# Tempogram-Only (3 secs) CNN

March 22, 2025

## 1 CNN for Tempogram (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, Normalization
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', 'tempograms (3 secs)')

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, [256, 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(32, (3, 3), activation='relu', input_shape=(256, 256, 1)),
```

```
    Normalization(),
    MaxPooling2D((2, 2)),

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

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

    Conv2D(256, (3, 3), activation='relu'),
    Normalization(),
    MaxPooling2D((2, 2)),

    Flatten(),

    Dense(512, activation='relu'),
    Dropout(0.5),

    Dense(256, activation='relu'),
    Dropout(0.5),

    Dense(128, activation='relu'),
    Dense(len(GENRES), activation='softmax')  # Output size matches number of␣
 ↪genres
])

# 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 01:34:41.395383: 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             687s 3s/step -
accuracy: 0.1026 - loss: 2.3002 - val_accuracy: 0.1505 - val_loss: 2.2288 -
learning_rate: 1.0000e-04
Epoch 2/20
250/250             899s 3s/step -
accuracy: 0.1690 - loss: 2.1934 - val_accuracy: 0.1800 - val_loss: 2.1742 -
learning_rate: 1.0000e-04
Epoch 3/20
250/250             910s 4s/step -
accuracy: 0.1840 - loss: 2.1405 - val_accuracy: 0.1900 - val_loss: 2.1881 -
learning_rate: 1.0000e-04
Epoch 4/20
250/250             832s 3s/step -
accuracy: 0.1941 - loss: 2.1355 - val_accuracy: 0.2055 - val_loss: 2.1446 -
learning_rate: 1.0000e-04
Epoch 5/20
250/250             836s 3s/step -
accuracy: 0.2114 - loss: 2.1179 - val_accuracy: 0.2025 - val_loss: 2.1344 -
learning_rate: 1.0000e-04
Epoch 6/20
250/250             833s 3s/step -
accuracy: 0.2195 - loss: 2.0986 - val_accuracy: 0.2125 - val_loss: 2.1128 -
learning_rate: 1.0000e-04
Epoch 7/20
250/250             834s 3s/step -
```

```
accuracy: 0.2303 - loss: 2.0737 - val_accuracy: 0.2305 - val_loss: 2.0977 -
learning_rate: 1.0000e-04
Epoch 8/20
250/250            835s 3s/step -
accuracy: 0.2591 - loss: 2.0378 - val_accuracy: 0.2390 - val_loss: 2.0707 -
learning_rate: 1.0000e-04
Epoch 9/20
250/250            823s 3s/step -
accuracy: 0.2680 - loss: 2.0238 - val_accuracy: 0.2465 - val_loss: 2.0978 -
learning_rate: 1.0000e-04
Epoch 10/20
250/250            686s 3s/step -
accuracy: 0.2793 - loss: 1.9875 - val_accuracy: 0.2780 - val_loss: 2.0210 -
learning_rate: 1.0000e-04
Epoch 11/20
250/250            651s 3s/step -
accuracy: 0.2951 - loss: 1.9616 - val_accuracy: 0.2980 - val_loss: 1.9908 -
learning_rate: 1.0000e-04
Epoch 12/20
250/250            707s 3s/step -
accuracy: 0.3081 - loss: 1.9441 - val_accuracy: 0.3175 - val_loss: 1.9736 -
learning_rate: 1.0000e-04
Epoch 13/20
250/250            666s 3s/step -
accuracy: 0.3187 - loss: 1.9149 - val_accuracy: 0.3290 - val_loss: 1.9536 -
learning_rate: 1.0000e-04
Epoch 14/20
250/250            680s 3s/step -
accuracy: 0.3326 - loss: 1.8933 - val_accuracy: 0.3235 - val_loss: 1.9679 -
learning_rate: 1.0000e-04
Epoch 15/20
250/250            672s 3s/step -
accuracy: 0.3558 - loss: 1.8339 - val_accuracy: 0.3305 - val_loss: 1.9391 -
learning_rate: 1.0000e-04
Epoch 16/20
250/250            682s 3s/step -
accuracy: 0.3540 - loss: 1.8298 - val_accuracy: 0.3275 - val_loss: 1.9517 -
learning_rate: 1.0000e-04
Epoch 17/20
250/250            668s 3s/step -
accuracy: 0.3609 - loss: 1.8169 - val_accuracy: 0.3290 - val_loss: 1.9783 -
learning_rate: 1.0000e-04
Epoch 18/20
250/250            683s 3s/step -
accuracy: 0.3726 - loss: 1.7848 - val_accuracy: 0.3280 - val_loss: 1.9740 -
learning_rate: 1.0000e-04
Epoch 19/20
250/250            675s 3s/step -
```

```
accuracy: 0.3924 - loss: 1.7524 - val_accuracy: 0.3410 - val_loss: 1.9456 -
learning_rate: 5.0000e-05
Epoch 20/20
250/250             674s 3s/step -
accuracy: 0.3804 - loss: 1.7556 - val_accuracy: 0.3345 - val_loss: 1.9340 -
learning_rate: 5.0000e-05
63/63             39s 614ms/step -
accuracy: 0.3040 - loss: 1.9422
Test accuracy: 0.335
```

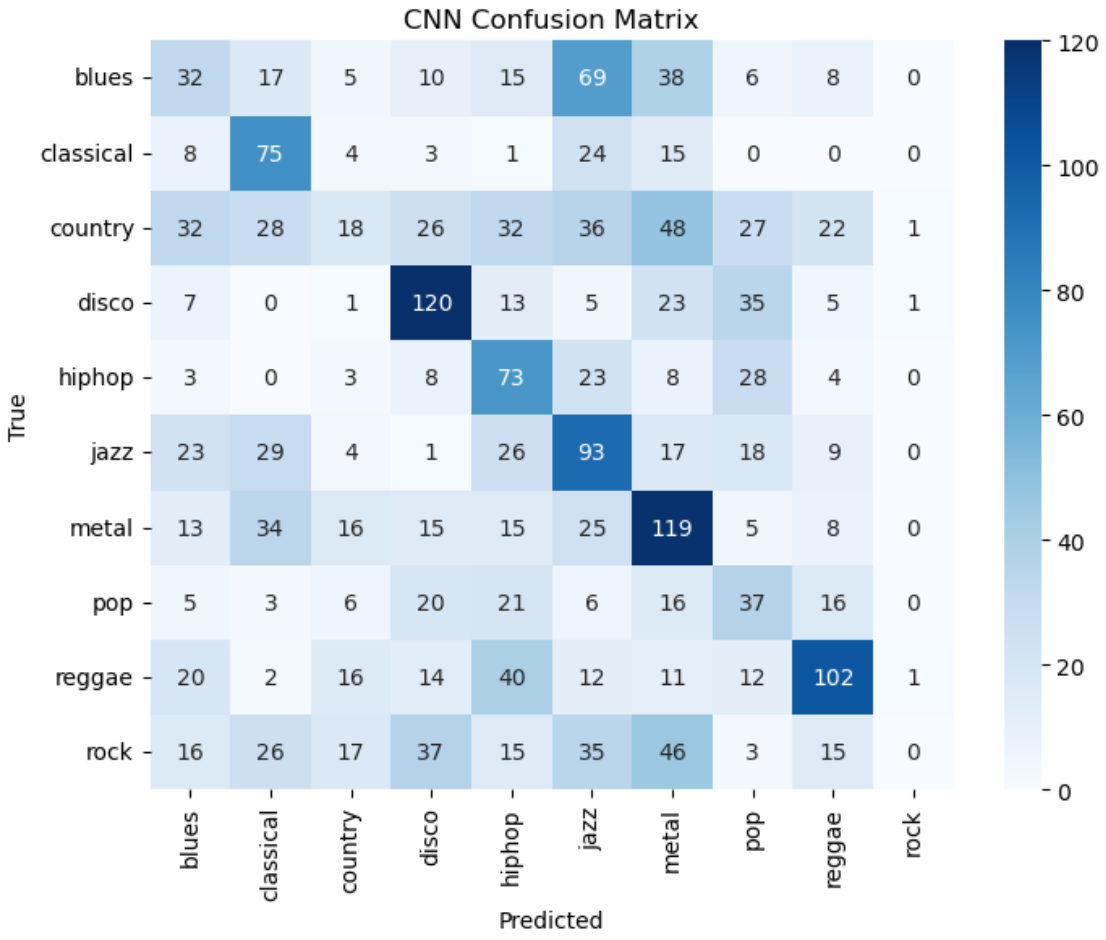
### 1.2 Apply the confusion matrix after the model

[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             33s 511ms/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, Normalization
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', 'tempograms (3 secs)')

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, [256, 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(32, (3, 3), activation='relu', input_shape=(256, 256, 1)),
    Normalization(),
    MaxPooling2D((2, 2)),

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

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

    Conv2D(256, (3, 3), activation='relu'),
    Normalization(),
    MaxPooling2D((2, 2)),

    Flatten(),

    Dense(512, activation='relu'),
    Dropout(0.5),

    Dense(256, activation='relu'),
    Dropout(0.5),

    Dense(128, activation='relu'),
    Dense(len(GENRES), activation='softmax')  # Output size matches number of␣
 ↪genres
])

# 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             147s 3s/step -
accuracy: 0.5146 - loss: 0.6957 - val_accuracy: 0.4750 - val_loss: 0.6932 -
learning_rate: 1.0000e-04
Epoch 2/20
50/50             141s 3s/step -
accuracy: 0.4980 - loss: 0.6933 - val_accuracy: 0.4750 - val_loss: 0.6935 -
learning_rate: 1.0000e-04
Epoch 3/20
50/50             111s 2s/step -
accuracy: 0.4931 - loss: 0.6931 - val_accuracy: 0.4750 - val_loss: 0.6925 -
learning_rate: 1.0000e-04
Epoch 4/20
50/50             120s 2s/step -
accuracy: 0.5517 - loss: 0.6904 - val_accuracy: 0.4750 - val_loss: 0.6943 -
learning_rate: 1.0000e-04
Epoch 5/20
50/50             102s 2s/step -
accuracy: 0.5476 - loss: 0.6893 - val_accuracy: 0.6075 - val_loss: 0.6674 -
learning_rate: 1.0000e-04
Epoch 6/20
```

```
50/50             126s 3s/step -
accuracy: 0.5728 - loss: 0.6691 - val_accuracy: 0.6425 - val_loss: 0.6414 -
learning_rate: 1.0000e-04
Epoch 7/20
50/50             129s 3s/step -
accuracy: 0.6637 - loss: 0.6198 - val_accuracy: 0.6675 - val_loss: 0.6363 -
learning_rate: 1.0000e-04
Epoch 8/20
50/50             130s 3s/step -
accuracy: 0.6786 - loss: 0.6166 - val_accuracy: 0.6525 - val_loss: 0.6131 -
learning_rate: 1.0000e-04
Epoch 9/20
50/50             135s 3s/step -
accuracy: 0.6938 - loss: 0.5894 - val_accuracy: 0.6525 - val_loss: 0.6012 -
learning_rate: 1.0000e-04
Epoch 10/20
50/50             136s 3s/step -
accuracy: 0.7086 - loss: 0.5586 - val_accuracy: 0.6700 - val_loss: 0.6460 -
learning_rate: 1.0000e-04
Epoch 11/20
50/50             129s 3s/step -
accuracy: 0.6816 - loss: 0.5860 - val_accuracy: 0.6525 - val_loss: 0.6096 -
learning_rate: 1.0000e-04
Epoch 12/20
50/50             143s 3s/step -
accuracy: 0.7071 - loss: 0.5629 - val_accuracy: 0.6700 - val_loss: 0.6115 -
learning_rate: 1.0000e-04
Epoch 13/20
50/50             137s 2s/step -
accuracy: 0.7127 - loss: 0.5479 - val_accuracy: 0.6750 - val_loss: 0.6185 -
learning_rate: 5.0000e-05
Epoch 14/20
50/50             132s 3s/step -
accuracy: 0.7295 - loss: 0.5204 - val_accuracy: 0.6825 - val_loss: 0.5829 -
learning_rate: 5.0000e-05
Epoch 15/20
50/50             132s 3s/step -
accuracy: 0.7304 - loss: 0.5187 - val_accuracy: 0.6650 - val_loss: 0.6064 -
learning_rate: 5.0000e-05
Epoch 16/20
50/50             138s 3s/step -
accuracy: 0.7351 - loss: 0.5276 - val_accuracy: 0.6950 - val_loss: 0.5760 -
learning_rate: 5.0000e-05
Epoch 17/20
50/50             128s 3s/step -
accuracy: 0.7637 - loss: 0.5007 - val_accuracy: 0.6900 - val_loss: 0.5725 -
learning_rate: 5.0000e-05
Epoch 18/20
```

```
50/50             143s 3s/step -
accuracy: 0.7726 - loss: 0.4784 - val_accuracy: 0.6850 - val_loss: 0.5772 -
learning_rate: 5.0000e-05
Epoch 19/20
50/50             131s 3s/step -
accuracy: 0.7655 - loss: 0.4811 - val_accuracy: 0.6850 - val_loss: 0.5706 -
learning_rate: 5.0000e-05
Epoch 20/20
50/50             132s 2s/step -
accuracy: 0.7917 - loss: 0.4616 - val_accuracy: 0.6525 - val_loss: 0.6041 -
learning_rate: 5.0000e-05
13/13             6s 484ms/step -
accuracy: 0.6293 - loss: 0.5918
Test accuracy: 0.652
```

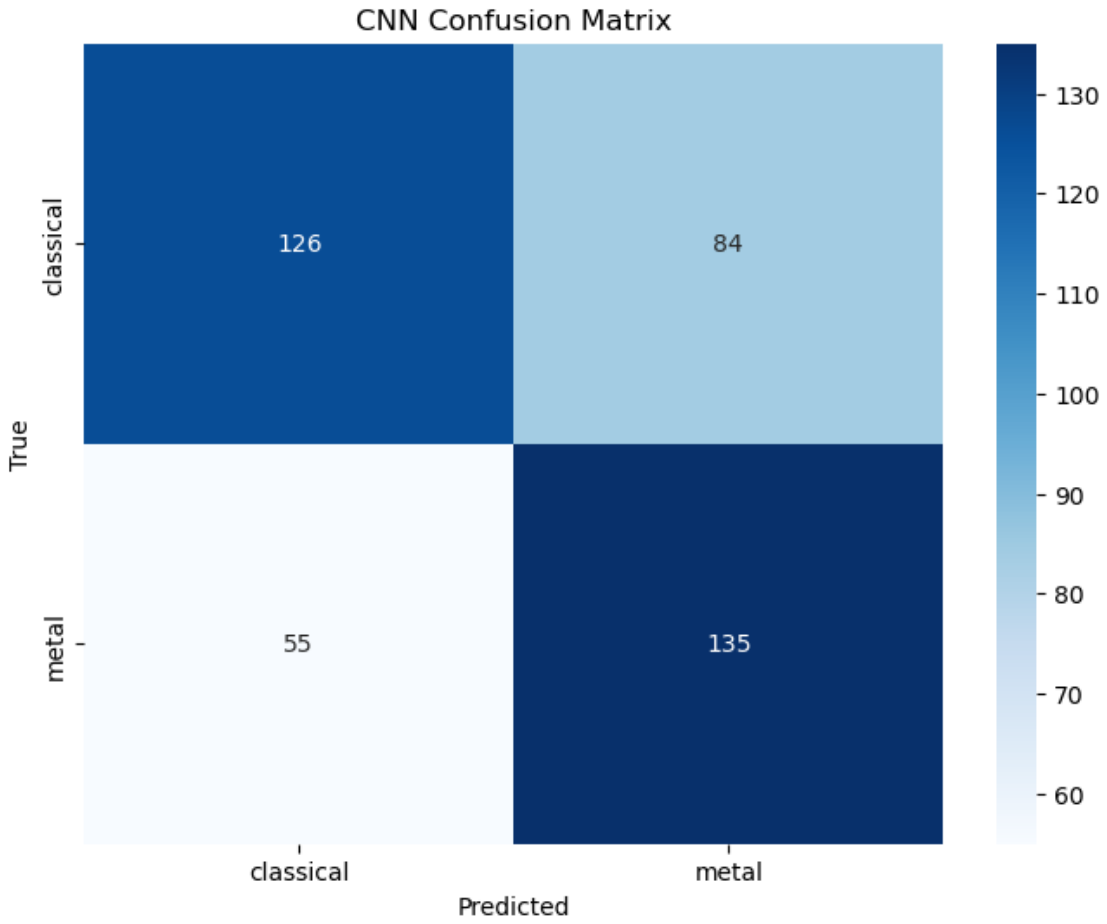
### 1.4 Confusion Matrix Easy (classical and metal)

[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             7s 488ms/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, Normalization
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', 'tempograms (3 secs)')

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, [256, 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(32, (3, 3), activation='relu', input_shape=(256, 256, 1)),
    Normalization(),
    MaxPooling2D((2, 2)),

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

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

    Conv2D(256, (3, 3), activation='relu'),
    Normalization(),
    MaxPooling2D((2, 2)),

    Flatten(),

    Dense(512, activation='relu'),
    Dropout(0.5),

    Dense(256, activation='relu'),
    Dropout(0.5),

    Dense(128, activation='relu'),
    Dense(len(GENRES), activation='softmax')  # Output size matches number of␣
 ↪genres
])

# 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             133s 3s/step -
accuracy: 0.5002 - loss: 0.6944 - val_accuracy: 0.4750 - val_loss: 0.6935 -
learning_rate: 1.0000e-04
Epoch 2/20
50/50             131s 3s/step -
accuracy: 0.5327 - loss: 0.6924 - val_accuracy: 0.5000 - val_loss: 0.6930 -
learning_rate: 1.0000e-04
Epoch 3/20
50/50             133s 3s/step -
accuracy: 0.5192 - loss: 0.6923 - val_accuracy: 0.4750 - val_loss: 0.6937 -
learning_rate: 1.0000e-04
Epoch 4/20
50/50             98s 2s/step -
accuracy: 0.5231 - loss: 0.6939 - val_accuracy: 0.5250 - val_loss: 0.6873 -
learning_rate: 1.0000e-04
Epoch 5/20
50/50             88s 2s/step -
accuracy: 0.5297 - loss: 0.6901 - val_accuracy: 0.5725 - val_loss: 0.6717 -
learning_rate: 1.0000e-04
Epoch 6/20
```

```
50/50            107s 2s/step -
accuracy: 0.5627 - loss: 0.6863 - val_accuracy: 0.6325 - val_loss: 0.6507 -
learning_rate: 1.0000e-04
Epoch 7/20
50/50            126s 3s/step -
accuracy: 0.6791 - loss: 0.6492 - val_accuracy: 0.6575 - val_loss: 0.6035 -
learning_rate: 1.0000e-04
Epoch 8/20
50/50            128s 3s/step -
accuracy: 0.6962 - loss: 0.5900 - val_accuracy: 0.6700 - val_loss: 0.5946 -
learning_rate: 1.0000e-04
Epoch 9/20
50/50            137s 2s/step -
accuracy: 0.7492 - loss: 0.5394 - val_accuracy: 0.6425 - val_loss: 0.6556 -
learning_rate: 1.0000e-04
Epoch 10/20
50/50            128s 3s/step -
accuracy: 0.7535 - loss: 0.5492 - val_accuracy: 0.7175 - val_loss: 0.5997 -
learning_rate: 1.0000e-04
Epoch 11/20
50/50            144s 3s/step -
accuracy: 0.7651 - loss: 0.5161 - val_accuracy: 0.6975 - val_loss: 0.6111 -
learning_rate: 1.0000e-04
Epoch 12/20
50/50            130s 3s/step -
accuracy: 0.7693 - loss: 0.4948 - val_accuracy: 0.7125 - val_loss: 0.6279 -
learning_rate: 5.0000e-05
Epoch 13/20
50/50            141s 3s/step -
accuracy: 0.7898 - loss: 0.4687 - val_accuracy: 0.6975 - val_loss: 0.6069 -
learning_rate: 5.0000e-05
Epoch 14/20
50/50            123s 2s/step -
accuracy: 0.8127 - loss: 0.4479 - val_accuracy: 0.7050 - val_loss: 0.6300 -
learning_rate: 5.0000e-05
Epoch 15/20
50/50            146s 3s/step -
accuracy: 0.8153 - loss: 0.4369 - val_accuracy: 0.7025 - val_loss: 0.6383 -
learning_rate: 2.5000e-05
Epoch 16/20
50/50            146s 3s/step -
accuracy: 0.8116 - loss: 0.4304 - val_accuracy: 0.7050 - val_loss: 0.6063 -
learning_rate: 2.5000e-05
Epoch 17/20
50/50            145s 3s/step -
accuracy: 0.8253 - loss: 0.4308 - val_accuracy: 0.7050 - val_loss: 0.6129 -
learning_rate: 2.5000e-05
Epoch 18/20
```

```
50/50              137s 3s/step -
accuracy: 0.8242 - loss: 0.4201 - val_accuracy: 0.7150 - val_loss: 0.6257 -
learning_rate: 1.2500e-05
Epoch 19/20
50/50              125s 2s/step -
accuracy: 0.8077 - loss: 0.4261 - val_accuracy: 0.7150 - val_loss: 0.6297 -
learning_rate: 1.2500e-05
Epoch 20/20
50/50              129s 3s/step -
accuracy: 0.8168 - loss: 0.4332 - val_accuracy: 0.7225 - val_loss: 0.6239 -
learning_rate: 1.2500e-05
13/13              6s 476ms/step -
accuracy: 0.6764 - loss: 0.7388
Test accuracy: 0.723
```

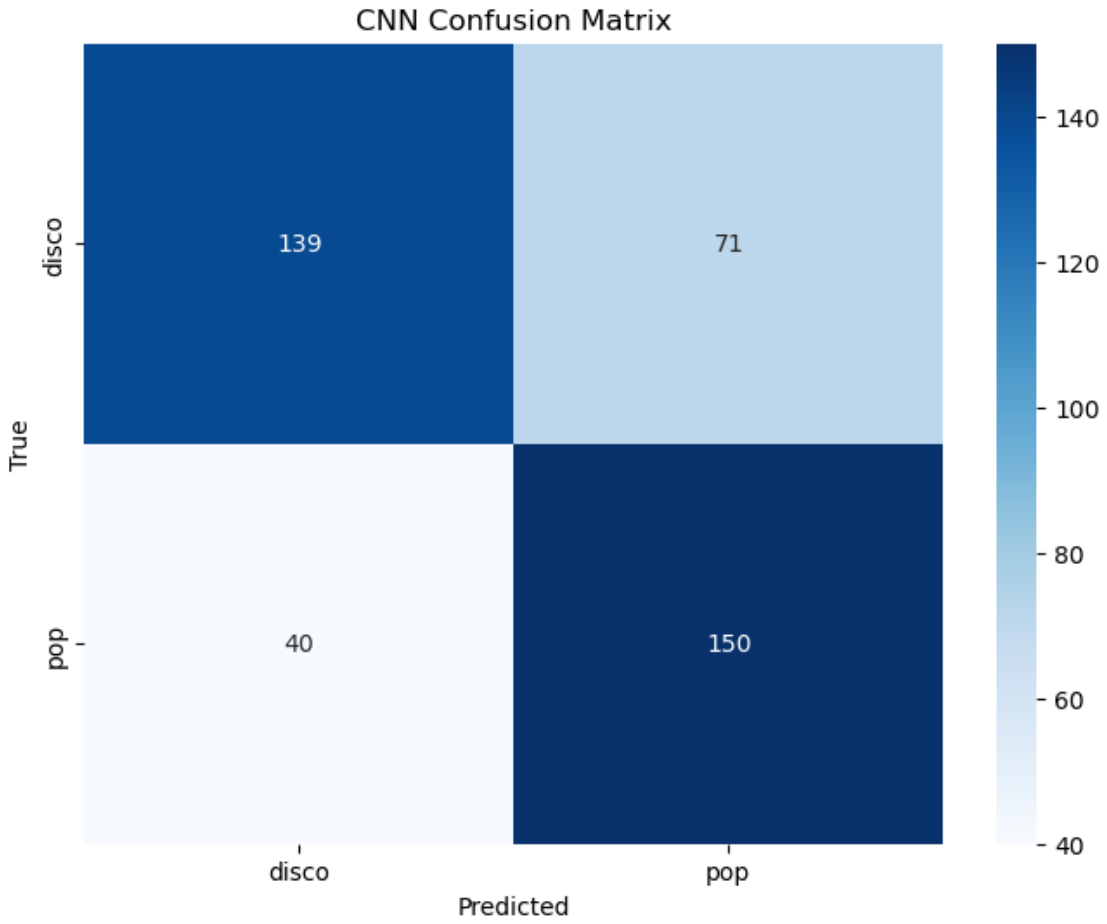
### 1.6 Confusion Matrix Hard (disco and pop)

[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              8s 558ms/step
```




### 1.7 4 - 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, Normalization
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', 'tempograms (3 secs)')

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, [256, 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(32, (3, 3), activation='relu', input_shape=(256, 256, 1)),
    Normalization(),
    MaxPooling2D((2, 2)),

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

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

    Conv2D(256, (3, 3), activation='relu'),
    Normalization(),
    MaxPooling2D((2, 2)),

    Flatten(),

    Dense(512, activation='relu'),
    Dropout(0.5),

    Dense(256, activation='relu'),
    Dropout(0.5),

    Dense(128, activation='relu'),
    Dense(len(GENRES), activation='softmax')  # Output size matches number of␣
 ↪genres
])
```

```
# 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}")
```

```
['disco', 'metal', 'blues', 'reggae', 'rock']
Processing genre: disco
Processing genre: metal
Processing genre: blues
Processing genre: reggae
Processing genre: rock
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             331s 3s/step -
accuracy: 0.1972 - loss: 1.6105 - val_accuracy: 0.1040 - val_loss: 1.6141 -
learning_rate: 1.0000e-04
Epoch 2/20
125/125             247s 2s/step -
accuracy: 0.2563 - loss: 1.5880 - val_accuracy: 0.2440 - val_loss: 1.5414 -
learning_rate: 1.0000e-04
Epoch 3/20
125/125             262s 2s/step -
accuracy: 0.2966 - loss: 1.5085 - val_accuracy: 0.2620 - val_loss: 1.5219 -
learning_rate: 1.0000e-04
Epoch 4/20
125/125             324s 3s/step -
accuracy: 0.3123 - loss: 1.4688 - val_accuracy: 0.2730 - val_loss: 1.4836 -
```

```
learning_rate: 1.0000e-04
Epoch 5/20
125/125             316s 3s/step -
accuracy: 0.3439 - loss: 1.4753 - val_accuracy: 0.2900 - val_loss: 1.5189 -
learning_rate: 1.0000e-04
Epoch 6/20
125/125             340s 3s/step -
accuracy: 0.3441 - loss: 1.4693 - val_accuracy: 0.2670 - val_loss: 1.5005 -
learning_rate: 1.0000e-04
Epoch 7/20
125/125             318s 3s/step -
accuracy: 0.3538 - loss: 1.4455 - val_accuracy: 0.3170 - val_loss: 1.4330 -
learning_rate: 1.0000e-04
Epoch 8/20
125/125             320s 3s/step -
accuracy: 0.3467 - loss: 1.4596 - val_accuracy: 0.3020 - val_loss: 1.4642 -
learning_rate: 1.0000e-04
Epoch 9/20
125/125             331s 3s/step -
accuracy: 0.3597 - loss: 1.4296 - val_accuracy: 0.3170 - val_loss: 1.4274 -
learning_rate: 1.0000e-04
Epoch 10/20
125/125             313s 3s/step -
accuracy: 0.3689 - loss: 1.4307 - val_accuracy: 0.3440 - val_loss: 1.4492 -
learning_rate: 1.0000e-04
Epoch 11/20
125/125             333s 3s/step -
accuracy: 0.3908 - loss: 1.4062 - val_accuracy: 0.4010 - val_loss: 1.4017 -
learning_rate: 1.0000e-04
Epoch 12/20
125/125             392s 3s/step -
accuracy: 0.4274 - loss: 1.3802 - val_accuracy: 0.4520 - val_loss: 1.3108 -
learning_rate: 1.0000e-04
Epoch 13/20
125/125             325s 3s/step -
accuracy: 0.4502 - loss: 1.3334 - val_accuracy: 0.3480 - val_loss: 1.5574 -
learning_rate: 1.0000e-04
Epoch 14/20
125/125             314s 3s/step -
accuracy: 0.4622 - loss: 1.2999 - val_accuracy: 0.4540 - val_loss: 1.2971 -
learning_rate: 1.0000e-04
Epoch 15/20
125/125             307s 2s/step -
accuracy: 0.4975 - loss: 1.2418 - val_accuracy: 0.4000 - val_loss: 1.3991 -
learning_rate: 1.0000e-04
Epoch 16/20
125/125             324s 3s/step -
accuracy: 0.4951 - loss: 1.2474 - val_accuracy: 0.4750 - val_loss: 1.2743 -
```

```
learning_rate: 1.0000e-04
Epoch 17/20
125/125             309s 2s/step -
accuracy: 0.5089 - loss: 1.2176 - val_accuracy: 0.4230 - val_loss: 1.3834 -
learning_rate: 1.0000e-04
Epoch 18/20
125/125             332s 3s/step -
accuracy: 0.4979 - loss: 1.2221 - val_accuracy: 0.4510 - val_loss: 1.3530 -
learning_rate: 1.0000e-04
Epoch 19/20
125/125             328s 3s/step -
accuracy: 0.5275 - loss: 1.1976 - val_accuracy: 0.4470 - val_loss: 1.2990 -
learning_rate: 1.0000e-04
Epoch 20/20
125/125             335s 3s/step -
accuracy: 0.5405 - loss: 1.1422 - val_accuracy: 0.4670 - val_loss: 1.2766 -
learning_rate: 5.0000e-05
32/32             19s 578ms/step -
accuracy: 0.5930 - loss: 1.1996
Test accuracy: 0.467
```

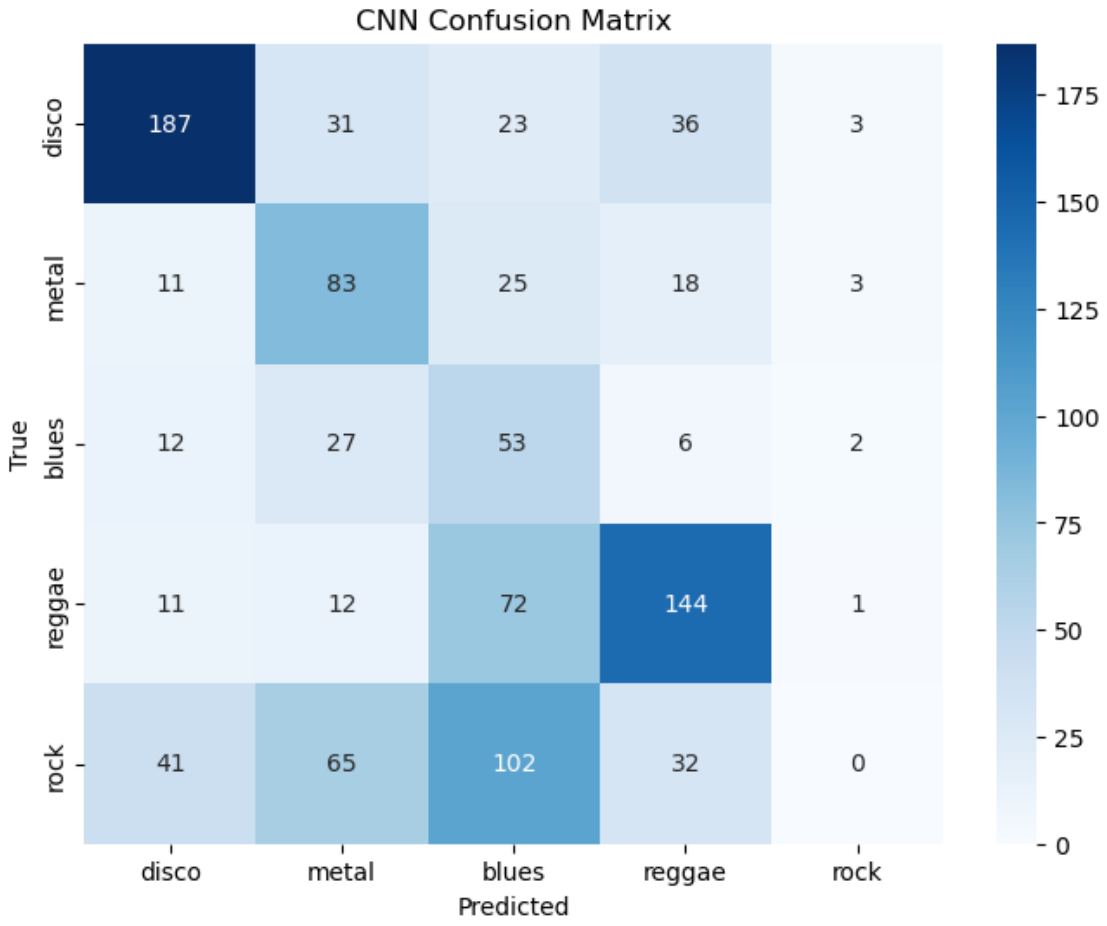
### 1.8 Confusion Matrix Medium (5 random)

[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             20s 602ms/step
```


CNN Confusion Matrix
True
Predicted
disco
metal
blues
reggae
rock
187 31 23 36 3
11 83 25 18 3
12 27 53 6 2
11 12 72 144 1
41 65 102 32 0
175
150
125
100
75
50
25
0