CNN for Spectrogram (3 secs)

1 - All 10

In [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, Dropou
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', '
FILE_PATH = os.path.join('Data', 'spectrograms (3 secs)', 'spectrogram_256')

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 ge
])

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.0001), loss='sparse_categorical_cro

# Learning rate adjustment
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3, min_lr

# Train the model
```

```python
model.fit(X_train, y_train, epochs=20, validation_data=(X_test, y_test), batch_s

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

```
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
```

```
c:\Users\berna\AppData\Local\Programs\Python\Python39\lib\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 ━━━━━━━━━━━━━━━━━━━━ 400s 2s/step - accuracy: 0.1356 - loss: 2.2691 - val
_accuracy: 0.2785 - val_loss: 1.9863 - learning_rate: 1.0000e-04
Epoch 2/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 380s 2s/step - accuracy: 0.3081 - loss: 1.9127 - val
_accuracy: 0.3350 - val_loss: 1.7063 - learning_rate: 1.0000e-04
Epoch 3/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 380s 2s/step - accuracy: 0.3830 - loss: 1.6695 - val
_accuracy: 0.4505 - val_loss: 1.4569 - learning_rate: 1.0000e-04
Epoch 4/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 378s 2s/step - accuracy: 0.4267 - loss: 1.5358 - val
_accuracy: 0.4650 - val_loss: 1.4106 - learning_rate: 1.0000e-04
Epoch 5/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 375s 2s/step - accuracy: 0.4729 - loss: 1.4234 - val
_accuracy: 0.4775 - val_loss: 1.3796 - learning_rate: 1.0000e-04
Epoch 6/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 285s 1s/step - accuracy: 0.5053 - loss: 1.3498 - val
_accuracy: 0.5190 - val_loss: 1.3365 - learning_rate: 1.0000e-04
Epoch 7/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 370s 1s/step - accuracy: 0.5309 - loss: 1.2962 - val
_accuracy: 0.5425 - val_loss: 1.2698 - learning_rate: 1.0000e-04
Epoch 8/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 352s 1s/step - accuracy: 0.5632 - loss: 1.2256 - val
_accuracy: 0.5475 - val_loss: 1.2784 - learning_rate: 1.0000e-04
Epoch 9/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 373s 1s/step - accuracy: 0.5877 - loss: 1.1599 - val
_accuracy: 0.5725 - val_loss: 1.2130 - learning_rate: 1.0000e-04
Epoch 10/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 346s 1s/step - accuracy: 0.5952 - loss: 1.1405 - val
_accuracy: 0.5310 - val_loss: 1.3173 - learning_rate: 1.0000e-04
Epoch 11/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 312s 1s/step - accuracy: 0.6199 - loss: 1.0907 - val
_accuracy: 0.5670 - val_loss: 1.2213 - learning_rate: 1.0000e-04
Epoch 12/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 372s 1s/step - accuracy: 0.6536 - loss: 1.0075 - val
_accuracy: 0.5850 - val_loss: 1.2293 - learning_rate: 1.0000e-04
Epoch 13/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 373s 1s/step - accuracy: 0.6535 - loss: 0.9748 - val
_accuracy: 0.5880 - val_loss: 1.1926 - learning_rate: 5.0000e-05
Epoch 14/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 371s 1s/step - accuracy: 0.6872 - loss: 0.9265 - val
_accuracy: 0.6360 - val_loss: 1.1117 - learning_rate: 5.0000e-05
Epoch 15/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 300s 1s/step - accuracy: 0.6881 - loss: 0.8893 - val
_accuracy: 0.6375 - val_loss: 1.0748 - learning_rate: 5.0000e-05
Epoch 16/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 281s 1s/step - accuracy: 0.7038 - loss: 0.8607 - val
_accuracy: 0.6190 - val_loss: 1.1446 - learning_rate: 5.0000e-05
Epoch 17/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 379s 2s/step - accuracy: 0.7226 - loss: 0.8142 - val
_accuracy: 0.6335 - val_loss: 1.0944 - learning_rate: 5.0000e-05
Epoch 18/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 377s 2s/step - accuracy: 0.7220 - loss: 0.8106 - val
_accuracy: 0.6435 - val_loss: 1.0509 - learning_rate: 5.0000e-05
Epoch 19/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 358s 1s/step - accuracy: 0.7255 - loss: 0.7839 - val
_accuracy: 0.6515 - val_loss: 1.0210 - learning_rate: 5.0000e-05
Epoch 20/20
250/250 ━━━━━━━━━━━━━━━━━━━━ 385s 2s/step - accuracy: 0.7399 - loss: 0.7688 - val
_accuracy: 0.6490 - val_loss: 1.0756 - learning_rate: 5.0000e-05
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 26s 414ms/step - accuracy: 0.7325 - loss: 0.8718
Test accuracy: 0.649
```

Apply the confusion matrix after the model

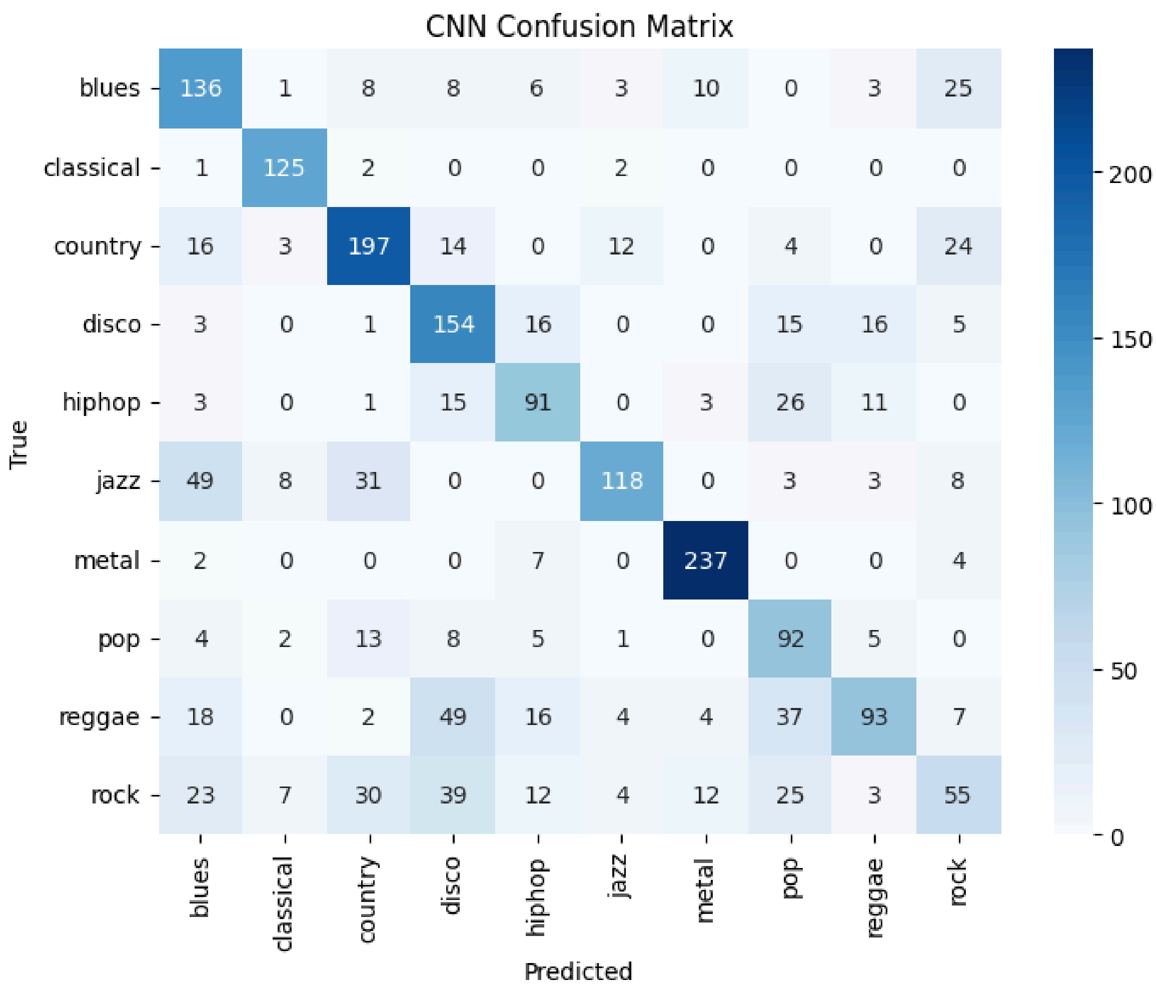
In [2]:

```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, ytick
plt.title("CNN Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()
```

```
63/63 ━━━━━━━━━━━━━━━━━━━━ 26s 402ms/step
```



2 - Limited Genres Easy (metal and classical)

In [3]:

```python
import os
import numpy as np
import tensorflow as tf
```

```python
from sklearn.model_selection import train_test_split
from tensorflow.keras import models
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropou
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau
import matplotlib.pyplot as plt

# Augmentation function
def augment_image(image):
    image = tf.image.random_flip_left_right(image)
    image = tf.image.random_brightness(image, max_delta=0.1)
    image = tf.image.random_contrast(image, 0.8, 1.2)
    return image

# Define the genres and file paths
GENRES = ['classical', 'metal']
FILE_PATH = os.path.join('Data', 'spectrograms (3 secs)', 'spectrogram_256')

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 ge
])

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.0001), loss='sparse_categorical_cro

# Learning rate adjustment
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3, min_lr

# Train the model
model.fit(X_train, y_train, epochs=20, validation_data=(X_test, y_test), batch_s

# 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
```

```
c:\Users\berna\AppData\Local\Programs\Python\Python39\lib\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 ━━━━━━━━━━━━━━━━━━━━ 78s 1s/step - accuracy: 0.6563 - loss: 0.6013 - val_ac
curacy: 0.9550 - val_loss: 0.1258 - learning_rate: 1.0000e-04
Epoch 2/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 83s 1s/step - accuracy: 0.9321 - loss: 0.1808 - val_ac
curacy: 0.9950 - val_loss: 0.0143 - learning_rate: 1.0000e-04
Epoch 3/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 73s 1s/step - accuracy: 0.9749 - loss: 0.0833 - val_ac
curacy: 0.9950 - val_loss: 0.0201 - learning_rate: 1.0000e-04
Epoch 4/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 71s 1s/step - accuracy: 0.9847 - loss: 0.0633 - val_ac
curacy: 1.0000 - val_loss: 0.0034 - learning_rate: 1.0000e-04
Epoch 5/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 73s 1s/step - accuracy: 0.9760 - loss: 0.0711 - val_ac
curacy: 1.0000 - val_loss: 0.0022 - learning_rate: 1.0000e-04
Epoch 6/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 72s 1s/step - accuracy: 0.9942 - loss: 0.0357 - val_ac
curacy: 1.0000 - val_loss: 0.0021 - learning_rate: 1.0000e-04
Epoch 7/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 73s 1s/step - accuracy: 0.9943 - loss: 0.0201 - val_ac
curacy: 1.0000 - val_loss: 0.0015 - learning_rate: 1.0000e-04
Epoch 8/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 73s 1s/step - accuracy: 0.9890 - loss: 0.0342 - val_ac
curacy: 1.0000 - val_loss: 0.0023 - learning_rate: 1.0000e-04
Epoch 9/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 72s 1s/step - accuracy: 0.9925 - loss: 0.0273 - val_ac
curacy: 1.0000 - val_loss: 0.0011 - learning_rate: 1.0000e-04
Epoch 10/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 73s 1s/step - accuracy: 0.9937 - loss: 0.0230 - val_ac
curacy: 1.0000 - val_loss: 7.0509e-04 - learning_rate: 1.0000e-04
Epoch 11/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 73s 1s/step - accuracy: 0.9977 - loss: 0.0096 - val_ac
curacy: 1.0000 - val_loss: 3.4998e-04 - learning_rate: 1.0000e-04
Epoch 12/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 71s 1s/step - accuracy: 0.9958 - loss: 0.0185 - val_ac
curacy: 1.0000 - val_loss: 3.0017e-04 - learning_rate: 1.0000e-04
Epoch 13/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 85s 1s/step - accuracy: 0.9955 - loss: 0.0102 - val_ac
curacy: 1.0000 - val_loss: 6.3605e-04 - learning_rate: 1.0000e-04
Epoch 14/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 62s 1s/step - accuracy: 0.9916 - loss: 0.0240 - val_ac
curacy: 1.0000 - val_loss: 2.5256e-04 - learning_rate: 1.0000e-04
Epoch 15/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 66s 1s/step - accuracy: 0.9992 - loss: 0.0071 - val_ac
curacy: 1.0000 - val_loss: 6.6068e-04 - learning_rate: 5.0000e-05
Epoch 16/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 72s 1s/step - accuracy: 0.9995 - loss: 0.0089 - val_ac
curacy: 1.0000 - val_loss: 3.8584e-04 - learning_rate: 5.0000e-05
Epoch 17/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 67s 1s/step - accuracy: 0.9998 - loss: 0.0049 - val_ac
curacy: 1.0000 - val_loss: 3.8319e-04 - learning_rate: 5.0000e-05
Epoch 18/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 71s 1s/step - accuracy: 0.9986 - loss: 0.0047 - val_ac
curacy: 1.0000 - val_loss: 2.5186e-04 - learning_rate: 2.5000e-05
Epoch 19/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 83s 1s/step - accuracy: 0.9995 - loss: 0.0031 - val_ac
curacy: 1.0000 - val_loss: 4.0043e-04 - learning_rate: 2.5000e-05
Epoch 20/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 71s 1s/step - accuracy: 0.9996 - loss: 0.0055 - val_ac
curacy: 1.0000 - val_loss: 2.9530e-04 - learning_rate: 2.5000e-05
```

```
13/13 ━━━━━━━━━━━━━━━━━━━━ 5s 400ms/step - accuracy: 1.0000 - loss: 3.5531e-04
Test accuracy: 1.000
```

Confusion Matrix Easy (classical and metal)

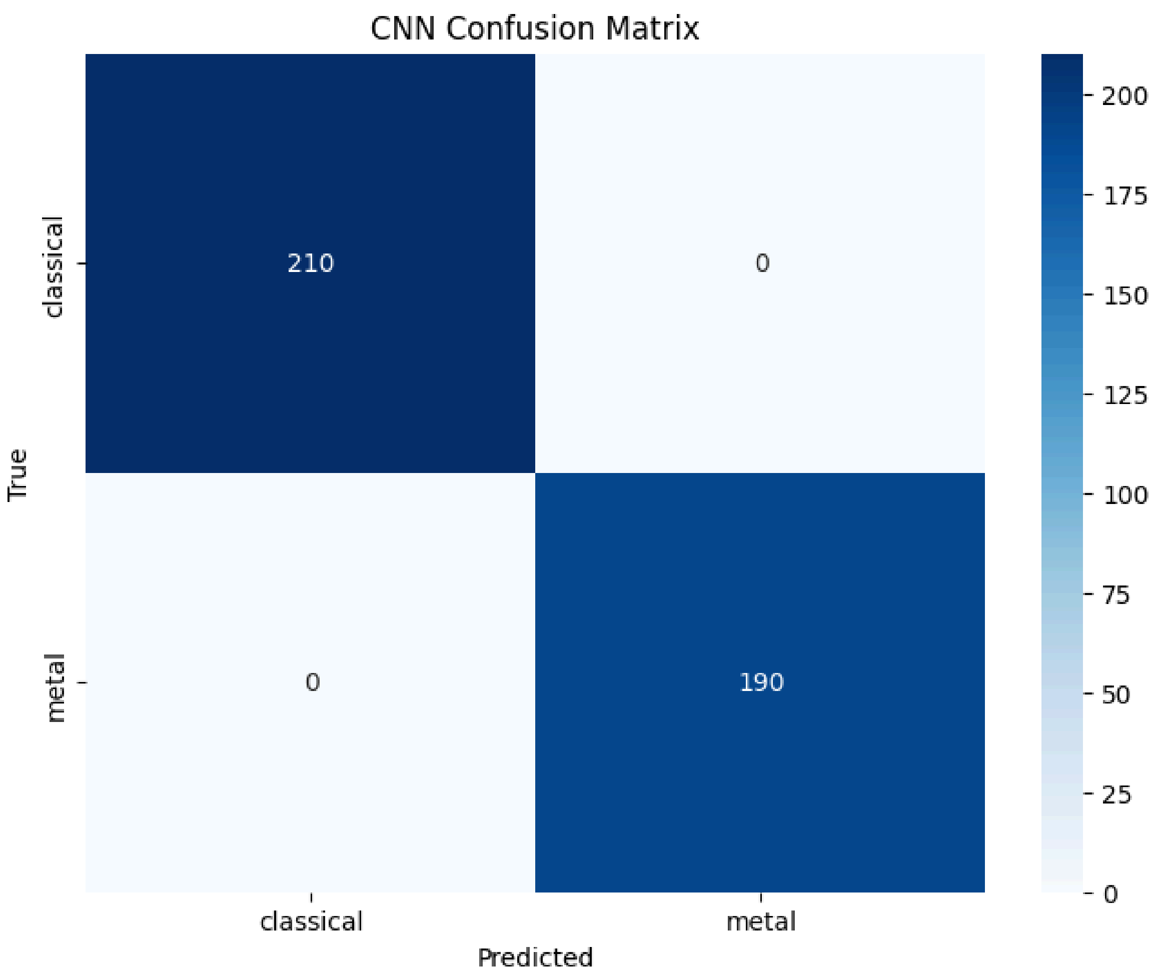
In [4]:

```
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, ytick
plt.title("CNN Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()
```

```
13/13 ━━━━━━━━━━━━━━━━━━━━ 6s 419ms/step
```



3 - Limited genres Hard (disco and pop)

In [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, Dropou
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau
import matplotlib.pyplot as plt

# Augmentation function
def augment_image(image):
    image = tf.image.random_flip_left_right(image)
    image = tf.image.random_brightness(image, max_delta=0.1)
    image = tf.image.random_contrast(image, 0.8, 1.2)
    return image

# Define the genres and file paths
GENRES = ['disco', 'pop']
FILE_PATH = os.path.join('Data', 'spectrograms (3 secs)', 'spectrogram_256')

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 ge
])

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.0001), loss='sparse_categorical_cro

# Learning rate adjustment
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3, min_lr

# Train the model
model.fit(X_train, y_train, epochs=20, validation_data=(X_test, y_test), batch_s

# 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
```

```
c:\Users\berna\AppData\Local\Programs\Python\Python39\lib\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 ━━━━━━━━━━━━━━━━━━━━ 83s 2s/step - accuracy: 0.4790 - loss: 0.6977 - val_ac
curacy: 0.8950 - val_loss: 0.6627 - learning_rate: 1.0000e-04
Epoch 2/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 83s 2s/step - accuracy: 0.7001 - loss: 0.6051 - val_ac
curacy: 0.8900 - val_loss: 0.3114 - learning_rate: 1.0000e-04
Epoch 3/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 142s 2s/step - accuracy: 0.7658 - loss: 0.4663 - val_a
ccuracy: 0.8350 - val_loss: 0.3510 - learning_rate: 1.0000e-04
Epoch 4/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 131s 3s/step - accuracy: 0.7725 - loss: 0.4690 - val_a
ccuracy: 0.9100 - val_loss: 0.2367 - learning_rate: 1.0000e-04
Epoch 5/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 125s 2s/step - accuracy: 0.8119 - loss: 0.3836 - val_a
ccuracy: 0.9075 - val_loss: 0.2502 - learning_rate: 1.0000e-04
Epoch 6/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 96s 1s/step - accuracy: 0.8026 - loss: 0.3976 - val_ac
curacy: 0.9175 - val_loss: 0.2238 - learning_rate: 1.0000e-04
Epoch 7/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 91s 2s/step - accuracy: 0.8056 - loss: 0.3659 - val_ac
curacy: 0.9125 - val_loss: 0.2147 - learning_rate: 1.0000e-04
Epoch 8/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 109s 1s/step - accuracy: 0.8286 - loss: 0.3425 - val_a
ccuracy: 0.9100 - val_loss: 0.2161 - learning_rate: 1.0000e-04
Epoch 9/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 35s 709ms/step - accuracy: 0.8446 - loss: 0.3208 - val
_accuracy: 0.9250 - val_loss: 0.1930 - learning_rate: 1.0000e-04
Epoch 10/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 46s 932ms/step - accuracy: 0.8460 - loss: 0.3296 - val
_accuracy: 0.9475 - val_loss: 0.1809 - learning_rate: 1.0000e-04
Epoch 11/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 148s 2s/step - accuracy: 0.8707 - loss: 0.2893 - val_a
ccuracy: 0.9325 - val_loss: 0.1627 - learning_rate: 1.0000e-04
Epoch 12/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 85s 1s/step - accuracy: 0.8695 - loss: 0.2826 - val_ac
curacy: 0.8950 - val_loss: 0.1869 - learning_rate: 1.0000e-04
Epoch 13/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 84s 2s/step - accuracy: 0.8775 - loss: 0.2699 - val_ac
curacy: 0.9425 - val_loss: 0.1548 - learning_rate: 1.0000e-04
Epoch 14/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 92s 689ms/step - accuracy: 0.9081 - loss: 0.2281 - val
_accuracy: 0.9450 - val_loss: 0.1442 - learning_rate: 1.0000e-04
Epoch 15/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 36s 723ms/step - accuracy: 0.9108 - loss: 0.2176 - val
_accuracy: 0.9475 - val_loss: 0.1898 - learning_rate: 1.0000e-04
Epoch 16/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 38s 767ms/step - accuracy: 0.9080 - loss: 0.2214 - val
_accuracy: 0.9500 - val_loss: 0.1476 - learning_rate: 1.0000e-04
Epoch 17/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 53s 1s/step - accuracy: 0.9294 - loss: 0.1853 - val_ac
curacy: 0.9475 - val_loss: 0.1506 - learning_rate: 1.0000e-04
Epoch 18/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 95s 2s/step - accuracy: 0.9415 - loss: 0.1615 - val_ac
curacy: 0.9400 - val_loss: 0.1540 - learning_rate: 5.0000e-05
Epoch 19/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 37s 732ms/step - accuracy: 0.9479 - loss: 0.1378 - val
_accuracy: 0.9475 - val_loss: 0.1436 - learning_rate: 5.0000e-05
Epoch 20/20
50/50 ━━━━━━━━━━━━━━━━━━━━ 37s 749ms/step - accuracy: 0.9516 - loss: 0.1171 - val
_accuracy: 0.9375 - val_loss: 0.1589 - learning_rate: 5.0000e-05
```

```
13/13 ━━━━━━━━━━━━━━━━━━━━ 3s 212ms/step - accuracy: 0.9114 - loss: 0.1904
Test accuracy: 0.938
```

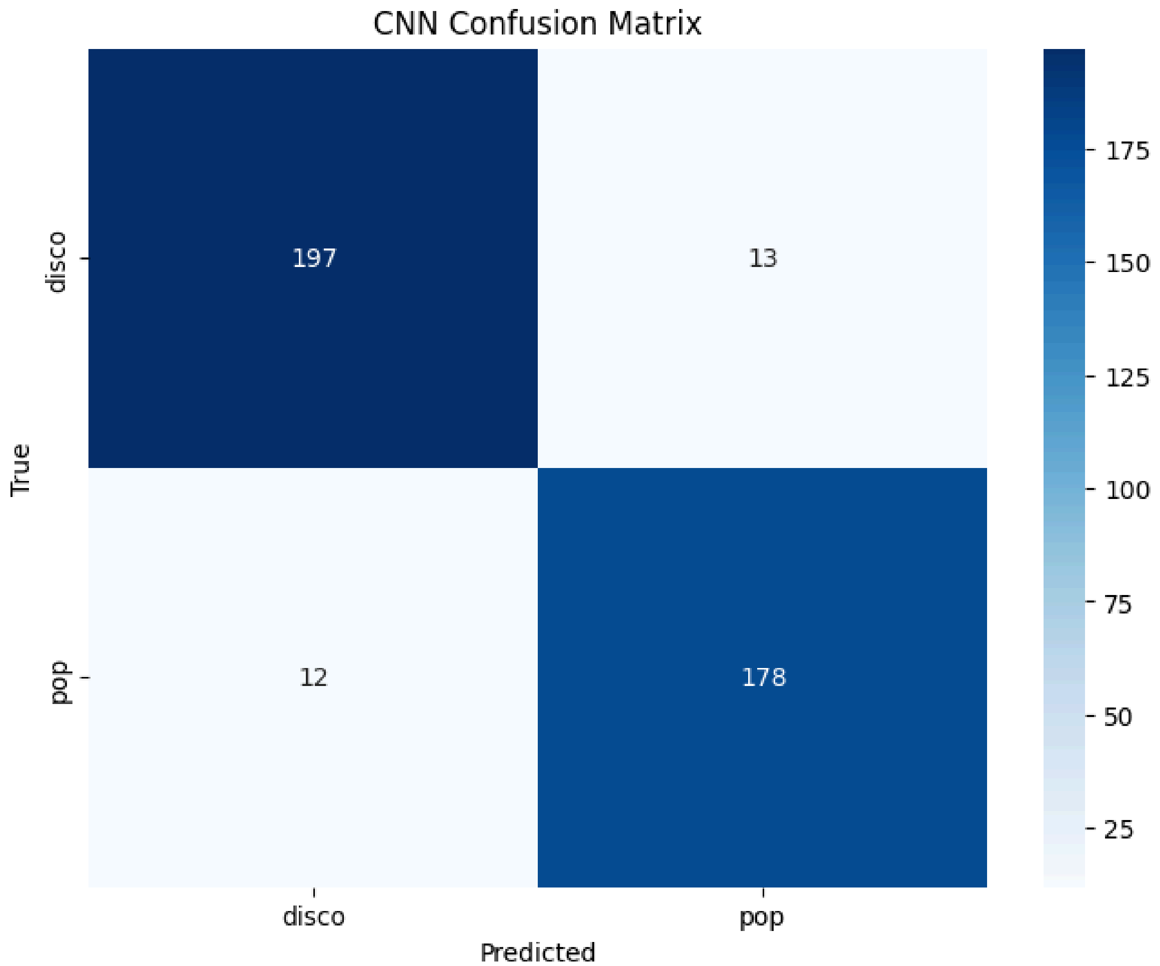
Confusion Matrix Hard (disco and pop)

In [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, ytick
plt.title("CNN Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()
```

```
13/13 ━━━━━━━━━━━━━━━━━━━━ 3s 224ms/step
```



4 - Limited Genres Medium (5 random)

In [7]:
```python
import os
import numpy as np
import tensorflow as tf
```

```python
from sklearn.model_selection import train_test_split
from tensorflow.keras import models
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropou
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau
import matplotlib.pyplot as plt
import random

# Augmentation function
def augment_image(image):
    image = tf.image.random_flip_left_right(image)
    image = tf.image.random_brightness(image, max_delta=0.1)
    image = tf.image.random_contrast(image, 0.8, 1.2)
    return image

GENRES = ['blues', 'classical', 'country', 'disco', 'hiphop', 'jazz', 'metal', '
GENRES = random.sample(GENRES, 5)
print(GENRES)
FILE_PATH = os.path.join('Data', 'spectrograms (3 secs)', 'spectrogram_256')

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 ge
])

# Compile the model
model.compile(optimizer=Adam(learning_rate=0.0001), loss='sparse_categorical_cro

# Learning rate adjustment
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3, min_lr

# Train the model
model.fit(X_train, y_train, epochs=20, validation_data=(X_test, y_test), batch_s

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

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

```
c:\Users\berna\AppData\Local\Programs\Python\Python39\lib\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 ━━━━━━━━━━━━━━━━━━━━ 147s 1s/step - accuracy: 0.2369 - loss: 1.5866 - val
_accuracy: 0.3940 - val_loss: 1.3013 - learning_rate: 1.0000e-04
Epoch 2/20
125/125 ━━━━━━━━━━━━━━━━━━━━ 97s 776ms/step - accuracy: 0.3942 - loss: 1.3604 - v
al_accuracy: 0.5050 - val_loss: 1.1816 - learning_rate: 1.0000e-04
Epoch 3/20
125/125 ━━━━━━━━━━━━━━━━━━━━ 97s 777ms/step - accuracy: 0.4789 - loss: 1.1932 - v
al_accuracy: 0.5690 - val_loss: 1.1159 - learning_rate: 1.0000e-04
Epoch 4/20
125/125 ━━━━━━━━━━━━━━━━━━━━ 95s 762ms/step - accuracy: 0.5451 - loss: 1.0784 - v
al_accuracy: 0.5840 - val_loss: 1.0536 - learning_rate: 1.0000e-04
Epoch 5/20
125/125 ━━━━━━━━━━━━━━━━━━━━ 97s 775ms/step - accuracy: 0.5712 - loss: 1.0006 - v
al_accuracy: 0.5520 - val_loss: 1.0673 - learning_rate: 1.0000e-04
Epoch 6/20
125/125 ━━━━━━━━━━━━━━━━━━━━ 97s 778ms/step - accuracy: 0.6093 - loss: 0.9384 - v
al_accuracy: 0.6110 - val_loss: 0.9325 - learning_rate: 1.0000e-04
Epoch 7/20
125/125 ━━━━━━━━━━━━━━━━━━━━ 97s 775ms/step - accuracy: 0.5754 - loss: 0.9923 - v
al_accuracy: 0.6190 - val_loss: 0.9757 - learning_rate: 1.0000e-04
Epoch 8/20
125/125 ━━━━━━━━━━━━━━━━━━━━ 97s 779ms/step - accuracy: 0.6253 - loss: 0.8820 - v
al_accuracy: 0.6480 - val_loss: 0.9333 - learning_rate: 1.0000e-04
Epoch 9/20
125/125 ━━━━━━━━━━━━━━━━━━━━ 97s 775ms/step - accuracy: 0.6650 - loss: 0.8084 - v
al_accuracy: 0.6380 - val_loss: 0.9084 - learning_rate: 1.0000e-04
Epoch 10/20
125/125 ━━━━━━━━━━━━━━━━━━━━ 96s 772ms/step - accuracy: 0.6794 - loss: 0.7913 - v
al_accuracy: 0.6780 - val_loss: 0.9182 - learning_rate: 1.0000e-04
Epoch 11/20
125/125 ━━━━━━━━━━━━━━━━━━━━ 96s 764ms/step - accuracy: 0.6906 - loss: 0.7769 - v
al_accuracy: 0.6330 - val_loss: 0.9856 - learning_rate: 1.0000e-04
Epoch 12/20
125/125 ━━━━━━━━━━━━━━━━━━━━ 96s 768ms/step - accuracy: 0.7322 - loss: 0.6818 - v
al_accuracy: 0.6370 - val_loss: 1.0242 - learning_rate: 1.0000e-04
Epoch 13/20
125/125 ━━━━━━━━━━━━━━━━━━━━ 97s 775ms/step - accuracy: 0.7425 - loss: 0.6478 - v
al_accuracy: 0.6580 - val_loss: 0.9945 - learning_rate: 5.0000e-05
Epoch 14/20
125/125 ━━━━━━━━━━━━━━━━━━━━ 97s 776ms/step - accuracy: 0.7532 - loss: 0.6328 - v
al_accuracy: 0.7120 - val_loss: 0.9237 - learning_rate: 5.0000e-05
Epoch 15/20
125/125 ━━━━━━━━━━━━━━━━━━━━ 96s 766ms/step - accuracy: 0.7550 - loss: 0.6079 - v
al_accuracy: 0.7190 - val_loss: 0.8639 - learning_rate: 5.0000e-05
Epoch 16/20
125/125 ━━━━━━━━━━━━━━━━━━━━ 137s 727ms/step - accuracy: 0.7908 - loss: 0.5683 -
val_accuracy: 0.7200 - val_loss: 0.9002 - learning_rate: 5.0000e-05
Epoch 17/20
125/125 ━━━━━━━━━━━━━━━━━━━━ 96s 771ms/step - accuracy: 0.7729 - loss: 0.5816 - v
al_accuracy: 0.6900 - val_loss: 0.9912 - learning_rate: 5.0000e-05
Epoch 18/20
125/125 ━━━━━━━━━━━━━━━━━━━━ 96s 765ms/step - accuracy: 0.7913 - loss: 0.5571 - v
al_accuracy: 0.7370 - val_loss: 0.8789 - learning_rate: 5.0000e-05
Epoch 19/20
125/125 ━━━━━━━━━━━━━━━━━━━━ 95s 762ms/step - accuracy: 0.8094 - loss: 0.5295 - v
al_accuracy: 0.7450 - val_loss: 0.9127 - learning_rate: 2.5000e-05
Epoch 20/20
125/125 ━━━━━━━━━━━━━━━━━━━━ 96s 770ms/step - accuracy: 0.8116 - loss: 0.4933 - v
al_accuracy: 0.7390 - val_loss: 0.9624 - learning_rate: 2.5000e-05
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 7s 216ms/step - accuracy: 0.7018 - loss: 1.0183
Test accuracy: 0.739
```

Confusion Matrix Medium (5 random)

In [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, ytick
plt.title("CNN Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()
```

```
32/32 ━━━━━━━━━━━━━━━━━━━━ 7s 219ms/step
```