Onset HeatMap Clips-Only (3 secs) CNN

March 22, 2025

1 CNN for Onset HeatMap Clip Images (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', 'onset_heatmaps (3 secs)')

GENRE_TO_INDEX = {genre: index for index, genre in enumerate(GENRES)}

# Organize data by song ID
song_to_clips = {}

for genre in GENRES:
    genre_dir = os.path.join(FILE_PATH, genre)
    print(f"Processing genre: {genre}")
    for file in os.listdir(genre_dir):
        if not file.endswith(".png"):
```

```
            continue

        song_id = file.split("_clip_")[0]  # Extract song ID (e.g., "blues.
↪00042")

        if song_id not in song_to_clips:
            song_to_clips[song_id] = []

        image = tf.io.read_file(os.path.join(genre_dir, file))
        image = tf.image.decode_png(image, channels=1)
        image = tf.image.convert_image_dtype(image, tf.float32)
        image = tf.image.resize(image, [256, 256])  # Resize to 256x256
        image = augment_image(image)  # Apply augmentation
        image = image.numpy()  # Convert to numpy array

        song_to_clips[song_id].append((image, GENRE_TO_INDEX[genre]))

# Convert dictionary to list format
song_ids = list(song_to_clips.keys())
train_ids, test_ids = train_test_split(song_ids, test_size=0.2, random_state=42)

X_train, y_train, X_test, y_test = [], [], [], []

# Assign clips based on the train-test split
for song_id in song_ids:
    clips = song_to_clips[song_id]
    if song_id in train_ids:
        for image, label in clips:
            X_train.append(image)
            y_train.append(label)
    else:
        for image, label in clips:
            X_test.append(image)
            y_test.append(label)

# Convert to numpy arrays
X_train = np.array(X_train)
y_train = np.array(y_train)
X_test = np.array(X_test)
y_test = np.array(y_test)

print(f"Train set: {len(X_train)} samples")
print(f"Test set: {len(X_test)} samples")

# Define the CNN model
model = models.Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(256, 256, 1)),
```

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

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

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

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

    Flatten(),

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

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

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

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.0001),␣
 ↪loss='sparse_categorical_crossentropy', metrics=['accuracy'])

# Learning rate adjustment
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3,␣
 ↪min_lr=1e-6)

# Train the model
model.fit(X_train, y_train, epochs=20, validation_data=(X_test, y_test),␣
 ↪batch_size=32, callbacks=[reduce_lr])

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

```
2025-03-22 01:34:32.561959: 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              623s 2s/step -
accuracy: 0.1283 - loss: 2.2970 - val_accuracy: 0.1870 - val_loss: 2.1840 -
learning_rate: 1.0000e-04
Epoch 2/20
250/250              820s 3s/step -
accuracy: 0.2007 - loss: 2.1431 - val_accuracy: 0.2205 - val_loss: 2.1061 -
learning_rate: 1.0000e-04
Epoch 3/20
250/250              886s 4s/step -
accuracy: 0.2201 - loss: 2.0877 - val_accuracy: 0.2485 - val_loss: 2.0619 -
learning_rate: 1.0000e-04
Epoch 4/20
250/250              827s 3s/step -
accuracy: 0.2381 - loss: 2.0543 - val_accuracy: 0.2870 - val_loss: 2.0087 -
learning_rate: 1.0000e-04
Epoch 5/20
250/250              852s 3s/step -
accuracy: 0.2866 - loss: 1.9709 - val_accuracy: 0.3175 - val_loss: 1.9437 -
learning_rate: 1.0000e-04
Epoch 6/20
250/250              852s 3s/step -
accuracy: 0.3101 - loss: 1.9245 - val_accuracy: 0.3545 - val_loss: 1.9213 -
learning_rate: 1.0000e-04
Epoch 7/20
250/250              860s 3s/step -
```

```
accuracy: 0.3465 - loss: 1.8533 - val_accuracy: 0.3730 - val_loss: 1.8612 -
learning_rate: 1.0000e-04
Epoch 8/20
250/250            831s 3s/step -
accuracy: 0.3773 - loss: 1.7844 - val_accuracy: 0.3580 - val_loss: 1.8713 -
learning_rate: 1.0000e-04
Epoch 9/20
250/250            821s 3s/step -
accuracy: 0.4117 - loss: 1.6845 - val_accuracy: 0.3755 - val_loss: 1.8405 -
learning_rate: 1.0000e-04
Epoch 10/20
250/250            695s 3s/step -
accuracy: 0.4487 - loss: 1.5756 - val_accuracy: 0.3975 - val_loss: 1.8282 -
learning_rate: 1.0000e-04
Epoch 11/20
250/250            667s 3s/step -
accuracy: 0.4763 - loss: 1.4993 - val_accuracy: 0.3620 - val_loss: 1.8928 -
learning_rate: 1.0000e-04
Epoch 12/20
250/250            689s 3s/step -
accuracy: 0.5062 - loss: 1.3998 - val_accuracy: 0.3675 - val_loss: 1.9349 -
learning_rate: 1.0000e-04
Epoch 13/20
250/250            672s 3s/step -
accuracy: 0.5432 - loss: 1.2983 - val_accuracy: 0.3510 - val_loss: 2.0937 -
learning_rate: 1.0000e-04
Epoch 14/20
250/250            687s 3s/step -
accuracy: 0.5932 - loss: 1.1399 - val_accuracy: 0.3490 - val_loss: 2.1161 -
learning_rate: 5.0000e-05
Epoch 15/20
250/250            675s 3s/step -
accuracy: 0.6430 - loss: 1.0211 - val_accuracy: 0.3525 - val_loss: 2.1776 -
learning_rate: 5.0000e-05
Epoch 16/20
250/250            687s 3s/step -
accuracy: 0.6782 - loss: 0.9563 - val_accuracy: 0.3450 - val_loss: 2.2251 -
learning_rate: 5.0000e-05
Epoch 17/20
250/250            678s 3s/step -
accuracy: 0.7083 - loss: 0.8472 - val_accuracy: 0.3360 - val_loss: 2.4100 -
learning_rate: 2.5000e-05
Epoch 18/20
250/250            672s 3s/step -
accuracy: 0.7243 - loss: 0.7991 - val_accuracy: 0.3395 - val_loss: 2.4176 -
learning_rate: 2.5000e-05
Epoch 19/20
250/250            695s 3s/step -
```

```
accuracy: 0.7355 - loss: 0.7714 - val_accuracy: 0.3495 - val_loss: 2.4411 -
learning_rate: 2.5000e-05
Epoch 20/20
250/250              685s 3s/step -
accuracy: 0.7554 - loss: 0.7095 - val_accuracy: 0.3425 - val_loss: 2.5106 -
learning_rate: 1.2500e-05
63/63              40s 628ms/step -
accuracy: 0.3338 - loss: 2.5294
Test accuracy: 0.343
```

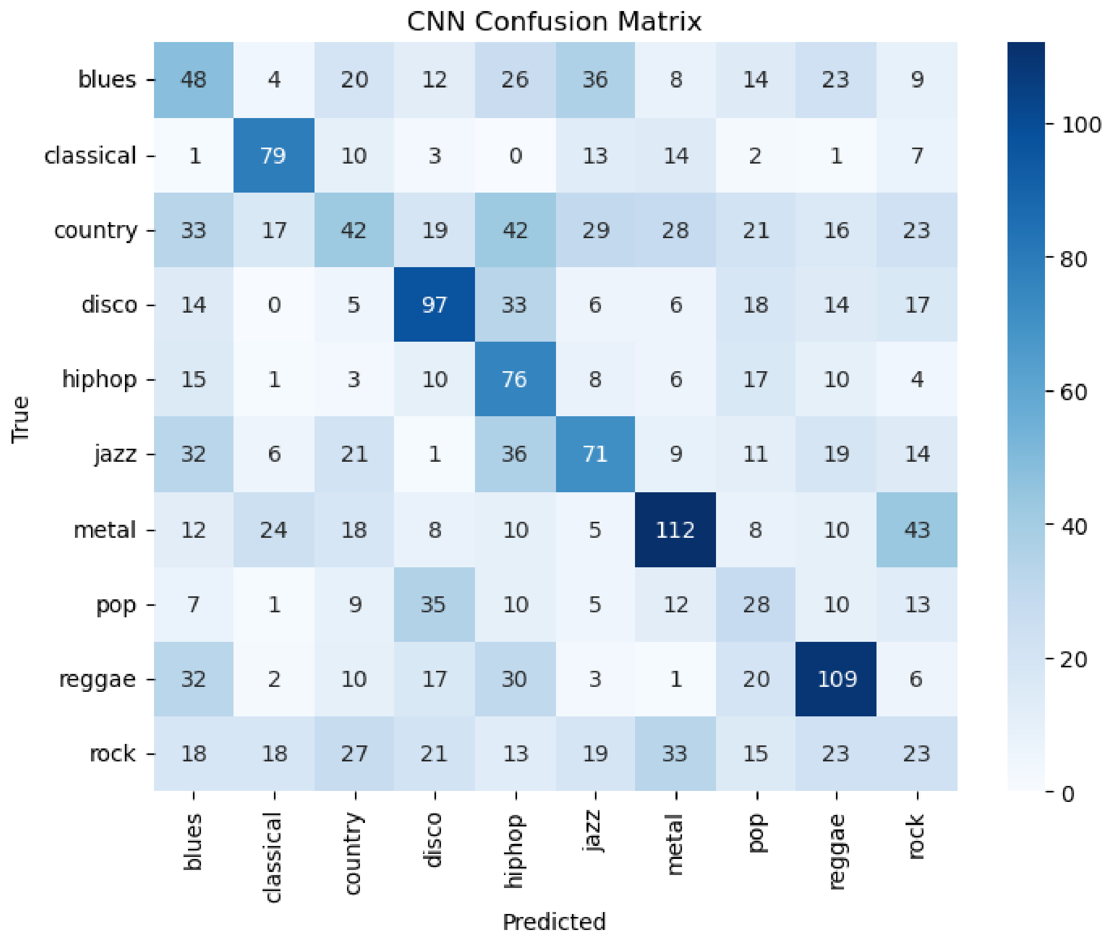
1.2 Apply the confusion matrix after the model

[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              40s 625ms/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', 'onset_heatmaps (3 secs)')

GENRE_TO_INDEX = {genre: index for index, genre in enumerate(GENRES)}

# Organize data by song ID
song_to_clips = {}

for genre in GENRES:
    genre_dir = os.path.join(FILE_PATH, genre)
    print(f"Processing genre: {genre}")
    for file in os.listdir(genre_dir):
        if not file.endswith(".png"):
            continue

        song_id = file.split("_clip_")[0]  # Extract song ID (e.g., "blues.
 ↪00042")

        if song_id not in song_to_clips:
            song_to_clips[song_id] = []

        image = tf.io.read_file(os.path.join(genre_dir, file))
        image = tf.image.decode_png(image, channels=1)
        image = tf.image.convert_image_dtype(image, tf.float32)
        image = tf.image.resize(image, [256, 256])  # Resize to 256x256
        image = augment_image(image)  # Apply augmentation
        image = image.numpy()  # Convert to numpy array

        song_to_clips[song_id].append((image, GENRE_TO_INDEX[genre]))

# Convert dictionary to list format
song_ids = list(song_to_clips.keys())
train_ids, test_ids = train_test_split(song_ids, test_size=0.2, random_state=42)

X_train, y_train, X_test, y_test = [], [], [], []

# Assign clips based on the train-test split
for song_id in song_ids:
    clips = song_to_clips[song_id]
    if song_id in train_ids:
        for image, label in clips:
            X_train.append(image)
            y_train.append(label)
    else:
```

```python
        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'])
```

```python
# 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                 136s 3s/step -
accuracy: 0.5216 - loss: 0.6912 - val_accuracy: 0.5750 - val_loss: 0.6640 -
learning_rate: 1.0000e-04
Epoch 2/20
50/50                 138s 3s/step -
accuracy: 0.6724 - loss: 0.6358 - val_accuracy: 0.6450 - val_loss: 0.6133 -
learning_rate: 1.0000e-04
Epoch 3/20
50/50                 140s 3s/step -
accuracy: 0.6864 - loss: 0.5801 - val_accuracy: 0.6675 - val_loss: 0.5758 -
learning_rate: 1.0000e-04
Epoch 4/20
50/50                 71s 1s/step -
accuracy: 0.7109 - loss: 0.5765 - val_accuracy: 0.7400 - val_loss: 0.5159 -
learning_rate: 1.0000e-04
Epoch 5/20
50/50                 89s 2s/step -
accuracy: 0.7532 - loss: 0.5213 - val_accuracy: 0.7900 - val_loss: 0.4619 -
learning_rate: 1.0000e-04
Epoch 6/20
50/50                 178s 3s/step -
accuracy: 0.7931 - loss: 0.4790 - val_accuracy: 0.7875 - val_loss: 0.4726 -
```

```
learning_rate: 1.0000e-04
Epoch 7/20
50/50            151s 3s/step -
accuracy: 0.8069 - loss: 0.4414 - val_accuracy: 0.7925 - val_loss: 0.4382 -
learning_rate: 1.0000e-04
Epoch 8/20
50/50            136s 3s/step -
accuracy: 0.8119 - loss: 0.4215 - val_accuracy: 0.7475 - val_loss: 0.4905 -
learning_rate: 1.0000e-04
Epoch 9/20
50/50            141s 3s/step -
accuracy: 0.8266 - loss: 0.3998 - val_accuracy: 0.8100 - val_loss: 0.4560 -
learning_rate: 1.0000e-04
Epoch 10/20
50/50            131s 3s/step -
accuracy: 0.8487 - loss: 0.3825 - val_accuracy: 0.8125 - val_loss: 0.4507 -
learning_rate: 1.0000e-04
Epoch 11/20
50/50            142s 3s/step -
accuracy: 0.8588 - loss: 0.3302 - val_accuracy: 0.8150 - val_loss: 0.4172 -
learning_rate: 5.0000e-05
Epoch 12/20
50/50            129s 3s/step -
accuracy: 0.8767 - loss: 0.3155 - val_accuracy: 0.8125 - val_loss: 0.4219 -
learning_rate: 5.0000e-05
Epoch 13/20
50/50            138s 2s/step -
accuracy: 0.8770 - loss: 0.3062 - val_accuracy: 0.8125 - val_loss: 0.4310 -
learning_rate: 5.0000e-05
Epoch 14/20
50/50            134s 3s/step -
accuracy: 0.8902 - loss: 0.2786 - val_accuracy: 0.8100 - val_loss: 0.4465 -
learning_rate: 5.0000e-05
Epoch 15/20
50/50            143s 3s/step -
accuracy: 0.8903 - loss: 0.2759 - val_accuracy: 0.8150 - val_loss: 0.4281 -
learning_rate: 2.5000e-05
Epoch 16/20
50/50            130s 3s/step -
accuracy: 0.8872 - loss: 0.2549 - val_accuracy: 0.8125 - val_loss: 0.4426 -
learning_rate: 2.5000e-05
Epoch 17/20
50/50            140s 3s/step -
accuracy: 0.9041 - loss: 0.2385 - val_accuracy: 0.8125 - val_loss: 0.4373 -
learning_rate: 2.5000e-05
Epoch 18/20
50/50            129s 3s/step -
accuracy: 0.9098 - loss: 0.2323 - val_accuracy: 0.8200 - val_loss: 0.4481 -
```

```
learning_rate: 1.2500e-05
Epoch 19/20
50/50              132s 3s/step -
accuracy: 0.8979 - loss: 0.2466 - val_accuracy: 0.8100 - val_loss: 0.4503 -
learning_rate: 1.2500e-05
Epoch 20/20
50/50              128s 2s/step -
accuracy: 0.9187 - loss: 0.2244 - val_accuracy: 0.8125 - val_loss: 0.4521 -
learning_rate: 1.2500e-05
13/13              8s 593ms/step -
accuracy: 0.7909 - loss: 0.4786
Test accuracy: 0.812
```

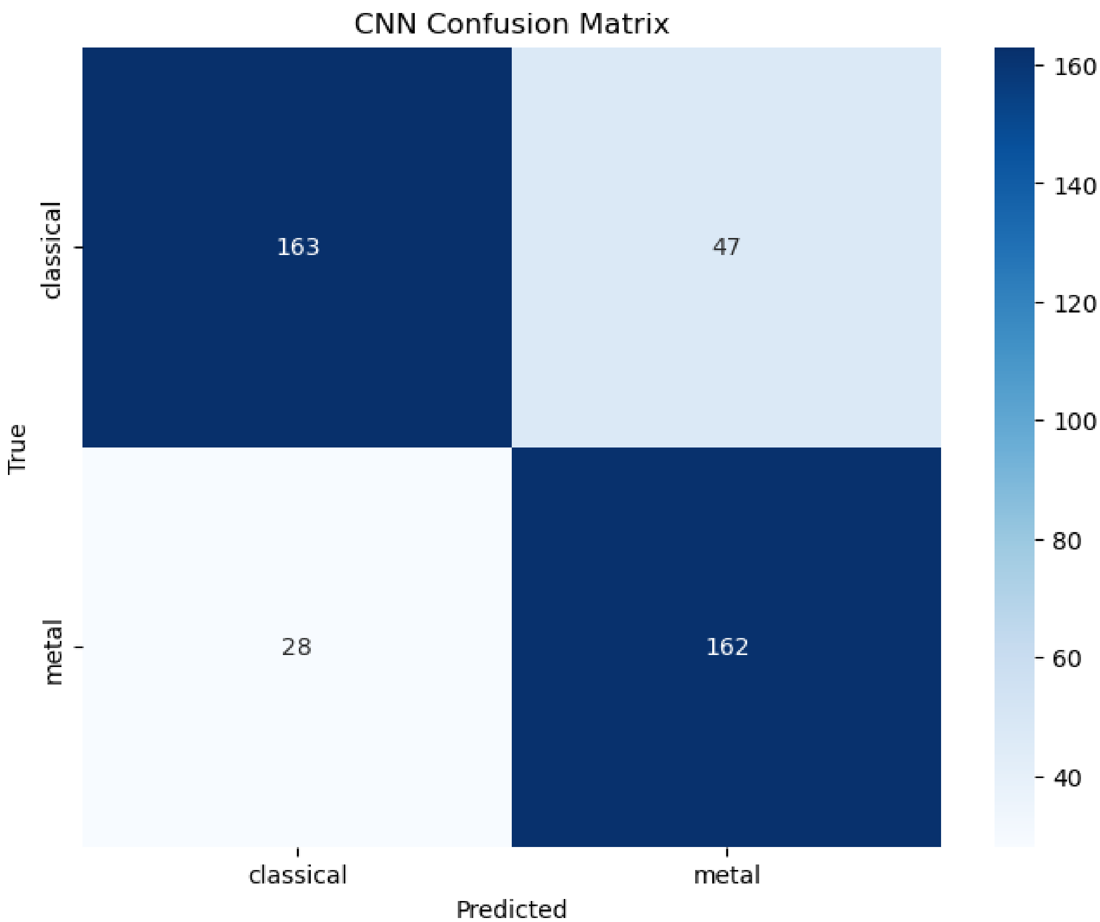
1.4 Confusion Matrix Easy (classical and metal)

[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              8s 609ms/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', 'onset_heatmaps (3 secs)')

GENRE_TO_INDEX = {genre: index for index, genre in enumerate(GENRES)}

# Organize data by song ID
song_to_clips = {}

for genre in GENRES:
    genre_dir = os.path.join(FILE_PATH, genre)
    print(f"Processing genre: {genre}")
    for file in os.listdir(genre_dir):
        if not file.endswith(".png"):
            continue

        song_id = file.split("_clip_")[0]  # Extract song ID (e.g., "blues.
 ↪00042")

        if song_id not in song_to_clips:
            song_to_clips[song_id] = []

        image = tf.io.read_file(os.path.join(genre_dir, file))
        image = tf.image.decode_png(image, channels=1)
        image = tf.image.convert_image_dtype(image, tf.float32)
        image = tf.image.resize(image, [256, 256])  # Resize to 256x256
        image = augment_image(image)  # Apply augmentation
        image = image.numpy()  # Convert to numpy array

        song_to_clips[song_id].append((image, GENRE_TO_INDEX[genre]))

# Convert dictionary to list format
song_ids = list(song_to_clips.keys())
train_ids, test_ids = train_test_split(song_ids, test_size=0.2, random_state=42)

X_train, y_train, X_test, y_test = [], [], [], []

# Assign clips based on the train-test split
for song_id in song_ids:
    clips = song_to_clips[song_id]
    if song_id in train_ids:
        for image, label in clips:
            X_train.append(image)
            y_train.append(label)
    else:
```

```
        for image, label in clips:
            X_test.append(image)
            y_test.append(label)

# Convert to numpy arrays
X_train = np.array(X_train)
y_train = np.array(y_train)
X_test = np.array(X_test)
y_test = np.array(y_test)

print(f"Train set: {len(X_train)} samples")
print(f"Test set: {len(X_test)} samples")

# Define the CNN model
model = models.Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(256, 256, 1)),
    Normalization(),
    MaxPooling2D((2, 2)),

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

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

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

    Flatten(),

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

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

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

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.0001),
 ↪loss='sparse_categorical_crossentropy', metrics=['accuracy'])
```

```python
# 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            113s 2s/step -
accuracy: 0.5213 - loss: 0.6936 - val_accuracy: 0.5350 - val_loss: 0.6916 -
learning_rate: 1.0000e-04
Epoch 2/20
50/50            125s 3s/step -
accuracy: 0.5187 - loss: 0.6935 - val_accuracy: 0.6300 - val_loss: 0.6869 -
learning_rate: 1.0000e-04
Epoch 3/20
50/50            131s 3s/step -
accuracy: 0.5630 - loss: 0.6824 - val_accuracy: 0.6925 - val_loss: 0.6357 -
learning_rate: 1.0000e-04
Epoch 4/20
50/50            113s 2s/step -
accuracy: 0.6308 - loss: 0.6463 - val_accuracy: 0.7075 - val_loss: 0.5872 -
learning_rate: 1.0000e-04
Epoch 5/20
50/50            78s 2s/step -
accuracy: 0.7213 - loss: 0.5650 - val_accuracy: 0.7075 - val_loss: 0.5848 -
learning_rate: 1.0000e-04
Epoch 6/20
50/50            109s 2s/step -
accuracy: 0.7383 - loss: 0.5488 - val_accuracy: 0.6575 - val_loss: 0.6828 -
```

```
learning_rate: 1.0000e-04
Epoch 7/20
50/50          163s 3s/step -
accuracy: 0.7597 - loss: 0.5317 - val_accuracy: 0.7100 - val_loss: 0.6312 -
learning_rate: 1.0000e-04
Epoch 8/20
50/50          143s 3s/step -
accuracy: 0.7648 - loss: 0.5111 - val_accuracy: 0.7525 - val_loss: 0.5666 -
learning_rate: 1.0000e-04
Epoch 9/20
50/50          127s 3s/step -
accuracy: 0.7894 - loss: 0.4674 - val_accuracy: 0.7675 - val_loss: 0.5627 -
learning_rate: 1.0000e-04
Epoch 10/20
50/50          145s 3s/step -
accuracy: 0.8142 - loss: 0.4495 - val_accuracy: 0.7525 - val_loss: 0.5991 -
learning_rate: 1.0000e-04
Epoch 11/20
50/50          129s 3s/step -
accuracy: 0.8056 - loss: 0.4404 - val_accuracy: 0.7200 - val_loss: 0.7150 -
learning_rate: 1.0000e-04
Epoch 12/20
50/50          147s 3s/step -
accuracy: 0.8223 - loss: 0.4266 - val_accuracy: 0.7250 - val_loss: 0.7272 -
learning_rate: 1.0000e-04
Epoch 13/20
50/50          130s 3s/step -
accuracy: 0.8501 - loss: 0.3687 - val_accuracy: 0.7375 - val_loss: 0.7397 -
learning_rate: 5.0000e-05
Epoch 14/20
50/50          124s 2s/step -
accuracy: 0.8612 - loss: 0.3396 - val_accuracy: 0.7350 - val_loss: 0.7400 -
learning_rate: 5.0000e-05
Epoch 15/20
50/50          138s 2s/step -
accuracy: 0.8893 - loss: 0.3004 - val_accuracy: 0.7475 - val_loss: 0.7158 -
learning_rate: 5.0000e-05
Epoch 16/20
50/50          127s 3s/step -
accuracy: 0.8972 - loss: 0.2583 - val_accuracy: 0.7350 - val_loss: 0.7591 -
learning_rate: 2.5000e-05
Epoch 17/20
50/50          129s 3s/step -
accuracy: 0.8990 - loss: 0.2639 - val_accuracy: 0.7325 - val_loss: 0.7861 -
learning_rate: 2.5000e-05
Epoch 18/20
50/50          129s 3s/step -
accuracy: 0.8999 - loss: 0.2499 - val_accuracy: 0.7275 - val_loss: 0.8149 -
```

```
learning_rate: 2.5000e-05
Epoch 19/20
50/50              124s 2s/step -
accuracy: 0.9146 - loss: 0.2057 - val_accuracy: 0.7350 - val_loss: 0.8333 -
learning_rate: 1.2500e-05
Epoch 20/20
50/50              152s 3s/step -
accuracy: 0.9141 - loss: 0.2293 - val_accuracy: 0.7350 - val_loss: 0.8624 -
learning_rate: 1.2500e-05
13/13              7s 546ms/step -
accuracy: 0.6850 - loss: 1.1571
Test accuracy: 0.735
```

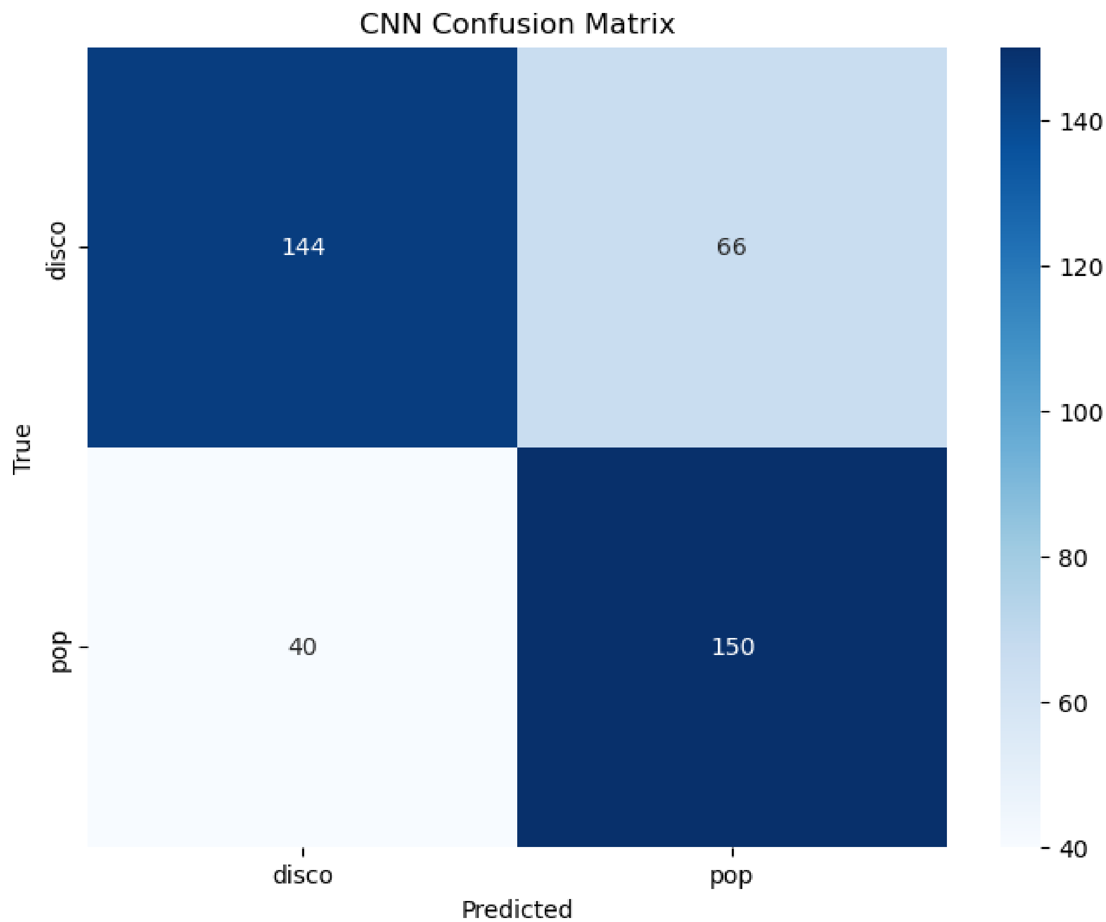
1.6 Confusion Matrix Hard (disco and pop)

[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              9s 686ms/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

GENRES = ['blues', 'classical', 'country', 'disco', 'hiphop', 'jazz', 'metal',␣
↪'pop', 'reggae', 'rock']
GENRES = random.sample(GENRES, 5)
print(GENRES)
FILE_PATH = os.path.join('Data', 'onset_heatmaps (3 secs)')

GENRE_TO_INDEX = {genre: index for index, genre in enumerate(GENRES)}

# Organize data by song ID
song_to_clips = {}

for genre in GENRES:
    genre_dir = os.path.join(FILE_PATH, genre)
    print(f"Processing genre: {genre}")
    for file in os.listdir(genre_dir):
        if not file.endswith(".png"):
            continue

        song_id = file.split("_clip_")[0]  # Extract song ID (e.g., "blues.
↪00042")

        if song_id not in song_to_clips:
            song_to_clips[song_id] = []

        image = tf.io.read_file(os.path.join(genre_dir, file))
        image = tf.image.decode_png(image, channels=1)
        image = tf.image.convert_image_dtype(image, tf.float32)
        image = tf.image.resize(image, [256, 256])  # Resize to 256x256
        image = augment_image(image)  # Apply augmentation
        image = image.numpy()  # Convert to numpy array

        song_to_clips[song_id].append((image, GENRE_TO_INDEX[genre]))

# Convert dictionary to list format
song_ids = list(song_to_clips.keys())
train_ids, test_ids = train_test_split(song_ids, test_size=0.2, random_state=42)

X_train, y_train, X_test, y_test = [], [], [], []

# Assign clips based on the train-test split
for song_id in song_ids:
    clips = song_to_clips[song_id]
    if song_id in train_ids:
        for image, label in clips:
```

```
            X_train.append(image)
            y_train.append(label)
    else:
        for image, label in clips:
            X_test.append(image)
            y_test.append(label)

# Convert to numpy arrays
X_train = np.array(X_train)
y_train = np.array(y_train)
X_test = np.array(X_test)
y_test = np.array(y_test)

print(f"Train set: {len(X_train)} samples")
print(f"Test set: {len(X_test)} samples")

# Define the CNN model
model = models.Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(256, 256, 1)),
    Normalization(),
    MaxPooling2D((2, 2)),

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

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

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

    Flatten(),

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

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

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

```python
# 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}")
```

```
['country', 'jazz', 'hiphop', 'disco', 'reggae']
Processing genre: country
Processing genre: jazz
Processing genre: hiphop
Processing genre: disco
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            330s 3s/step -
accuracy: 0.2076 - loss: 1.6097 - val_accuracy: 0.1000 - val_loss: 1.6511 -
learning_rate: 1.0000e-04
Epoch 2/20
125/125            219s 2s/step -
accuracy: 0.2229 - loss: 1.6037 - val_accuracy: 0.2700 - val_loss: 1.6034 -
learning_rate: 1.0000e-04
Epoch 3/20
125/125            212s 2s/step -
accuracy: 0.2700 - loss: 1.5749 - val_accuracy: 0.2340 - val_loss: 1.5848 -
learning_rate: 1.0000e-04
Epoch 4/20
125/125            375s 3s/step -
accuracy: 0.3787 - loss: 1.4700 - val_accuracy: 0.4320 - val_loss: 1.4167 -
learning_rate: 1.0000e-04
```

```
Epoch 5/20
125/125              330s 3s/step -
accuracy: 0.4567 - loss: 1.3365 - val_accuracy: 0.4390 - val_loss: 1.3384 -
learning_rate: 1.0000e-04
Epoch 6/20
125/125              393s 3s/step -
accuracy: 0.5011 - loss: 1.2533 - val_accuracy: 0.4600 - val_loss: 1.2765 -
learning_rate: 1.0000e-04
Epoch 7/20
125/125              368s 3s/step -
accuracy: 0.5366 - loss: 1.2016 - val_accuracy: 0.5480 - val_loss: 1.1390 -
learning_rate: 1.0000e-04
Epoch 8/20
125/125              378s 3s/step -
accuracy: 0.5409 - loss: 1.1505 - val_accuracy: 0.5170 - val_loss: 1.1822 -
learning_rate: 1.0000e-04
Epoch 9/20
125/125              324s 3s/step -
accuracy: 0.5787 - loss: 1.1323 - val_accuracy: 0.5440 - val_loss: 1.1463 -
learning_rate: 1.0000e-04
Epoch 10/20
125/125              323s 3s/step -
accuracy: 0.6054 - loss: 1.0316 - val_accuracy: 0.5450 - val_loss: 1.1617 -
learning_rate: 1.0000e-04
Epoch 11/20
125/125              385s 3s/step -
accuracy: 0.6421 - loss: 0.9453 - val_accuracy: 0.5080 - val_loss: 1.2160 -
learning_rate: 5.0000e-05
Epoch 12/20
125/125              330s 3s/step -
accuracy: 0.6670 - loss: 0.8826 - val_accuracy: 0.5230 - val_loss: 1.2263 -
learning_rate: 5.0000e-05
Epoch 13/20
125/125              320s 3s/step -
accuracy: 0.7094 - loss: 0.8227 - val_accuracy: 0.5090 - val_loss: 1.2627 -
learning_rate: 5.0000e-05
Epoch 14/20
125/125              320s 3s/step -
accuracy: 0.7318 - loss: 0.7424 - val_accuracy: 0.5450 - val_loss: 1.2526 -
learning_rate: 2.5000e-05
Epoch 15/20
125/125              315s 3s/step -
accuracy: 0.7595 - loss: 0.6870 - val_accuracy: 0.4990 - val_loss: 1.3579 -
learning_rate: 2.5000e-05
Epoch 16/20
125/125              330s 3s/step -
accuracy: 0.7522 - loss: 0.6719 - val_accuracy: 0.5290 - val_loss: 1.3440 -
learning_rate: 2.5000e-05
```

```
Epoch 17/20
125/125              304s 2s/step -
accuracy: 0.7693 - loss: 0.6151 - val_accuracy: 0.5150 - val_loss: 1.3466 -
learning_rate: 1.2500e-05
Epoch 18/20
125/125              346s 3s/step -
accuracy: 0.7845 - loss: 0.6032 - val_accuracy: 0.5160 - val_loss: 1.3903 -
learning_rate: 1.2500e-05
Epoch 19/20
125/125              394s 3s/step -
accuracy: 0.7825 - loss: 0.5935 - val_accuracy: 0.5160 - val_loss: 1.4124 -
learning_rate: 1.2500e-05
Epoch 20/20
125/125              301s 2s/step -
accuracy: 0.7922 - loss: 0.5670 - val_accuracy: 0.5340 - val_loss: 1.3790 -
learning_rate: 6.2500e-06
32/32              15s 484ms/step -
accuracy: 0.4441 - loss: 1.5364
Test accuracy: 0.534
```

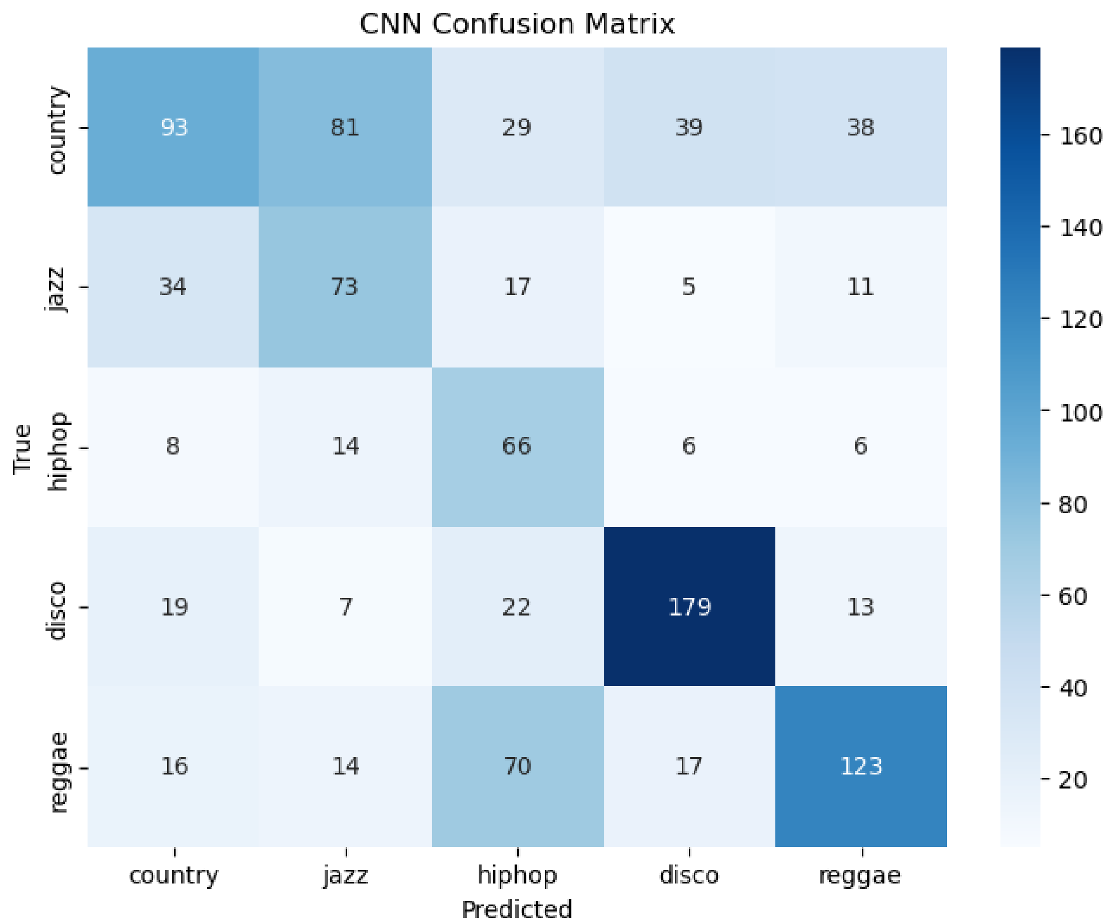
1.8 Confusion Matrix Medium (5 random)

[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              17s 504ms/step
```


CNN Confusion Matrix
country
jazz
hiphop
disco
reggae
True
93
81
29
39
38
34
73
17
5
11
8
14
66
6
6
19
7
22
179
13
16
14
70
17
123
country
jazz
hiphop
disco
reggae
Predicted
160
140
120
100
80
60
40
20