Mel-Spectrogram-Only (3 secs) CNN

March 22, 2025

1 CNN for Mel-Spectrogram (3 secs)

1.1 1 - All 10

[1]:
```python
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', 'mel_spectrograms (3 secs)',␣
↪'mel_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([
```

```python
    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 01:34:29.539964: 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              818s 3s/step -
accuracy: 0.1273 - loss: 2.2744 - val_accuracy: 0.2445 - val_loss: 2.0385 -
learning_rate: 1.0000e-04
Epoch 2/20
250/250              925s 4s/step -
accuracy: 0.2898 - loss: 1.9442 - val_accuracy: 0.4125 - val_loss: 1.5736 -
learning_rate: 1.0000e-04
Epoch 3/20
250/250              882s 4s/step -
accuracy: 0.3960 - loss: 1.6165 - val_accuracy: 0.4500 - val_loss: 1.4110 -
learning_rate: 1.0000e-04
Epoch 4/20
250/250              844s 3s/step -
accuracy: 0.4741 - loss: 1.4444 - val_accuracy: 0.4750 - val_loss: 1.3296 -
learning_rate: 1.0000e-04
Epoch 5/20
250/250              833s 3s/step -
accuracy: 0.5080 - loss: 1.3485 - val_accuracy: 0.5355 - val_loss: 1.2397 -
learning_rate: 1.0000e-04
Epoch 6/20
250/250              828s 3s/step -
accuracy: 0.5569 - loss: 1.2529 - val_accuracy: 0.5795 - val_loss: 1.1709 -
learning_rate: 1.0000e-04
Epoch 7/20
```

```
250/250 803s 3s/step -
accuracy: 0.5872 - loss: 1.1728 - val_accuracy: 0.5195 - val_loss: 1.3018 -
learning_rate: 1.0000e-04
Epoch 8/20
250/250 920s 3s/step -
accuracy: 0.6071 - loss: 1.1081 - val_accuracy: 0.6215 - val_loss: 1.0494 -
learning_rate: 1.0000e-04
Epoch 9/20
250/250 773s 3s/step -
accuracy: 0.6550 - loss: 1.0158 - val_accuracy: 0.6425 - val_loss: 1.0253 -
learning_rate: 1.0000e-04
Epoch 10/20
250/250 699s 3s/step -
accuracy: 0.6653 - loss: 0.9695 - val_accuracy: 0.6200 - val_loss: 1.0582 -
learning_rate: 1.0000e-04
Epoch 11/20
250/250 712s 3s/step -
accuracy: 0.6758 - loss: 0.9305 - val_accuracy: 0.6560 - val_loss: 1.0008 -
learning_rate: 1.0000e-04
Epoch 12/20
250/250 705s 3s/step -
accuracy: 0.7045 - loss: 0.8546 - val_accuracy: 0.6640 - val_loss: 0.9721 -
learning_rate: 1.0000e-04
Epoch 13/20
250/250 689s 3s/step -
accuracy: 0.7166 - loss: 0.8087 - val_accuracy: 0.6665 - val_loss: 0.9582 -
learning_rate: 1.0000e-04
Epoch 14/20
250/250 694s 3s/step -
accuracy: 0.7382 - loss: 0.7526 - val_accuracy: 0.6670 - val_loss: 0.9679 -
learning_rate: 1.0000e-04
Epoch 15/20
250/250 680s 3s/step -
accuracy: 0.7570 - loss: 0.7173 - val_accuracy: 0.6310 - val_loss: 1.0394 -
learning_rate: 1.0000e-04
Epoch 16/20
250/250 702s 3s/step -
accuracy: 0.7824 - loss: 0.6422 - val_accuracy: 0.6710 - val_loss: 0.9810 -
learning_rate: 1.0000e-04
Epoch 17/20
250/250 727s 3s/step -
accuracy: 0.8012 - loss: 0.5802 - val_accuracy: 0.6845 - val_loss: 0.9525 -
learning_rate: 5.0000e-05
Epoch 18/20
250/250 754s 3s/step -
accuracy: 0.8206 - loss: 0.5179 - val_accuracy: 0.6540 - val_loss: 1.0576 -
learning_rate: 5.0000e-05
Epoch 19/20
```

```
250/250 740s 3s/step -
accuracy: 0.8219 - loss: 0.5293 - val_accuracy: 0.6870 - val_loss: 0.9679 -
learning_rate: 5.0000e-05
Epoch 20/20
250/250 647s 3s/step -
accuracy: 0.8326 - loss: 0.4864 - val_accuracy: 0.7000 - val_loss: 0.9511 -
learning_rate: 5.0000e-05
63/63 37s 575ms/step -
accuracy: 0.7285 - loss: 0.8502
Test accuracy: 0.700
```

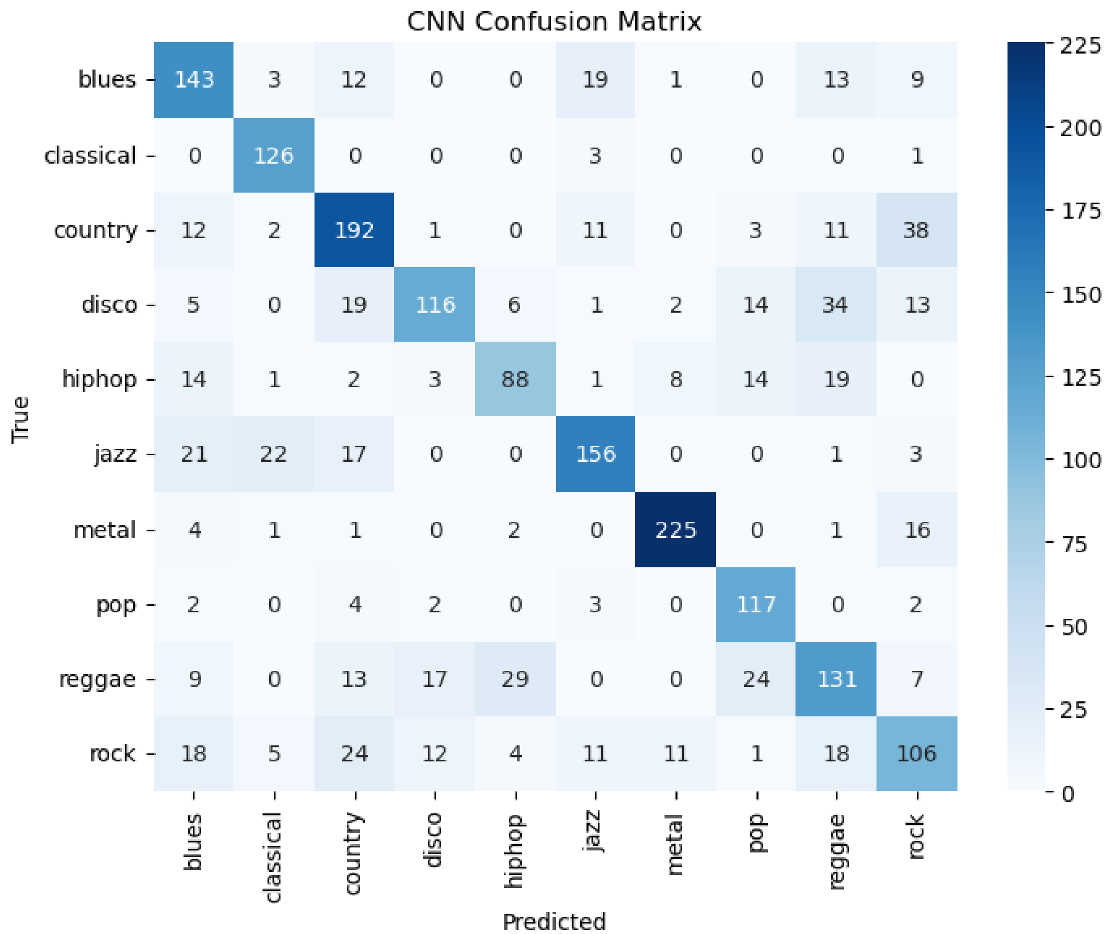
1.2 Apply the confusion matrix after the model

[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 28s 412ms/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

# 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)
```

```python
    return image

# Define the genres and file paths
GENRES = ['classical', 'metal']
FILE_PATH = os.path.join('Data', 'mel_spectrograms (3 secs)',␣
↪'mel_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)
```

```python
    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
```

```python
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            124s 2s/step -
accuracy: 0.7027 - loss: 0.5824 - val_accuracy: 0.9200 - val_loss: 0.2278 -
learning_rate: 1.0000e-04
Epoch 2/20
50/50            131s 3s/step -
accuracy: 0.9430 - loss: 0.1705 - val_accuracy: 0.9825 - val_loss: 0.0713 -
learning_rate: 1.0000e-04
Epoch 3/20
50/50            134s 3s/step -
accuracy: 0.9658 - loss: 0.0986 - val_accuracy: 0.9950 - val_loss: 0.0344 -
learning_rate: 1.0000e-04
Epoch 4/20
50/50            141s 3s/step -
accuracy: 0.9863 - loss: 0.0443 - val_accuracy: 0.9825 - val_loss: 0.0506 -
learning_rate: 1.0000e-04
Epoch 5/20
50/50            135s 3s/step -
accuracy: 0.9878 - loss: 0.0299 - val_accuracy: 0.9975 - val_loss: 0.0232 -
learning_rate: 1.0000e-04
Epoch 6/20
```

```
50/50            136s 3s/step -
accuracy: 0.9952 - loss: 0.0214 - val_accuracy: 0.9950 - val_loss: 0.0272 -
learning_rate: 1.0000e-04
Epoch 7/20
50/50            135s 3s/step -
accuracy: 0.9991 - loss: 0.0077 - val_accuracy: 0.9850 - val_loss: 0.0360 -
learning_rate: 1.0000e-04
Epoch 8/20
50/50            142s 3s/step -
accuracy: 0.9974 - loss: 0.0110 - val_accuracy: 0.9950 - val_loss: 0.0256 -
learning_rate: 1.0000e-04
Epoch 9/20
50/50            143s 3s/step -
accuracy: 0.9982 - loss: 0.0119 - val_accuracy: 0.9875 - val_loss: 0.0282 -
learning_rate: 5.0000e-05
Epoch 10/20
50/50            132s 3s/step -
accuracy: 0.9988 - loss: 0.0059 - val_accuracy: 0.9850 - val_loss: 0.0349 -
learning_rate: 5.0000e-05
Epoch 11/20
50/50            128s 3s/step -
accuracy: 0.9999 - loss: 0.0036 - val_accuracy: 0.9850 - val_loss: 0.0372 -
learning_rate: 5.0000e-05
Epoch 12/20
50/50            127s 3s/step -
accuracy: 0.9978 - loss: 0.0061 - val_accuracy: 0.9850 - val_loss: 0.0348 -
learning_rate: 2.5000e-05
Epoch 13/20
50/50            144s 3s/step -
accuracy: 0.9978 - loss: 0.0063 - val_accuracy: 0.9975 - val_loss: 0.0187 -
learning_rate: 2.5000e-05
Epoch 14/20
50/50            132s 3s/step -
accuracy: 0.9990 - loss: 0.0053 - val_accuracy: 0.9925 - val_loss: 0.0233 -
learning_rate: 2.5000e-05
Epoch 15/20
50/50            140s 3s/step -
accuracy: 0.9998 - loss: 0.0024 - val_accuracy: 0.9825 - val_loss: 0.0536 -
learning_rate: 2.5000e-05
Epoch 16/20
50/50            116s 2s/step -
accuracy: 0.9961 - loss: 0.0066 - val_accuracy: 0.9925 - val_loss: 0.0248 -
learning_rate: 2.5000e-05
Epoch 17/20
50/50            156s 2s/step -
accuracy: 1.0000 - loss: 0.0020 - val_accuracy: 0.9925 - val_loss: 0.0248 -
learning_rate: 1.2500e-05
Epoch 18/20
```

```
50/50                135s 3s/step -
accuracy: 0.9990 - loss: 0.0038 - val_accuracy: 0.9975 - val_loss: 0.0214 -
learning_rate: 1.2500e-05
Epoch 19/20
50/50                139s 3s/step -
accuracy: 0.9999 - loss: 0.0019 - val_accuracy: 0.9875 - val_loss: 0.0302 -
learning_rate: 1.2500e-05
Epoch 20/20
50/50                96s 2s/step -
accuracy: 0.9984 - loss: 0.0036 - val_accuracy: 0.9850 - val_loss: 0.0355 -
learning_rate: 6.2500e-06
13/13                5s 384ms/step -
accuracy: 0.9729 - loss: 0.0556
Test accuracy: 0.985
```

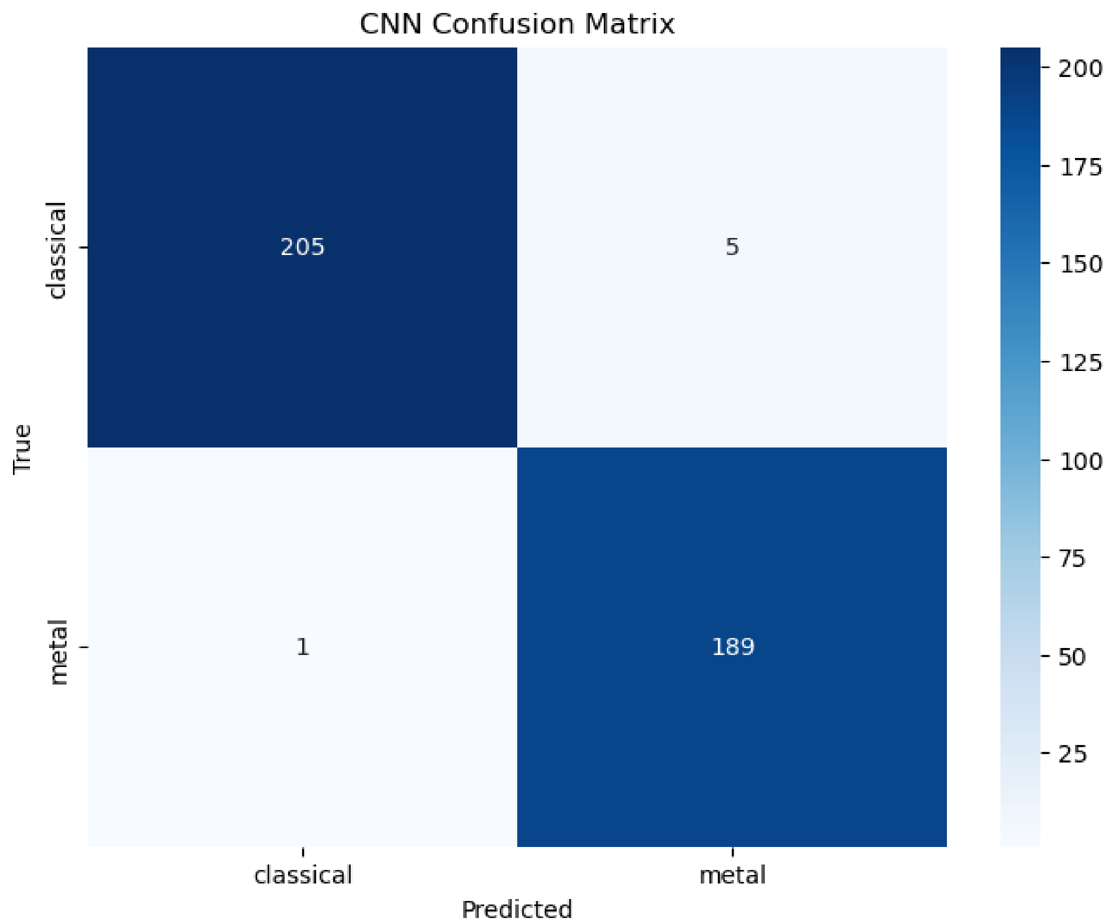
1.4 Confusion Matrix Easy (classical and metal)

[4]:
```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()
```

```
13/13                6s 407ms/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

# 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)
```

```python
    return image

# Define the genres and file paths
GENRES = ['disco', 'pop']
FILE_PATH = os.path.join('Data', 'mel_spectrograms (3 secs)',␣
↪'mel_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
```

```python
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                126s 2s/step -
accuracy: 0.5060 - loss: 0.6966 - val_accuracy: 0.4750 - val_loss: 0.6925 -
learning_rate: 1.0000e-04
Epoch 2/20
50/50                128s 3s/step -
accuracy: 0.5376 - loss: 0.6863 - val_accuracy: 0.7600 - val_loss: 0.5162 -
learning_rate: 1.0000e-04
Epoch 3/20
50/50                141s 3s/step -
accuracy: 0.7613 - loss: 0.5047 - val_accuracy: 0.7600 - val_loss: 0.4415 -
learning_rate: 1.0000e-04
Epoch 4/20
50/50                141s 3s/step -
accuracy: 0.8075 - loss: 0.4014 - val_accuracy: 0.8125 - val_loss: 0.3646 -
learning_rate: 1.0000e-04
Epoch 5/20
50/50                128s 3s/step -
accuracy: 0.8136 - loss: 0.3866 - val_accuracy: 0.8225 - val_loss: 0.3380 -
learning_rate: 1.0000e-04
Epoch 6/20
```

```
50/50            134s 3s/step -
accuracy: 0.8306 - loss: 0.3362 - val_accuracy: 0.8175 - val_loss: 0.3692 -
learning_rate: 1.0000e-04
Epoch 7/20
50/50            136s 3s/step -
accuracy: 0.8570 - loss: 0.3051 - val_accuracy: 0.8175 - val_loss: 0.3869 -
learning_rate: 1.0000e-04
Epoch 8/20
50/50            127s 3s/step -
accuracy: 0.8549 - loss: 0.2874 - val_accuracy: 0.8475 - val_loss: 0.3393 -
learning_rate: 1.0000e-04
Epoch 9/20
50/50            136s 2s/step -
accuracy: 0.8927 - loss: 0.2590 - val_accuracy: 0.8375 - val_loss: 0.3917 -
learning_rate: 5.0000e-05
Epoch 10/20
50/50            146s 3s/step -
accuracy: 0.8995 - loss: 0.2265 - val_accuracy: 0.8475 - val_loss: 0.3881 -
learning_rate: 5.0000e-05
Epoch 11/20
50/50            131s 3s/step -
accuracy: 0.9134 - loss: 0.2085 - val_accuracy: 0.8575 - val_loss: 0.3392 -
learning_rate: 5.0000e-05
Epoch 12/20
50/50            134s 3s/step -
accuracy: 0.8996 - loss: 0.2215 - val_accuracy: 0.8525 - val_loss: 0.3517 -
learning_rate: 2.5000e-05
Epoch 13/20
50/50            136s 3s/step -
accuracy: 0.9240 - loss: 0.1811 - val_accuracy: 0.8700 - val_loss: 0.3138 -
learning_rate: 2.5000e-05
Epoch 14/20
50/50            145s 3s/step -
accuracy: 0.9355 - loss: 0.1831 - val_accuracy: 0.8575 - val_loss: 0.3855 -
learning_rate: 2.5000e-05
Epoch 15/20
50/50            113s 2s/step -
accuracy: 0.9372 - loss: 0.1557 - val_accuracy: 0.8500 - val_loss: 0.3818 -
learning_rate: 2.5000e-05
Epoch 16/20
50/50            94s 2s/step -
accuracy: 0.9312 - loss: 0.1680 - val_accuracy: 0.8550 - val_loss: 0.3907 -
learning_rate: 2.5000e-05
Epoch 17/20
50/50            110s 2s/step -
accuracy: 0.9424 - loss: 0.1472 - val_accuracy: 0.8550 - val_loss: 0.4060 -
learning_rate: 1.2500e-05
Epoch 18/20
```

```
50/50                135s 3s/step -
accuracy: 0.9371 - loss: 0.1623 - val_accuracy: 0.8575 - val_loss: 0.3687 -
learning_rate: 1.2500e-05
Epoch 19/20
50/50                128s 3s/step -
accuracy: 0.9416 - loss: 0.1467 - val_accuracy: 0.8575 - val_loss: 0.4238 -
learning_rate: 1.2500e-05
Epoch 20/20
50/50                98s 2s/step -
accuracy: 0.9401 - loss: 0.1520 - val_accuracy: 0.8550 - val_loss: 0.3949 -
learning_rate: 6.2500e-06
13/13                5s 377ms/step -
accuracy: 0.7550 - loss: 0.6441
Test accuracy: 0.855
```

1.6 Confusion Matrix Hard (disco and pop)

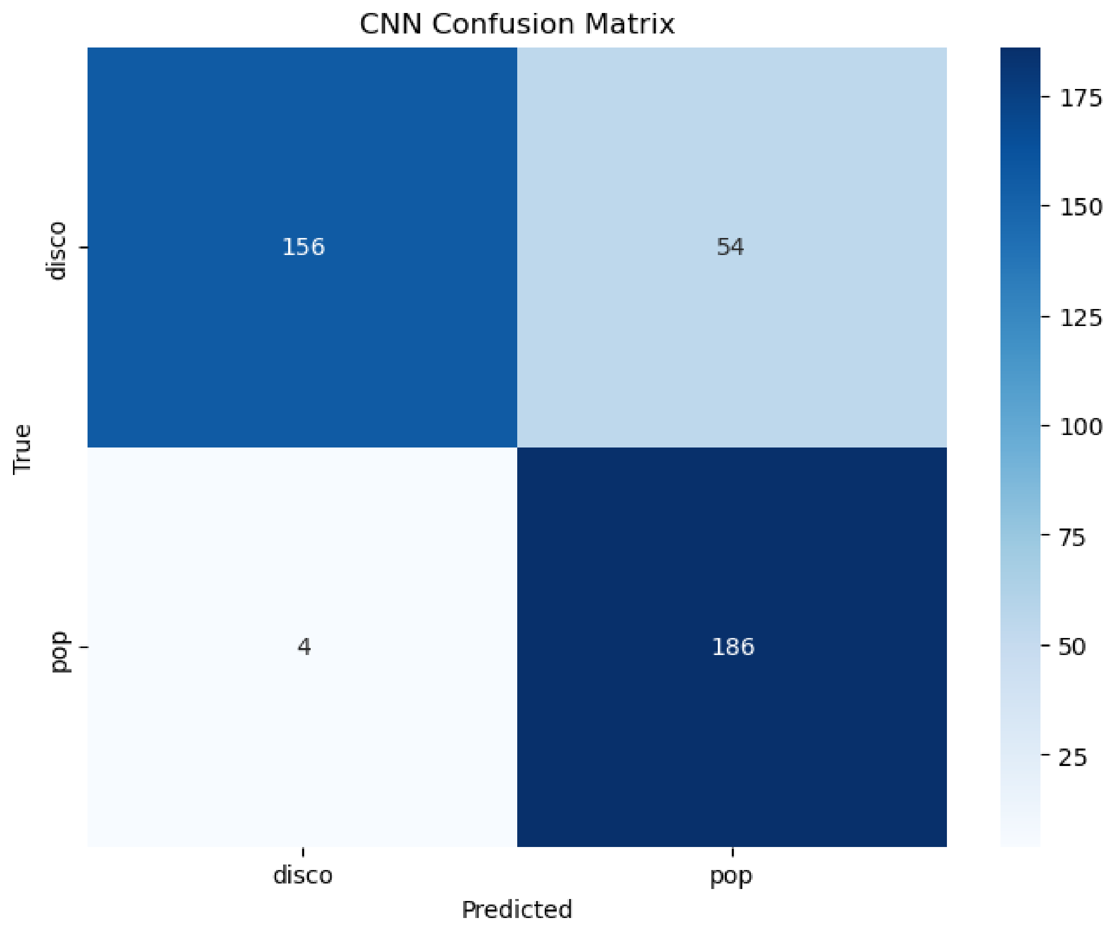
[6]:

```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()
```

```
13/13                6s 435ms/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 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

# Define the genres and file paths
GENRES = ['blues', 'classical', 'country', 'disco', 'hiphop', 'jazz', 'metal',␣
 ↪'pop', 'reggae', 'rock']
GENRES = random.sample(GENRES, 5)
print(GENRES)
FILE_PATH = os.path.join('Data', 'mel_spectrograms (3 secs)',␣
 ↪'mel_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}")
```

```
['disco', 'country', 'metal', 'blues', 'reggae']
Processing genre: disco
Processing genre: country
Processing genre: metal
Processing genre: blues
Processing genre: reggae
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              326s 3s/step -
accuracy: 0.2373 - loss: 1.5844 - val_accuracy: 0.2750 - val_loss: 1.4777 -
learning_rate: 1.0000e-04
Epoch 2/20
125/125              331s 3s/step -
accuracy: 0.3848 - loss: 1.3818 - val_accuracy: 0.4080 - val_loss: 1.3078 -
learning_rate: 1.0000e-04
Epoch 3/20
125/125              334s 3s/step -
accuracy: 0.4703 - loss: 1.1893 - val_accuracy: 0.4060 - val_loss: 1.3604 -
learning_rate: 1.0000e-04
```

```
Epoch 4/20
125/125             334s 3s/step -
accuracy: 0.5132 - loss: 1.1196 - val_accuracy: 0.6080 - val_loss: 0.9877 -
learning_rate: 1.0000e-04
Epoch 5/20
125/125             373s 3s/step -
accuracy: 0.5938 - loss: 0.9629 - val_accuracy: 0.5640 - val_loss: 1.0057 -
learning_rate: 1.0000e-04
Epoch 6/20
125/125             394s 3s/step -
accuracy: 0.6760 - loss: 0.8099 - val_accuracy: 0.6780 - val_loss: 0.8479 -
learning_rate: 1.0000e-04
Epoch 7/20
125/125             369s 3s/step -
accuracy: 0.7075 - loss: 0.7492 - val_accuracy: 0.7250 - val_loss: 0.7566 -
learning_rate: 1.0000e-04
Epoch 8/20
125/125             314s 3s/step -
accuracy: 0.7425 - loss: 0.6760 - val_accuracy: 0.7260 - val_loss: 0.6982 -
learning_rate: 1.0000e-04
Epoch 9/20
125/125             339s 3s/step -
accuracy: 0.7616 - loss: 0.6412 - val_accuracy: 0.7550 - val_loss: 0.6306 -
learning_rate: 1.0000e-04
Epoch 10/20
125/125             331s 3s/step -
accuracy: 0.7901 - loss: 0.5625 - val_accuracy: 0.7530 - val_loss: 0.6632 -
learning_rate: 1.0000e-04
Epoch 11/20
125/125             380s 3s/step -
accuracy: 0.8013 - loss: 0.5578 - val_accuracy: 0.7830 - val_loss: 0.5826 -
learning_rate: 1.0000e-04
Epoch 12/20
125/125             384s 3s/step -
accuracy: 0.8242 - loss: 0.4761 - val_accuracy: 0.7780 - val_loss: 0.6002 -
learning_rate: 1.0000e-04
Epoch 13/20
125/125             331s 3s/step -
accuracy: 0.8575 - loss: 0.4181 - val_accuracy: 0.7610 - val_loss: 0.6754 -
learning_rate: 1.0000e-04
Epoch 14/20
125/125             377s 3s/step -
accuracy: 0.8622 - loss: 0.3909 - val_accuracy: 0.7860 - val_loss: 0.5866 -
learning_rate: 1.0000e-04
Epoch 15/20
125/125             388s 3s/step -
accuracy: 0.8916 - loss: 0.3237 - val_accuracy: 0.8080 - val_loss: 0.5396 -
learning_rate: 5.0000e-05
```

```
Epoch 16/20
125/125              338s 3s/step -
accuracy: 0.8958 - loss: 0.3110 - val_accuracy: 0.7740 - val_loss: 0.6009 -
learning_rate: 5.0000e-05
Epoch 17/20
125/125              314s 2s/step -
accuracy: 0.9001 - loss: 0.2955 - val_accuracy: 0.7970 - val_loss: 0.6126 -
learning_rate: 5.0000e-05
Epoch 18/20
125/125              228s 2s/step -
accuracy: 0.9069 - loss: 0.2740 - val_accuracy: 0.8150 - val_loss: 0.5890 -
learning_rate: 5.0000e-05
Epoch 19/20
125/125              206s 2s/step -
accuracy: 0.9148 - loss: 0.2365 - val_accuracy: 0.8020 - val_loss: 0.6006 -
learning_rate: 2.5000e-05
Epoch 20/20
125/125              140s 1s/step -
accuracy: 0.9197 - loss: 0.2430 - val_accuracy: 0.8160 - val_loss: 0.5940 -
learning_rate: 2.5000e-05
32/32              7s 222ms/step -
accuracy: 0.7766 - loss: 0.6681
Test accuracy: 0.816
```

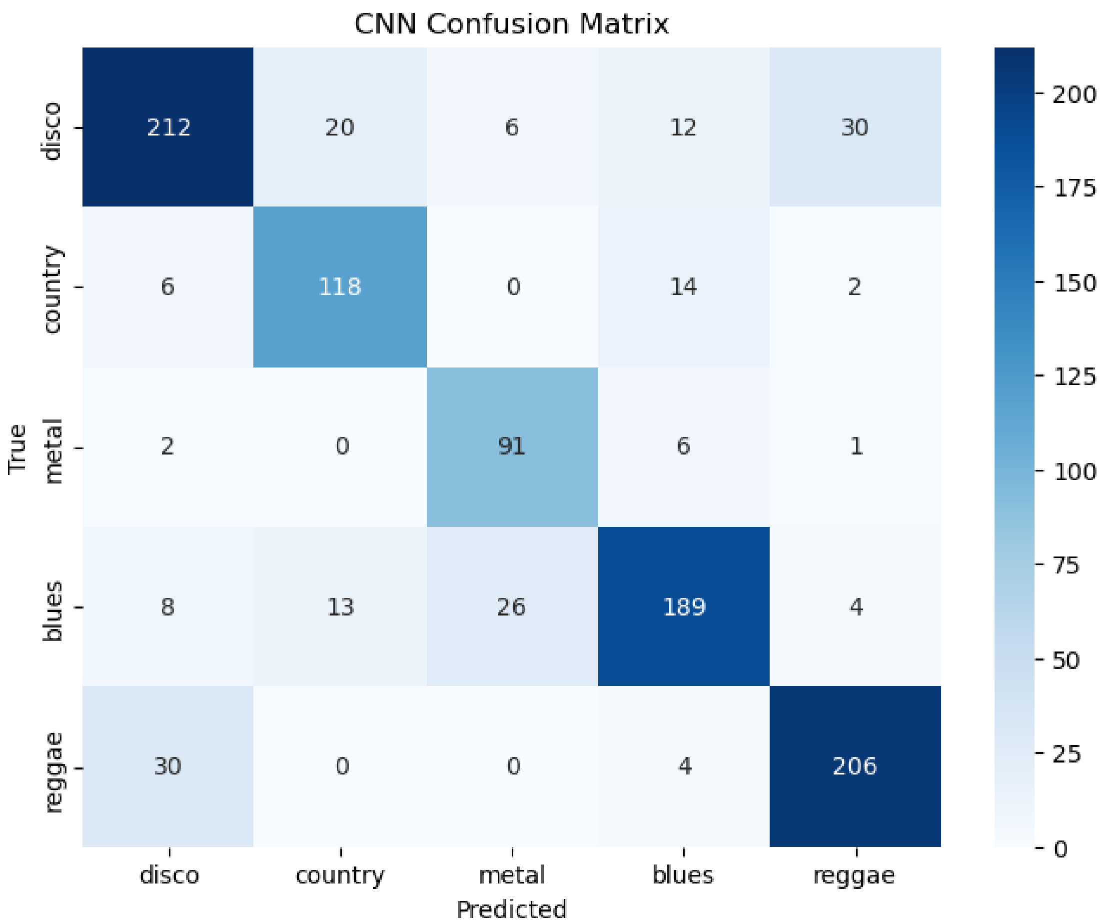
1.8 Confusion Matrix Medium (5 random)

[8]:
```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()
```

```
32/32              8s 228ms/step
```


CNN Confusion Matrix
disco
country
metal
blues
reggae
True
212
20
6
12
30
6
118
0
14
2
2
0
91
6
1
8
13
26
189
4
30
0
0
4
206
disco
country
metal
blues
reggae
Predicted
200
175
150
125
100
75
50
25
0