Chromagram-Only (3 secs) CNN

March 22, 2025

1 CNN for Chromagram only (3 secs)

1.1 1 - All 10

[1]:
```
import os
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from tensorflow.keras import models
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
↪Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau
import matplotlib.pyplot as plt

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

# Define the genres and file paths
GENRES = ['blues', 'classical', 'country', 'disco', 'hiphop', 'jazz', 'metal',␣
↪'pop', 'reggae', 'rock']
FILE_PATH = os.path.join('Data', 'chromagrams (3 secs)', 'chromagram_12')

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

# Organize data by song ID
song_to_clips = {}

for genre in GENRES:
    genre_dir = os.path.join(FILE_PATH, genre)
    print(f"Processing genre: {genre}")
    for file in os.listdir(genre_dir):
        if not file.endswith(".png"):
```

```
            continue

        song_id = file.split("_clip_")[0]  # Extract song ID (e.g., "blues.
↪00042")

        if song_id not in song_to_clips:
            song_to_clips[song_id] = []

        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])  # Resize to 256x256
        image = augment_image(image)  # Apply augmentation
        image = image.numpy()  # Convert to numpy array

        song_to_clips[song_id].append((image, GENRE_TO_INDEX[genre]))

# Convert dictionary to list format
song_ids = list(song_to_clips.keys())
train_ids, test_ids = train_test_split(song_ids, test_size=0.2, random_state=42)

X_train, y_train, X_test, y_test = [], [], [], []

# Assign clips based on the train-test split
for song_id in song_ids:
    clips = song_to_clips[song_id]
    if song_id in train_ids:
        for image, label in clips:
            X_train.append(image)
            y_train.append(label)
    else:
        for image, label in clips:
            X_test.append(image)
            y_test.append(label)

# Convert to numpy arrays
X_train = np.array(X_train)
y_train = np.array(y_train)
X_test = np.array(X_test)
y_test = np.array(y_test)

print(f"Train set: {len(X_train)} samples")
print(f"Test set: {len(X_test)} samples")

# Define the CNN model
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')
])

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

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

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

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

```
2025-03-22 01:34:19.393342: 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.

Processing genre: blues
Processing genre: classical
Processing genre: country
Processing genre: disco
Processing genre: hiphop
Processing genre: jazz
Processing genre: metal
Processing genre: pop
Processing genre: reggae
Processing genre: rock
Train set: 8000 samples
Test set: 2000 samples
```

```
/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
250/250             143s 547ms/step -
accuracy: 0.1024 - loss: 2.4198 - val_accuracy: 0.0650 - val_loss: 2.3038 -
learning_rate: 1.0000e-04
Epoch 2/20
250/250             132s 527ms/step -
accuracy: 0.1060 - loss: 2.3027 - val_accuracy: 0.0650 - val_loss: 2.3055 -
learning_rate: 1.0000e-04
Epoch 3/20
250/250             142s 567ms/step -
accuracy: 0.1077 - loss: 2.3021 - val_accuracy: 0.0650 - val_loss: 2.3110 -
learning_rate: 1.0000e-04
Epoch 4/20
250/250             156s 623ms/step -
accuracy: 0.1055 - loss: 2.3020 - val_accuracy: 0.0650 - val_loss: 2.3087 -
learning_rate: 1.0000e-04
Epoch 5/20
250/250             196s 599ms/step -
accuracy: 0.1068 - loss: 2.3009 - val_accuracy: 0.0650 - val_loss: 2.3094 -
learning_rate: 5.0000e-05
Epoch 6/20
250/250             147s 586ms/step -
accuracy: 0.1113 - loss: 2.3011 - val_accuracy: 0.0650 - val_loss: 2.3100 -
learning_rate: 5.0000e-05
Epoch 7/20
250/250             164s 657ms/step -
accuracy: 0.1122 - loss: 2.3012 - val_accuracy: 0.0650 - val_loss: 2.3107 -
learning_rate: 5.0000e-05
Epoch 8/20
250/250             196s 635ms/step -
accuracy: 0.1075 - loss: 2.3013 - val_accuracy: 0.0650 - val_loss: 2.3110 -
learning_rate: 2.5000e-05
Epoch 9/20
250/250             208s 660ms/step -
accuracy: 0.1098 - loss: 2.3012 - val_accuracy: 0.0650 - val_loss: 2.3112 -
learning_rate: 2.5000e-05
Epoch 10/20
250/250             207s 678ms/step -
accuracy: 0.1180 - loss: 2.3004 - val_accuracy: 0.0650 - val_loss: 2.3115 -
learning_rate: 2.5000e-05
Epoch 11/20
```

```
250/250             198s 663ms/step -
accuracy: 0.1154 - loss: 2.3006 - val_accuracy: 0.0650 - val_loss: 2.3116 -
learning_rate: 1.2500e-05
Epoch 12/20
250/250             173s 691ms/step -
accuracy: 0.1110 - loss: 2.3007 - val_accuracy: 0.0650 - val_loss: 2.3118 -
learning_rate: 1.2500e-05
Epoch 13/20
250/250             198s 674ms/step -
accuracy: 0.1103 - loss: 2.3013 - val_accuracy: 0.0650 - val_loss: 2.3119 -
learning_rate: 1.2500e-05
Epoch 14/20
250/250             158s 632ms/step -
accuracy: 0.1067 - loss: 2.3012 - val_accuracy: 0.0650 - val_loss: 2.3120 -
learning_rate: 6.2500e-06
Epoch 15/20
250/250             202s 632ms/step -
accuracy: 0.1099 - loss: 2.3016 - val_accuracy: 0.0650 - val_loss: 2.3120 -
learning_rate: 6.2500e-06
Epoch 16/20
250/250             207s 654ms/step -
accuracy: 0.1059 - loss: 2.3018 - val_accuracy: 0.0650 - val_loss: 2.3121 -
learning_rate: 6.2500e-06
Epoch 17/20
250/250             205s 667ms/step -
accuracy: 0.1101 - loss: 2.3008 - val_accuracy: 0.0650 - val_loss: 2.3121 -
learning_rate: 3.1250e-06
Epoch 18/20
250/250             205s 677ms/step -
accuracy: 0.1058 - loss: 2.3011 - val_accuracy: 0.0650 - val_loss: 2.3122 -
learning_rate: 3.1250e-06
Epoch 19/20
250/250             161s 642ms/step -
accuracy: 0.1164 - loss: 2.3006 - val_accuracy: 0.0650 - val_loss: 2.3122 -
learning_rate: 3.1250e-06
Epoch 20/20
250/250             199s 632ms/step -
accuracy: 0.1010 - loss: 2.3016 - val_accuracy: 0.0650 - val_loss: 2.3122 -
learning_rate: 1.5625e-06
63/63              6s 96ms/step -
accuracy: 0.1310 - loss: 2.3084
Test accuracy: 0.065
```

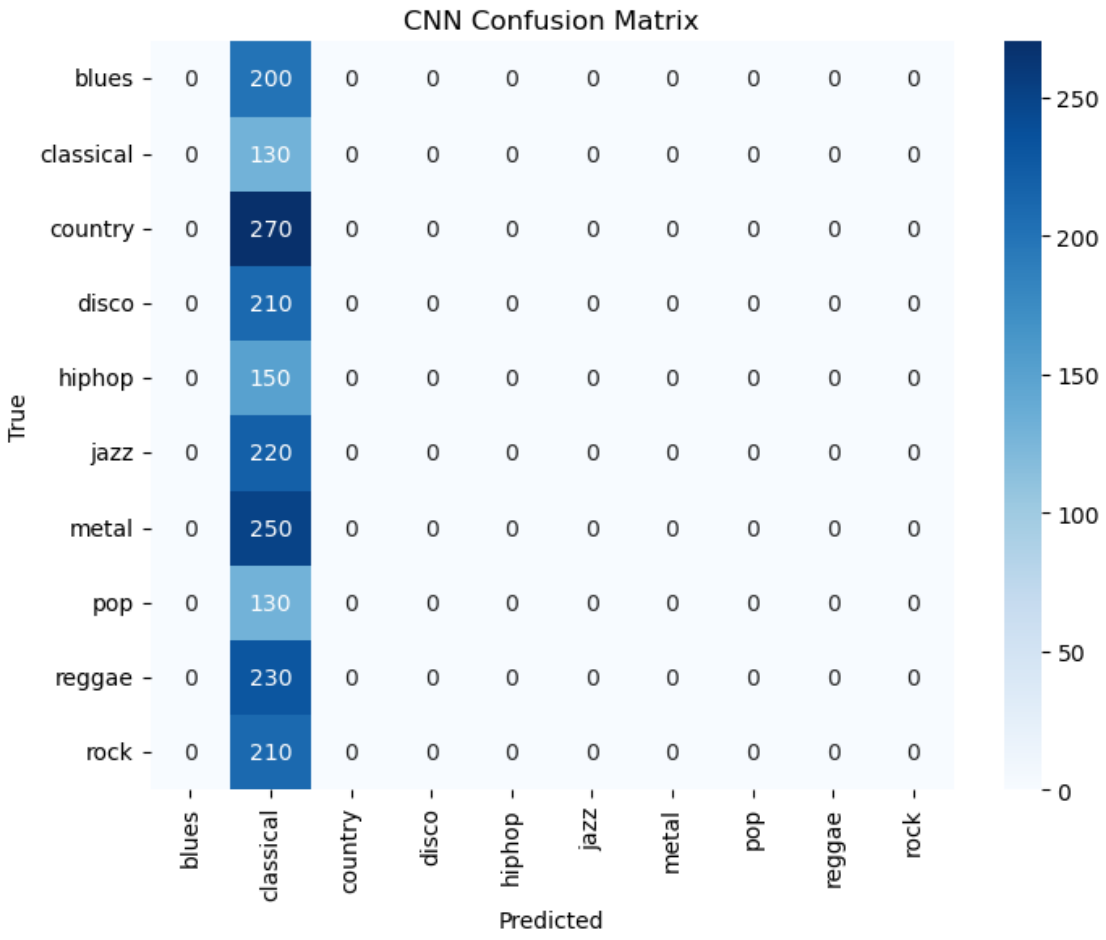
1.2 Apply the confusion matrix after the model

[2]:
```
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                6s 93ms/step
```



1.3 2 - Limited Genres Easy (metal and classical)

[3]:
```
import os
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from tensorflow.keras import models
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
 ↪Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau
import matplotlib.pyplot as plt

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

# Define the genres and file paths
GENRES = ['classical', 'metal']
FILE_PATH = os.path.join('Data', 'chromagrams (3 secs)', 'chromagram_12')

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

# Organize data by song ID
song_to_clips = {}

for genre in GENRES:
    genre_dir = os.path.join(FILE_PATH, genre)
    print(f"Processing genre: {genre}")
    for file in os.listdir(genre_dir):
        if not file.endswith(".png"):
            continue

        song_id = file.split("_clip_")[0]  # Extract song ID (e.g., "blues.
 ↪00042")

        if song_id not in song_to_clips:
            song_to_clips[song_id] = []

        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])  # Resize to 256x256
        image = augment_image(image)  # Apply augmentation
        image = image.numpy()  # Convert to numpy array

        song_to_clips[song_id].append((image, GENRE_TO_INDEX[genre]))

# Convert dictionary to list format
song_ids = list(song_to_clips.keys())
train_ids, test_ids = train_test_split(song_ids, test_size=0.2, random_state=42)

X_train, y_train, X_test, y_test = [], [], [], []

# Assign clips based on the train-test split
for song_id in song_ids:
    clips = song_to_clips[song_id]
    if song_id in train_ids:
        for image, label in clips:
            X_train.append(image)
            y_train.append(label)
    else:
        for image, label in clips:
            X_test.append(image)
            y_test.append(label)

# Convert to numpy arrays
X_train = np.array(X_train)
y_train = np.array(y_train)
X_test = np.array(X_test)
y_test = np.array(y_test)

print(f"Train set: {len(X_train)} samples")
print(f"Test set: {len(X_test)} samples")

# Define the CNN model
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')
])

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

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

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

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

```
Processing genre: classical
Processing genre: metal
Train set: 1600 samples
Test set: 400 samples
```

```
/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             39s 650ms/step -
accuracy: 0.4665 - loss: 1.2953 - val_accuracy: 0.5250 - val_loss: 0.8456 -
learning_rate: 1.0000e-04
Epoch 2/20
50/50             33s 655ms/step -
accuracy: 0.5162 - loss: 0.9208 - val_accuracy: 0.5250 - val_loss: 0.7112 -
learning_rate: 1.0000e-04
Epoch 3/20
50/50             30s 600ms/step -
accuracy: 0.5785 - loss: 0.7466 - val_accuracy: 0.5500 - val_loss: 0.6528 -
learning_rate: 1.0000e-04
Epoch 4/20
50/50             30s 591ms/step -
accuracy: 0.6466 - loss: 0.6751 - val_accuracy: 0.8975 - val_loss: 0.4590 -
learning_rate: 1.0000e-04
```

```
Epoch 5/20
50/50             32s 638ms/step -
accuracy: 0.7096 - loss: 0.5861 - val_accuracy: 0.8425 - val_loss: 0.3765 -
learning_rate: 1.0000e-04
Epoch 6/20
50/50             28s 567ms/step -
accuracy: 0.7791 - loss: 0.4861 - val_accuracy: 0.9075 - val_loss: 0.2984 -
learning_rate: 1.0000e-04
Epoch 7/20
50/50             30s 599ms/step -
accuracy: 0.8053 - loss: 0.4428 - val_accuracy: 0.9025 - val_loss: 0.2808 -
learning_rate: 1.0000e-04
Epoch 8/20
50/50             43s 645ms/step -
accuracy: 0.8557 - loss: 0.3730 - val_accuracy: 0.9250 - val_loss: 0.2201 -
learning_rate: 1.0000e-04
Epoch 9/20
50/50             41s 640ms/step -
accuracy: 0.8784 - loss: 0.3214 - val_accuracy: 0.9300 - val_loss: 0.2305 -
learning_rate: 1.0000e-04
Epoch 10/20
50/50             33s 663ms/step -
accuracy: 0.8668 - loss: 0.3536 - val_accuracy: 0.9200 - val_loss: 0.2148 -
learning_rate: 1.0000e-04
Epoch 11/20
50/50             30s 591ms/step -
accuracy: 0.9145 - loss: 0.2563 - val_accuracy: 0.9350 - val_loss: 0.1846 -
learning_rate: 1.0000e-04
Epoch 12/20
50/50             30s 605ms/step -
accuracy: 0.9214 - loss: 0.2406 - val_accuracy: 0.9400 - val_loss: 0.1736 -
learning_rate: 1.0000e-04
Epoch 13/20
50/50             32s 643ms/step -
accuracy: 0.9249 - loss: 0.2154 - val_accuracy: 0.9275 - val_loss: 0.1849 -
learning_rate: 1.0000e-04
Epoch 14/20
50/50             33s 650ms/step -
accuracy: 0.9274 - loss: 0.2165 - val_accuracy: 0.9450 - val_loss: 0.1495 -
learning_rate: 1.0000e-04
Epoch 15/20
50/50             32s 633ms/step -
accuracy: 0.9430 - loss: 0.1907 - val_accuracy: 0.9475 - val_loss: 0.1494 -
learning_rate: 1.0000e-04
Epoch 16/20
50/50             31s 621ms/step -
accuracy: 0.9376 - loss: 0.1808 - val_accuracy: 0.9425 - val_loss: 0.1596 -
learning_rate: 1.0000e-04
```

```
Epoch 17/20
50/50             32s 646ms/step -
accuracy: 0.9684 - loss: 0.1099 - val_accuracy: 0.9425 - val_loss: 0.1451 -
learning_rate: 1.0000e-04
Epoch 18/20
50/50             33s 660ms/step -
accuracy: 0.9458 - loss: 0.1558 - val_accuracy: 0.9525 - val_loss: 0.1359 -
learning_rate: 1.0000e-04
Epoch 19/20
50/50             30s 599ms/step -
accuracy: 0.9605 - loss: 0.1231 - val_accuracy: 0.9500 - val_loss: 0.1393 -
learning_rate: 1.0000e-04
Epoch 20/20
50/50             29s 583ms/step -
accuracy: 0.9637 - loss: 0.1144 - val_accuracy: 0.9475 - val_loss: 0.1374 -
learning_rate: 1.0000e-04
13/13             1s 92ms/step -
accuracy: 0.9354 - loss: 0.1542
Test accuracy: 0.947
```

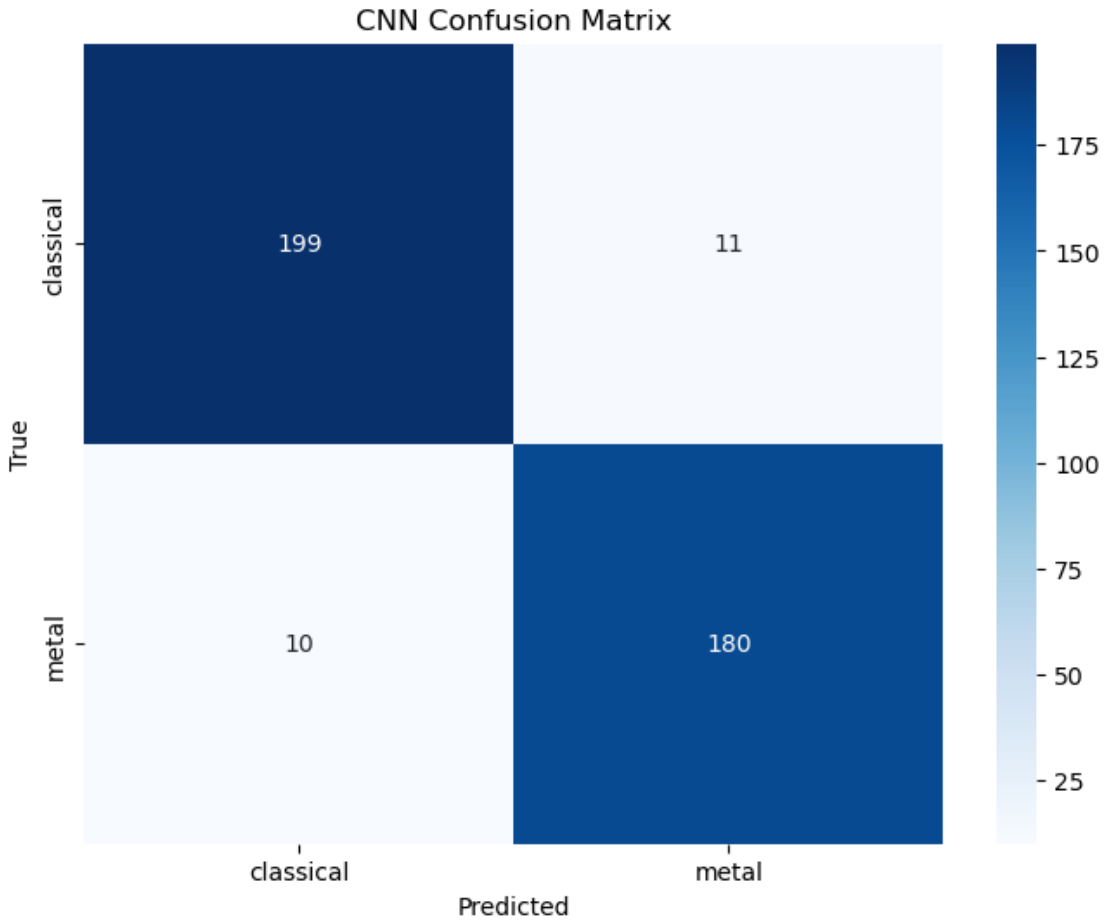
1.4 Confusion Matrix Easy (classical and metal)

[4]:
```
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             1s 85ms/step
```



1.5 3 - Limited genres Hard (disco and pop)

[5]:
```
import os
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from tensorflow.keras import models
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
↪Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau
import matplotlib.pyplot as plt

# Augmentation function
def augment_image(image):
    image = tf.image.random_flip_left_right(image)
    image = tf.image.random_brightness(image, max_delta=0.1)
```

```
    image = tf.image.random_contrast(image, 0.8, 1.2)
    return image

# Define the genres and file paths
GENRES = ['disco', 'pop']
FILE_PATH = os.path.join('Data', 'chromagrams (3 secs)', 'chromagram_12')

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

# Organize data by song ID
song_to_clips = {}

for genre in GENRES:
    genre_dir = os.path.join(FILE_PATH, genre)
    print(f"Processing genre: {genre}")
    for file in os.listdir(genre_dir):
        if not file.endswith(".png"):
            continue

        song_id = file.split("_clip_")[0]  # Extract song ID (e.g., "blues.
 ↪00042")

        if song_id not in song_to_clips:
            song_to_clips[song_id] = []

        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])  # Resize to 256x256
        image = augment_image(image)  # Apply augmentation
        image = image.numpy()  # Convert to numpy array

        song_to_clips[song_id].append((image, GENRE_TO_INDEX[genre]))

# Convert dictionary to list format
song_ids = list(song_to_clips.keys())
train_ids, test_ids = train_test_split(song_ids, test_size=0.2, random_state=42)

X_train, y_train, X_test, y_test = [], [], [], []

# Assign clips based on the train-test split
for song_id in song_ids:
    clips = song_to_clips[song_id]
    if song_id in train_ids:
        for image, label in clips:
            X_train.append(image)
            y_train.append(label)
```

```
    else:
        for image, label in clips:
            X_test.append(image)
            y_test.append(label)

# Convert to numpy arrays
X_train = np.array(X_train)
y_train = np.array(y_train)
X_test = np.array(X_test)
y_test = np.array(y_test)

print(f"Train set: {len(X_train)} samples")
print(f"Test set: {len(X_test)} samples")

# Define the CNN model
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')
])

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

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

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

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

```
Processing genre: disco
Processing genre: pop
Train set: 1600 samples
Test set: 400 samples

/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             38s 644ms/step -
accuracy: 0.4577 - loss: 1.2118 - val_accuracy: 0.4750 - val_loss: 0.7505 -
learning_rate: 1.0000e-04
Epoch 2/20
50/50             31s 627ms/step -
accuracy: 0.4945 - loss: 0.8667 - val_accuracy: 0.4750 - val_loss: 0.7591 -
learning_rate: 1.0000e-04
Epoch 3/20
50/50             32s 642ms/step -
accuracy: 0.4957 - loss: 0.8270 - val_accuracy: 0.4750 - val_loss: 0.7557 -
learning_rate: 1.0000e-04
Epoch 4/20
50/50             33s 650ms/step -
accuracy: 0.5011 - loss: 0.8108 - val_accuracy: 0.4750 - val_loss: 0.7631 -
learning_rate: 1.0000e-04
Epoch 5/20
50/50             33s 650ms/step -
accuracy: 0.4974 - loss: 0.7734 - val_accuracy: 0.4750 - val_loss: 0.7319 -
learning_rate: 5.0000e-05
Epoch 6/20
50/50             41s 659ms/step -
accuracy: 0.5312 - loss: 0.7385 - val_accuracy: 0.4750 - val_loss: 0.7209 -
learning_rate: 5.0000e-05
Epoch 7/20
50/50             31s 607ms/step -
accuracy: 0.4942 - loss: 0.7513 - val_accuracy: 0.4750 - val_loss: 0.7301 -
learning_rate: 5.0000e-05
Epoch 8/20
50/50             42s 634ms/step -
accuracy: 0.5134 - loss: 0.7452 - val_accuracy: 0.4750 - val_loss: 0.7351 -
learning_rate: 5.0000e-05
Epoch 9/20
50/50             30s 587ms/step -
accuracy: 0.5248 - loss: 0.7260 - val_accuracy: 0.4750 - val_loss: 0.7194 -
learning_rate: 5.0000e-05
Epoch 10/20
```

```
50/50            29s 583ms/step -
accuracy: 0.5099 - loss: 0.7431 - val_accuracy: 0.4750 - val_loss: 0.7137 -
learning_rate: 5.0000e-05
Epoch 11/20
50/50            43s 630ms/step -
accuracy: 0.5234 - loss: 0.7253 - val_accuracy: 0.4750 - val_loss: 0.7110 -
learning_rate: 5.0000e-05
Epoch 12/20
50/50            35s 695ms/step -
accuracy: 0.5004 - loss: 0.7350 - val_accuracy: 0.4750 - val_loss: 0.7102 -
learning_rate: 5.0000e-05
Epoch 13/20
50/50            35s 689ms/step -
accuracy: 0.5306 - loss: 0.7105 - val_accuracy: 0.4800 - val_loss: 0.7048 -
learning_rate: 5.0000e-05
Epoch 14/20
50/50            36s 590ms/step -
accuracy: 0.5258 - loss: 0.7145 - val_accuracy: 0.4750 - val_loss: 0.7082 -
learning_rate: 5.0000e-05
Epoch 15/20
50/50            45s 672ms/step -
accuracy: 0.5377 - loss: 0.7168 - val_accuracy: 0.4750 - val_loss: 0.7134 -
learning_rate: 5.0000e-05
Epoch 16/20
50/50            33s 661ms/step -
accuracy: 0.5100 - loss: 0.7166 - val_accuracy: 0.4750 - val_loss: 0.7194 -
learning_rate: 5.0000e-05
Epoch 17/20
50/50            40s 646ms/step -
accuracy: 0.5139 - loss: 0.7193 - val_accuracy: 0.4750 - val_loss: 0.7157 -
learning_rate: 2.5000e-05
Epoch 18/20
50/50            40s 636ms/step -
accuracy: 0.5315 - loss: 0.7083 - val_accuracy: 0.4750 - val_loss: 0.7110 -
learning_rate: 2.5000e-05
Epoch 19/20
50/50            33s 651ms/step -
accuracy: 0.5147 - loss: 0.7142 - val_accuracy: 0.4750 - val_loss: 0.7084 -
learning_rate: 2.5000e-05
Epoch 20/20
50/50            41s 655ms/step -
accuracy: 0.5194 - loss: 0.7076 - val_accuracy: 0.4750 - val_loss: 0.7067 -
learning_rate: 1.2500e-05
13/13            1s 86ms/step -
accuracy: 0.1902 - loss: 0.7699
Test accuracy: 0.475
```

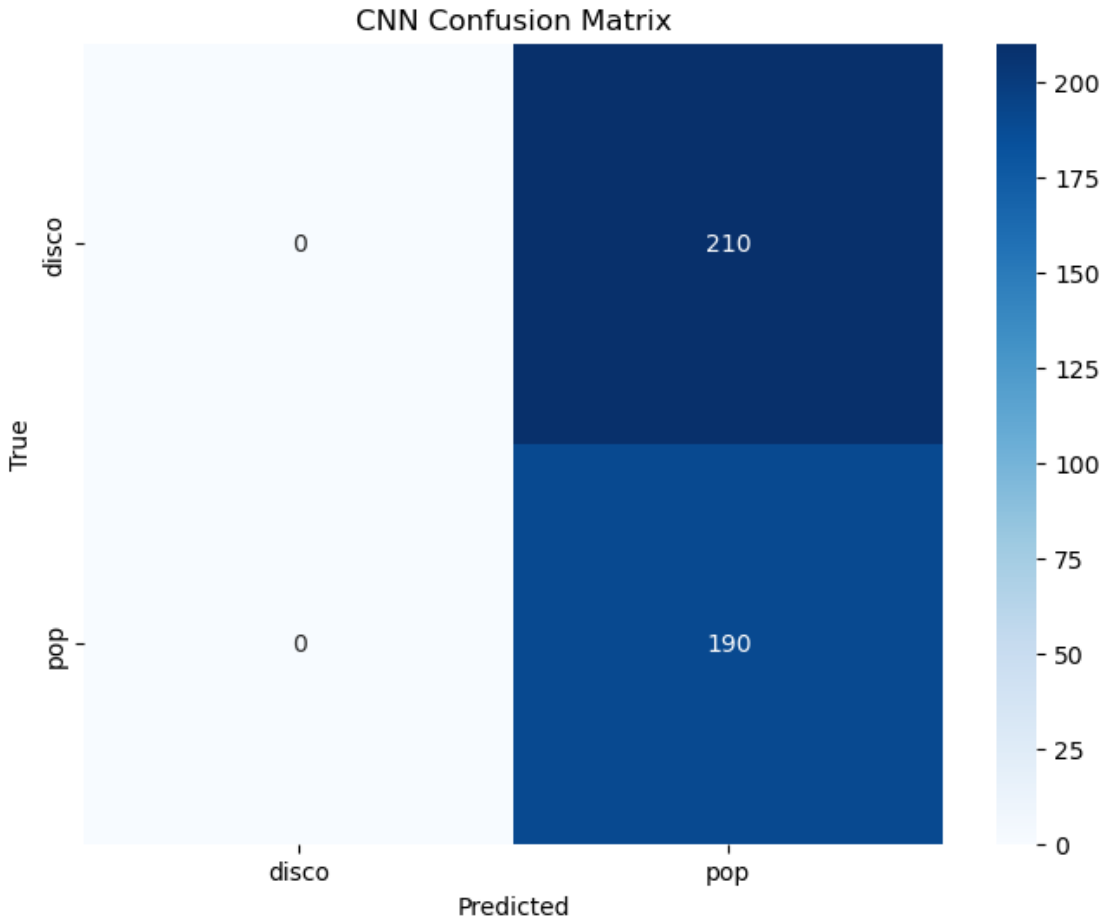
1.6 12 - Confusion Matrix Hard (disco and pop)

[6]:
```
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                2s 120ms/step
```



1.7 13 - Limited Genres Medium (5 random)

[7]:
```
import os
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from tensorflow.keras import models
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
 ↪Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau
import matplotlib.pyplot as plt
import random

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

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 (3 secs)', 'chromagram_12')

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

# Organize data by song ID
song_to_clips = {}

for genre in GENRES:
    genre_dir = os.path.join(FILE_PATH, genre)
    print(f"Processing genre: {genre}")
    for file in os.listdir(genre_dir):
        if not file.endswith(".png"):
            continue

        song_id = file.split("_clip_")[0]  # Extract song ID (e.g., "blues.
 ↪00042")

        if song_id not in song_to_clips:
            song_to_clips[song_id] = []
```

```
        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])  # Resize to 256x256
        image = augment_image(image)  # Apply augmentation
        image = image.numpy()  # Convert to numpy array

        song_to_clips[song_id].append((image, GENRE_TO_INDEX[genre]))

# Convert dictionary to list format
song_ids = list(song_to_clips.keys())
train_ids, test_ids = train_test_split(song_ids, test_size=0.2, random_state=42)

X_train, y_train, X_test, y_test = [], [], [], []

# Assign clips based on the train-test split
for song_id in song_ids:
    clips = song_to_clips[song_id]
    if song_id in train_ids:
        for image, label in clips:
            X_train.append(image)
            y_train.append(label)
    else:
        for image, label in clips:
            X_test.append(image)
            y_test.append(label)

# Convert to numpy arrays
X_train = np.array(X_train)
y_train = np.array(y_train)
X_test = np.array(X_test)
y_test = np.array(y_test)

print(f"Train set: {len(X_train)} samples")
print(f"Test set: {len(X_test)} samples")

# Define the CNN model
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')
])

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

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

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

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

```
['rock', 'blues', 'disco', 'hiphop', 'classical']
Processing genre: rock
Processing genre: blues
Processing genre: disco
Processing genre: hiphop
Processing genre: classical
Train set: 4000 samples
Test set: 1000 samples
```

```
/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             87s 651ms/step -
accuracy: 0.1751 - loss: 2.1139 - val_accuracy: 0.1000 - val_loss: 1.7369 -
learning_rate: 1.0000e-04
Epoch 2/20
125/125             84s 671ms/step -
accuracy: 0.2072 - loss: 1.7757 - val_accuracy: 0.1000 - val_loss: 1.7374 -
learning_rate: 1.0000e-04
Epoch 3/20
```

```
125/125             85s 677ms/step -
accuracy: 0.2033 - loss: 1.7353 - val_accuracy: 0.1000 - val_loss: 1.6729 -
learning_rate: 1.0000e-04
Epoch 4/20
125/125             78s 619ms/step -
accuracy: 0.2276 - loss: 1.6752 - val_accuracy: 0.2770 - val_loss: 1.5640 -
learning_rate: 1.0000e-04
Epoch 5/20
125/125             89s 672ms/step -
accuracy: 0.2644 - loss: 1.5998 - val_accuracy: 0.3620 - val_loss: 1.5286 -
learning_rate: 1.0000e-04
Epoch 6/20
125/125             140s 659ms/step -
accuracy: 0.3090 - loss: 1.5271 - val_accuracy: 0.4560 - val_loss: 1.3848 -
learning_rate: 1.0000e-04
Epoch 7/20
125/125             137s 619ms/step -
accuracy: 0.3600 - loss: 1.4508 - val_accuracy: 0.4400 - val_loss: 1.2978 -
learning_rate: 1.0000e-04
Epoch 8/20
125/125             80s 604ms/step -
accuracy: 0.3948 - loss: 1.3851 - val_accuracy: 0.5140 - val_loss: 1.2276 -
learning_rate: 1.0000e-04
Epoch 9/20
125/125             85s 679ms/step -
accuracy: 0.4016 - loss: 1.3338 - val_accuracy: 0.4600 - val_loss: 1.1799 -
learning_rate: 1.0000e-04
Epoch 10/20
125/125             84s 674ms/step -
accuracy: 0.4522 - loss: 1.2747 - val_accuracy: 0.5220 - val_loss: 1.1519 -
learning_rate: 1.0000e-04
Epoch 11/20
125/125             80s 637ms/step -
accuracy: 0.4807 - loss: 1.2151 - val_accuracy: 0.5250 - val_loss: 1.1623 -
learning_rate: 1.0000e-04
Epoch 12/20
125/125             85s 665ms/step -
accuracy: 0.4783 - loss: 1.1982 - val_accuracy: 0.5100 - val_loss: 1.1774 -
learning_rate: 1.0000e-04
Epoch 13/20
125/125             141s 658ms/step -
accuracy: 0.4969 - loss: 1.1696 - val_accuracy: 0.5270 - val_loss: 1.1389 -
learning_rate: 1.0000e-04
Epoch 14/20
125/125             77s 617ms/step -
accuracy: 0.5041 - loss: 1.1549 - val_accuracy: 0.5320 - val_loss: 1.1623 -
learning_rate: 1.0000e-04
Epoch 15/20
```

```
125/125             83s 666ms/step -
accuracy: 0.5236 - loss: 1.1238 - val_accuracy: 0.5430 - val_loss: 1.1160 -
learning_rate: 1.0000e-04
Epoch 16/20
125/125             83s 662ms/step -
accuracy: 0.5353 - loss: 1.1079 - val_accuracy: 0.5640 - val_loss: 1.1371 -
learning_rate: 1.0000e-04
Epoch 17/20
125/125             142s 664ms/step -
accuracy: 0.5349 - loss: 1.1179 - val_accuracy: 0.5810 - val_loss: 1.1351 -
learning_rate: 1.0000e-04
Epoch 18/20
125/125             81s 648ms/step -
accuracy: 0.5476 - loss: 1.0621 - val_accuracy: 0.5710 - val_loss: 1.0927 -
learning_rate: 1.0000e-04
Epoch 19/20
125/125             83s 660ms/step -
accuracy: 0.5568 - loss: 1.0599 - val_accuracy: 0.5330 - val_loss: 1.2034 -
learning_rate: 1.0000e-04
Epoch 20/20
125/125             142s 661ms/step -
accuracy: 0.5663 - loss: 1.0521 - val_accuracy: 0.5600 - val_loss: 1.0927 -
learning_rate: 1.0000e-04
32/32              3s 89ms/step -
accuracy: 0.4182 - loss: 1.1740
Test accuracy: 0.560
```

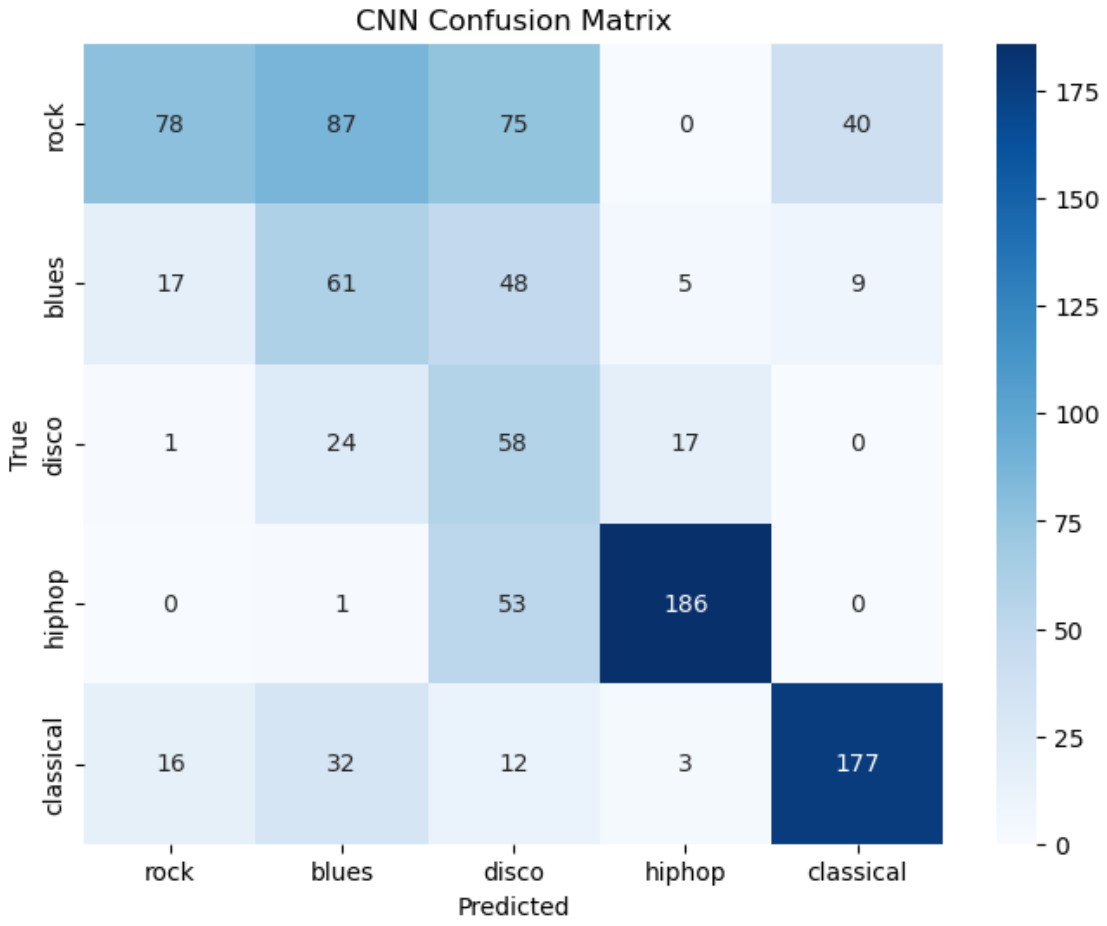
1.8 14 - Confusion Matrix Medium (5 random)

[8]:
```
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              3s 101ms/step
```


CNN Confusion Matrix
True
Predicted
rock
blues
disco
hiphop
classical
78 87 75 0 40
17 61 48 5 9
1 24 58 17 0
0 1 53 186 0
16 32 12 3 177
175
150
125
100
75
50
25
0