Onset Images-Only (3 secs) CNN

March 22, 2025

1 CNN for Onset Images (3 secs)

1.1 1 - All the imports

[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_images (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 01:34:35.073856: 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             870s 3s/step -
accuracy: 0.1066 - loss: 2.3004 - val_accuracy: 0.0835 - val_loss: 2.2941 -
learning_rate: 1.0000e-04
Epoch 2/20
250/250             859s 3s/step -
accuracy: 0.1531 - loss: 2.2435 - val_accuracy: 0.1240 - val_loss: 2.2314 -
learning_rate: 1.0000e-04
Epoch 3/20
250/250             867s 3s/step -
accuracy: 0.1891 - loss: 2.1783 - val_accuracy: 0.1765 - val_loss: 2.1637 -
learning_rate: 1.0000e-04
Epoch 4/20
250/250             817s 3s/step -
accuracy: 0.2002 - loss: 2.1276 - val_accuracy: 0.1800 - val_loss: 2.1610 -
learning_rate: 1.0000e-04
Epoch 5/20
250/250             883s 3s/step -
accuracy: 0.2045 - loss: 2.1104 - val_accuracy: 0.2085 - val_loss: 2.0995 -
learning_rate: 1.0000e-04
Epoch 6/20
250/250             807s 3s/step -
accuracy: 0.2378 - loss: 2.0561 - val_accuracy: 0.2520 - val_loss: 2.0749 -
learning_rate: 1.0000e-04
Epoch 7/20
250/250             823s 3s/step -
```

```
accuracy: 0.2708 - loss: 2.0054 - val_accuracy: 0.2530 - val_loss: 2.0321 -
learning_rate: 1.0000e-04
Epoch 8/20
250/250            820s 3s/step -
accuracy: 0.2969 - loss: 1.9610 - val_accuracy: 0.2665 - val_loss: 2.0170 -
learning_rate: 1.0000e-04
Epoch 9/20
250/250            708s 3s/step -
accuracy: 0.3152 - loss: 1.9159 - val_accuracy: 0.2850 - val_loss: 1.9843 -
learning_rate: 1.0000e-04
Epoch 10/20
250/250            659s 3s/step -
accuracy: 0.3379 - loss: 1.8570 - val_accuracy: 0.3145 - val_loss: 1.9427 -
learning_rate: 1.0000e-04
Epoch 11/20
250/250            670s 3s/step -
accuracy: 0.3583 - loss: 1.7908 - val_accuracy: 0.3065 - val_loss: 1.9476 -
learning_rate: 1.0000e-04
Epoch 12/20
250/250            691s 3s/step -
accuracy: 0.3880 - loss: 1.7320 - val_accuracy: 0.2990 - val_loss: 1.9530 -
learning_rate: 1.0000e-04
Epoch 13/20
250/250            684s 3s/step -
accuracy: 0.4178 - loss: 1.6386 - val_accuracy: 0.3025 - val_loss: 1.9700 -
learning_rate: 1.0000e-04
Epoch 14/20
250/250            686s 3s/step -
accuracy: 0.4494 - loss: 1.5592 - val_accuracy: 0.3175 - val_loss: 1.9825 -
learning_rate: 5.0000e-05
Epoch 15/20
250/250            734s 3s/step -
accuracy: 0.4957 - loss: 1.4576 - val_accuracy: 0.3185 - val_loss: 2.0186 -
learning_rate: 5.0000e-05
Epoch 16/20
250/250            682s 3s/step -
accuracy: 0.5041 - loss: 1.4130 - val_accuracy: 0.3060 - val_loss: 2.0678 -
learning_rate: 5.0000e-05
Epoch 17/20
250/250            664s 3s/step -
accuracy: 0.5480 - loss: 1.3096 - val_accuracy: 0.3170 - val_loss: 2.1081 -
learning_rate: 2.5000e-05
Epoch 18/20
250/250            676s 3s/step -
accuracy: 0.5599 - loss: 1.2615 - val_accuracy: 0.3045 - val_loss: 2.1244 -
learning_rate: 2.5000e-05
Epoch 19/20
250/250            675s 3s/step -
```

```
accuracy: 0.5741 - loss: 1.2268 - val_accuracy: 0.3130 - val_loss: 2.1490 -
learning_rate: 2.5000e-05
Epoch 20/20
250/250            634s 3s/step -
accuracy: 0.5933 - loss: 1.1557 - val_accuracy: 0.3105 - val_loss: 2.1536 -
learning_rate: 1.2500e-05
63/63            40s 635ms/step -
accuracy: 0.2953 - loss: 2.2385
Test accuracy: 0.310
```

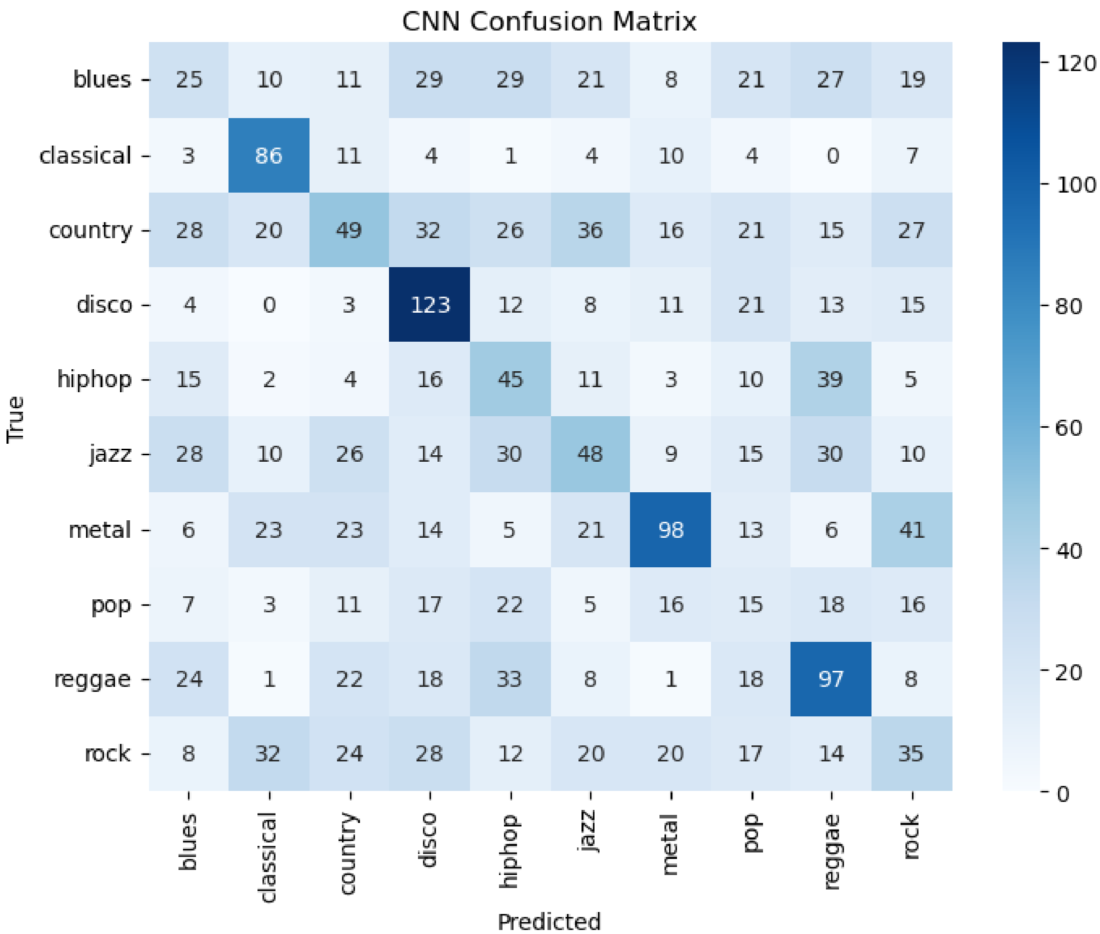
1.2 Apply the confusion matrix after the model

[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            41s 631ms/step
```



1.3 2 - Limited Genres Easy (metal and classical)

[3]:
```
import os
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from tensorflow.keras import models
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
↪Dropout, Normalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau
import matplotlib.pyplot as plt

# Augmentation function
def augment_image(image):
    image = tf.image.random_flip_left_right(image)
    image = tf.image.random_brightness(image, max_delta=0.1)
```

```python
    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_images (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)
```

```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              109s 2s/step -
accuracy: 0.4802 - loss: 0.6957 - val_accuracy: 0.4750 - val_loss: 0.6932 -
learning_rate: 1.0000e-04
Epoch 2/20
50/50              126s 3s/step -
accuracy: 0.5063 - loss: 0.6931 - val_accuracy: 0.5975 - val_loss: 0.6870 -
learning_rate: 1.0000e-04
Epoch 3/20
50/50              130s 3s/step -
accuracy: 0.5623 - loss: 0.6853 - val_accuracy: 0.5625 - val_loss: 0.6879 -
learning_rate: 1.0000e-04
Epoch 4/20
50/50              130s 3s/step -
accuracy: 0.5859 - loss: 0.6780 - val_accuracy: 0.5925 - val_loss: 0.6637 -
learning_rate: 1.0000e-04
Epoch 5/20
50/50              131s 3s/step -
accuracy: 0.6408 - loss: 0.6455 - val_accuracy: 0.5850 - val_loss: 0.6635 -
learning_rate: 1.0000e-04
Epoch 6/20
```

```
50/50            128s 3s/step -
accuracy: 0.6751 - loss: 0.6224 - val_accuracy: 0.6000 - val_loss: 0.6697 -
learning_rate: 1.0000e-04
Epoch 7/20
50/50            132s 3s/step -
accuracy: 0.6811 - loss: 0.6016 - val_accuracy: 0.6625 - val_loss: 0.6085 -
learning_rate: 1.0000e-04
Epoch 8/20
50/50            137s 3s/step -
accuracy: 0.7092 - loss: 0.5526 - val_accuracy: 0.7075 - val_loss: 0.5732 -
learning_rate: 1.0000e-04
Epoch 9/20
50/50            125s 3s/step -
accuracy: 0.7658 - loss: 0.5288 - val_accuracy: 0.7225 - val_loss: 0.5699 -
learning_rate: 1.0000e-04
Epoch 10/20
50/50            149s 3s/step -
accuracy: 0.7622 - loss: 0.4709 - val_accuracy: 0.7400 - val_loss: 0.5450 -
learning_rate: 1.0000e-04
Epoch 11/20
50/50            131s 3s/step -
accuracy: 0.7918 - loss: 0.4658 - val_accuracy: 0.7500 - val_loss: 0.5466 -
learning_rate: 1.0000e-04
Epoch 12/20
50/50            143s 3s/step -
accuracy: 0.7981 - loss: 0.4425 - val_accuracy: 0.7250 - val_loss: 0.5706 -
learning_rate: 1.0000e-04
Epoch 13/20
50/50            125s 3s/step -
accuracy: 0.8347 - loss: 0.4024 - val_accuracy: 0.7325 - val_loss: 0.5869 -
learning_rate: 1.0000e-04
Epoch 14/20
50/50            146s 3s/step -
accuracy: 0.8431 - loss: 0.3955 - val_accuracy: 0.7400 - val_loss: 0.5527 -
learning_rate: 5.0000e-05
Epoch 15/20
50/50            145s 3s/step -
accuracy: 0.8356 - loss: 0.3920 - val_accuracy: 0.7350 - val_loss: 0.5548 -
learning_rate: 5.0000e-05
Epoch 16/20
50/50            126s 3s/step -
accuracy: 0.8573 - loss: 0.3462 - val_accuracy: 0.7450 - val_loss: 0.5599 -
learning_rate: 5.0000e-05
Epoch 17/20
50/50            111s 2s/step -
accuracy: 0.8787 - loss: 0.3161 - val_accuracy: 0.7575 - val_loss: 0.5655 -
learning_rate: 2.5000e-05
Epoch 18/20
```

```
50/50                 104s 2s/step -
accuracy: 0.8727 - loss: 0.3203 - val_accuracy: 0.7625 - val_loss: 0.6119 -
learning_rate: 2.5000e-05
Epoch 19/20
50/50                 170s 3s/step -
accuracy: 0.8929 - loss: 0.2822 - val_accuracy: 0.7625 - val_loss: 0.6092 -
learning_rate: 2.5000e-05
Epoch 20/20
50/50                 130s 3s/step -
accuracy: 0.8613 - loss: 0.3196 - val_accuracy: 0.7575 - val_loss: 0.5733 -
learning_rate: 1.2500e-05
13/13                 8s 623ms/step -
accuracy: 0.7420 - loss: 0.6049
Test accuracy: 0.757
```

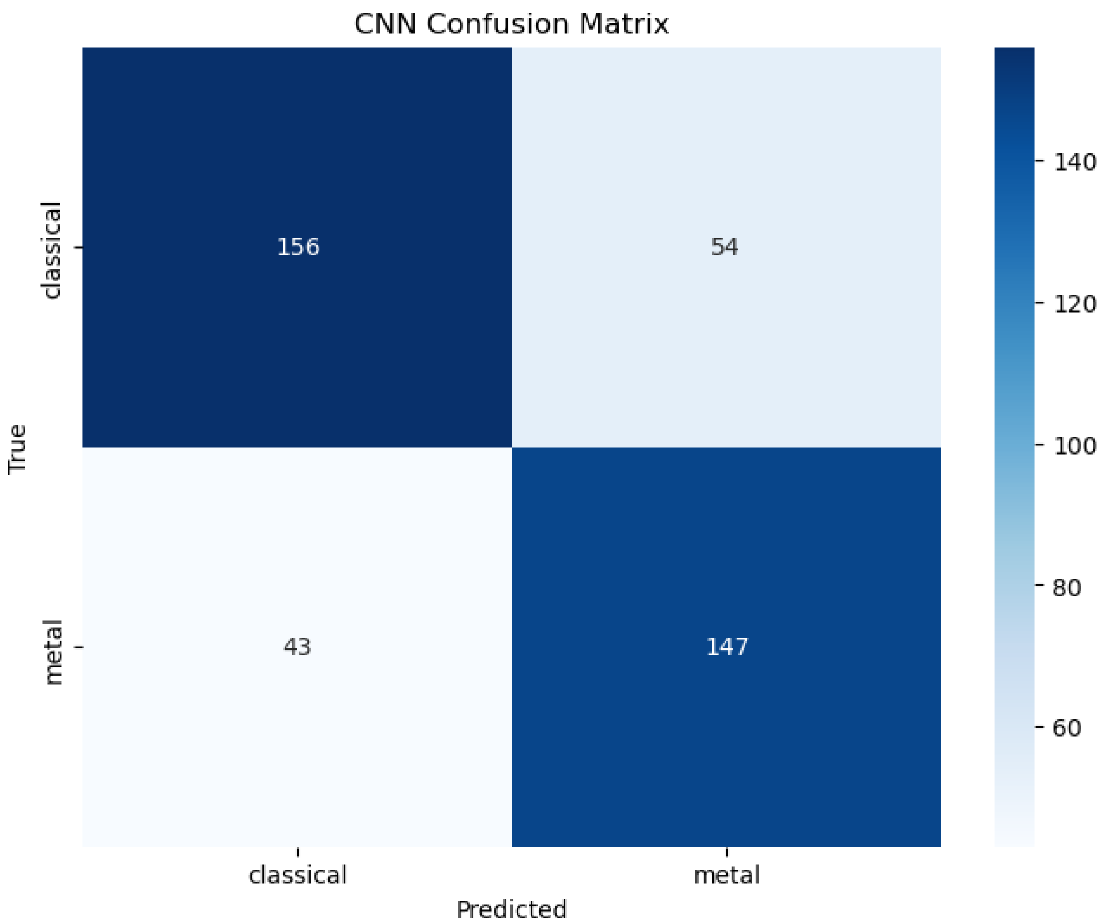
1.4 Confusion Matrix Easy (classical and metal)

[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                 9s 636ms/step
```



1.5 3 - Limited genres Hard (disco and pop)

[5]:
```
import os
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
from tensorflow.keras import models
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
↪Dropout, Normalization
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau
import matplotlib.pyplot as plt

# Augmentation function
def augment_image(image):
    image = tf.image.random_flip_left_right(image)
    image = tf.image.random_brightness(image, max_delta=0.1)
```

```python
    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', 'onset_images (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)
```

```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: 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               114s 2s/step -
accuracy: 0.4937 - loss: 0.6968 - val_accuracy: 0.4750 - val_loss: 0.6956 -
learning_rate: 1.0000e-04
Epoch 2/20
50/50               117s 2s/step -
accuracy: 0.5004 - loss: 0.6937 - val_accuracy: 0.4750 - val_loss: 0.6945 -
learning_rate: 1.0000e-04
Epoch 3/20
50/50               125s 3s/step -
accuracy: 0.5399 - loss: 0.6892 - val_accuracy: 0.4750 - val_loss: 0.7269 -
learning_rate: 1.0000e-04
Epoch 4/20
50/50               122s 2s/step -
accuracy: 0.5216 - loss: 0.6912 - val_accuracy: 0.4775 - val_loss: 0.6946 -
learning_rate: 1.0000e-04
Epoch 5/20
50/50               121s 2s/step -
accuracy: 0.5886 - loss: 0.6755 - val_accuracy: 0.5350 - val_loss: 0.6897 -
learning_rate: 1.0000e-04
Epoch 6/20
```

```
50/50            126s 3s/step -
accuracy: 0.6214 - loss: 0.6550 - val_accuracy: 0.5525 - val_loss: 0.6827 -
learning_rate: 1.0000e-04
Epoch 7/20
50/50            132s 3s/step -
accuracy: 0.6561 - loss: 0.6281 - val_accuracy: 0.5625 - val_loss: 0.7170 -
learning_rate: 1.0000e-04
Epoch 8/20
50/50            129s 3s/step -
accuracy: 0.6722 - loss: 0.6041 - val_accuracy: 0.6200 - val_loss: 0.6898 -
learning_rate: 1.0000e-04
Epoch 9/20
50/50            142s 3s/step -
accuracy: 0.6812 - loss: 0.5854 - val_accuracy: 0.5650 - val_loss: 0.7744 -
learning_rate: 1.0000e-04
Epoch 10/20
50/50            132s 2s/step -
accuracy: 0.7570 - loss: 0.5337 - val_accuracy: 0.6125 - val_loss: 0.7322 -
learning_rate: 5.0000e-05
Epoch 11/20
50/50            146s 2s/step -
accuracy: 0.7399 - loss: 0.5122 - val_accuracy: 0.5700 - val_loss: 0.9103 -
learning_rate: 5.0000e-05
Epoch 12/20
50/50            147s 3s/step -
accuracy: 0.7546 - loss: 0.4935 - val_accuracy: 0.6225 - val_loss: 0.8109 -
learning_rate: 5.0000e-05
Epoch 13/20
50/50            129s 3s/step -
accuracy: 0.7856 - loss: 0.4716 - val_accuracy: 0.6275 - val_loss: 0.8065 -
learning_rate: 2.5000e-05
Epoch 14/20
50/50            127s 3s/step -
accuracy: 0.7937 - loss: 0.4618 - val_accuracy: 0.6450 - val_loss: 0.8170 -
learning_rate: 2.5000e-05
Epoch 15/20
50/50            139s 2s/step -
accuracy: 0.7954 - loss: 0.4433 - val_accuracy: 0.6225 - val_loss: 0.8795 -
learning_rate: 2.5000e-05
Epoch 16/20
50/50            124s 2s/step -
accuracy: 0.8052 - loss: 0.4348 - val_accuracy: 0.6550 - val_loss: 0.8297 -
learning_rate: 1.2500e-05
Epoch 17/20
50/50            90s 2s/step -
accuracy: 0.8333 - loss: 0.4046 - val_accuracy: 0.6625 - val_loss: 0.8288 -
learning_rate: 1.2500e-05
Epoch 18/20
```

```
50/50                 96s 2s/step -
accuracy: 0.8092 - loss: 0.4028 - val_accuracy: 0.6525 - val_loss: 0.8656 -
learning_rate: 1.2500e-05
Epoch 19/20
50/50                 127s 3s/step -
accuracy: 0.8284 - loss: 0.3939 - val_accuracy: 0.6500 - val_loss: 0.8865 -
learning_rate: 6.2500e-06
Epoch 20/20
50/50                 130s 3s/step -
accuracy: 0.8265 - loss: 0.3963 - val_accuracy: 0.6500 - val_loss: 0.8853 -
learning_rate: 6.2500e-06
13/13                 8s 622ms/step -
accuracy: 0.5659 - loss: 1.1148
Test accuracy: 0.650
```

1.6 Confusion Matrix Hard (disco and pop)

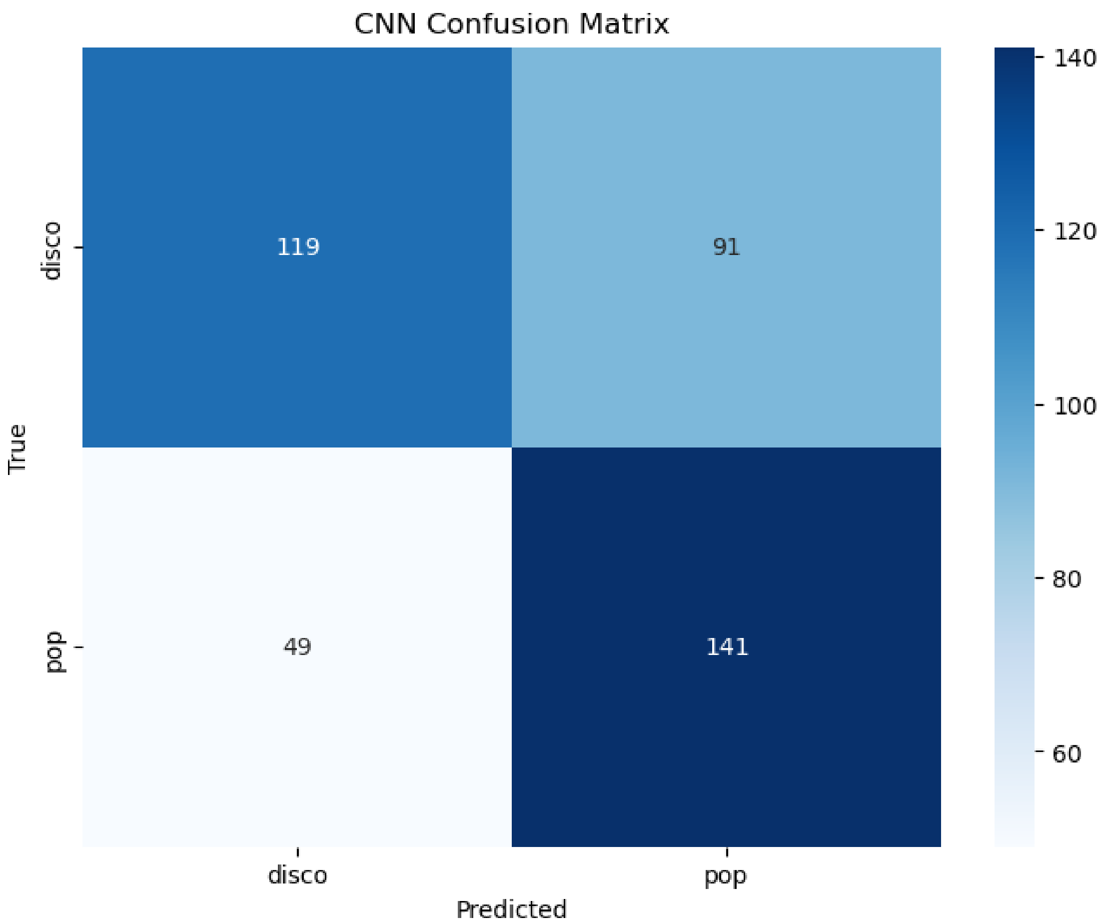
[6]:

```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 653ms/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)
```

```python
    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_images (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:
```

```python
            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}")
```

```
['blues', 'hiphop', 'disco', 'rock', 'classical']
Processing genre: blues
Processing genre: hiphop
Processing genre: disco
Processing genre: rock
Processing genre: classical
Train set: 4000 samples
Test set: 1000 samples
```

```
/opt/conda/lib/python3.12/site-
packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

```
Epoch 1/20
125/125              306s 2s/step -
accuracy: 0.2047 - loss: 1.6093 - val_accuracy: 0.2630 - val_loss: 1.5967 -
learning_rate: 1.0000e-04
Epoch 2/20
125/125              356s 3s/step -
accuracy: 0.2709 - loss: 1.5675 - val_accuracy: 0.2970 - val_loss: 1.4521 -
learning_rate: 1.0000e-04
Epoch 3/20
125/125              316s 3s/step -
accuracy: 0.3057 - loss: 1.5278 - val_accuracy: 0.2950 - val_loss: 1.4032 -
learning_rate: 1.0000e-04
Epoch 4/20
125/125              329s 3s/step -
accuracy: 0.3518 - loss: 1.4622 - val_accuracy: 0.3240 - val_loss: 1.3421 -
learning_rate: 1.0000e-04
```

```
Epoch 5/20
125/125             313s 3s/step -
accuracy: 0.3636 - loss: 1.3936 - val_accuracy: 0.3310 - val_loss: 1.3206 -
learning_rate: 1.0000e-04
Epoch 6/20
125/125             319s 3s/step -
accuracy: 0.3885 - loss: 1.3689 - val_accuracy: 0.3320 - val_loss: 1.3177 -
learning_rate: 1.0000e-04
Epoch 7/20
125/125             325s 3s/step -
accuracy: 0.4218 - loss: 1.3438 - val_accuracy: 0.3860 - val_loss: 1.3076 -
learning_rate: 1.0000e-04
Epoch 8/20
125/125             376s 3s/step -
accuracy: 0.4410 - loss: 1.2997 - val_accuracy: 0.4640 - val_loss: 1.2310 -
learning_rate: 1.0000e-04
Epoch 9/20
125/125             317s 3s/step -
accuracy: 0.4749 - loss: 1.2492 - val_accuracy: 0.4550 - val_loss: 1.2499 -
learning_rate: 1.0000e-04
Epoch 10/20
125/125             331s 3s/step -
accuracy: 0.5109 - loss: 1.2035 - val_accuracy: 0.4460 - val_loss: 1.2492 -
learning_rate: 1.0000e-04
Epoch 11/20
125/125             378s 3s/step -
accuracy: 0.5258 - loss: 1.1490 - val_accuracy: 0.4940 - val_loss: 1.2168 -
learning_rate: 1.0000e-04
Epoch 12/20
125/125             365s 2s/step -
accuracy: 0.5607 - loss: 1.1089 - val_accuracy: 0.4540 - val_loss: 1.2803 -
learning_rate: 1.0000e-04
Epoch 13/20
125/125             333s 3s/step -
accuracy: 0.5868 - loss: 1.0464 - val_accuracy: 0.4620 - val_loss: 1.2714 -
learning_rate: 1.0000e-04
Epoch 14/20
125/125             362s 3s/step -
accuracy: 0.6048 - loss: 1.0094 - val_accuracy: 0.4810 - val_loss: 1.2923 -
learning_rate: 1.0000e-04
Epoch 15/20
125/125             329s 3s/step -
accuracy: 0.6478 - loss: 0.9177 - val_accuracy: 0.4820 - val_loss: 1.3100 -
learning_rate: 5.0000e-05
Epoch 16/20
125/125             315s 3s/step -
accuracy: 0.6808 - loss: 0.8540 - val_accuracy: 0.4840 - val_loss: 1.3238 -
learning_rate: 5.0000e-05
```

```
Epoch 17/20
125/125              330s 3s/step -
accuracy: 0.6888 - loss: 0.8110 - val_accuracy: 0.4830 - val_loss: 1.3420 -
learning_rate: 5.0000e-05
Epoch 18/20
125/125              286s 2s/step -
accuracy: 0.7113 - loss: 0.7654 - val_accuracy: 0.4880 - val_loss: 1.3851 -
learning_rate: 2.5000e-05
Epoch 19/20
125/125              217s 2s/step -
accuracy: 0.7304 - loss: 0.7180 - val_accuracy: 0.4880 - val_loss: 1.4154 -
learning_rate: 2.5000e-05
Epoch 20/20
125/125              219s 2s/step -
accuracy: 0.7287 - loss: 0.6931 - val_accuracy: 0.4950 - val_loss: 1.4499 -
learning_rate: 2.5000e-05
32/32              12s 379ms/step -
accuracy: 0.4091 - loss: 1.6126
Test accuracy: 0.495
```

1.8 Confusion Matrix Medium (5 random)

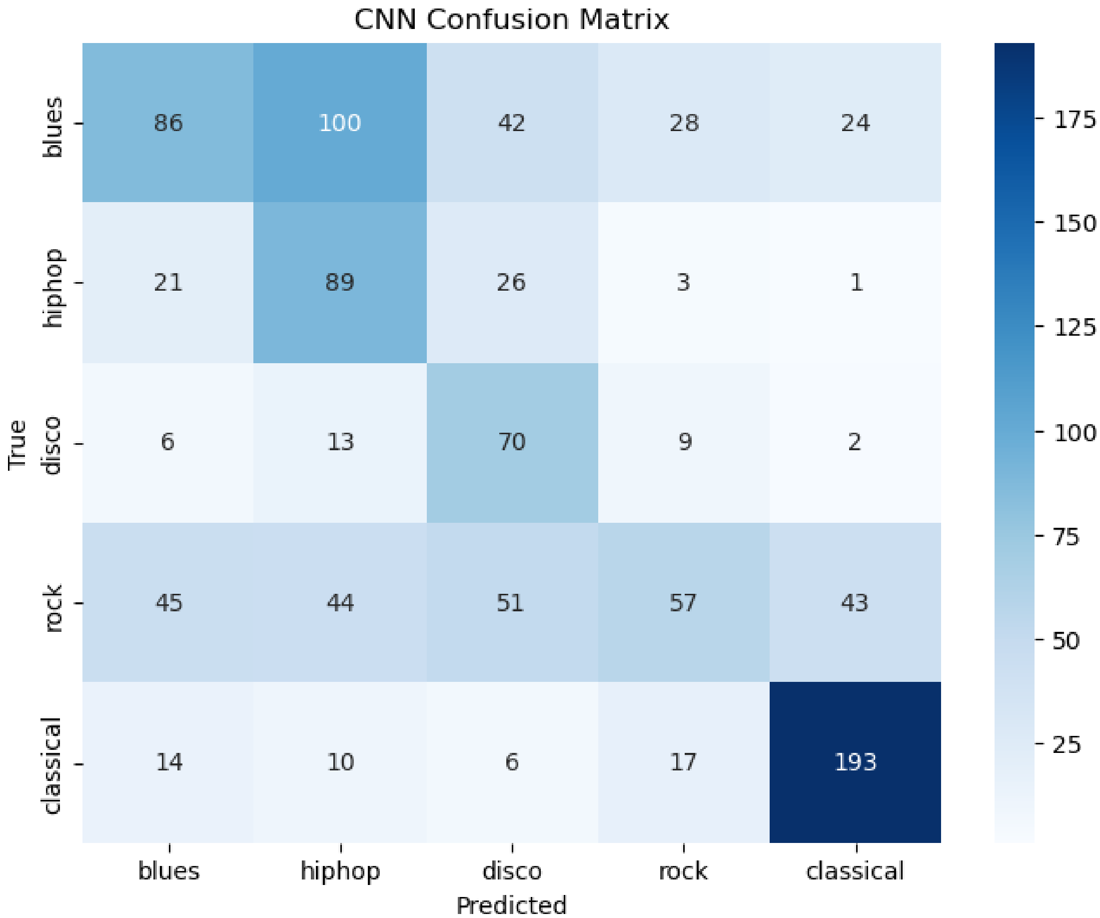
[8]:

```
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              13s 397ms/step
```


CNN Confusion Matrix
blues
hiphop
disco
rock
classical
True
Predicted
86
100
42
28
24
21
89
26
3
1
6
13
70
9
2
45
44
51
57
43
14
10
6
17
193
175
150
125
100
75
50
25