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, 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', 'spectrograms', '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 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}")
```

```
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: 800 samples
Test set: 200 samples
```

```
/Users/conorwoollatt/.pyenv/versions/3.9.6/lib/python3.9/site-packages/keras/src/layers/convolutional/base_conv.py:1
07: 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
25/25 ━━━━━━━━━━━━━━━━━━━━ 53s 2s/step - accuracy: 0.1032 - loss: 2.3031 - val_accuracy: 0.1400 - val_loss: 2.2998 -
learning_rate: 1.0000e-04
Epoch 2/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 47s 2s/step - accuracy: 0.1146 - loss: 2.2993 - val_accuracy: 0.0850 - val_loss: 2.2917 -
learning_rate: 1.0000e-04
Epoch 3/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 49s 2s/step - accuracy: 0.1225 - loss: 2.2822 - val_accuracy: 0.1200 - val_loss: 2.2538 -
learning_rate: 1.0000e-04
Epoch 4/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 47s 2s/step - accuracy: 0.1672 - loss: 2.2301 - val_accuracy: 0.2500 - val_loss: 2.2133 -
learning_rate: 1.0000e-04
Epoch 5/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 83s 2s/step - accuracy: 0.1919 - loss: 2.1657 - val_accuracy: 0.2400 - val_loss: 2.1746 -
learning_rate: 1.0000e-04
Epoch 6/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 47s 2s/step - accuracy: 0.2609 - loss: 2.0865 - val_accuracy: 0.2550 - val_loss: 2.0515 -
learning_rate: 1.0000e-04
Epoch 7/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 47s 2s/step - accuracy: 0.2935 - loss: 1.9865 - val_accuracy: 0.2650 - val_loss: 2.0503 -
learning_rate: 1.0000e-04
Epoch 8/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 45s 2s/step - accuracy: 0.2839 - loss: 1.9593 - val_accuracy: 0.2800 - val_loss: 1.9448 -
learning_rate: 1.0000e-04
Epoch 9/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 46s 2s/step - accuracy: 0.2778 - loss: 1.9522 - val_accuracy: 0.2600 - val_loss: 1.9263 -
learning_rate: 1.0000e-04
Epoch 10/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 45s 2s/step - accuracy: 0.3088 - loss: 1.8980 - val_accuracy: 0.2850 - val_loss: 1.8720 -
learning_rate: 1.0000e-04
Epoch 11/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 48s 2s/step - accuracy: 0.3316 - loss: 1.8579 - val_accuracy: 0.2700 - val_loss: 1.8718 -
learning_rate: 1.0000e-04
Epoch 12/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 46s 2s/step - accuracy: 0.3245 - loss: 1.7608 - val_accuracy: 0.3200 - val_loss: 1.7792 -
learning_rate: 1.0000e-04
Epoch 13/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 48s 2s/step - accuracy: 0.3162 - loss: 1.7483 - val_accuracy: 0.3450 - val_loss: 1.7431 -
learning_rate: 1.0000e-04
Epoch 14/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 46s 2s/step - accuracy: 0.3252 - loss: 1.7569 - val_accuracy: 0.3350 - val_loss: 1.7633 -
learning_rate: 1.0000e-04
Epoch 15/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 47s 2s/step - accuracy: 0.3925 - loss: 1.6682 - val_accuracy: 0.3650 - val_loss: 1.6461 -
learning_rate: 1.0000e-04
Epoch 16/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 45s 2s/step - accuracy: 0.3835 - loss: 1.6865 - val_accuracy: 0.3600 - val_loss: 1.6824 -
learning_rate: 1.0000e-04
Epoch 17/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 48s 2s/step - accuracy: 0.4129 - loss: 1.6514 - val_accuracy: 0.4000 - val_loss: 1.6270 -
learning_rate: 1.0000e-04
Epoch 18/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 39s 2s/step - accuracy: 0.4295 - loss: 1.5276 - val_accuracy: 0.4000 - val_loss: 1.5525 -
learning_rate: 1.0000e-04
Epoch 19/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 40s 2s/step - accuracy: 0.3664 - loss: 1.6627 - val_accuracy: 0.4000 - val_loss: 1.6968 -
learning_rate: 1.0000e-04
Epoch 20/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 39s 2s/step - accuracy: 0.3907 - loss: 1.6326 - val_accuracy: 0.4050 - val_loss: 1.5546 -
learning_rate: 1.0000e-04
7/7 ━━━━━━━━━━━━━━━━━━━━ 3s 367ms/step - accuracy: 0.4320 - loss: 1.4950
Test accuracy: 0.405
```

Apply the confusion matrix after the model

In [2]:

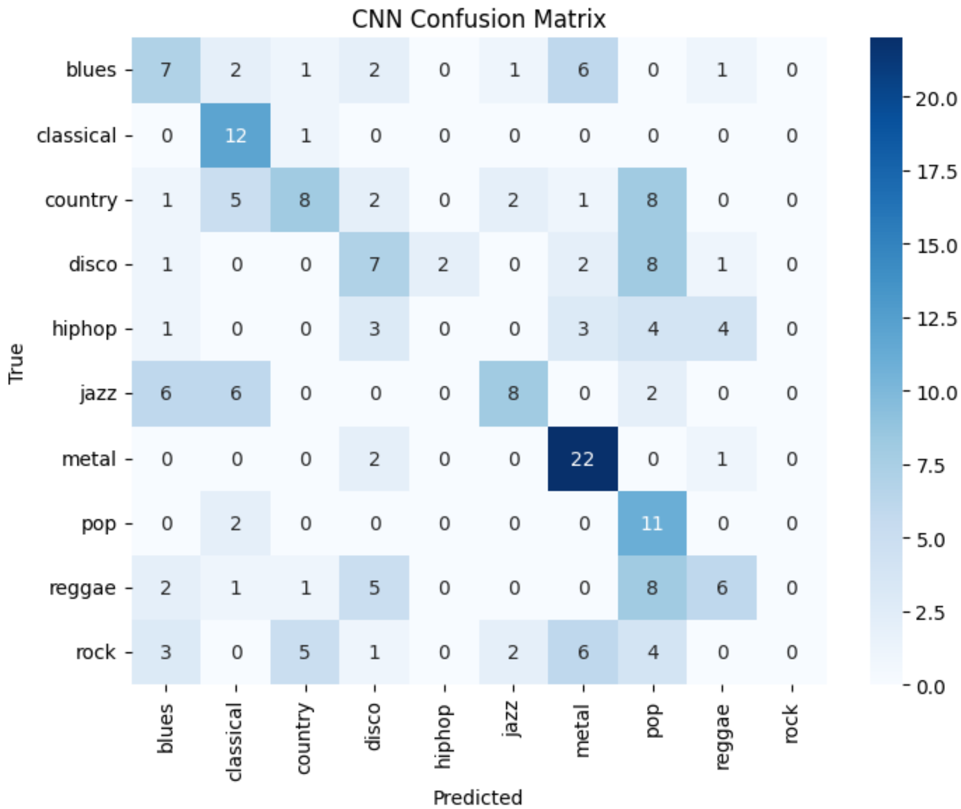
```
import seaborn as sns
# from sklearn.metrics import confusion
import numpy as NP
from sklearn.metrics import confusion_matrix

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()
```

```
7/7 ━━━━━━━━━━━━━━━━━━━━ 3s 394ms/step
```



2 - Limited Genres Easy (metal and classical)

In [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)
    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', '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 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}")
```

```
Processing genre: classical
Processing genre: metal
Train set: 160 samples
Test set: 40 samples
```

```
/Users/conorwoollatt/.pyenv/versions/3.9.6/lib/python3.9/site-packages/keras/src/layers/convolutional/base_conv.py:1
07: 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
5/5 ━━━━━━━━━━━━━━━━━━━━ 10s 2s/step - accuracy: 0.6016 - loss: 0.6744 - val_accuracy: 0.5500 - val_loss: 0.6178 - l
earning_rate: 1.0000e-04
Epoch 2/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 11s 2s/step - accuracy: 0.6523 - loss: 0.6254 - val_accuracy: 1.0000 - val_loss: 0.5034 - l
earning_rate: 1.0000e-04
Epoch 3/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.8244 - loss: 0.5410 - val_accuracy: 0.9750 - val_loss: 0.3196 - le
arning_rate: 1.0000e-04
Epoch 4/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.8447 - loss: 0.4198 - val_accuracy: 1.0000 - val_loss: 0.1658 - le
arning_rate: 1.0000e-04
Epoch 5/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.9014 - loss: 0.2858 - val_accuracy: 1.0000 - val_loss: 0.0907 - le
arning_rate: 1.0000e-04
Epoch 6/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.9055 - loss: 0.2530 - val_accuracy: 1.0000 - val_loss: 0.0327 - le
arning_rate: 1.0000e-04
Epoch 7/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.9341 - loss: 0.1771 - val_accuracy: 0.9500 - val_loss: 0.0839 - le
arning_rate: 1.0000e-04
Epoch 8/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.9436 - loss: 0.1945 - val_accuracy: 1.0000 - val_loss: 0.0163 - le
arning_rate: 1.0000e-04
Epoch 9/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.9257 - loss: 0.1906 - val_accuracy: 1.0000 - val_loss: 0.0351 - le
arning_rate: 1.0000e-04
Epoch 10/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.9114 - loss: 0.2553 - val_accuracy: 1.0000 - val_loss: 0.0175 - le
arning_rate: 1.0000e-04
Epoch 11/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.9054 - loss: 0.2258 - val_accuracy: 1.0000 - val_loss: 0.0321 - le
arning_rate: 1.0000e-04
Epoch 12/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.9636 - loss: 0.1210 - val_accuracy: 1.0000 - val_loss: 0.0179 - le
arning_rate: 5.0000e-05
Epoch 13/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.9546 - loss: 0.0951 - val_accuracy: 1.0000 - val_loss: 0.0127 - le
arning_rate: 5.0000e-05
Epoch 14/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.9433 - loss: 0.1192 - val_accuracy: 1.0000 - val_loss: 0.0133 - le
arning_rate: 5.0000e-05
Epoch 15/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 10s 2s/step - accuracy: 0.9843 - loss: 0.0812 - val_accuracy: 1.0000 - val_loss: 0.0068 - l
earning_rate: 5.0000e-05
Epoch 16/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.9666 - loss: 0.0915 - val_accuracy: 1.0000 - val_loss: 0.0044 - le
arning_rate: 5.0000e-05
Epoch 17/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.9688 - loss: 0.0585 - val_accuracy: 1.0000 - val_loss: 0.0031 - le
arning_rate: 5.0000e-05
Epoch 18/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.9770 - loss: 0.0698 - val_accuracy: 1.0000 - val_loss: 0.0043 - le
arning_rate: 5.0000e-05
Epoch 19/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 11s 2s/step - accuracy: 0.9979 - loss: 0.0265 - val_accuracy: 1.0000 - val_loss: 0.0022 - l
earning_rate: 5.0000e-05
Epoch 20/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.9855 - loss: 0.0338 - val_accuracy: 1.0000 - val_loss: 0.0020 - le
arning_rate: 5.0000e-05
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 148ms/step - accuracy: 1.0000 - loss: 0.0020
Test accuracy: 1.000
```

Confusion Matrix Easy (classical and metal)

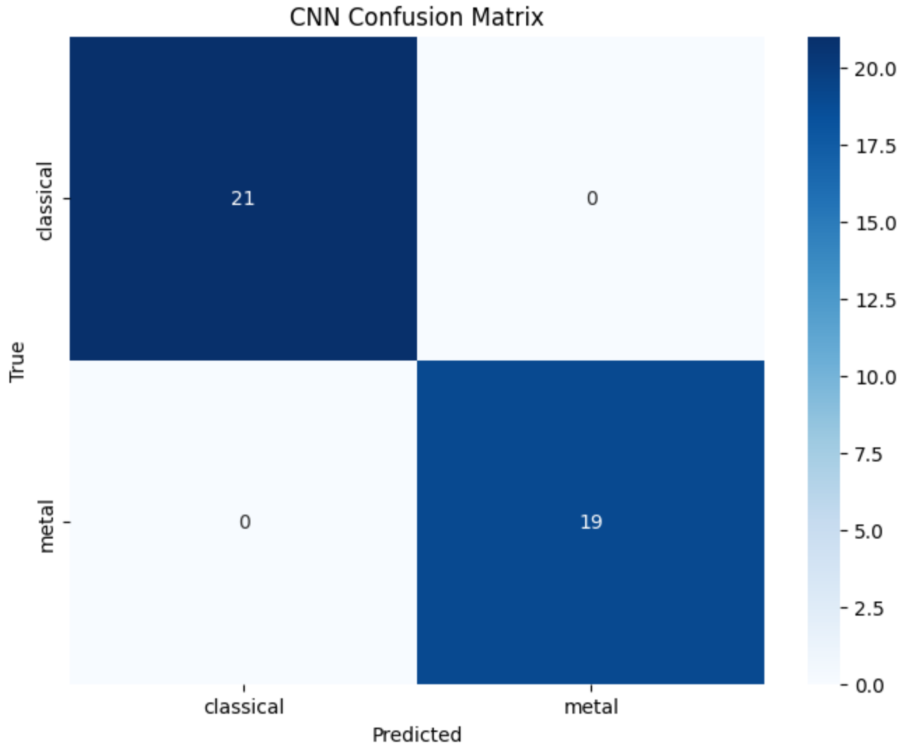
In [4]:
```
import seaborn as sns
# from sklearn.metrics import confusion
import numpy as NP
from sklearn.metrics import confusion_matrix

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

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 168ms/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, 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 = ['disco', 'pop']
FILE_PATH = os.path.join('Data', 'spectrograms', '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 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])
```

```python
# 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: 160 samples
Test set: 40 samples
```

```
/Users/conorwoollatt/.pyenv/versions/3.9.6/lib/python3.9/site-packages/keras/src/layers/convolutional/base_conv.py:1
07: 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
5/5 ━━━━━━━━━━━━━━━━━━━━ 11s 2s/step - accuracy: 0.5100 - loss: 0.6950 - val_accuracy: 0.8000 - val_loss: 0.6892 - l
earning_rate: 1.0000e-04
Epoch 2/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.5526 - loss: 0.6848 - val_accuracy: 0.5750 - val_loss: 0.6864 - le
arning_rate: 1.0000e-04
Epoch 3/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.5654 - loss: 0.6941 - val_accuracy: 0.4750 - val_loss: 0.6845 - le
arning_rate: 1.0000e-04
Epoch 4/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.4780 - loss: 0.6919 - val_accuracy: 0.5250 - val_loss: 0.6802 - le
arning_rate: 1.0000e-04
Epoch 5/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.5151 - loss: 0.6932 - val_accuracy: 0.5250 - val_loss: 0.6770 - le
arning_rate: 1.0000e-04
Epoch 6/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 10s 2s/step - accuracy: 0.5552 - loss: 0.6847 - val_accuracy: 0.5250 - val_loss: 0.6713 - l
earning_rate: 1.0000e-04
Epoch 7/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.5589 - loss: 0.6811 - val_accuracy: 0.6500 - val_loss: 0.6600 - le
arning_rate: 1.0000e-04
Epoch 8/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.6069 - loss: 0.6749 - val_accuracy: 0.7500 - val_loss: 0.6429 - le
arning_rate: 1.0000e-04
Epoch 9/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.6562 - loss: 0.6531 - val_accuracy: 0.7500 - val_loss: 0.6157 - le
arning_rate: 1.0000e-04
Epoch 10/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 10s 2s/step - accuracy: 0.7037 - loss: 0.6253 - val_accuracy: 0.8000 - val_loss: 0.5672 - l
earning_rate: 1.0000e-04
Epoch 11/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.7556 - loss: 0.5887 - val_accuracy: 0.7750 - val_loss: 0.5414 - le
arning_rate: 1.0000e-04
Epoch 12/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.7682 - loss: 0.5549 - val_accuracy: 0.7750 - val_loss: 0.4609 - le
arning_rate: 1.0000e-04
Epoch 13/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.8223 - loss: 0.4703 - val_accuracy: 0.8000 - val_loss: 0.4511 - le
arning_rate: 1.0000e-04
Epoch 14/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.8478 - loss: 0.4124 - val_accuracy: 0.8000 - val_loss: 0.3839 - le
arning_rate: 1.0000e-04
Epoch 15/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.8421 - loss: 0.4029 - val_accuracy: 0.7750 - val_loss: 0.5107 - le
arning_rate: 1.0000e-04
Epoch 16/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.8210 - loss: 0.4270 - val_accuracy: 0.7750 - val_loss: 0.4046 - le
arning_rate: 1.0000e-04
Epoch 17/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.8056 - loss: 0.4268 - val_accuracy: 0.8000 - val_loss: 0.4143 - le
arning_rate: 1.0000e-04
Epoch 18/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.8123 - loss: 0.3839 - val_accuracy: 0.8000 - val_loss: 0.4228 - le
arning_rate: 5.0000e-05
Epoch 19/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.8286 - loss: 0.3989 - val_accuracy: 0.8250 - val_loss: 0.3827 - le
arning_rate: 5.0000e-05
Epoch 20/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.8284 - loss: 0.3617 - val_accuracy: 0.8000 - val_loss: 0.3971 - le
arning_rate: 5.0000e-05
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 137ms/step - accuracy: 0.8146 - loss: 0.3817
Test accuracy: 0.800
```

Confusion Matrix Hard (disco and pop)

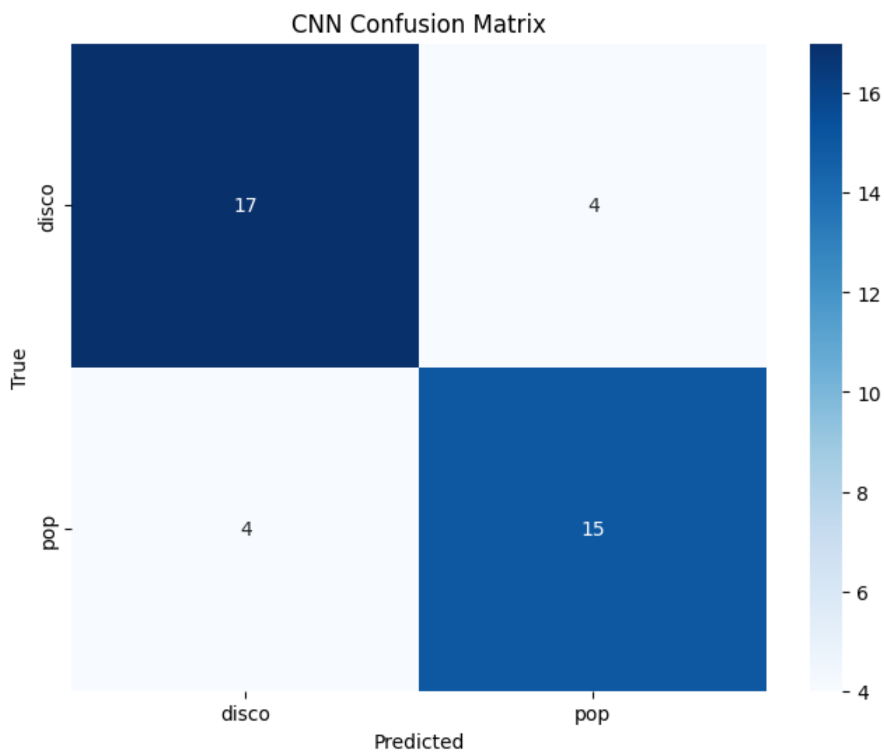
In [6]:
```python
import seaborn as sns
# from sklearn.metrics import confusion
```

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

```
WARNING:tensorflow:5 out of the last 10 calls to <function TensorFlowTrainer.make_predict_function.<locals>.one_step
_on_data_distributed at 0x3d0b55040> triggered tf.function retracing. Tracing is expensive and the excessive number
of tracings could be due to (1) creating @tf.function repeatedly in a loop, (2) passing tensors with different shape
s, (3) passing Python objects instead of tensors. For (1), please define your @tf.function outside of the loop. For
(2), @tf.function has reduce_retracing=True option that can avoid unnecessary retracing. For (3), please refer to ht
tps://www.tensorflow.org/guide/function#controlling_retracing and https://www.tensorflow.org/api_docs/python/tf/func
tion for  more details.
1/2 ━━━━━━━━━━━━━━━━━━━━ 0s 524ms/stepWARNING:tensorflow:6 out of the last 11 calls to <function TensorFlowTrainer.m
ake_predict_function.<locals>.one_step_on_data_distributed at 0x3d0b55040> triggered tf.function retracing. Tracing
is expensive and the excessive number of tracings could be due to (1) creating @tf.function repeatedly in a loop,
(2) passing tensors with different shapes, (3) passing Python objects instead of tensors. For (1), please define you
r @tf.function outside of the loop. For (2), @tf.function has reduce_retracing=True option that can avoid unnecessar
y retracing. For (3), please refer to https://www.tensorflow.org/guide/function#controlling_retracing and https://ww
w.tensorflow.org/api_docs/python/tf/function for  more details.
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 222ms/step
```



4 - Limited Genres Medium (5 random)

In [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 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', 'pop', 'reggae', 'rock']
GENRES = random.sample(GENRES, 5)
print(GENRES)
FILE_PATH = os.path.join('Data', 'spectrograms', '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 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}")
```

```
['metal', 'disco', 'country', 'blues', 'rock']
Processing genre: metal
Processing genre: disco
Processing genre: country
Processing genre: blues
Processing genre: rock
Train set: 400 samples
Test set: 100 samples
```

```
/Users/conorwoollatt/.pyenv/versions/3.9.6/lib/python3.9/site-packages/keras/src/layers/convolutional/base_conv.py:1
07: 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
13/13 ━━━━━━━━━━━━━━━━━━━━ 23s 2s/step - accuracy: 0.2177 - loss: 1.6135 - val_accuracy: 0.1000 - val_loss: 1.6172 -
learning_rate: 1.0000e-04
Epoch 2/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 20s 2s/step - accuracy: 0.2251 - loss: 1.6055 - val_accuracy: 0.1000 - val_loss: 1.6177 -
learning_rate: 1.0000e-04
Epoch 3/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 20s 2s/step - accuracy: 0.2361 - loss: 1.6064 - val_accuracy: 0.1000 - val_loss: 1.6063 -
learning_rate: 1.0000e-04
Epoch 4/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 20s 2s/step - accuracy: 0.1882 - loss: 1.6054 - val_accuracy: 0.2100 - val_loss: 1.5746 -
learning_rate: 1.0000e-04
Epoch 5/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 20s 2s/step - accuracy: 0.2260 - loss: 1.5781 - val_accuracy: 0.2900 - val_loss: 1.5324 -
learning_rate: 1.0000e-04
Epoch 6/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 21s 2s/step - accuracy: 0.2374 - loss: 1.5311 - val_accuracy: 0.4000 - val_loss: 1.4648 -
learning_rate: 1.0000e-04
Epoch 7/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 21s 2s/step - accuracy: 0.3666 - loss: 1.4602 - val_accuracy: 0.3800 - val_loss: 1.3820 -
learning_rate: 1.0000e-04
Epoch 8/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 20s 2s/step - accuracy: 0.3432 - loss: 1.4784 - val_accuracy: 0.4100 - val_loss: 1.3642 -
learning_rate: 1.0000e-04
Epoch 9/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 20s 2s/step - accuracy: 0.3295 - loss: 1.5096 - val_accuracy: 0.5300 - val_loss: 1.3076 -
learning_rate: 1.0000e-04
Epoch 10/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 20s 2s/step - accuracy: 0.3696 - loss: 1.4425 - val_accuracy: 0.5500 - val_loss: 1.2912 -
learning_rate: 1.0000e-04
Epoch 11/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 20s 2s/step - accuracy: 0.4066 - loss: 1.4221 - val_accuracy: 0.4000 - val_loss: 1.2919 -
learning_rate: 1.0000e-04
Epoch 12/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 21s 2s/step - accuracy: 0.3881 - loss: 1.4100 - val_accuracy: 0.4900 - val_loss: 1.1971 -
learning_rate: 1.0000e-04
Epoch 13/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 20s 2s/step - accuracy: 0.3940 - loss: 1.3668 - val_accuracy: 0.4400 - val_loss: 1.1717 -
learning_rate: 1.0000e-04
Epoch 14/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 21s 2s/step - accuracy: 0.4175 - loss: 1.2869 - val_accuracy: 0.5300 - val_loss: 1.1135 -
learning_rate: 1.0000e-04
Epoch 15/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 20s 2s/step - accuracy: 0.4421 - loss: 1.2459 - val_accuracy: 0.4400 - val_loss: 1.1738 -
learning_rate: 1.0000e-04
Epoch 16/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 21s 2s/step - accuracy: 0.4919 - loss: 1.2574 - val_accuracy: 0.4500 - val_loss: 1.1843 -
learning_rate: 1.0000e-04
Epoch 17/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 21s 2s/step - accuracy: 0.4528 - loss: 1.2793 - val_accuracy: 0.4900 - val_loss: 1.1055 -
learning_rate: 1.0000e-04
Epoch 18/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 20s 2s/step - accuracy: 0.4269 - loss: 1.2633 - val_accuracy: 0.4900 - val_loss: 1.1079 -
learning_rate: 1.0000e-04
Epoch 19/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 21s 2s/step - accuracy: 0.5283 - loss: 1.1316 - val_accuracy: 0.4500 - val_loss: 1.1282 -
learning_rate: 1.0000e-04
Epoch 20/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 20s 2s/step - accuracy: 0.4464 - loss: 1.1758 - val_accuracy: 0.5400 - val_loss: 1.0639 -
learning_rate: 1.0000e-04
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 274ms/step - accuracy: 0.5962 - loss: 0.9621
Test accuracy: 0.540
```

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, yticklabels=GENRES)
plt.title("CNN Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("True")
```

```
plt.show()
```

```
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 341ms/step
```