CNN for Spectrogram (3 secs)

1 - All 10

In [10]:
```
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_512')

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
Epoch 1/20
```

```
/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)
```

```
25/25 ━━━━━━━━━━━━━━━━━━━━ 32s 1s/step - accuracy: 0.1026 - loss: 2.3072 - val_accuracy: 0.1100 - val_loss: 2.3043 -
learning_rate: 1.0000e-04
Epoch 2/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 41s 1s/step - accuracy: 0.1243 - loss: 2.2975 - val_accuracy: 0.1050 - val_loss: 2.2956 -
learning_rate: 1.0000e-04
Epoch 3/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 40s 2s/step - accuracy: 0.1640 - loss: 2.2802 - val_accuracy: 0.1000 - val_loss: 2.2701 -
learning_rate: 1.0000e-04
Epoch 4/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 42s 2s/step - accuracy: 0.1886 - loss: 2.2533 - val_accuracy: 0.1850 - val_loss: 2.2088 -
learning_rate: 1.0000e-04
Epoch 5/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 41s 2s/step - accuracy: 0.2195 - loss: 2.1435 - val_accuracy: 0.2250 - val_loss: 2.1487 -
learning_rate: 1.0000e-04
Epoch 6/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 43s 2s/step - accuracy: 0.2585 - loss: 2.1122 - val_accuracy: 0.2400 - val_loss: 2.0907 -
learning_rate: 1.0000e-04
Epoch 7/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 40s 2s/step - accuracy: 0.2481 - loss: 2.0288 - val_accuracy: 0.2600 - val_loss: 2.0168 -
learning_rate: 1.0000e-04
Epoch 8/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 42s 2s/step - accuracy: 0.2848 - loss: 1.9908 - val_accuracy: 0.2850 - val_loss: 1.9368 -
learning_rate: 1.0000e-04
Epoch 9/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 41s 2s/step - accuracy: 0.3246 - loss: 1.8856 - val_accuracy: 0.2750 - val_loss: 1.9313 -
learning_rate: 1.0000e-04
Epoch 10/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 41s 2s/step - accuracy: 0.3005 - loss: 1.8813 - val_accuracy: 0.2850 - val_loss: 1.8914 -
learning_rate: 1.0000e-04
Epoch 11/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 46s 2s/step - accuracy: 0.3404 - loss: 1.7701 - val_accuracy: 0.2650 - val_loss: 1.9336 -
learning_rate: 1.0000e-04
Epoch 12/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 54s 2s/step - accuracy: 0.3384 - loss: 1.7970 - val_accuracy: 0.3250 - val_loss: 1.7737 -
learning_rate: 1.0000e-04
Epoch 13/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 56s 2s/step - accuracy: 0.3465 - loss: 1.7331 - val_accuracy: 0.3000 - val_loss: 1.7928 -
learning_rate: 1.0000e-04
Epoch 14/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 52s 2s/step - accuracy: 0.3757 - loss: 1.7147 - val_accuracy: 0.3400 - val_loss: 1.6776 -
learning_rate: 1.0000e-04
Epoch 15/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 49s 2s/step - accuracy: 0.3765 - loss: 1.6480 - val_accuracy: 0.3750 - val_loss: 1.6383 -
learning_rate: 1.0000e-04
Epoch 16/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 87s 2s/step - accuracy: 0.3801 - loss: 1.6586 - val_accuracy: 0.3700 - val_loss: 1.6348 -
learning_rate: 1.0000e-04
Epoch 17/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 49s 2s/step - accuracy: 0.3996 - loss: 1.5864 - val_accuracy: 0.3850 - val_loss: 1.6140 -
learning_rate: 1.0000e-04
Epoch 18/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 41s 2s/step - accuracy: 0.3746 - loss: 1.6334 - val_accuracy: 0.3800 - val_loss: 1.5466 -
learning_rate: 1.0000e-04
Epoch 19/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 42s 2s/step - accuracy: 0.4386 - loss: 1.5327 - val_accuracy: 0.3450 - val_loss: 1.6302 -
learning_rate: 1.0000e-04
Epoch 20/20
25/25 ━━━━━━━━━━━━━━━━━━━━ 81s 2s/step - accuracy: 0.3758 - loss: 1.6021 - val_accuracy: 0.3900 - val_loss: 1.5477 -
learning_rate: 1.0000e-04
7/7 ━━━━━━━━━━━━━━━━━━━━ 3s 403ms/step - accuracy: 0.4427 - loss: 1.4565
Test accuracy: 0.390
```

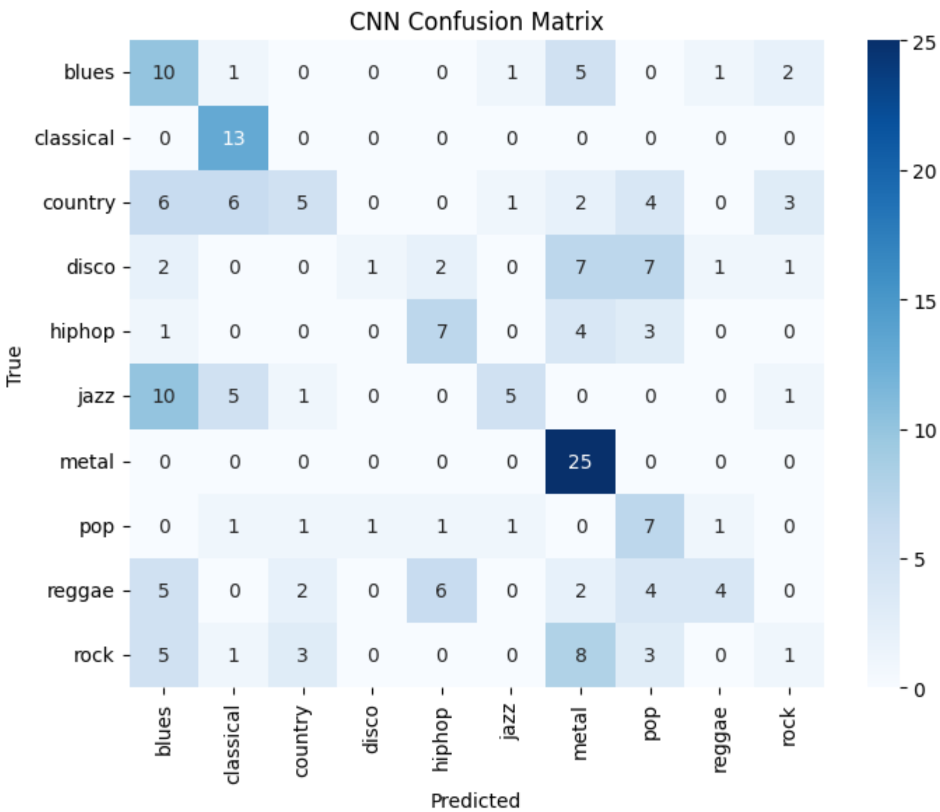
Apply the confusion matrix after the model

In [11]:
```
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 388ms/step
```



2 - Limited Genres Easy (metal and classical)

In [12]:
```
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_512')

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

# Organize data by song ID
```

```
song_to_clips = {}

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

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

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

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

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

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

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

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

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

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

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

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

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

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

    Flatten(),

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

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

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

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

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

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

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

```
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 ━━━━━━━━━━━━━━━━━━━━ 11s 2s/step - accuracy: 0.5634 - loss: 0.6826 - val_accuracy: 0.5000 - val_loss: 0.6214 - l
earning_rate: 1.0000e-04
Epoch 2/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.6078 - loss: 0.6259 - val_accuracy: 0.7500 - val_loss: 0.5154 - le
arning_rate: 1.0000e-04
Epoch 3/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.7771 - loss: 0.5574 - val_accuracy: 1.0000 - val_loss: 0.3475 - le
arning_rate: 1.0000e-04
Epoch 4/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.8875 - loss: 0.4331 - val_accuracy: 1.0000 - val_loss: 0.1889 - le
arning_rate: 1.0000e-04
Epoch 5/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.8260 - loss: 0.3968 - val_accuracy: 1.0000 - val_loss: 0.1112 - le
arning_rate: 1.0000e-04
Epoch 6/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.8906 - loss: 0.2693 - val_accuracy: 1.0000 - val_loss: 0.0618 - le
arning_rate: 1.0000e-04
Epoch 7/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.9002 - loss: 0.2210 - val_accuracy: 1.0000 - val_loss: 0.0299 - le
arning_rate: 1.0000e-04
Epoch 8/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.8978 - loss: 0.2394 - val_accuracy: 1.0000 - val_loss: 0.0217 - le
arning_rate: 1.0000e-04
Epoch 9/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.9475 - loss: 0.1449 - val_accuracy: 1.0000 - val_loss: 0.0186 - le
arning_rate: 1.0000e-04
Epoch 10/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.9770 - loss: 0.1176 - val_accuracy: 1.0000 - val_loss: 0.0201 - le
arning_rate: 1.0000e-04
Epoch 11/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.9373 - loss: 0.1323 - val_accuracy: 1.0000 - val_loss: 0.0089 - le
arning_rate: 1.0000e-04
Epoch 12/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.9405 - loss: 0.1421 - val_accuracy: 1.0000 - val_loss: 0.0027 - le
arning_rate: 1.0000e-04
Epoch 13/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.9795 - loss: 0.0677 - val_accuracy: 1.0000 - val_loss: 0.0022 - le
arning_rate: 1.0000e-04
Epoch 14/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.9782 - loss: 0.0656 - val_accuracy: 1.0000 - val_loss: 0.0020 - le
arning_rate: 1.0000e-04
Epoch 15/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.9949 - loss: 0.0444 - val_accuracy: 1.0000 - val_loss: 0.0011 - le
arning_rate: 1.0000e-04
Epoch 16/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.9979 - loss: 0.0456 - val_accuracy: 1.0000 - val_loss: 8.3942e-04
- learning_rate: 1.0000e-04
Epoch 17/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.9923 - loss: 0.0325 - val_accuracy: 1.0000 - val_loss: 4.4720e-04
- learning_rate: 1.0000e-04
Epoch 18/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 1.0000 - loss: 0.0185 - val_accuracy: 1.0000 - val_loss: 2.4891e-04
- learning_rate: 1.0000e-04
Epoch 19/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.9966 - loss: 0.0250 - val_accuracy: 1.0000 - val_loss: 1.5008e-04
- learning_rate: 1.0000e-04
Epoch 20/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.9898 - loss: 0.0170 - val_accuracy: 1.0000 - val_loss: 2.2307e-04
- learning_rate: 1.0000e-04
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 155ms/step - accuracy: 1.0000 - loss: 2.3692e-04
Test accuracy: 1.000
```

Confusion Matrix Easy (classical and metal)

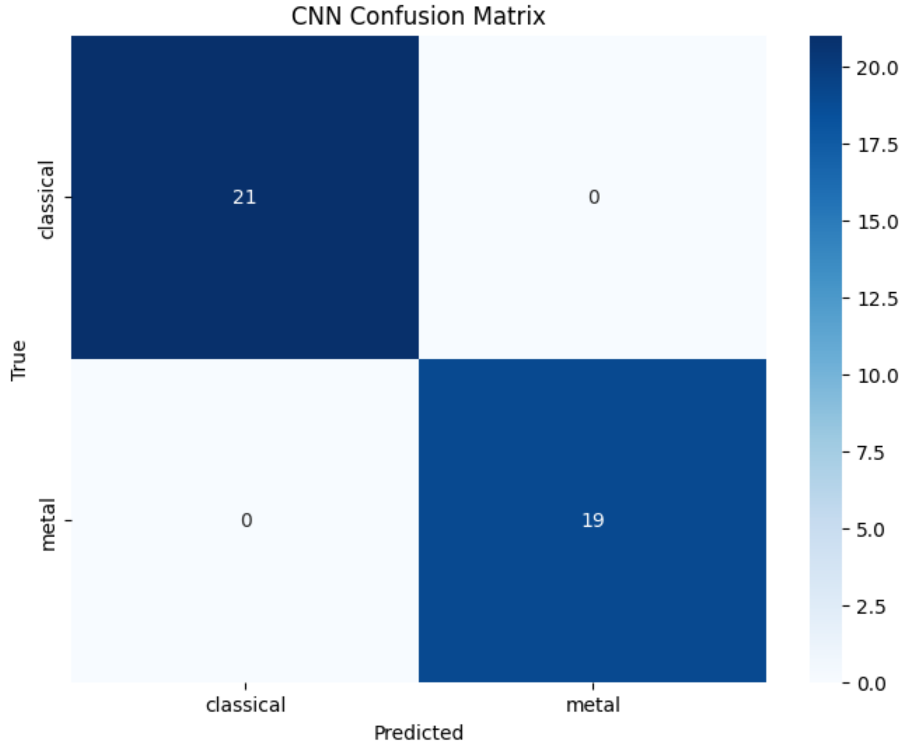
In [13]:
```
import seaborn as sns
# from sklearn.metrics import confusion
import numpy as NP
from sklearn.metrics import confusion_matrix

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 217ms/step
```



3 - Limited genres Hard (disco and pop)

In [14]:

```
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_512')

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: disco
Processing genre: pop
Train set: 160 samples
Test set: 40 samples
Epoch 1/20
```

```
/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)
5/5 ━━━━━━━━━━━━━━━━━━━━ 12s 2s/step - accuracy: 0.5313 - loss: 0.6987 - val_accuracy: 0.5250 - val_loss: 0.6890 - l
earning_rate: 1.0000e-04
Epoch 2/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.4990 - loss: 0.7050 - val_accuracy: 0.5250 - val_loss: 0.6910 - le
arning_rate: 1.0000e-04
Epoch 3/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.5268 - loss: 0.6912 - val_accuracy: 0.5250 - val_loss: 0.6901 - le
arning_rate: 1.0000e-04
Epoch 4/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 10s 2s/step - accuracy: 0.5072 - loss: 0.6941 - val_accuracy: 0.5750 - val_loss: 0.6895 - l
earning_rate: 1.0000e-04
Epoch 5/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.5964 - loss: 0.6849 - val_accuracy: 0.8000 - val_loss: 0.6888 - le
arning_rate: 5.0000e-05
Epoch 6/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.5357 - loss: 0.6878 - val_accuracy: 0.7250 - val_loss: 0.6878 - le
arning_rate: 5.0000e-05
Epoch 7/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.4533 - loss: 0.6947 - val_accuracy: 0.6500 - val_loss: 0.6866 - le
arning_rate: 5.0000e-05
Epoch 8/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.6569 - loss: 0.6775 - val_accuracy: 0.6750 - val_loss: 0.6839 - le
arning_rate: 5.0000e-05
Epoch 9/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.5951 - loss: 0.6824 - val_accuracy: 0.6750 - val_loss: 0.6806 - le
arning_rate: 5.0000e-05
Epoch 10/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.6589 - loss: 0.6700 - val_accuracy: 0.7000 - val_loss: 0.6729 - le
arning_rate: 5.0000e-05
Epoch 11/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.7177 - loss: 0.6615 - val_accuracy: 0.7500 - val_loss: 0.6607 - le
arning_rate: 5.0000e-05
Epoch 12/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.6555 - loss: 0.6607 - val_accuracy: 0.8000 - val_loss: 0.6396 - le
arning_rate: 5.0000e-05
Epoch 13/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 9s 2s/step - accuracy: 0.6817 - loss: 0.6449 - val_accuracy: 0.7750 - val_loss: 0.6115 - le
arning_rate: 5.0000e-05
Epoch 14/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 10s 2s/step - accuracy: 0.6942 - loss: 0.6050 - val_accuracy: 0.7750 - val_loss: 0.5744 - l
earning_rate: 5.0000e-05
Epoch 15/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.6996 - loss: 0.5942 - val_accuracy: 0.7750 - val_loss: 0.5305 - le
arning_rate: 5.0000e-05
Epoch 16/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.7808 - loss: 0.5350 - val_accuracy: 0.7750 - val_loss: 0.4827 - le
arning_rate: 5.0000e-05
Epoch 17/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.7689 - loss: 0.5242 - val_accuracy: 0.7750 - val_loss: 0.4407 - le
arning_rate: 5.0000e-05
Epoch 18/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 7s 1s/step - accuracy: 0.7949 - loss: 0.4776 - val_accuracy: 0.7750 - val_loss: 0.4147 - le
arning_rate: 5.0000e-05
Epoch 19/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.7656 - loss: 0.4501 - val_accuracy: 0.7750 - val_loss: 0.3992 - le
arning_rate: 5.0000e-05
Epoch 20/20
5/5 ━━━━━━━━━━━━━━━━━━━━ 8s 2s/step - accuracy: 0.7523 - loss: 0.4228 - val_accuracy: 0.7500 - val_loss: 0.3872 - le
arning_rate: 5.0000e-05
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 143ms/step - accuracy: 0.7500 - loss: 0.3834
Test accuracy: 0.750
```

Confusion Matrix Hard (disco and pop)

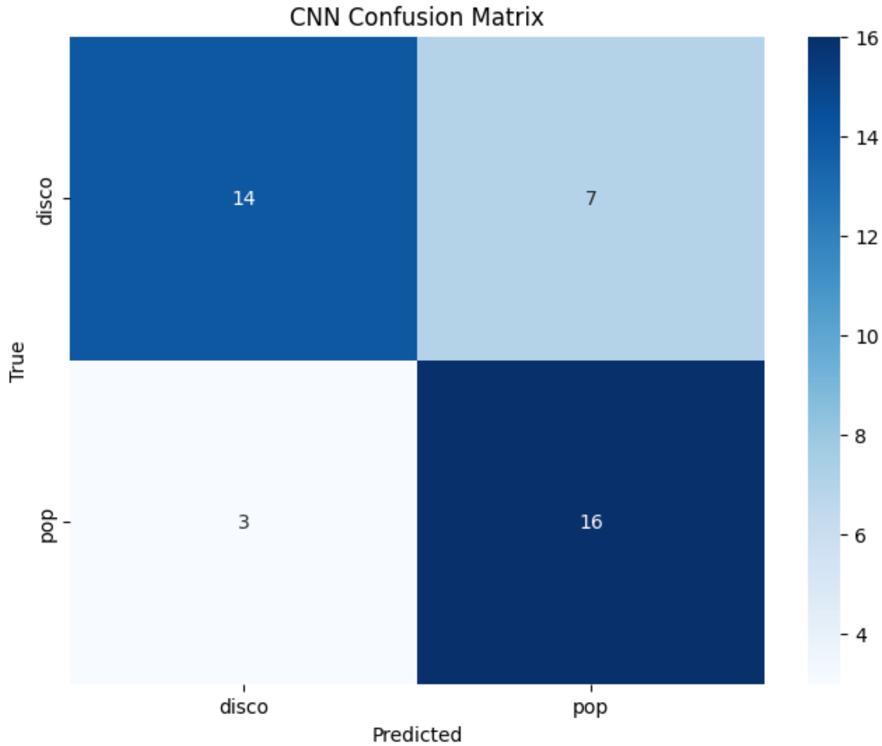
In [15]:
```python
import seaborn as sns
# from sklearn.metrics import confusion
import numpy as NP
from sklearn.metrics import confusion_matrix

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()
```

```
2/2 ━━━━━━━━━━━━━━━━━━━━ 1s 168ms/step
```



4 - Limited Genres Medium (5 random)

In [16]:
```
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_512')

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}")
```

```
['disco', 'jazz', 'rock', 'country', 'hiphop']
Processing genre: disco
Processing genre: jazz
Processing genre: rock
Processing genre: country
Processing genre: hiphop
Train set: 400 samples
Test set: 100 samples
Epoch 1/20
/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)
13/13 ━━━━━━━━━━━━━━━━━━━━ 23s 2s/step - accuracy: 0.1994 - loss: 1.6067 - val_accuracy: 0.2800 - val_loss: 1.6053 -
learning_rate: 1.0000e-04
Epoch 2/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 20s 2s/step - accuracy: 0.2594 - loss: 1.6072 - val_accuracy: 0.1000 - val_loss: 1.6172 -
learning_rate: 1.0000e-04
Epoch 3/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 20s 2s/step - accuracy: 0.2134 - loss: 1.6044 - val_accuracy: 0.0900 - val_loss: 1.6215 -
learning_rate: 1.0000e-04
Epoch 4/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 20s 2s/step - accuracy: 0.2440 - loss: 1.6030 - val_accuracy: 0.1200 - val_loss: 1.6329 -
learning_rate: 1.0000e-04
Epoch 5/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 20s 2s/step - accuracy: 0.2682 - loss: 1.5787 - val_accuracy: 0.1200 - val_loss: 1.6233 -
learning_rate: 5.0000e-05
Epoch 6/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 21s 2s/step - accuracy: 0.2800 - loss: 1.5810 - val_accuracy: 0.1200 - val_loss: 1.6276 -
learning_rate: 5.0000e-05
Epoch 7/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 20s 2s/step - accuracy: 0.2947 - loss: 1.5746 - val_accuracy: 0.3100 - val_loss: 1.5948 -
learning_rate: 5.0000e-05
Epoch 8/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 20s 2s/step - accuracy: 0.2553 - loss: 1.5634 - val_accuracy: 0.1400 - val_loss: 1.6444 -
learning_rate: 5.0000e-05
Epoch 9/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 20s 2s/step - accuracy: 0.3289 - loss: 1.5609 - val_accuracy: 0.1900 - val_loss: 1.5931 -
learning_rate: 5.0000e-05
Epoch 10/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 21s 2s/step - accuracy: 0.3193 - loss: 1.5315 - val_accuracy: 0.2000 - val_loss: 1.6031 -
learning_rate: 5.0000e-05
Epoch 11/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 20s 2s/step - accuracy: 0.2995 - loss: 1.5330 - val_accuracy: 0.2400 - val_loss: 1.5936 -
learning_rate: 5.0000e-05
Epoch 12/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 21s 2s/step - accuracy: 0.2874 - loss: 1.4983 - val_accuracy: 0.2300 - val_loss: 1.5806 -
learning_rate: 5.0000e-05
Epoch 13/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 20s 2s/step - accuracy: 0.3923 - loss: 1.4635 - val_accuracy: 0.2000 - val_loss: 1.5772 -
learning_rate: 5.0000e-05
Epoch 14/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 21s 2s/step - accuracy: 0.3480 - loss: 1.4929 - val_accuracy: 0.2900 - val_loss: 1.5200 -
learning_rate: 5.0000e-05
Epoch 15/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 20s 2s/step - accuracy: 0.4328 - loss: 1.4170 - val_accuracy: 0.2500 - val_loss: 1.5361 -
learning_rate: 5.0000e-05
Epoch 16/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 20s 2s/step - accuracy: 0.4413 - loss: 1.3735 - val_accuracy: 0.3500 - val_loss: 1.4481 -
learning_rate: 5.0000e-05
Epoch 17/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 20s 2s/step - accuracy: 0.3858 - loss: 1.3919 - val_accuracy: 0.2800 - val_loss: 1.4530 -
learning_rate: 5.0000e-05
Epoch 18/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 21s 2s/step - accuracy: 0.4020 - loss: 1.3864 - val_accuracy: 0.4000 - val_loss: 1.3779 -
learning_rate: 5.0000e-05
Epoch 19/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 20s 2s/step - accuracy: 0.4320 - loss: 1.3128 - val_accuracy: 0.4000 - val_loss: 1.3609 -
learning_rate: 5.0000e-05
Epoch 20/20
13/13 ━━━━━━━━━━━━━━━━━━━━ 20s 2s/step - accuracy: 0.4298 - loss: 1.2953 - val_accuracy: 0.3500 - val_loss: 1.3899 -
learning_rate: 5.0000e-05
4/4 ━━━━━━━━━━━━━━━━━━━━ 1s 338ms/step - accuracy: 0.2723 - loss: 1.4357
Test accuracy: 0.350
```

Confusion Matrix Medium (5 random)

In [17]:

```python
import seaborn as sns
# from sklearn.metrics import confusion
```

```python
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 342ms/step
```