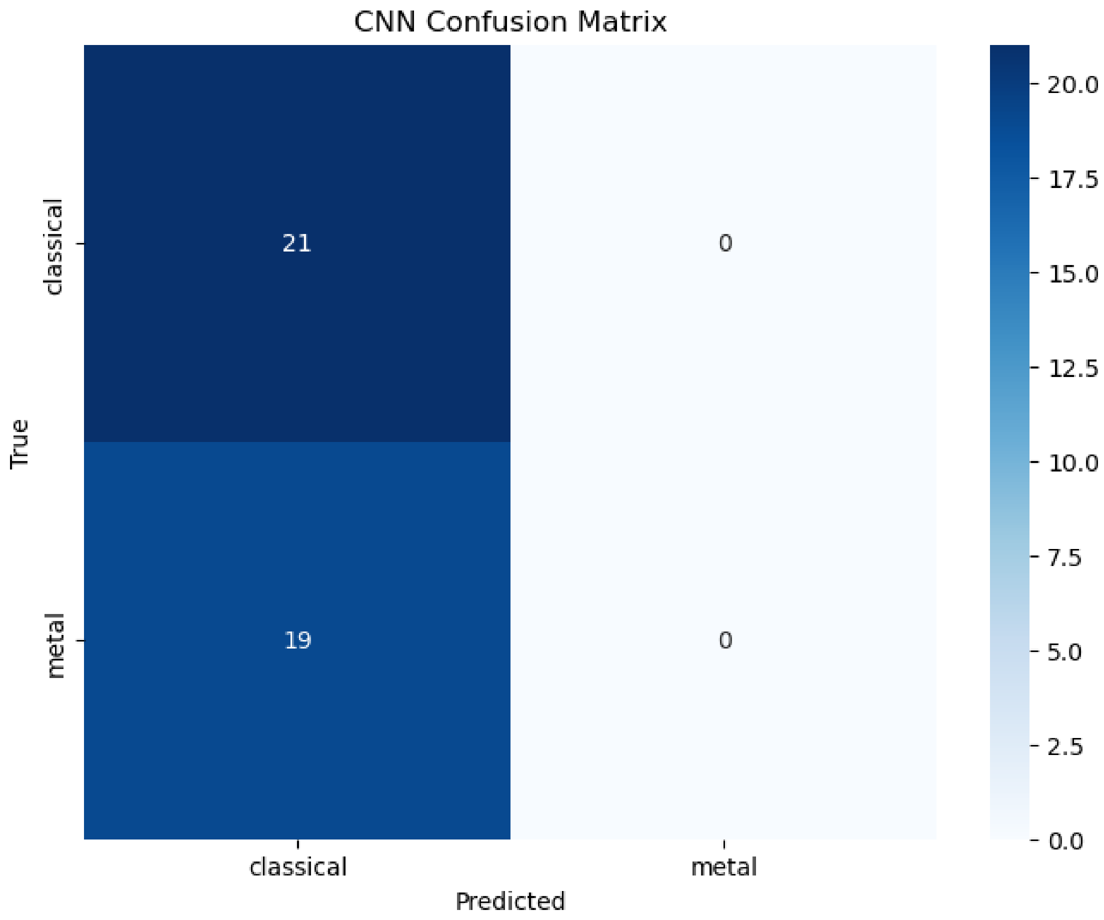
```
5/5              5s 657ms/step -
accuracy: 0.5617 - loss: 0.7787 - val_accuracy: 0.5250 - val_loss: 0.7194 -
learning_rate: 6.2500e-06
Epoch 18/20
5/5              5s 706ms/step -
accuracy: 0.5549 - loss: 0.8388 - val_accuracy: 0.5250 - val_loss: 0.7193 -
learning_rate: 6.2500e-06
Epoch 19/20
5/5              5s 673ms/step -
accuracy: 0.4526 - loss: 0.8574 - val_accuracy: 0.5250 - val_loss: 0.7183 -
learning_rate: 6.2500e-06
Epoch 20/20
5/5              5s 677ms/step -
accuracy: 0.5134 - loss: 0.9232 - val_accuracy: 0.5250 - val_loss: 0.7176 -
learning_rate: 3.1250e-06
2/2              0s 35ms/step -
accuracy: 0.5375 - loss: 0.7107
Test accuracy: 0.525
```

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()
```

```
2/2              0s 209ms/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', 'chromagrams', 'chromagram_36')
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):
        try:
            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, [36, 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])
        except:
            continue

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(36, (3, 3), activation='relu', input_shape=(36, 256, 1)),
    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               9s 684ms/step -
accuracy: 0.3586 - loss: 1.7353 - val_accuracy: 0.5250 - val_loss: 0.7731 -
learning_rate: 1.0000e-04
Epoch 2/20
5/5               5s 593ms/step -
accuracy: 0.4812 - loss: 1.2302 - val_accuracy: 0.4750 - val_loss: 0.7205 -
learning_rate: 1.0000e-04
Epoch 3/20
5/5               3s 578ms/step -
accuracy: 0.5442 - loss: 1.1762 - val_accuracy: 0.4750 - val_loss: 0.7089 -
learning_rate: 1.0000e-04
Epoch 4/20
5/5               5s 597ms/step -
accuracy: 0.5174 - loss: 1.0011 - val_accuracy: 0.4750 - val_loss: 0.7286 -
learning_rate: 1.0000e-04
Epoch 5/20
5/5               3s 585ms/step -
accuracy: 0.5332 - loss: 1.1260 - val_accuracy: 0.4750 - val_loss: 0.7189 -
learning_rate: 1.0000e-04
Epoch 6/20
```

```
5/5            3s 651ms/step -
accuracy: 0.4221 - loss: 1.1033 - val_accuracy: 0.4750 - val_loss: 0.7362 -
learning_rate: 1.0000e-04
Epoch 7/20
5/5            3s 557ms/step -
accuracy: 0.4797 - loss: 1.0079 - val_accuracy: 0.4750 - val_loss: 0.7543 -
learning_rate: 5.0000e-05
Epoch 8/20
5/5            5s 559ms/step -
accuracy: 0.4394 - loss: 1.1952 - val_accuracy: 0.4750 - val_loss: 0.7532 -
learning_rate: 5.0000e-05
Epoch 9/20
5/5            3s 587ms/step -
accuracy: 0.4943 - loss: 0.9226 - val_accuracy: 0.4750 - val_loss: 0.7434 -
learning_rate: 5.0000e-05
Epoch 10/20
5/5            6s 705ms/step -
accuracy: 0.5376 - loss: 0.9564 - val_accuracy: 0.4750 - val_loss: 0.7390 -
learning_rate: 2.5000e-05
Epoch 11/20
5/5            5s 658ms/step -
accuracy: 0.4997 - loss: 0.8991 - val_accuracy: 0.4750 - val_loss: 0.7375 -
learning_rate: 2.5000e-05
Epoch 12/20
5/5            3s 698ms/step -
accuracy: 0.5333 - loss: 0.8613 - val_accuracy: 0.4750 - val_loss: 0.7368 -
learning_rate: 2.5000e-05
Epoch 13/20
5/5            3s 582ms/step -
accuracy: 0.3935 - loss: 1.0626 - val_accuracy: 0.4750 - val_loss: 0.7382 -
learning_rate: 1.2500e-05
Epoch 14/20
5/5            6s 696ms/step -
accuracy: 0.4749 - loss: 0.9155 - val_accuracy: 0.4750 - val_loss: 0.7374 -
learning_rate: 1.2500e-05
Epoch 15/20
5/5            5s 718ms/step -
accuracy: 0.3780 - loss: 1.0891 - val_accuracy: 0.4750 - val_loss: 0.7388 -
learning_rate: 1.2500e-05
Epoch 16/20
5/5            5s 714ms/step -
accuracy: 0.5343 - loss: 0.9041 - val_accuracy: 0.4750 - val_loss: 0.7391 -
learning_rate: 6.2500e-06
Epoch 17/20
5/5            3s 625ms/step -
accuracy: 0.5332 - loss: 0.8844 - val_accuracy: 0.4750 - val_loss: 0.7397 -
learning_rate: 6.2500e-06
Epoch 18/20
```

```
5/5             3s 673ms/step -
accuracy: 0.4250 - loss: 0.9915 - val_accuracy: 0.4750 - val_loss: 0.7405 -
learning_rate: 6.2500e-06
Epoch 19/20
5/5             5s 652ms/step -
accuracy: 0.5087 - loss: 0.9428 - val_accuracy: 0.4750 - val_loss: 0.7411 -
learning_rate: 3.1250e-06
Epoch 20/20
5/5             5s 636ms/step -
accuracy: 0.4782 - loss: 0.8978 - val_accuracy: 0.4750 - val_loss: 0.7407 -
learning_rate: 3.1250e-06
2/2             0s 49ms/step -
accuracy: 0.4625 - loss: 0.7434
Test accuracy: 0.475
```

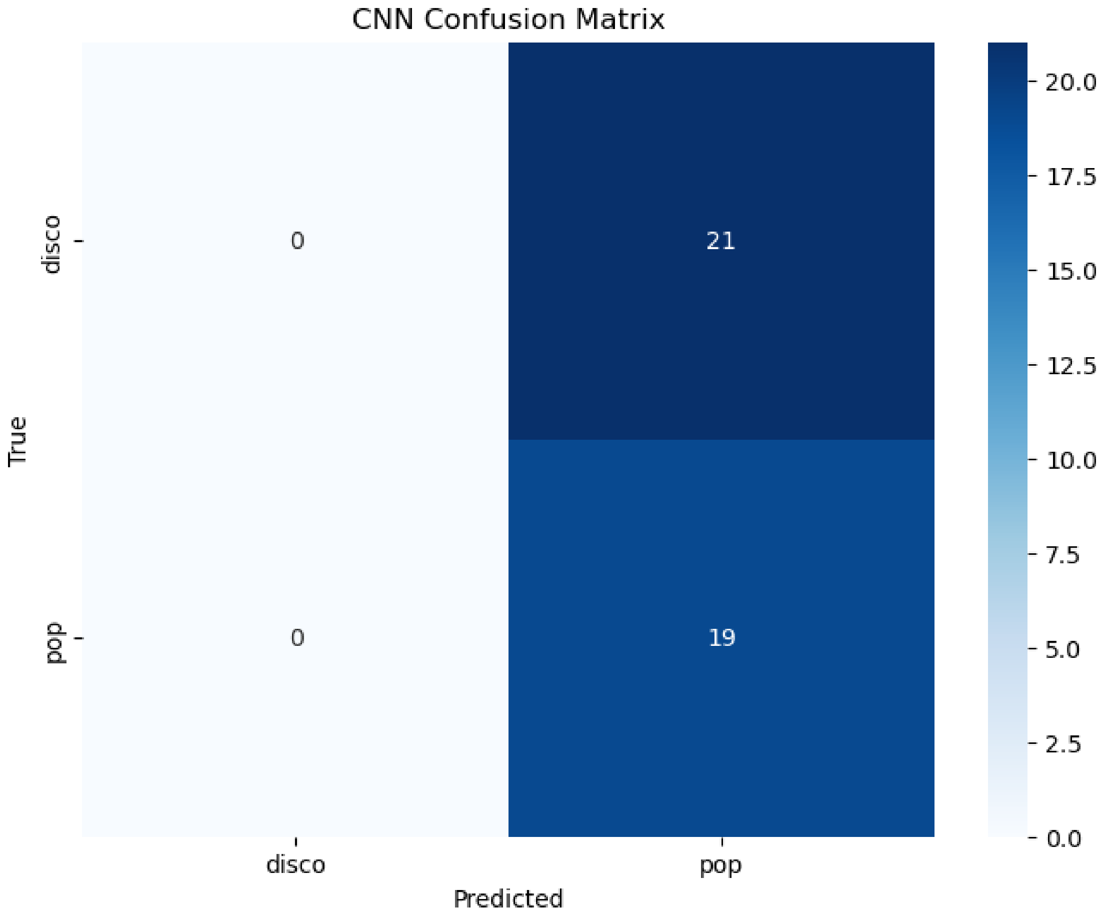
4.4 12 - Confusion Matrix Hard (disco and pop)

[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
0x7f90b07893a0> 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             0s
370ms/stepWARNING:tensorflow:6 out of the last 11 calls to <function
TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at
0x7f90b07893a0> 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             1s 440ms/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
```

```python
GENRES = ['blues', 'classical', 'country', 'disco', 'hiphop', 'jazz', 'metal', 'pop', 'reggae', 'rock']
GENRES = random.sample(GENRES, 5)
print(GENRES)
FILE_PATH = os.path.join('Data', 'chromagrams', 'chromagram_36')
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):
        try:
            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, [36, 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])
        except:
            continue

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
```

```python
model = models.Sequential([
    Conv2D(36, (3, 3), activation='relu', input_shape=(36, 256, 1)),
    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}")
```

```
['reggae', 'pop', 'jazz', 'blues', 'hiphop']
Going through reggae
Going through pop
Going through jazz
Going through blues
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                  14s 686ms/step -
accuracy: 0.1441 - loss: 2.2368 - val_accuracy: 0.1400 - val_loss: 1.8219 -
```

```
learning_rate: 1.0000e-04
Epoch 2/20
13/13             9s 666ms/step -
accuracy: 0.1910 - loss: 2.0577 - val_accuracy: 0.1000 - val_loss: 1.8728 -
learning_rate: 1.0000e-04
Epoch 3/20
13/13             9s 653ms/step -
accuracy: 0.2305 - loss: 1.9831 - val_accuracy: 0.1400 - val_loss: 1.7738 -
learning_rate: 1.0000e-04
Epoch 4/20
13/13             10s 629ms/step -
accuracy: 0.1846 - loss: 1.9247 - val_accuracy: 0.1000 - val_loss: 1.8802 -
learning_rate: 1.0000e-04
Epoch 5/20
13/13             9s 537ms/step -
accuracy: 0.1937 - loss: 1.8909 - val_accuracy: 0.1000 - val_loss: 1.8156 -
learning_rate: 1.0000e-04
Epoch 6/20
13/13             11s 558ms/step -
accuracy: 0.1958 - loss: 1.8502 - val_accuracy: 0.1000 - val_loss: 1.8489 -
learning_rate: 1.0000e-04
Epoch 7/20
13/13             10s 568ms/step -
accuracy: 0.1955 - loss: 1.8599 - val_accuracy: 0.1000 - val_loss: 1.8579 -
learning_rate: 5.0000e-05
Epoch 8/20
13/13             11s 600ms/step -
accuracy: 0.2076 - loss: 1.8512 - val_accuracy: 0.1000 - val_loss: 1.8223 -
learning_rate: 5.0000e-05
Epoch 9/20
13/13             8s 599ms/step -
accuracy: 0.2078 - loss: 1.8040 - val_accuracy: 0.1000 - val_loss: 1.8352 -
learning_rate: 5.0000e-05
Epoch 10/20
13/13             8s 605ms/step -
accuracy: 0.1917 - loss: 1.8259 - val_accuracy: 0.1000 - val_loss: 1.8367 -
learning_rate: 2.5000e-05
Epoch 11/20
13/13             10s 617ms/step -
accuracy: 0.2060 - loss: 1.7998 - val_accuracy: 0.1100 - val_loss: 1.8123 -
learning_rate: 2.5000e-05
Epoch 12/20
13/13             8s 580ms/step -
accuracy: 0.2220 - loss: 1.7942 - val_accuracy: 0.1200 - val_loss: 1.8006 -
learning_rate: 2.5000e-05
Epoch 13/20
13/13             8s 586ms/step -
accuracy: 0.2000 - loss: 1.8172 - val_accuracy: 0.0900 - val_loss: 1.8048 -
```

```
learning_rate: 1.2500e-05
Epoch 14/20
13/13                 10s 558ms/step -
accuracy: 0.2095 - loss: 1.7522 - val_accuracy: 0.1000 - val_loss: 1.8045 -
learning_rate: 1.2500e-05
Epoch 15/20
13/13                 10s 580ms/step -
accuracy: 0.1628 - loss: 1.8296 - val_accuracy: 0.1000 - val_loss: 1.7934 -
learning_rate: 1.2500e-05
Epoch 16/20
13/13                 7s 570ms/step -
accuracy: 0.2259 - loss: 1.7633 - val_accuracy: 0.1000 - val_loss: 1.7880 -
learning_rate: 6.2500e-06
Epoch 17/20
13/13                 8s 590ms/step -
accuracy: 0.2286 - loss: 1.7458 - val_accuracy: 0.1000 - val_loss: 1.7852 -
learning_rate: 6.2500e-06
Epoch 18/20
13/13                 7s 552ms/step -
accuracy: 0.2568 - loss: 1.7470 - val_accuracy: 0.1000 - val_loss: 1.7894 -
learning_rate: 6.2500e-06
Epoch 19/20
13/13                 7s 569ms/step -
accuracy: 0.2391 - loss: 1.7286 - val_accuracy: 0.1000 - val_loss: 1.7898 -
learning_rate: 3.1250e-06
Epoch 20/20
13/13                 7s 549ms/step -
accuracy: 0.2045 - loss: 1.7391 - val_accuracy: 0.1000 - val_loss: 1.7904 -
learning_rate: 3.1250e-06
4/4                 0s 30ms/step -
accuracy: 0.0890 - loss: 1.7922
Test accuracy: 0.100
```

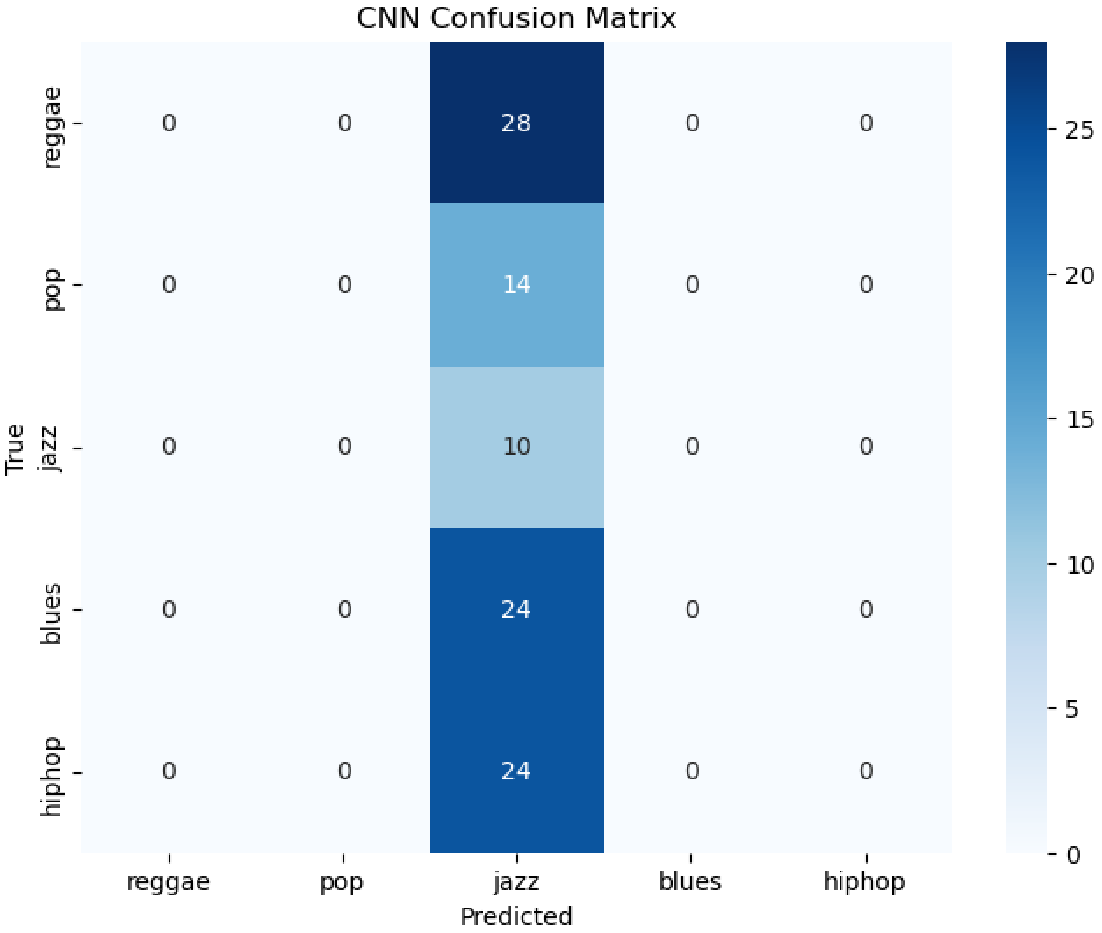
4.6 14 - Confusion Matrix Medium (5 random)

[16]:

```
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                1s 104ms/step
```



```
[ ]:
```