Chromagram-Only CNN

March 23, 2025

1 CNN for Chromagram (30 secs)

1.1 1 - All the imports

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

```
2025-03-23 00:53:49.625232: I tensorflow/core/platform/cpu_feature_guard.cc:210]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild
TensorFlow with the appropriate compiler flags.
```

1.2 2 - Put the data within the model

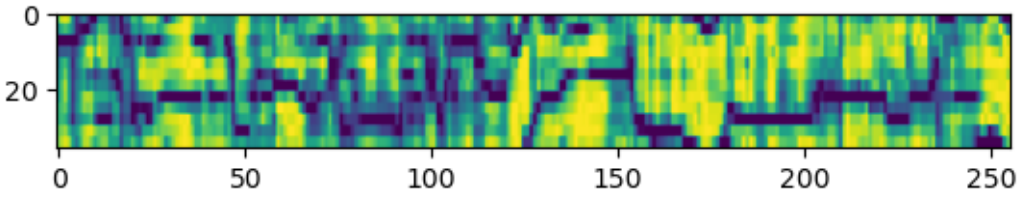
[2]:
```
# Import a single image and save it to be read by the model

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

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

# Convert to a numpy array
image = tf.image.decode_png(image, channels=1)
image = tf.image.convert_image_dtype(image, tf.float32)
image = tf.image.resize(image, [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]:
```
# 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_24')
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):
```

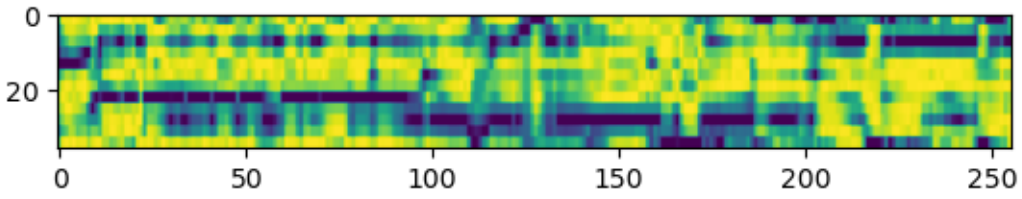
```
        try:
            image = tf.io.read_file(os.path.join(genre_dir, file))
            image = tf.image.decode_png(image, channels=1)
            image = tf.image.convert_image_dtype(image, tf.float32)
            image = tf.image.resize(image, [36, 256])
            image = image.numpy()
            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
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             8s 243ms/step -
accuracy: 0.1014 - loss: 5.1020 - val_accuracy: 0.0850 - val_loss: 2.3226
Epoch 2/20
25/25             6s 232ms/step -
accuracy: 0.1047 - loss: 2.3707 - val_accuracy: 0.0800 - val_loss: 2.3117
Epoch 3/20
25/25             6s 226ms/step -
accuracy: 0.1017 - loss: 2.3154 - val_accuracy: 0.0650 - val_loss: 2.3074
Epoch 4/20
25/25             6s 224ms/step -
accuracy: 0.0764 - loss: 2.3064 - val_accuracy: 0.1500 - val_loss: 2.3038
Epoch 5/20
25/25             6s 221ms/step -
accuracy: 0.1167 - loss: 2.3003 - val_accuracy: 0.1450 - val_loss: 2.3025
Epoch 6/20
25/25             6s 221ms/step -
accuracy: 0.1202 - loss: 2.2898 - val_accuracy: 0.0750 - val_loss: 2.2954
Epoch 7/20
25/25             6s 221ms/step -
accuracy: 0.1500 - loss: 2.2726 - val_accuracy: 0.1250 - val_loss: 2.2791
Epoch 8/20
25/25             6s 227ms/step -
accuracy: 0.2137 - loss: 2.2218 - val_accuracy: 0.1200 - val_loss: 2.2534
Epoch 9/20
25/25             6s 220ms/step -
accuracy: 0.2190 - loss: 2.1608 - val_accuracy: 0.2100 - val_loss: 2.1533
Epoch 10/20
25/25             6s 222ms/step -
accuracy: 0.2753 - loss: 2.0052 - val_accuracy: 0.2250 - val_loss: 2.0479
Epoch 11/20
25/25             6s 221ms/step -
accuracy: 0.3047 - loss: 1.8244 - val_accuracy: 0.2200 - val_loss: 2.0274
Epoch 12/20
25/25             6s 223ms/step -
accuracy: 0.3744 - loss: 1.7106 - val_accuracy: 0.2500 - val_loss: 2.0426
Epoch 13/20
25/25             6s 225ms/step -
accuracy: 0.4189 - loss: 1.5493 - val_accuracy: 0.2200 - val_loss: 2.1877
Epoch 14/20
25/25             6s 222ms/step -
accuracy: 0.4571 - loss: 1.4623 - val_accuracy: 0.2300 - val_loss: 2.1527
Epoch 15/20
25/25             6s 218ms/step -
accuracy: 0.5285 - loss: 1.2679 - val_accuracy: 0.2500 - val_loss: 2.1360
Epoch 16/20
25/25             6s 220ms/step -
accuracy: 0.6056 - loss: 1.1083 - val_accuracy: 0.2100 - val_loss: 2.2748
```

```
Epoch 17/20
25/25             6s 220ms/step -
accuracy: 0.6328 - loss: 1.0569 - val_accuracy: 0.2450 - val_loss: 2.2527
Epoch 18/20
25/25             6s 222ms/step -
accuracy: 0.6844 - loss: 0.9294 - val_accuracy: 0.2150 - val_loss: 2.3980
Epoch 19/20
25/25             5s 218ms/step -
accuracy: 0.6957 - loss: 0.8547 - val_accuracy: 0.2700 - val_loss: 2.4271
Epoch 20/20
25/25             6s 227ms/step -
accuracy: 0.7680 - loss: 0.6657 - val_accuracy: 0.2500 - val_loss: 2.4708
```

[8]:
```
<keras.src.callbacks.history.History at 0x7f453c1cbce0>
```

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

```
7/7             0s 21ms/step -
accuracy: 0.2742 - loss: 2.3897
Test accuracy: 0.250
```

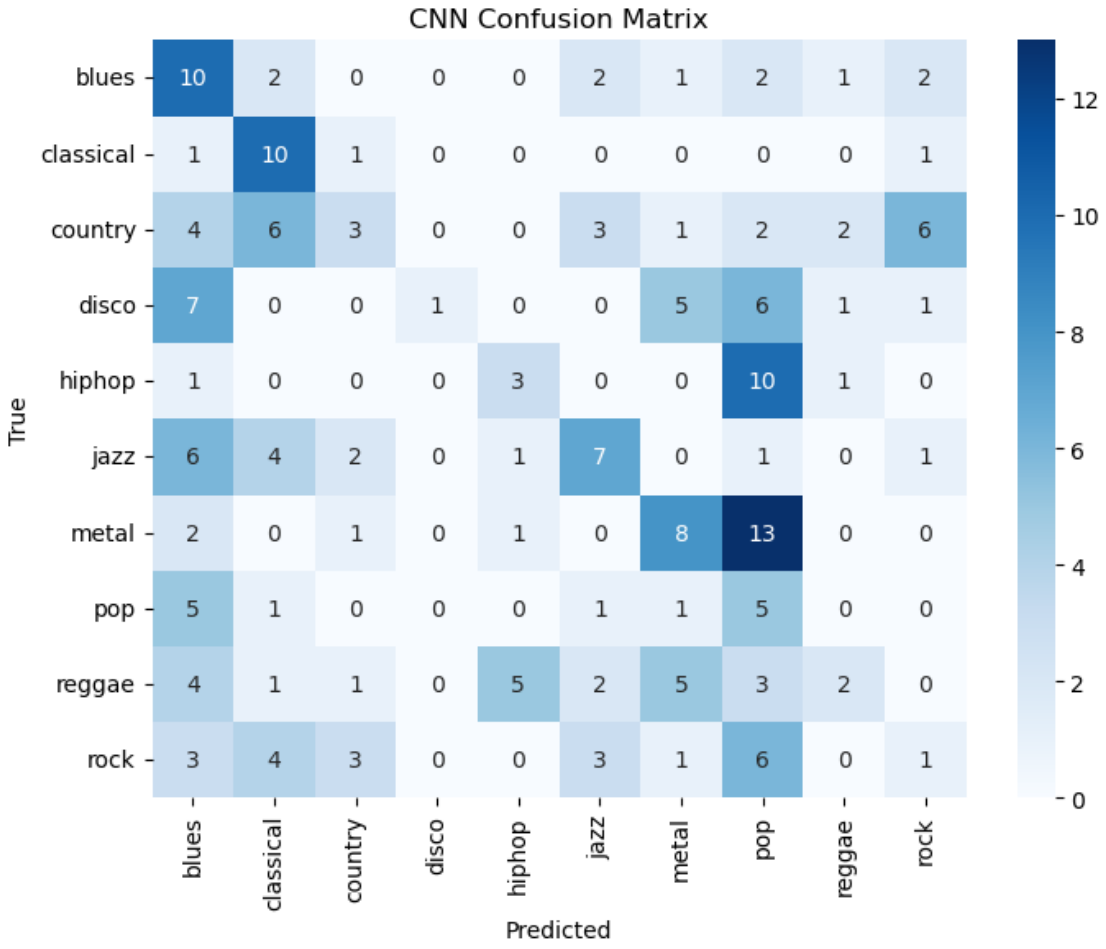
4 Apply the confusion matrix after the model

[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             0s 25ms/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_24')
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 classical
Going through metal
Epoch 1/20
/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)
5/5             4s 287ms/step -
accuracy: 0.3340 - loss: 1.7033 - val_accuracy: 0.5250 - val_loss: 0.7077 -
learning_rate: 1.0000e-04
Epoch 2/20
5/5             1s 236ms/step -
accuracy: 0.5170 - loss: 1.0387 - val_accuracy: 0.4750 - val_loss: 0.8331 -
learning_rate: 1.0000e-04
Epoch 3/20
5/5             1s 226ms/step -
accuracy: 0.4319 - loss: 1.3540 - val_accuracy: 0.5250 - val_loss: 0.6840 -
learning_rate: 1.0000e-04
Epoch 4/20
5/5             1s 229ms/step -
accuracy: 0.5292 - loss: 1.1100 - val_accuracy: 0.5250 - val_loss: 0.6997 -
learning_rate: 1.0000e-04
Epoch 5/20
5/5             1s 234ms/step -
accuracy: 0.5501 - loss: 0.9799 - val_accuracy: 0.4750 - val_loss: 0.6944 -
learning_rate: 1.0000e-04
Epoch 6/20
```

```
5/5            1s 232ms/step -
accuracy: 0.4783 - loss: 1.1280 - val_accuracy: 0.4750 - val_loss: 0.7100 -
learning_rate: 1.0000e-04
Epoch 7/20
5/5            1s 228ms/step -
accuracy: 0.5218 - loss: 1.0522 - val_accuracy: 0.4750 - val_loss: 0.7016 -
learning_rate: 5.0000e-05
Epoch 8/20
5/5            1s 233ms/step -
accuracy: 0.5344 - loss: 0.9565 - val_accuracy: 0.5250 - val_loss: 0.6973 -
learning_rate: 5.0000e-05
Epoch 9/20
5/5            1s 236ms/step -
accuracy: 0.5197 - loss: 0.8900 - val_accuracy: 0.5250 - val_loss: 0.6952 -
learning_rate: 5.0000e-05
Epoch 10/20
5/5            1s 238ms/step -
accuracy: 0.4516 - loss: 0.9227 - val_accuracy: 0.6250 - val_loss: 0.6944 -
learning_rate: 2.5000e-05
Epoch 11/20
5/5            1s 235ms/step -
accuracy: 0.5172 - loss: 0.9622 - val_accuracy: 0.8750 - val_loss: 0.6964 -
learning_rate: 2.5000e-05
Epoch 12/20
5/5            1s 235ms/step -
accuracy: 0.4589 - loss: 0.9575 - val_accuracy: 0.4750 - val_loss: 0.7014 -
learning_rate: 2.5000e-05
Epoch 13/20
5/5            1s 225ms/step -
accuracy: 0.5248 - loss: 0.9377 - val_accuracy: 0.4750 - val_loss: 0.7051 -
learning_rate: 1.2500e-05
Epoch 14/20
5/5            1s 234ms/step -
accuracy: 0.4797 - loss: 0.9780 - val_accuracy: 0.4750 - val_loss: 0.7067 -
learning_rate: 1.2500e-05
Epoch 15/20
5/5            1s 241ms/step -
accuracy: 0.4681 - loss: 0.9432 - val_accuracy: 0.4750 - val_loss: 0.7091 -
learning_rate: 1.2500e-05
Epoch 16/20
5/5            1s 230ms/step -
accuracy: 0.5631 - loss: 0.7753 - val_accuracy: 0.4750 - val_loss: 0.7102 -
learning_rate: 6.2500e-06
Epoch 17/20
5/5            1s 236ms/step -
accuracy: 0.4733 - loss: 0.9172 - val_accuracy: 0.4750 - val_loss: 0.7105 -
learning_rate: 6.2500e-06
Epoch 18/20
```

```
5/5             1s 241ms/step -
accuracy: 0.4742 - loss: 0.9381 - val_accuracy: 0.4750 - val_loss: 0.7091 -
learning_rate: 6.2500e-06
Epoch 19/20
5/5             1s 237ms/step -
accuracy: 0.4819 - loss: 0.8910 - val_accuracy: 0.4750 - val_loss: 0.7084 -
learning_rate: 3.1250e-06
Epoch 20/20
5/5             1s 241ms/step -
accuracy: 0.4944 - loss: 0.9199 - val_accuracy: 0.4750 - val_loss: 0.7079 -
learning_rate: 3.1250e-06
2/2             0s 12ms/step -
accuracy: 0.4625 - loss: 0.7104
Test accuracy: 0.475
```

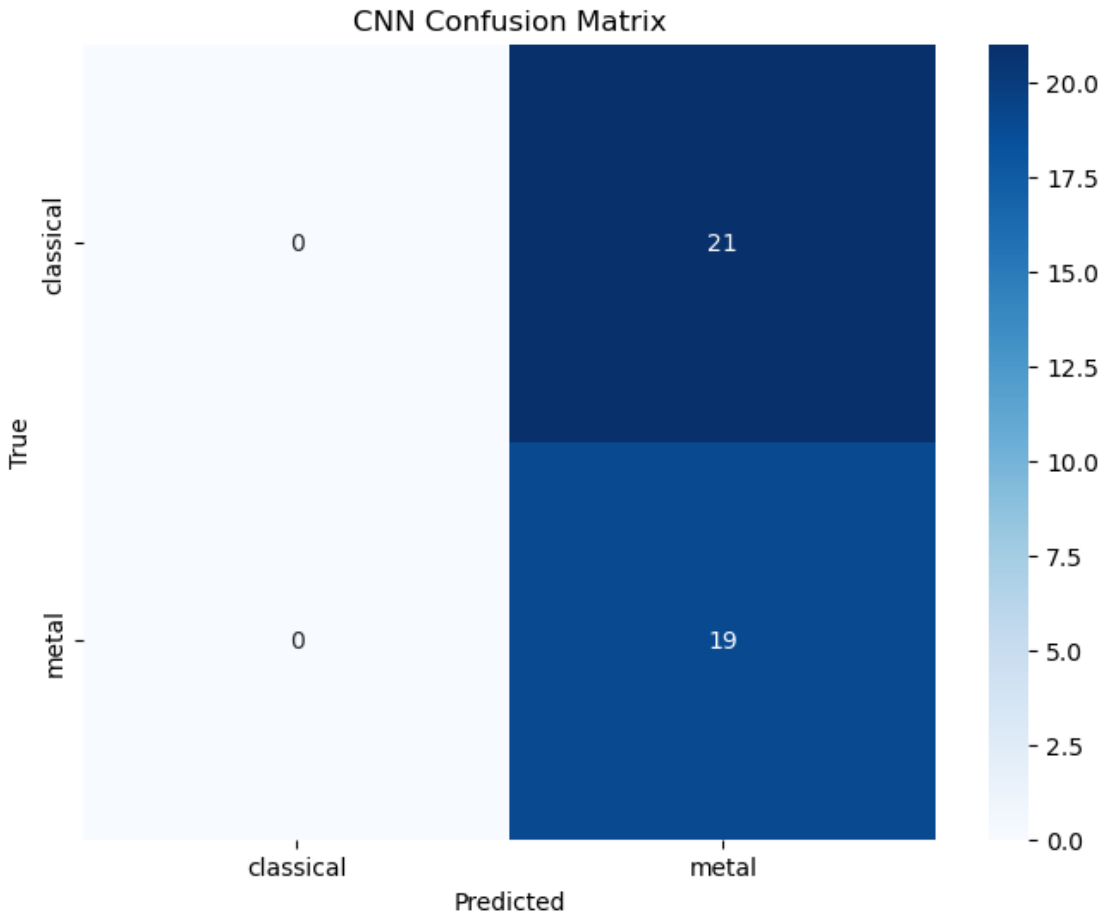
4.2 10 - Confusion Matrix Easy (classical and metal)

[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             0s 74ms/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_24')
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
Epoch 1/20
```

```
/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)
```

```
5/5             3s 277ms/step -
accuracy: 0.1928 - loss: 1.9740 - val_accuracy: 0.4750 - val_loss: 0.9608 -
learning_rate: 1.0000e-04
Epoch 2/20
5/5             1s 225ms/step -
accuracy: 0.4950 - loss: 1.2212 - val_accuracy: 0.4750 - val_loss: 0.8810 -
learning_rate: 1.0000e-04
Epoch 3/20
5/5             1s 240ms/step -
accuracy: 0.5171 - loss: 1.3551 - val_accuracy: 0.4750 - val_loss: 0.8399 -
learning_rate: 1.0000e-04
Epoch 4/20
5/5             1s 228ms/step -
accuracy: 0.4961 - loss: 1.1132 - val_accuracy: 0.4750 - val_loss: 0.8405 -
learning_rate: 1.0000e-04
Epoch 5/20
5/5             1s 237ms/step -
accuracy: 0.4424 - loss: 1.0624 - val_accuracy: 0.4750 - val_loss: 0.8511 -
learning_rate: 1.0000e-04
Epoch 6/20
```

```
5/5             1s 232ms/step -
accuracy: 0.4462 - loss: 1.2143 - val_accuracy: 0.4750 - val_loss: 0.7762 -
learning_rate: 1.0000e-04
Epoch 7/20
5/5             1s 229ms/step -
accuracy: 0.5994 - loss: 0.8669 - val_accuracy: 0.4750 - val_loss: 0.7699 -
learning_rate: 1.0000e-04
Epoch 8/20
5/5             1s 226ms/step -
accuracy: 0.5282 - loss: 1.0770 - val_accuracy: 0.4750 - val_loss: 0.7946 -
learning_rate: 1.0000e-04
Epoch 9/20
5/5             1s 223ms/step -
accuracy: 0.4477 - loss: 0.9819 - val_accuracy: 0.4750 - val_loss: 0.8142 -
learning_rate: 1.0000e-04
Epoch 10/20
5/5             1s 228ms/step -
accuracy: 0.5492 - loss: 0.8298 - val_accuracy: 0.4750 - val_loss: 0.7620 -
learning_rate: 1.0000e-04
Epoch 11/20
5/5             1s 225ms/step -
accuracy: 0.5530 - loss: 0.9722 - val_accuracy: 0.4750 - val_loss: 0.7400 -
learning_rate: 1.0000e-04
Epoch 12/20
5/5             1s 220ms/step -
accuracy: 0.5245 - loss: 0.9239 - val_accuracy: 0.4750 - val_loss: 0.7601 -
learning_rate: 1.0000e-04
Epoch 13/20
5/5             1s 220ms/step -
accuracy: 0.5392 - loss: 0.9345 - val_accuracy: 0.4750 - val_loss: 0.8045 -
learning_rate: 1.0000e-04
Epoch 14/20
5/5             1s 234ms/step -
accuracy: 0.4549 - loss: 1.0629 - val_accuracy: 0.4750 - val_loss: 0.8449 -
learning_rate: 1.0000e-04
Epoch 15/20
5/5             1s 227ms/step -
accuracy: 0.4546 - loss: 1.0473 - val_accuracy: 0.4750 - val_loss: 0.8633 -
learning_rate: 5.0000e-05
Epoch 16/20
5/5             1s 223ms/step -
accuracy: 0.4591 - loss: 0.9554 - val_accuracy: 0.4750 - val_loss: 0.8676 -
learning_rate: 5.0000e-05
Epoch 17/20
5/5             1s 223ms/step -
accuracy: 0.5253 - loss: 0.8380 - val_accuracy: 0.4750 - val_loss: 0.8564 -
learning_rate: 5.0000e-05
Epoch 18/20
```

```
5/5             1s 226ms/step -
accuracy: 0.5237 - loss: 0.9044 - val_accuracy: 0.4750 - val_loss: 0.8457 -
learning_rate: 2.5000e-05
Epoch 19/20
5/5             1s 233ms/step -
accuracy: 0.5201 - loss: 0.9314 - val_accuracy: 0.4750 - val_loss: 0.8393 -
learning_rate: 2.5000e-05
Epoch 20/20
5/5             1s 228ms/step -
accuracy: 0.5831 - loss: 0.8485 - val_accuracy: 0.4750 - val_loss: 0.8308 -
learning_rate: 2.5000e-05
2/2             0s 10ms/step -
accuracy: 0.4625 - loss: 0.8417
Test accuracy: 0.475
```

4.4 12 - Confusion Matrix Hard (disco and pop)

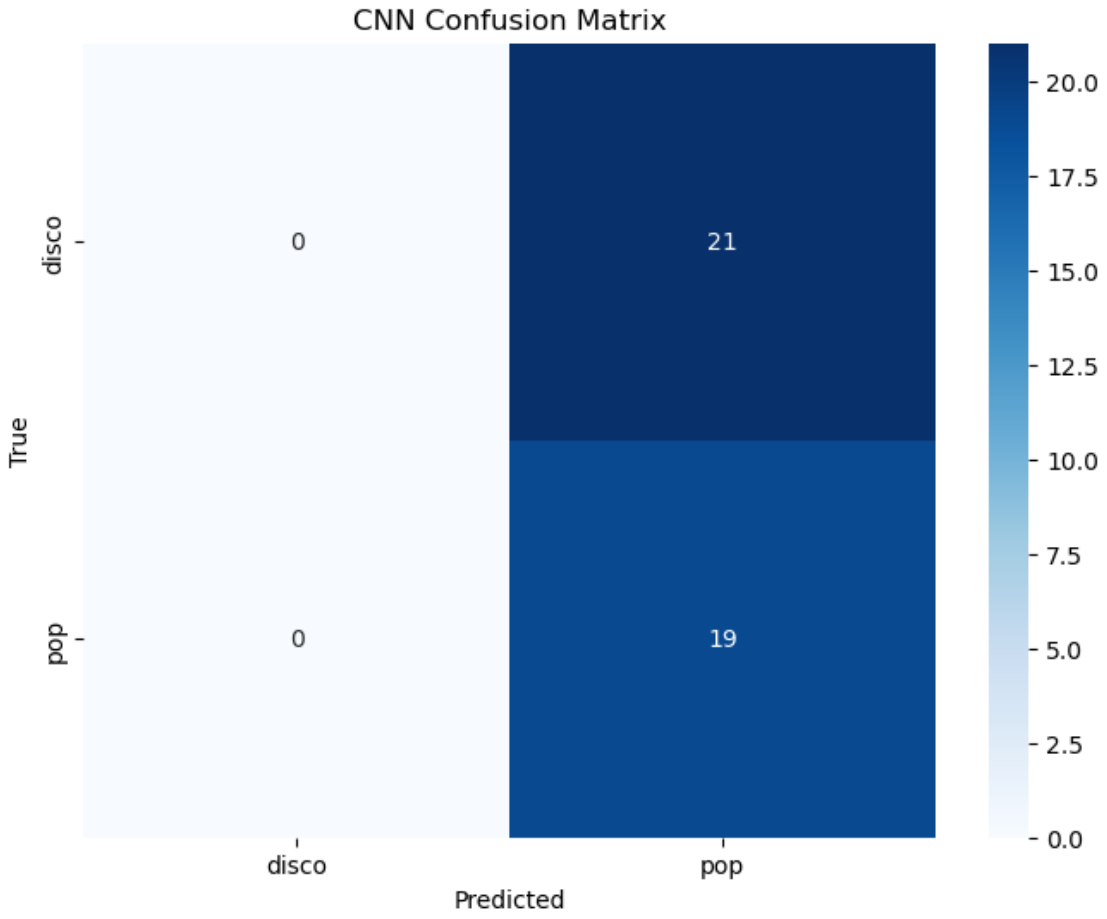
[14]:
```python
import seaborn as sns
# from sklearn.metrics import confusion
import numpy as NP
from sklearn.metrics import confusion_matrix

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

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

```
WARNING:tensorflow:5 out of the last 10 calls to <function
TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at
0x7f4510371080> 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
141ms/stepWARNING:tensorflow:6 out of the last 11 calls to <function
TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at
0x7f4510371080> 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             0s 159ms/step
```



4.5 13 - Limited Genres Medium (5 random)

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

```
GENRES = ['blues', 'classical', 'country', 'disco', 'hiphop', 'jazz', 'metal',␣
 ↪'pop', 'reggae', 'rock']
GENRES = random.sample(GENRES, 5)
print(GENRES)
FILE_PATH = os.path.join('Data', 'chromagrams', 'chromagram_24')
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}")
```

```
['rock', 'pop', 'country', 'reggae', 'classical']
Going through rock
Going through pop
Going through country
Going through reggae
Going through classical
```

```
/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             8s 381ms/step -
accuracy: 0.2045 - loss: 2.1218 - val_accuracy: 0.2400 - val_loss: 1.6974 -
```

```
learning_rate: 1.0000e-04
Epoch 2/20
13/13            4s 324ms/step -
accuracy: 0.1720 - loss: 2.1241 - val_accuracy: 0.2800 - val_loss: 1.6907 -
learning_rate: 1.0000e-04
Epoch 3/20
13/13            4s 339ms/step -
accuracy: 0.1987 - loss: 1.9797 - val_accuracy: 0.1000 - val_loss: 1.7258 -
learning_rate: 1.0000e-04
Epoch 4/20
13/13            4s 335ms/step -
accuracy: 0.1864 - loss: 1.9222 - val_accuracy: 0.1000 - val_loss: 1.7630 -
learning_rate: 1.0000e-04
Epoch 5/20
13/13            4s 332ms/step -
accuracy: 0.2375 - loss: 1.8345 - val_accuracy: 0.2000 - val_loss: 1.6912 -
learning_rate: 1.0000e-04
Epoch 6/20
13/13            4s 340ms/step -
accuracy: 0.2017 - loss: 1.8527 - val_accuracy: 0.1100 - val_loss: 1.7221 -
learning_rate: 5.0000e-05
Epoch 7/20
13/13            5s 344ms/step -
accuracy: 0.1798 - loss: 1.7907 - val_accuracy: 0.1000 - val_loss: 1.7232 -
learning_rate: 5.0000e-05
Epoch 8/20
13/13            5s 348ms/step -
accuracy: 0.1763 - loss: 1.7685 - val_accuracy: 0.1000 - val_loss: 1.7213 -
learning_rate: 5.0000e-05
Epoch 9/20
13/13            5s 348ms/step -
accuracy: 0.2262 - loss: 1.7923 - val_accuracy: 0.1000 - val_loss: 1.7238 -
learning_rate: 2.5000e-05
Epoch 10/20
13/13            4s 339ms/step -
accuracy: 0.1963 - loss: 1.7610 - val_accuracy: 0.1000 - val_loss: 1.7211 -
learning_rate: 2.5000e-05
Epoch 11/20
13/13            5s 346ms/step -
accuracy: 0.2108 - loss: 1.7973 - val_accuracy: 0.1000 - val_loss: 1.7147 -
learning_rate: 2.5000e-05
Epoch 12/20
13/13            5s 361ms/step -
accuracy: 0.2150 - loss: 1.7779 - val_accuracy: 0.1000 - val_loss: 1.7095 -
learning_rate: 1.2500e-05
Epoch 13/20
13/13            7s 460ms/step -
accuracy: 0.2445 - loss: 1.7248 - val_accuracy: 0.1000 - val_loss: 1.7056 -
```

```
learning_rate: 1.2500e-05
Epoch 14/20
13/13            5s 395ms/step -
accuracy: 0.2327 - loss: 1.7318 - val_accuracy: 0.1000 - val_loss: 1.7032 -
learning_rate: 1.2500e-05
Epoch 15/20
13/13            5s 375ms/step -
accuracy: 0.2062 - loss: 1.8017 - val_accuracy: 0.1000 - val_loss: 1.7064 -
learning_rate: 6.2500e-06
Epoch 16/20
13/13            5s 390ms/step -
accuracy: 0.2543 - loss: 1.7272 - val_accuracy: 0.1000 - val_loss: 1.7041 -
learning_rate: 6.2500e-06
Epoch 17/20
13/13            5s 380ms/step -
accuracy: 0.2485 - loss: 1.7268 - val_accuracy: 0.1000 - val_loss: 1.7037 -
learning_rate: 6.2500e-06
Epoch 18/20
13/13            6s 473ms/step -
accuracy: 0.2050 - loss: 1.7424 - val_accuracy: 0.1000 - val_loss: 1.7025 -
learning_rate: 3.1250e-06
Epoch 19/20
13/13            6s 470ms/step -
accuracy: 0.2378 - loss: 1.7495 - val_accuracy: 0.1000 - val_loss: 1.7012 -
learning_rate: 3.1250e-06
Epoch 20/20
13/13            6s 453ms/step -
accuracy: 0.2476 - loss: 1.6992 - val_accuracy: 0.1000 - val_loss: 1.7027 -
learning_rate: 3.1250e-06
4/4            0s 37ms/step -
accuracy: 0.0890 - loss: 1.7119
Test accuracy: 0.100
```

4.6 14 - Confusion Matrix Medium (5 random)

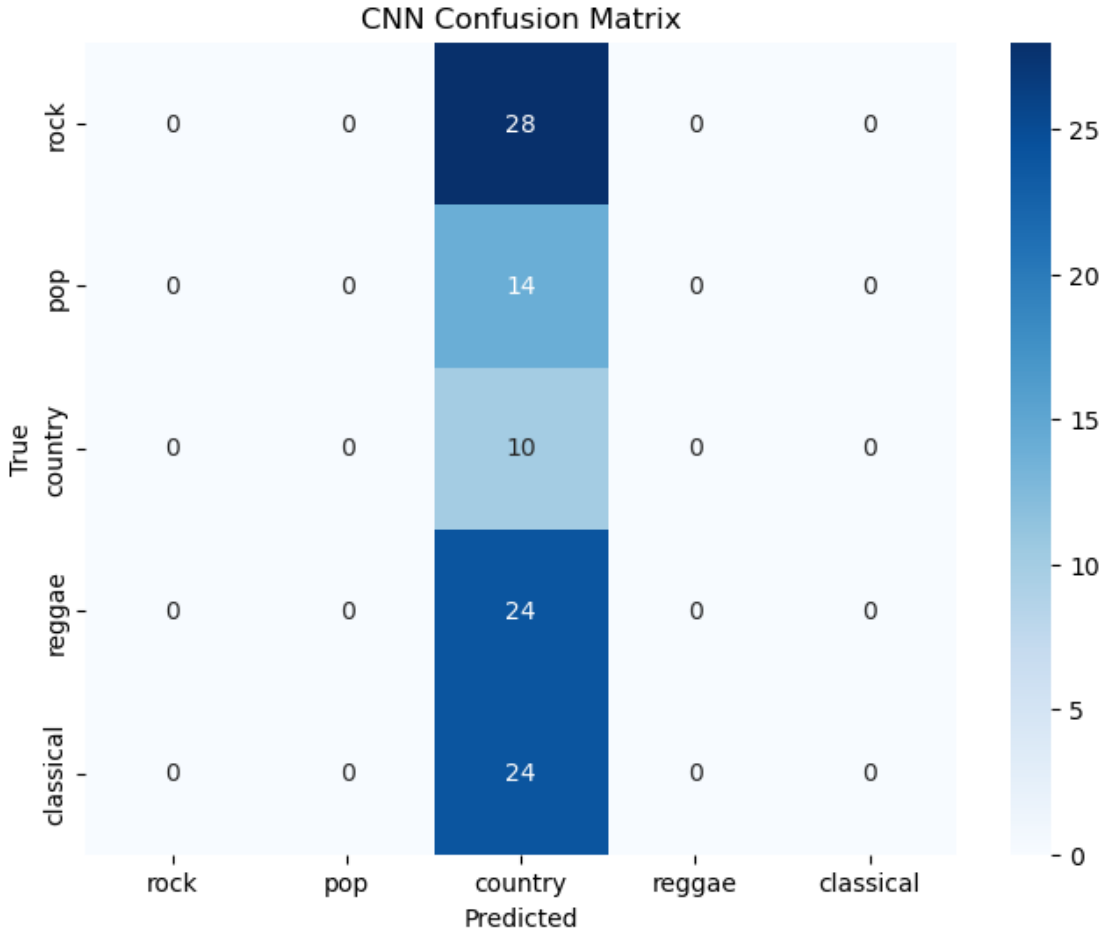
[16]:
```python
import seaborn as sns
# from sklearn.metrics import confusion
import numpy as NP
from sklearn.metrics import confusion_matrix

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                 0s 99ms/step
```



[ ]: