Mel-Spectrogram-Only (3 secs) CNN

March 23, 2025

1 CNN for Mel-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', 'mel_spectrograms (3 secs)',␣
↪'mel_spectrogram_32')

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}")
```

```
2025-03-23 00:54:43.335860: 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             503s 2s/step -
accuracy: 0.1136 - loss: 2.2968 - val_accuracy: 0.1730 - val_loss: 2.1680 -
learning_rate: 1.0000e-04
Epoch 2/20
250/250             526s 2s/step -
accuracy: 0.2494 - loss: 2.0508 - val_accuracy: 0.3610 - val_loss: 1.7624 -
learning_rate: 1.0000e-04
Epoch 3/20
250/250             561s 2s/step -
accuracy: 0.3535 - loss: 1.7847 - val_accuracy: 0.4580 - val_loss: 1.4814 -
learning_rate: 1.0000e-04
Epoch 4/20
250/250             531s 2s/step -
accuracy: 0.4451 - loss: 1.5529 - val_accuracy: 0.5050 - val_loss: 1.4202 -
learning_rate: 1.0000e-04
Epoch 5/20
250/250             447s 2s/step -
accuracy: 0.4797 - loss: 1.4636 - val_accuracy: 0.5150 - val_loss: 1.3609 -
learning_rate: 1.0000e-04
Epoch 6/20
250/250             417s 2s/step -
accuracy: 0.5204 - loss: 1.3738 - val_accuracy: 0.5445 - val_loss: 1.3133 -
learning_rate: 1.0000e-04
Epoch 7/20
```

```
250/250              446s 2s/step -
accuracy: 0.5410 - loss: 1.2993 - val_accuracy: 0.5370 - val_loss: 1.3499 -
learning_rate: 1.0000e-04
Epoch 8/20
250/250              481s 2s/step -
accuracy: 0.5718 - loss: 1.2408 - val_accuracy: 0.5475 - val_loss: 1.2907 -
learning_rate: 1.0000e-04
Epoch 9/20
250/250              389s 2s/step -
accuracy: 0.5905 - loss: 1.1811 - val_accuracy: 0.5610 - val_loss: 1.2800 -
learning_rate: 1.0000e-04
Epoch 10/20
250/250              339s 1s/step -
accuracy: 0.5961 - loss: 1.1427 - val_accuracy: 0.5910 - val_loss: 1.1903 -
learning_rate: 1.0000e-04
Epoch 11/20
250/250              327s 1s/step -
accuracy: 0.6275 - loss: 1.0863 - val_accuracy: 0.5750 - val_loss: 1.2527 -
learning_rate: 1.0000e-04
Epoch 12/20
250/250              336s 1s/step -
accuracy: 0.6398 - loss: 1.0360 - val_accuracy: 0.5465 - val_loss: 1.3996 -
learning_rate: 1.0000e-04
Epoch 13/20
250/250              337s 1s/step -
accuracy: 0.6605 - loss: 0.9944 - val_accuracy: 0.5940 - val_loss: 1.1862 -
learning_rate: 1.0000e-04
Epoch 14/20
250/250              338s 1s/step -
accuracy: 0.6825 - loss: 0.9164 - val_accuracy: 0.5950 - val_loss: 1.2341 -
learning_rate: 1.0000e-04
Epoch 15/20
250/250              337s 1s/step -
accuracy: 0.6907 - loss: 0.8665 - val_accuracy: 0.5965 - val_loss: 1.1765 -
learning_rate: 1.0000e-04
Epoch 16/20
250/250              337s 1s/step -
accuracy: 0.7158 - loss: 0.8165 - val_accuracy: 0.6210 - val_loss: 1.1514 -
learning_rate: 1.0000e-04
Epoch 17/20
250/250              338s 1s/step -
accuracy: 0.7388 - loss: 0.7608 - val_accuracy: 0.6185 - val_loss: 1.1352 -
learning_rate: 1.0000e-04
Epoch 18/20
250/250              326s 1s/step -
accuracy: 0.7542 - loss: 0.7059 - val_accuracy: 0.6240 - val_loss: 1.1824 -
learning_rate: 1.0000e-04
Epoch 19/20
```

```
250/250              338s 1s/step -
accuracy: 0.7636 - loss: 0.6801 - val_accuracy: 0.6300 - val_loss: 1.1486 -
learning_rate: 1.0000e-04
Epoch 20/20
250/250              338s 1s/step -
accuracy: 0.7939 - loss: 0.6035 - val_accuracy: 0.6290 - val_loss: 1.1801 -
learning_rate: 1.0000e-04
63/63              19s 292ms/step -
accuracy: 0.6723 - loss: 1.0212
Test accuracy: 0.629
```

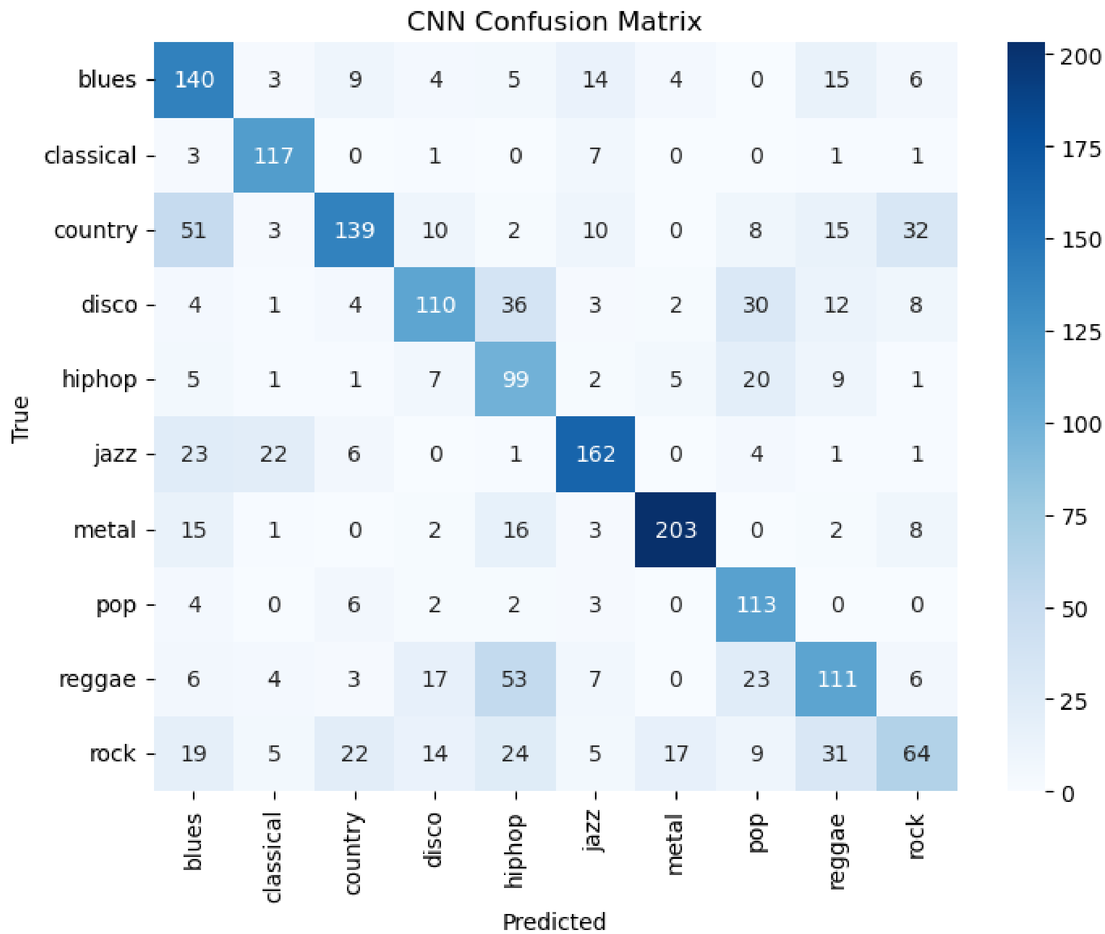
1.2 Apply the confusion matrix after the model

[2]:
```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()
```

```
63/63              20s 314ms/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)
```

```
    return image

# Define the genres and file paths
GENRES = ['classical', 'metal']
FILE_PATH = os.path.join('Data', 'mel_spectrograms (3 secs)',␣
↪'mel_spectrogram_32')

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: 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                 73s 1s/step -
accuracy: 0.6182 - loss: 0.6347 - val_accuracy: 0.8100 - val_loss: 0.3909 -
learning_rate: 1.0000e-04
Epoch 2/20
50/50                 67s 1s/step -
accuracy: 0.8880 - loss: 0.2961 - val_accuracy: 0.9175 - val_loss: 0.2093 -
learning_rate: 1.0000e-04
Epoch 3/20
50/50                 69s 1s/step -
accuracy: 0.9418 - loss: 0.1652 - val_accuracy: 0.9375 - val_loss: 0.1405 -
learning_rate: 1.0000e-04
Epoch 4/20
50/50                 68s 1s/step -
accuracy: 0.9714 - loss: 0.0915 - val_accuracy: 0.9675 - val_loss: 0.0816 -
learning_rate: 1.0000e-04
Epoch 5/20
50/50                 64s 1s/step -
accuracy: 0.9831 - loss: 0.0621 - val_accuracy: 0.9475 - val_loss: 0.1076 -
learning_rate: 1.0000e-04
Epoch 6/20
```

```
50/50              55s 1s/step -
accuracy: 0.9861 - loss: 0.0425 - val_accuracy: 0.9675 - val_loss: 0.0653 -
learning_rate: 1.0000e-04
Epoch 7/20
50/50              68s 1s/step -
accuracy: 0.9885 - loss: 0.0331 - val_accuracy: 0.9725 - val_loss: 0.0599 -
learning_rate: 1.0000e-04
Epoch 8/20
50/50              81s 1s/step -
accuracy: 0.9899 - loss: 0.0261 - val_accuracy: 0.9350 - val_loss: 0.1568 -
learning_rate: 1.0000e-04
Epoch 9/20
50/50              68s 1s/step -
accuracy: 0.9894 - loss: 0.0423 - val_accuracy: 0.9825 - val_loss: 0.0544 -
learning_rate: 1.0000e-04
Epoch 10/20
50/50              82s 1s/step -
accuracy: 0.9964 - loss: 0.0128 - val_accuracy: 0.9800 - val_loss: 0.0494 -
learning_rate: 1.0000e-04
Epoch 11/20
50/50              68s 1s/step -
accuracy: 0.9952 - loss: 0.0182 - val_accuracy: 0.9825 - val_loss: 0.0464 -
learning_rate: 1.0000e-04
Epoch 12/20
50/50              69s 1s/step -
accuracy: 0.9900 - loss: 0.0248 - val_accuracy: 0.9775 - val_loss: 0.0838 -
learning_rate: 1.0000e-04
Epoch 13/20
50/50              67s 1s/step -
accuracy: 0.9918 - loss: 0.0350 - val_accuracy: 0.9900 - val_loss: 0.0286 -
learning_rate: 1.0000e-04
Epoch 14/20
50/50              67s 1s/step -
accuracy: 0.9968 - loss: 0.0171 - val_accuracy: 0.9900 - val_loss: 0.0217 -
learning_rate: 1.0000e-04
Epoch 15/20
50/50              67s 1s/step -
accuracy: 0.9951 - loss: 0.0100 - val_accuracy: 0.9800 - val_loss: 0.0481 -
learning_rate: 1.0000e-04
Epoch 16/20
50/50              67s 1s/step -
accuracy: 0.9943 - loss: 0.0164 - val_accuracy: 0.9900 - val_loss: 0.0398 -
learning_rate: 1.0000e-04
Epoch 17/20
50/50              68s 1s/step -
accuracy: 0.9974 - loss: 0.0071 - val_accuracy: 0.9825 - val_loss: 0.0472 -
learning_rate: 1.0000e-04
Epoch 18/20
```

```
50/50                68s 1s/step -
accuracy: 0.9990 - loss: 0.0057 - val_accuracy: 0.9875 - val_loss: 0.0254 -
learning_rate: 5.0000e-05
Epoch 19/20
50/50                68s 1s/step -
accuracy: 0.9966 - loss: 0.0095 - val_accuracy: 0.9875 - val_loss: 0.0232 -
learning_rate: 5.0000e-05
Epoch 20/20
50/50                67s 1s/step -
accuracy: 0.9957 - loss: 0.0090 - val_accuracy: 0.9800 - val_loss: 0.0566 -
learning_rate: 5.0000e-05
13/13                4s 285ms/step -
accuracy: 0.9904 - loss: 0.0314
Test accuracy: 0.980
```

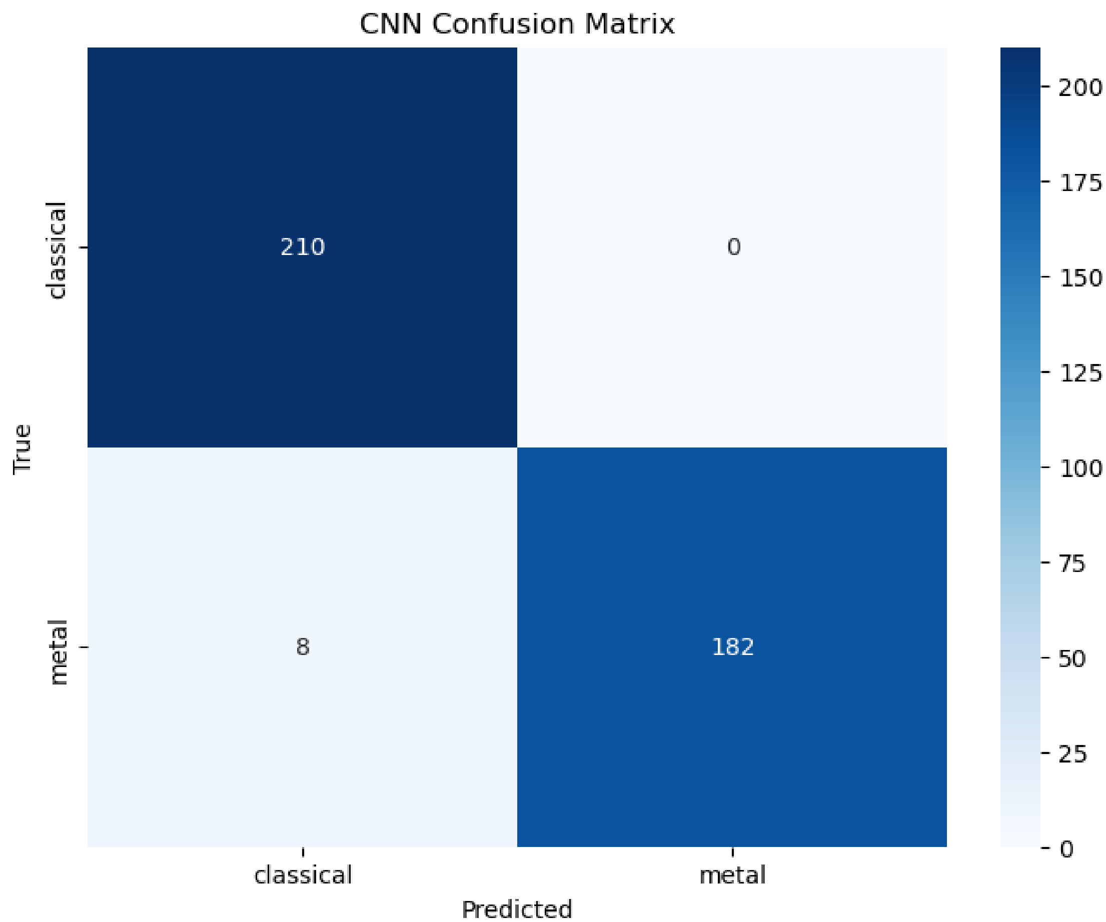
1.4 Confusion Matrix Easy (classical and metal)

[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                4s 305ms/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)
```

```
    return image

# Define the genres and file paths
GENRES = ['disco', 'pop']
FILE_PATH = os.path.join('Data', 'mel_spectrograms (3 secs)',␣
 ↪'mel_spectrogram_32')

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                72s 1s/step -
accuracy: 0.5056 - loss: 0.6929 - val_accuracy: 0.5625 - val_loss: 0.6858 -
learning_rate: 1.0000e-04
Epoch 2/20
50/50                68s 1s/step -
accuracy: 0.5888 - loss: 0.6763 - val_accuracy: 0.6475 - val_loss: 0.6472 -
learning_rate: 1.0000e-04
Epoch 3/20
50/50                68s 1s/step -
accuracy: 0.6916 - loss: 0.5834 - val_accuracy: 0.6800 - val_loss: 0.6024 -
learning_rate: 1.0000e-04
Epoch 4/20
50/50                83s 1s/step -
accuracy: 0.7574 - loss: 0.5227 - val_accuracy: 0.7850 - val_loss: 0.4732 -
learning_rate: 1.0000e-04
Epoch 5/20
50/50                55s 1s/step -
accuracy: 0.8184 - loss: 0.3910 - val_accuracy: 0.7875 - val_loss: 0.4325 -
learning_rate: 1.0000e-04
Epoch 6/20
```

```
50/50              61s 1s/step -
accuracy: 0.8543 - loss: 0.3382 - val_accuracy: 0.8075 - val_loss: 0.3713 -
learning_rate: 1.0000e-04
Epoch 7/20
50/50              66s 1s/step -
accuracy: 0.8734 - loss: 0.2907 - val_accuracy: 0.8475 - val_loss: 0.3267 -
learning_rate: 1.0000e-04
Epoch 8/20
50/50              67s 1s/step -
accuracy: 0.8672 - loss: 0.2826 - val_accuracy: 0.8350 - val_loss: 0.3778 -
learning_rate: 1.0000e-04
Epoch 9/20
50/50              68s 1s/step -
accuracy: 0.9000 - loss: 0.2502 - val_accuracy: 0.8450 - val_loss: 0.3517 -
learning_rate: 1.0000e-04
Epoch 10/20
50/50              66s 1s/step -
accuracy: 0.8934 - loss: 0.2262 - val_accuracy: 0.8425 - val_loss: 0.4106 -
learning_rate: 1.0000e-04
Epoch 11/20
50/50              66s 1s/step -
accuracy: 0.9146 - loss: 0.2146 - val_accuracy: 0.8475 - val_loss: 0.3574 -
learning_rate: 5.0000e-05
Epoch 12/20
50/50              67s 1s/step -
accuracy: 0.9054 - loss: 0.2294 - val_accuracy: 0.8475 - val_loss: 0.3888 -
learning_rate: 5.0000e-05
Epoch 13/20
50/50              67s 1s/step -
accuracy: 0.9226 - loss: 0.1920 - val_accuracy: 0.8525 - val_loss: 0.4068 -
learning_rate: 5.0000e-05
Epoch 14/20
50/50              68s 1s/step -
accuracy: 0.9252 - loss: 0.1753 - val_accuracy: 0.8500 - val_loss: 0.3792 -
learning_rate: 2.5000e-05
Epoch 15/20
50/50              67s 1s/step -
accuracy: 0.9299 - loss: 0.1679 - val_accuracy: 0.8450 - val_loss: 0.4010 -
learning_rate: 2.5000e-05
Epoch 16/20
50/50              67s 1s/step -
accuracy: 0.9316 - loss: 0.1667 - val_accuracy: 0.8450 - val_loss: 0.4144 -
learning_rate: 2.5000e-05
Epoch 17/20
50/50              67s 1s/step -
accuracy: 0.9288 - loss: 0.1851 - val_accuracy: 0.8500 - val_loss: 0.3970 -
learning_rate: 1.2500e-05
Epoch 18/20
```

```
50/50              68s 1s/step -
accuracy: 0.9388 - loss: 0.1666 - val_accuracy: 0.8475 - val_loss: 0.3886 -
learning_rate: 1.2500e-05
Epoch 19/20
50/50              68s 1s/step -
accuracy: 0.9402 - loss: 0.1520 - val_accuracy: 0.8450 - val_loss: 0.3768 -
learning_rate: 1.2500e-05
Epoch 20/20
50/50              67s 1s/step -
accuracy: 0.9545 - loss: 0.1438 - val_accuracy: 0.8475 - val_loss: 0.4142 -
learning_rate: 6.2500e-06
13/13              4s 269ms/step -
accuracy: 0.7524 - loss: 0.7477
Test accuracy: 0.848
```

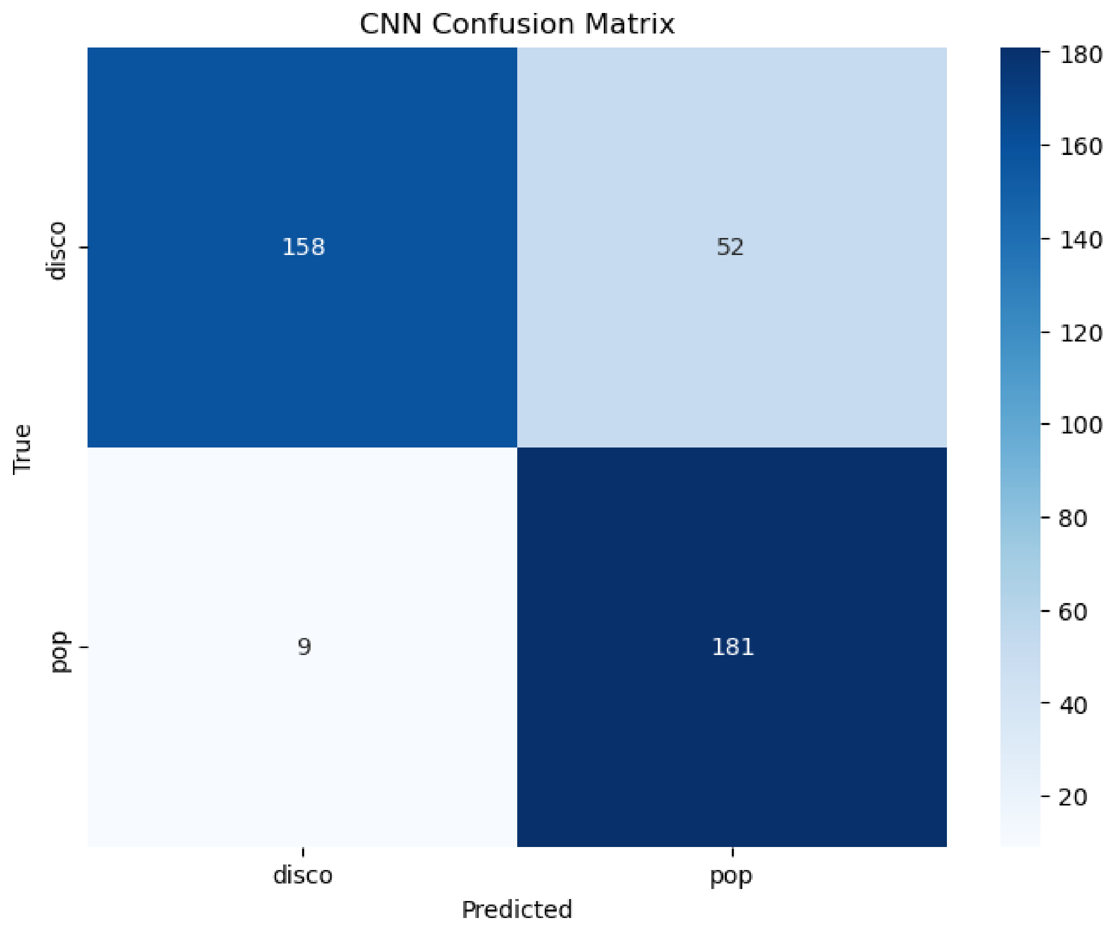
1.6 Confusion Matrix Hard (disco and pop)

[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              5s 330ms/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_32')

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}")
```

```
['reggae', 'jazz', 'blues', 'disco', 'rock']
Processing genre: reggae
Processing genre: jazz
Processing genre: blues
Processing genre: disco
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             169s 1s/step -
accuracy: 0.2147 - loss: 1.6134 - val_accuracy: 0.1400 - val_loss: 1.6476 -
learning_rate: 1.0000e-04
Epoch 2/20
125/125             148s 1s/step -
accuracy: 0.2588 - loss: 1.5801 - val_accuracy: 0.3980 - val_loss: 1.4582 -
learning_rate: 1.0000e-04
Epoch 3/20
125/125             144s 1s/step -
accuracy: 0.3661 - loss: 1.4292 - val_accuracy: 0.5030 - val_loss: 1.2236 -
learning_rate: 1.0000e-04
```

```
Epoch 4/20
125/125              164s 1s/step -
accuracy: 0.4910 - loss: 1.2591 - val_accuracy: 0.5070 - val_loss: 1.1443 -
learning_rate: 1.0000e-04
Epoch 5/20
125/125              157s 1s/step -
accuracy: 0.5168 - loss: 1.1505 - val_accuracy: 0.5690 - val_loss: 1.0259 -
learning_rate: 1.0000e-04
Epoch 6/20
125/125              165s 1s/step -
accuracy: 0.5626 - loss: 1.0732 - val_accuracy: 0.5590 - val_loss: 1.0161 -
learning_rate: 1.0000e-04
Epoch 7/20
125/125              161s 1s/step -
accuracy: 0.5946 - loss: 1.0161 - val_accuracy: 0.6140 - val_loss: 0.9416 -
learning_rate: 1.0000e-04
Epoch 8/20
125/125              207s 1s/step -
accuracy: 0.6157 - loss: 0.9640 - val_accuracy: 0.6440 - val_loss: 0.8905 -
learning_rate: 1.0000e-04
Epoch 9/20
125/125              165s 1s/step -
accuracy: 0.6359 - loss: 0.9347 - val_accuracy: 0.6170 - val_loss: 0.9752 -
learning_rate: 1.0000e-04
Epoch 10/20
125/125              167s 1s/step -
accuracy: 0.6527 - loss: 0.8611 - val_accuracy: 0.6460 - val_loss: 0.9383 -
learning_rate: 1.0000e-04
Epoch 11/20
125/125              165s 1s/step -
accuracy: 0.6735 - loss: 0.8301 - val_accuracy: 0.6220 - val_loss: 0.9369 -
learning_rate: 1.0000e-04
Epoch 12/20
125/125              167s 1s/step -
accuracy: 0.7278 - loss: 0.7252 - val_accuracy: 0.6560 - val_loss: 0.8634 -
learning_rate: 5.0000e-05
Epoch 13/20
125/125              166s 1s/step -
accuracy: 0.7452 - loss: 0.6722 - val_accuracy: 0.6740 - val_loss: 0.8172 -
learning_rate: 5.0000e-05
Epoch 14/20
125/125              165s 1s/step -
accuracy: 0.7700 - loss: 0.6372 - val_accuracy: 0.6600 - val_loss: 0.8364 -
learning_rate: 5.0000e-05
Epoch 15/20
125/125              205s 1s/step -
accuracy: 0.7792 - loss: 0.5974 - val_accuracy: 0.6780 - val_loss: 0.8124 -
learning_rate: 5.0000e-05
```

```
Epoch 16/20
125/125              202s 1s/step -
accuracy: 0.7808 - loss: 0.5813 - val_accuracy: 0.6960 - val_loss: 0.7883 -
learning_rate: 5.0000e-05
Epoch 17/20
125/125              169s 1s/step -
accuracy: 0.7917 - loss: 0.5564 - val_accuracy: 0.6840 - val_loss: 0.8497 -
learning_rate: 5.0000e-05
Epoch 18/20
125/125              167s 1s/step -
accuracy: 0.8141 - loss: 0.5136 - val_accuracy: 0.6910 - val_loss: 0.8106 -
learning_rate: 5.0000e-05
Epoch 19/20
125/125              167s 1s/step -
accuracy: 0.8184 - loss: 0.4979 - val_accuracy: 0.7090 - val_loss: 0.7794 -
learning_rate: 5.0000e-05
Epoch 20/20
125/125              168s 1s/step -
accuracy: 0.8350 - loss: 0.4642 - val_accuracy: 0.7060 - val_loss: 0.8191 -
learning_rate: 5.0000e-05
32/32              9s 290ms/step -
accuracy: 0.7696 - loss: 0.6710
Test accuracy: 0.706
```

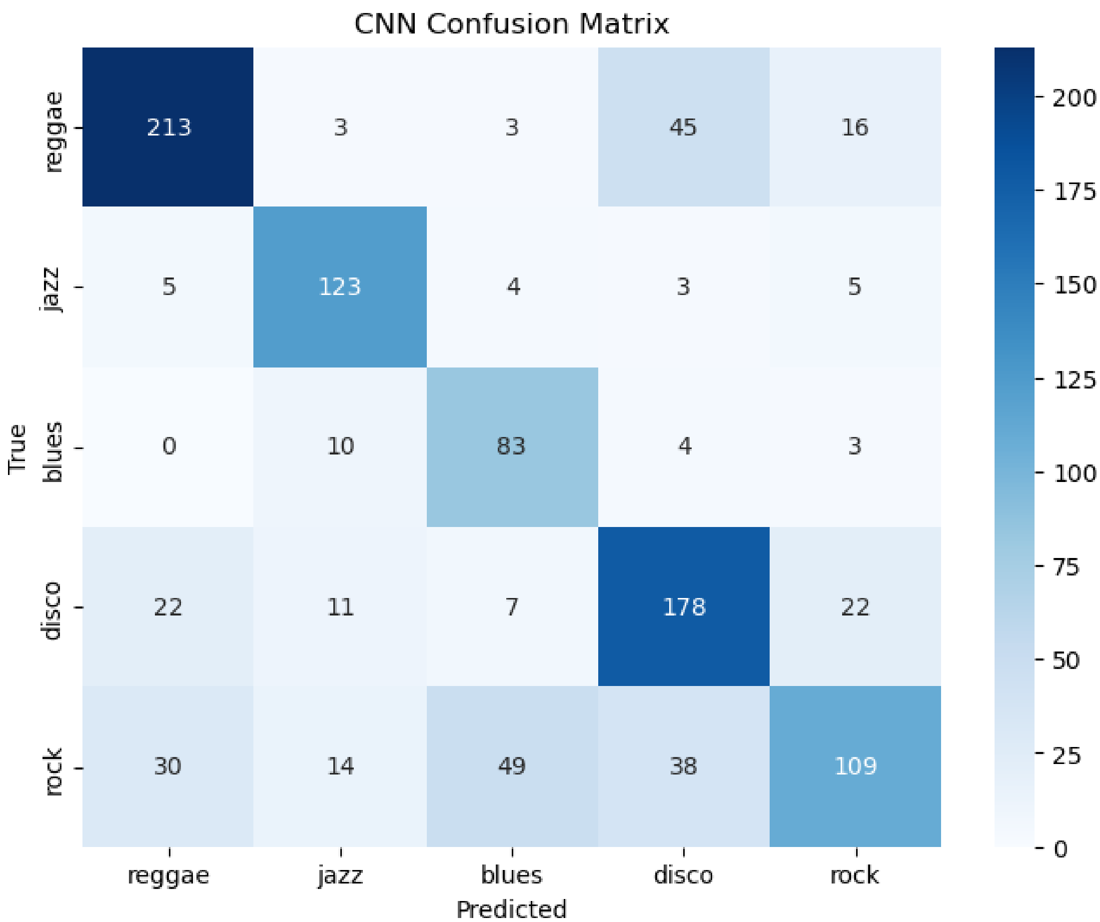
1.8 Confusion Matrix Medium (5 random)

[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 292ms/step
```



CNN Confusion Matrix
True
Predicted
reggae
jazz
blues
disco
rock
213
3
3
45
16
5
123
4
3
5
0
10
83
4
3
22
11
7
178
22
30
14
49
38
109
200
175
150
125
100
75
50
25
0