Spectrogram-Only (3 secs) CNN

March 22, 2025

1 CNN for Spectrogram (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', 'spectrograms (3 secs)', 'spectrogram_512')

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 01:34:38.608579: 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             833s 3s/step -
accuracy: 0.1373 - loss: 2.2631 - val_accuracy: 0.3095 - val_loss: 1.8501 -
learning_rate: 1.0000e-04
Epoch 2/20
250/250             865s 3s/step -
accuracy: 0.3290 - loss: 1.8363 - val_accuracy: 0.3965 - val_loss: 1.5511 -
learning_rate: 1.0000e-04
Epoch 3/20
250/250             875s 4s/step -
accuracy: 0.3942 - loss: 1.6289 - val_accuracy: 0.4370 - val_loss: 1.4853 -
learning_rate: 1.0000e-04
Epoch 4/20
250/250             888s 3s/step -
accuracy: 0.4431 - loss: 1.5210 - val_accuracy: 0.4820 - val_loss: 1.3832 -
learning_rate: 1.0000e-04
Epoch 5/20
250/250             884s 3s/step -
accuracy: 0.4981 - loss: 1.4001 - val_accuracy: 0.5345 - val_loss: 1.2726 -
learning_rate: 1.0000e-04
Epoch 6/20
250/250             826s 3s/step -
accuracy: 0.5132 - loss: 1.3495 - val_accuracy: 0.5250 - val_loss: 1.2488 -
learning_rate: 1.0000e-04
Epoch 7/20
250/250             844s 3s/step -
```

```
accuracy: 0.5477 - loss: 1.2857 - val_accuracy: 0.5580 - val_loss: 1.2250 -
learning_rate: 1.0000e-04
Epoch 8/20
250/250            909s 3s/step -
accuracy: 0.5765 - loss: 1.2106 - val_accuracy: 0.5940 - val_loss: 1.1636 -
learning_rate: 1.0000e-04
Epoch 9/20
250/250            770s 3s/step -
accuracy: 0.6034 - loss: 1.1455 - val_accuracy: 0.5845 - val_loss: 1.1860 -
learning_rate: 1.0000e-04
Epoch 10/20
250/250            684s 3s/step -
accuracy: 0.6263 - loss: 1.0658 - val_accuracy: 0.6425 - val_loss: 1.0521 -
learning_rate: 1.0000e-04
Epoch 11/20
250/250            742s 3s/step -
accuracy: 0.6502 - loss: 1.0100 - val_accuracy: 0.6300 - val_loss: 1.0750 -
learning_rate: 1.0000e-04
Epoch 12/20
250/250            694s 3s/step -
accuracy: 0.6743 - loss: 0.9450 - val_accuracy: 0.6405 - val_loss: 1.0358 -
learning_rate: 1.0000e-04
Epoch 13/20
250/250            749s 3s/step -
accuracy: 0.6861 - loss: 0.9069 - val_accuracy: 0.6670 - val_loss: 0.9802 -
learning_rate: 1.0000e-04
Epoch 14/20
250/250            689s 3s/step -
accuracy: 0.7320 - loss: 0.8009 - val_accuracy: 0.6735 - val_loss: 0.9701 -
learning_rate: 1.0000e-04
Epoch 15/20
250/250            687s 3s/step -
accuracy: 0.7274 - loss: 0.7718 - val_accuracy: 0.7050 - val_loss: 0.9467 -
learning_rate: 1.0000e-04
Epoch 16/20
250/250            693s 3s/step -
accuracy: 0.7727 - loss: 0.6884 - val_accuracy: 0.6970 - val_loss: 0.9383 -
learning_rate: 1.0000e-04
Epoch 17/20
250/250            668s 3s/step -
accuracy: 0.7797 - loss: 0.6376 - val_accuracy: 0.6865 - val_loss: 1.0315 -
learning_rate: 1.0000e-04
Epoch 18/20
250/250            689s 3s/step -
accuracy: 0.7913 - loss: 0.6099 - val_accuracy: 0.7155 - val_loss: 0.9230 -
learning_rate: 1.0000e-04
Epoch 19/20
250/250            686s 3s/step -
```

```
accuracy: 0.8064 - loss: 0.5626 - val_accuracy: 0.6915 - val_loss: 0.9619 -
learning_rate: 1.0000e-04
Epoch 20/20
250/250             650s 3s/step -
accuracy: 0.8391 - loss: 0.4776 - val_accuracy: 0.7260 - val_loss: 0.9182 -
learning_rate: 1.0000e-04
63/63             40s 626ms/step -
accuracy: 0.7607 - loss: 0.7900
Test accuracy: 0.726
```

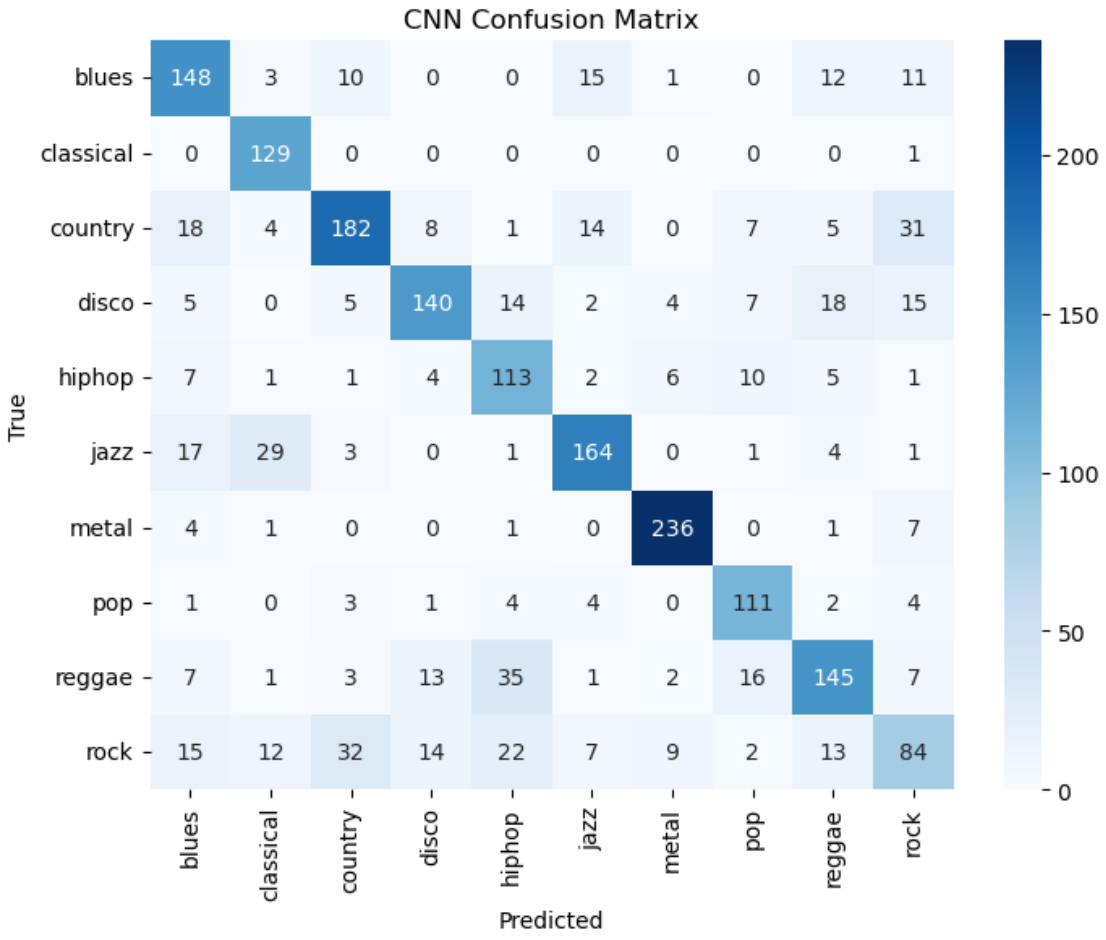
1.2 Apply the confusion matrix after the model

[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             34s 515ms/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', 'spectrograms (3 secs)', 'spectrogram_512')

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             113s 2s/step -
accuracy: 0.6454 - loss: 0.5985 - val_accuracy: 0.9000 - val_loss: 0.2571 -
learning_rate: 1.0000e-04
Epoch 2/20
50/50             164s 3s/step -
accuracy: 0.9332 - loss: 0.2167 - val_accuracy: 0.9825 - val_loss: 0.0584 -
learning_rate: 1.0000e-04
Epoch 3/20
50/50             137s 3s/step -
accuracy: 0.9743 - loss: 0.0749 - val_accuracy: 0.9775 - val_loss: 0.0873 -
learning_rate: 1.0000e-04
Epoch 4/20
50/50             130s 3s/step -
accuracy: 0.9871 - loss: 0.0441 - val_accuracy: 0.9850 - val_loss: 0.0426 -
learning_rate: 1.0000e-04
Epoch 5/20
50/50             135s 3s/step -
accuracy: 0.9918 - loss: 0.0264 - val_accuracy: 0.9875 - val_loss: 0.0279 -
learning_rate: 1.0000e-04
Epoch 6/20
```

```
50/50            136s 3s/step -
accuracy: 0.9899 - loss: 0.0296 - val_accuracy: 0.9900 - val_loss: 0.0348 -
learning_rate: 1.0000e-04
Epoch 7/20
50/50            129s 3s/step -
accuracy: 0.9987 - loss: 0.0139 - val_accuracy: 0.9900 - val_loss: 0.0291 -
learning_rate: 1.0000e-04
Epoch 8/20
50/50            130s 3s/step -
accuracy: 0.9973 - loss: 0.0107 - val_accuracy: 0.9925 - val_loss: 0.0176 -
learning_rate: 1.0000e-04
Epoch 9/20
50/50            123s 2s/step -
accuracy: 0.9975 - loss: 0.0077 - val_accuracy: 0.9975 - val_loss: 0.0133 -
learning_rate: 1.0000e-04
Epoch 10/20
50/50            134s 3s/step -
accuracy: 0.9970 - loss: 0.0085 - val_accuracy: 0.9950 - val_loss: 0.0189 -
learning_rate: 1.0000e-04
Epoch 11/20
50/50            132s 3s/step -
accuracy: 0.9959 - loss: 0.0088 - val_accuracy: 0.9800 - val_loss: 0.0835 -
learning_rate: 1.0000e-04
Epoch 12/20
50/50            134s 3s/step -
accuracy: 0.9957 - loss: 0.0091 - val_accuracy: 0.9800 - val_loss: 0.0731 -
learning_rate: 1.0000e-04
Epoch 13/20
50/50            133s 2s/step -
accuracy: 0.9975 - loss: 0.0063 - val_accuracy: 0.9800 - val_loss: 0.0862 -
learning_rate: 5.0000e-05
Epoch 14/20
50/50            130s 3s/step -
accuracy: 0.9998 - loss: 0.0025 - val_accuracy: 0.9775 - val_loss: 0.1298 -
learning_rate: 5.0000e-05
Epoch 15/20
50/50            131s 3s/step -
accuracy: 0.9923 - loss: 0.0177 - val_accuracy: 0.9850 - val_loss: 0.0474 -
learning_rate: 5.0000e-05
Epoch 16/20
50/50            126s 3s/step -
accuracy: 0.9998 - loss: 0.0030 - val_accuracy: 0.9825 - val_loss: 0.0519 -
learning_rate: 2.5000e-05
Epoch 17/20
50/50            116s 2s/step -
accuracy: 1.0000 - loss: 0.0018 - val_accuracy: 0.9925 - val_loss: 0.0302 -
learning_rate: 2.5000e-05
Epoch 18/20
```

```
50/50              172s 3s/step -
accuracy: 1.0000 - loss: 0.0014 - val_accuracy: 0.9825 - val_loss: 0.0544 -
learning_rate: 2.5000e-05
Epoch 19/20
50/50              135s 3s/step -
accuracy: 0.9986 - loss: 0.0035 - val_accuracy: 0.9900 - val_loss: 0.0370 -
learning_rate: 1.2500e-05
Epoch 20/20
50/50              134s 3s/step -
accuracy: 0.9981 - loss: 0.0028 - val_accuracy: 0.9900 - val_loss: 0.0363 -
learning_rate: 1.2500e-05
13/13              7s 504ms/step -
accuracy: 0.9830 - loss: 0.0519
Test accuracy: 0.990
```

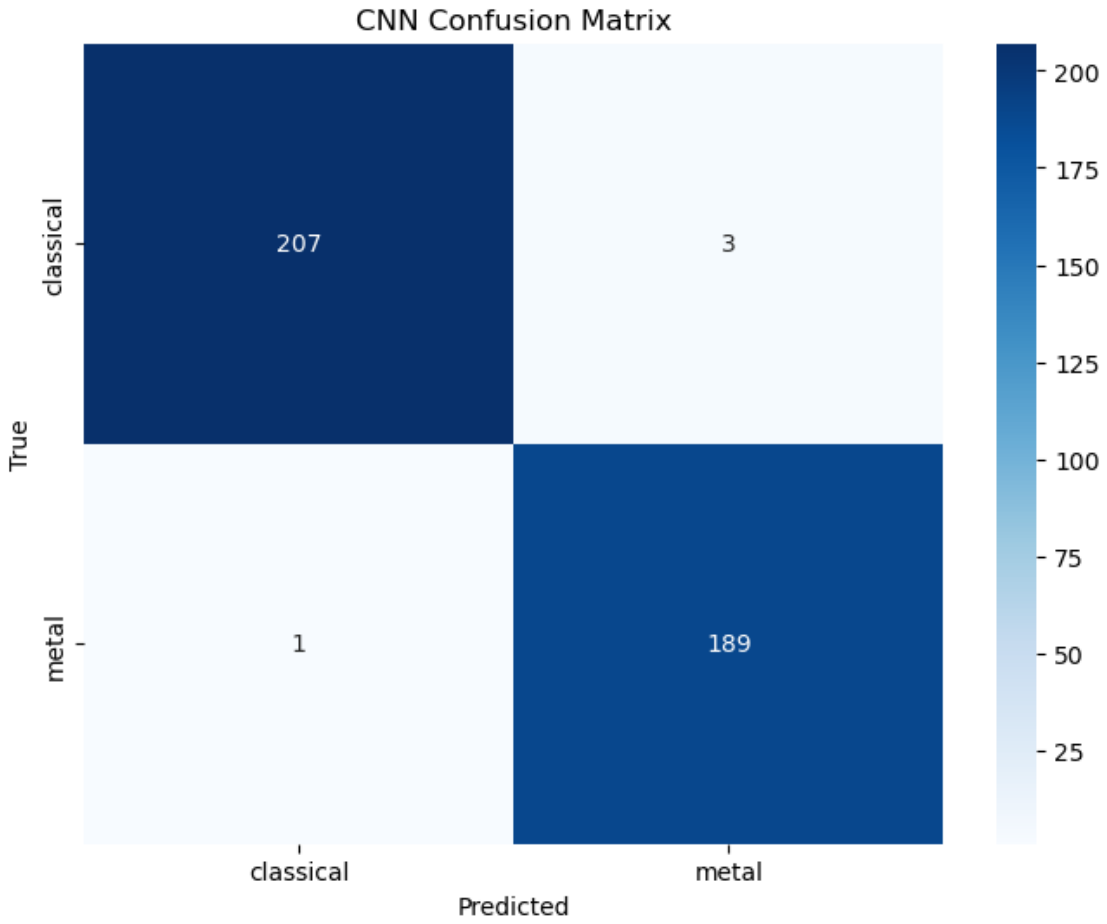
1.4 Confusion Matrix Easy (classical and metal)

[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 514ms/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', 'spectrograms (3 secs)', 'spectrogram_512')

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             121s 2s/step -
accuracy: 0.5520 - loss: 0.6876 - val_accuracy: 0.7425 - val_loss: 0.5479 -
learning_rate: 1.0000e-04
Epoch 2/20
50/50             158s 3s/step -
accuracy: 0.7378 - loss: 0.5189 - val_accuracy: 0.7750 - val_loss: 0.4052 -
learning_rate: 1.0000e-04
Epoch 3/20
50/50             139s 3s/step -
accuracy: 0.8065 - loss: 0.4113 - val_accuracy: 0.8100 - val_loss: 0.3718 -
learning_rate: 1.0000e-04
Epoch 4/20
50/50             139s 3s/step -
accuracy: 0.8226 - loss: 0.3717 - val_accuracy: 0.8175 - val_loss: 0.3584 -
learning_rate: 1.0000e-04
Epoch 5/20
50/50             150s 3s/step -
accuracy: 0.8371 - loss: 0.3634 - val_accuracy: 0.8000 - val_loss: 0.4047 -
learning_rate: 1.0000e-04
Epoch 6/20
```

```
50/50             139s 3s/step -
accuracy: 0.8515 - loss: 0.3278 - val_accuracy: 0.8375 - val_loss: 0.3464 -
learning_rate: 1.0000e-04
Epoch 7/20
50/50             129s 3s/step -
accuracy: 0.8544 - loss: 0.3020 - val_accuracy: 0.8325 - val_loss: 0.3597 -
learning_rate: 1.0000e-04
Epoch 8/20
50/50             129s 3s/step -
accuracy: 0.8620 - loss: 0.3191 - val_accuracy: 0.8500 - val_loss: 0.3254 -
learning_rate: 1.0000e-04
Epoch 9/20
50/50             133s 2s/step -
accuracy: 0.8968 - loss: 0.2550 - val_accuracy: 0.8450 - val_loss: 0.3398 -
learning_rate: 1.0000e-04
Epoch 10/20
50/50             124s 2s/step -
accuracy: 0.8938 - loss: 0.2470 - val_accuracy: 0.8425 - val_loss: 0.3658 -
learning_rate: 1.0000e-04
Epoch 11/20
50/50             129s 3s/step -
accuracy: 0.8949 - loss: 0.2419 - val_accuracy: 0.8600 - val_loss: 0.3489 -
learning_rate: 1.0000e-04
Epoch 12/20
50/50             132s 3s/step -
accuracy: 0.9013 - loss: 0.2110 - val_accuracy: 0.8450 - val_loss: 0.4212 -
learning_rate: 5.0000e-05
Epoch 13/20
50/50             133s 3s/step -
accuracy: 0.9077 - loss: 0.2080 - val_accuracy: 0.8625 - val_loss: 0.3315 -
learning_rate: 5.0000e-05
Epoch 14/20
50/50             134s 3s/step -
accuracy: 0.9238 - loss: 0.1823 - val_accuracy: 0.8475 - val_loss: 0.4295 -
learning_rate: 5.0000e-05
Epoch 15/20
50/50             135s 3s/step -
accuracy: 0.9343 - loss: 0.1748 - val_accuracy: 0.8475 - val_loss: 0.4262 -
learning_rate: 2.5000e-05
Epoch 16/20
50/50             100s 2s/step -
accuracy: 0.9296 - loss: 0.1787 - val_accuracy: 0.8650 - val_loss: 0.3502 -
learning_rate: 2.5000e-05
Epoch 17/20
50/50             99s 2s/step -
accuracy: 0.9323 - loss: 0.1639 - val_accuracy: 0.8500 - val_loss: 0.4218 -
learning_rate: 2.5000e-05
Epoch 18/20
```

```
50/50              179s 3s/step -
accuracy: 0.9254 - loss: 0.1785 - val_accuracy: 0.8525 - val_loss: 0.4156 -
learning_rate: 1.2500e-05
Epoch 19/20
50/50              137s 3s/step -
accuracy: 0.9443 - loss: 0.1380 - val_accuracy: 0.8650 - val_loss: 0.3830 -
learning_rate: 1.2500e-05
Epoch 20/20
50/50              101s 2s/step -
accuracy: 0.9416 - loss: 0.1505 - val_accuracy: 0.8525 - val_loss: 0.4041 -
learning_rate: 1.2500e-05
13/13              6s 476ms/step -
accuracy: 0.7435 - loss: 0.7243
Test accuracy: 0.853
```

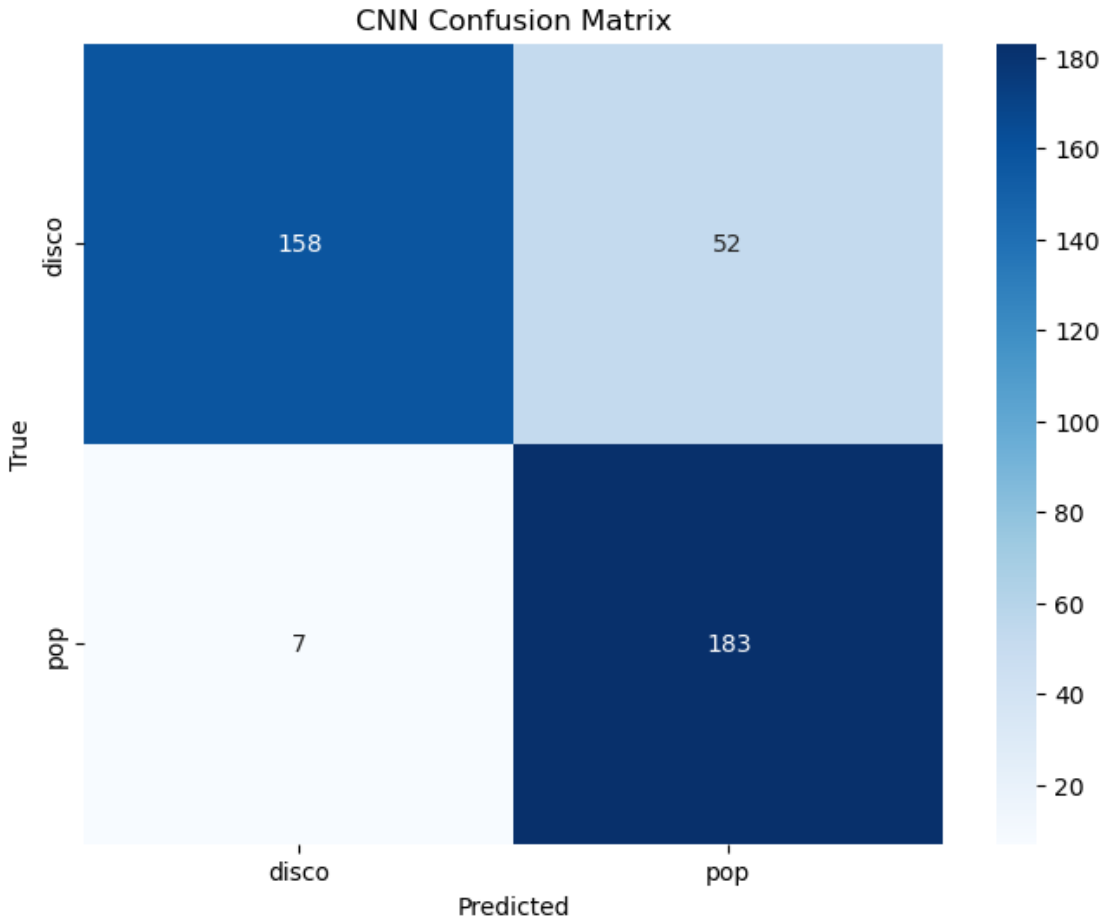
1.6 Confusion Matrix Hard (disco and pop)

[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              7s 557ms/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', 'spectrograms (3 secs)', 'spectrogram_512')

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}")
```

```
['metal', 'blues', 'disco', 'country', 'classical']
Processing genre: metal
Processing genre: blues
Processing genre: disco
Processing genre: country
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             303s 2s/step -
accuracy: 0.2512 - loss: 1.5737 - val_accuracy: 0.2630 - val_loss: 1.4406 -
learning_rate: 1.0000e-04
Epoch 2/20
125/125             358s 3s/step -
accuracy: 0.4178 - loss: 1.3306 - val_accuracy: 0.5400 - val_loss: 1.1575 -
learning_rate: 1.0000e-04
Epoch 3/20
125/125             377s 3s/step -
accuracy: 0.5948 - loss: 1.0159 - val_accuracy: 0.7080 - val_loss: 0.7385 -
learning_rate: 1.0000e-04
Epoch 4/20
125/125             329s 3s/step -
accuracy: 0.6632 - loss: 0.8531 - val_accuracy: 0.7130 - val_loss: 0.7044 -
```

```
learning_rate: 1.0000e-04
Epoch 5/20
125/125             318s 3s/step -
accuracy: 0.6892 - loss: 0.8033 - val_accuracy: 0.7340 - val_loss: 0.6691 -
learning_rate: 1.0000e-04
Epoch 6/20
125/125             326s 3s/step -
accuracy: 0.7228 - loss: 0.7316 - val_accuracy: 0.7670 - val_loss: 0.5988 -
learning_rate: 1.0000e-04
Epoch 7/20
125/125             319s 3s/step -
accuracy: 0.7526 - loss: 0.6740 - val_accuracy: 0.8300 - val_loss: 0.4730 -
learning_rate: 1.0000e-04
Epoch 8/20
125/125             321s 3s/step -
accuracy: 0.7758 - loss: 0.5877 - val_accuracy: 0.7870 - val_loss: 0.5173 -
learning_rate: 1.0000e-04
Epoch 9/20
125/125             321s 3s/step -
accuracy: 0.7869 - loss: 0.5670 - val_accuracy: 0.7950 - val_loss: 0.5032 -
learning_rate: 1.0000e-04
Epoch 10/20
125/125             332s 3s/step -
accuracy: 0.8112 - loss: 0.5240 - val_accuracy: 0.8060 - val_loss: 0.4803 -
learning_rate: 1.0000e-04
Epoch 11/20
125/125             320s 3s/step -
accuracy: 0.8362 - loss: 0.4619 - val_accuracy: 0.8550 - val_loss: 0.3615 -
learning_rate: 5.0000e-05
Epoch 12/20
125/125             327s 3s/step -
accuracy: 0.8481 - loss: 0.4440 - val_accuracy: 0.8270 - val_loss: 0.4334 -
learning_rate: 5.0000e-05
Epoch 13/20
125/125             392s 3s/step -
accuracy: 0.8720 - loss: 0.3738 - val_accuracy: 0.7980 - val_loss: 0.5868 -
learning_rate: 5.0000e-05
Epoch 14/20
125/125             326s 3s/step -
accuracy: 0.8631 - loss: 0.3900 - val_accuracy: 0.8590 - val_loss: 0.3583 -
learning_rate: 5.0000e-05
Epoch 15/20
125/125             386s 3s/step -
accuracy: 0.8725 - loss: 0.3706 - val_accuracy: 0.8210 - val_loss: 0.4956 -
learning_rate: 5.0000e-05
Epoch 16/20
125/125             323s 3s/step -
accuracy: 0.8881 - loss: 0.3326 - val_accuracy: 0.8560 - val_loss: 0.3667 -
```

```
learning_rate: 5.0000e-05
Epoch 17/20
125/125             336s 3s/step -
accuracy: 0.8787 - loss: 0.3402 - val_accuracy: 0.8200 - val_loss: 0.5124 -
learning_rate: 5.0000e-05
Epoch 18/20
125/125             289s 2s/step -
accuracy: 0.8931 - loss: 0.3200 - val_accuracy: 0.8420 - val_loss: 0.4219 -
learning_rate: 2.5000e-05
Epoch 19/20
125/125             227s 2s/step -
accuracy: 0.8918 - loss: 0.2951 - val_accuracy: 0.8410 - val_loss: 0.4342 -
learning_rate: 2.5000e-05
Epoch 20/20
125/125             228s 2s/step -
accuracy: 0.9042 - loss: 0.2777 - val_accuracy: 0.8290 - val_loss: 0.4888 -
learning_rate: 2.5000e-05
32/32             10s 306ms/step -
accuracy: 0.8870 - loss: 0.3268
Test accuracy: 0.829
```

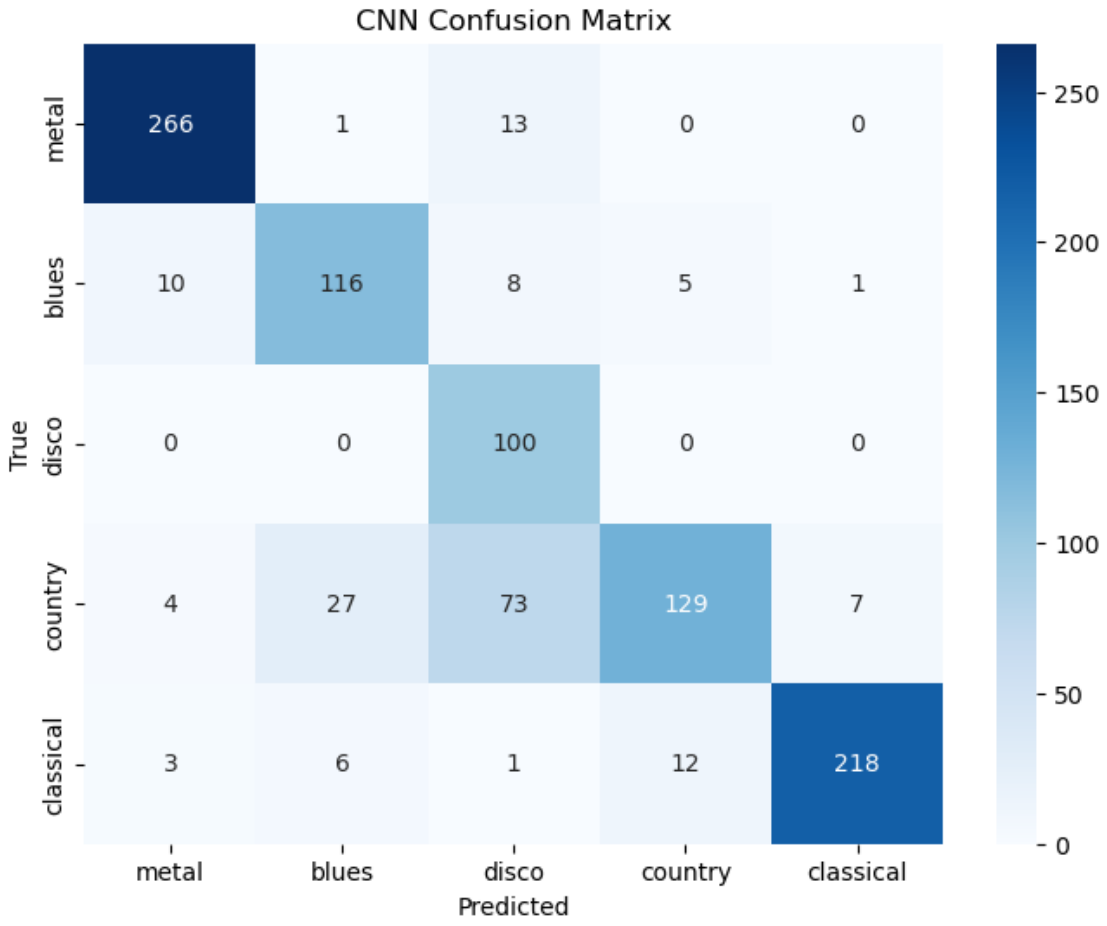
1.8 Confusion Matrix Medium (5 random)

[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             10s 309ms/step
```


CNN Confusion Matrix
True
Predicted
metal
blues
disco
country
classical
266
1
13
0
0
10
116
8
5
1
0
0
100
0
0
4
27
73
129
7
3
6
1
12
218
250
200
150
100
50
0