

CNN for Mel-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
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', 'mel_spectrograms (3 secs)', 'mel_spectrogram_1

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: 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 ————— 465s 2s/step - accuracy: 0.1310 - loss: 2.2690 - val
_accuracy: 0.2610 - val_loss: 2.0012 - learning_rate: 1.0000e-04

Epoch 2/20
250/250 ————— 351s 1s/step - accuracy: 0.2887 - loss: 1.9262 - val
_accuracy: 0.3970 - val_loss: 1.5779 - learning_rate: 1.0000e-04

Epoch 3/20
250/250 ————— 345s 1s/step - accuracy: 0.4069 - loss: 1.5975 - val
_accuracy: 0.4765 - val_loss: 1.4141 - learning_rate: 1.0000e-04

Epoch 4/20
250/250 ————— 329s 1s/step - accuracy: 0.4635 - loss: 1.4652 - val
_accuracy: 0.5335 - val_loss: 1.3086 - learning_rate: 1.0000e-04

Epoch 5/20
250/250 ————— 352s 1s/step - accuracy: 0.5108 - loss: 1.3445 - val
_accuracy: 0.5140 - val_loss: 1.3318 - learning_rate: 1.0000e-04

Epoch 6/20
250/250 ————— 421s 2s/step - accuracy: 0.5605 - loss: 1.2289 - val
_accuracy: 0.5500 - val_loss: 1.2753 - learning_rate: 1.0000e-04

Epoch 7/20
250/250 ————— 200s 799ms/step - accuracy: 0.5820 - loss: 1.1736 -
val_accuracy: 0.5615 - val_loss: 1.1939 - learning_rate: 1.0000e-04

Epoch 8/20
250/250 ————— 195s 782ms/step - accuracy: 0.6190 - loss: 1.0718 -
val_accuracy: 0.5715 - val_loss: 1.1740 - learning_rate: 1.0000e-04

Epoch 9/20
250/250 ————— 193s 773ms/step - accuracy: 0.6558 - loss: 0.9806 -
val_accuracy: 0.5890 - val_loss: 1.1583 - learning_rate: 1.0000e-04

Epoch 10/20
250/250 ————— 193s 772ms/step - accuracy: 0.6750 - loss: 0.9386 -
val_accuracy: 0.6180 - val_loss: 1.1030 - learning_rate: 1.0000e-04

Epoch 11/20
250/250 ————— 193s 772ms/step - accuracy: 0.6947 - loss: 0.8593 -
val_accuracy: 0.6445 - val_loss: 1.0498 - learning_rate: 1.0000e-04

Epoch 12/20
250/250 ————— 201s 804ms/step - accuracy: 0.7216 - loss: 0.7978 -
val_accuracy: 0.6390 - val_loss: 1.0621 - learning_rate: 1.0000e-04

Epoch 13/20
250/250 ————— 206s 825ms/step - accuracy: 0.7285 - loss: 0.7831 -
val_accuracy: 0.6405 - val_loss: 1.1063 - learning_rate: 1.0000e-04

Epoch 14/20
250/250 ————— 208s 830ms/step - accuracy: 0.7573 - loss: 0.6956 -
val_accuracy: 0.6540 - val_loss: 1.1323 - learning_rate: 1.0000e-04

Epoch 15/20
250/250 ————— 219s 878ms/step - accuracy: 0.7998 - loss: 0.5891 -
val_accuracy: 0.6625 - val_loss: 1.0852 - learning_rate: 5.0000e-05

Epoch 16/20
250/250 ————— 389s 2s/step - accuracy: 0.8047 - loss: 0.5551 - val
_accuracy: 0.6625 - val_loss: 1.1316 - learning_rate: 5.0000e-05

Epoch 17/20
250/250 ————— 382s 2s/step - accuracy: 0.8151 - loss: 0.5350 - val
_accuracy: 0.6575 - val_loss: 1.1428 - learning_rate: 5.0000e-05

Epoch 18/20
250/250 ————— 379s 2s/step - accuracy: 0.8341 - loss: 0.4776 - val
_accuracy: 0.6755 - val_loss: 1.1336 - learning_rate: 2.5000e-05

Epoch 19/20
250/250 ————— 379s 2s/step - accuracy: 0.8536 - loss: 0.4386 - val
_accuracy: 0.6655 - val_loss: 1.1618 - learning_rate: 2.5000e-05

Epoch 20/20
250/250 ————— 376s 2s/step - accuracy: 0.8588 - loss: 0.4138 - val
_accuracy: 0.6520 - val_loss: 1.1950 - learning_rate: 2.5000e-05

63/63 ————— 27s 421ms/step - accuracy: 0.6641 - loss: 1.1388
Test accuracy: 0.652

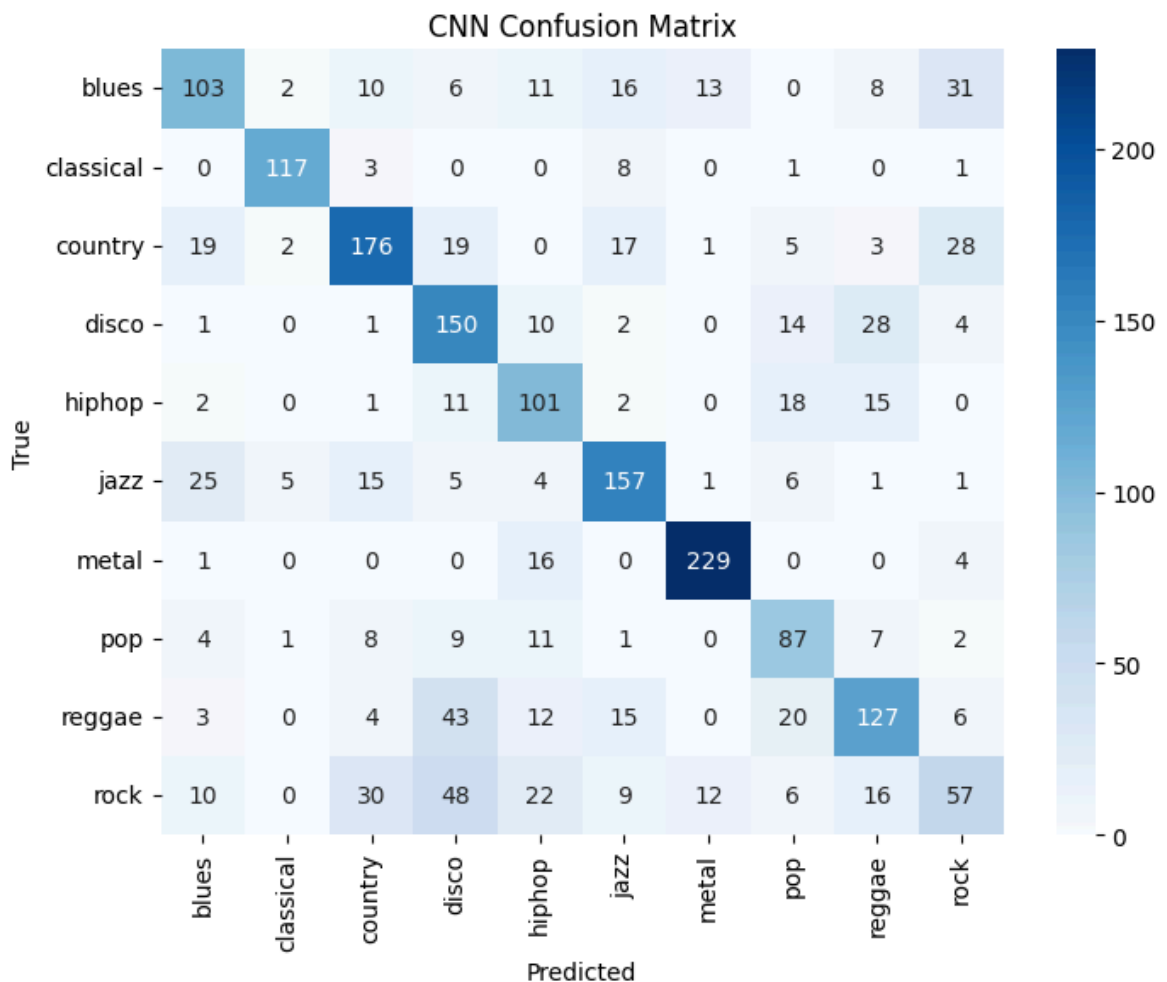
Apply the confusion matrix after the model

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

63/63 ————— 27s 429ms/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
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau

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

# Define the genres and file paths
GENRES = ['classical', 'metal']
FILE_PATH = os.path.join('Data', 'mel_spectrograms (3 secs)', 'mel_spectrogram_1

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  **82s** 2s/step - accuracy: 0.5755 - loss: 0.6513 - val_accuracy: 0.9675 - val_loss: 0.1900 - learning_rate: 1.0000e-04


Epoch 2/20
50/50  **73s** 1s/step - accuracy: 0.9095 - loss: 0.2484 - val_accuracy: 0.9750 - val_loss: 0.0829 - learning_rate: 1.0000e-04


Epoch 3/20
50/50  **76s** 2s/step - accuracy: 0.9563 - loss: 0.1271 - val_accuracy: 0.9875 - val_loss: 0.0400 - learning_rate: 1.0000e-04


Epoch 4/20
50/50  **75s** 1s/step - accuracy: 0.9683 - loss: 0.0949 - val_accuracy: 0.9900 - val_loss: 0.0253 - learning_rate: 1.0000e-04


Epoch 5/20
50/50  **76s** 2s/step - accuracy: 0.9807 - loss: 0.0582 - val_accuracy: 0.9950 - val_loss: 0.0120 - learning_rate: 1.0000e-04


Epoch 6/20
50/50  **75s** 2s/step - accuracy: 0.9879 - loss: 0.0451 - val_accuracy: 0.9925 - val_loss: 0.0138 - learning_rate: 1.0000e-04


Epoch 7/20
50/50  **78s** 2s/step - accuracy: 0.9900 - loss: 0.0384 - val_accuracy: 1.0000 - val_loss: 0.0028 - learning_rate: 1.0000e-04


Epoch 8/20
50/50  **79s** 2s/step - accuracy: 0.9952 - loss: 0.0122 - val_accuracy: 1.0000 - val_loss: 0.0030 - learning_rate: 1.0000e-04


Epoch 9/20
50/50  **80s** 2s/step - accuracy: 0.9951 - loss: 0.0159 - val_accuracy: 1.0000 - val_loss: 4.4660e-04 - learning_rate: 1.0000e-04


Epoch 10/20
50/50  **81s** 2s/step - accuracy: 0.9969 - loss: 0.0129 - val_accuracy: 1.0000 - val_loss: 4.9909e-04 - learning_rate: 1.0000e-04


Epoch 11/20
50/50  **73s** 1s/step - accuracy: 0.9965 - loss: 0.0103 - val_accuracy: 1.0000 - val_loss: 3.1585e-04 - learning_rate: 1.0000e-04


Epoch 12/20
50/50  **77s** 2s/step - accuracy: 0.9966 - loss: 0.0097 - val_accuracy: 1.0000 - val_loss: 8.3719e-04 - learning_rate: 1.0000e-04


Epoch 13/20
50/50  **75s** 2s/step - accuracy: 0.9949 - loss: 0.0091 - val_accuracy: 1.0000 - val_loss: 0.0015 - learning_rate: 1.0000e-04


Epoch 14/20
50/50  **76s** 2s/step - accuracy: 0.9981 - loss: 0.0046 - val_accuracy: 1.0000 - val_loss: 9.2895e-06 - learning_rate: 1.0000e-04


Epoch 15/20
50/50  **76s** 2s/step - accuracy: 0.9934 - loss: 0.0202 - val_accuracy: 1.0000 - val_loss: 6.5351e-05 - learning_rate: 1.0000e-04

Epoch 16/20
50/50  **72s** 1s/step - accuracy: 0.9987 - loss: 0.0042 - val_accuracy: 1.0000 - val_loss: 2.8363e-04 - learning_rate: 1.0000e-04

Epoch 17/20
50/50  **75s** 2s/step - accuracy: 0.9978 - loss: 0.0051 - val_accuracy: 1.0000 - val_loss: 3.0982e-05 - learning_rate: 1.0000e-04

Epoch 18/20
50/50  **76s** 2s/step - accuracy: 0.9963 - loss: 0.0116 - val_accuracy: 1.0000 - val_loss: 1.2814e-04 - learning_rate: 5.0000e-05

Epoch 19/20
50/50  **76s** 2s/step - accuracy: 0.9995 - loss: 0.0038 - val_accuracy: 1.0000 - val_loss: 7.6435e-05 - learning_rate: 5.0000e-05

Epoch 20/20
50/50  **80s** 2s/step - accuracy: 0.9996 - loss: 0.0032 - val_accuracy: 1.0000 - val_loss: 2.7869e-04 - learning_rate: 5.0000e-05

13/13 ————— 5s 405ms/step - accuracy: 1.0000 - loss: 4.3954e-04
Test accuracy: 1.000

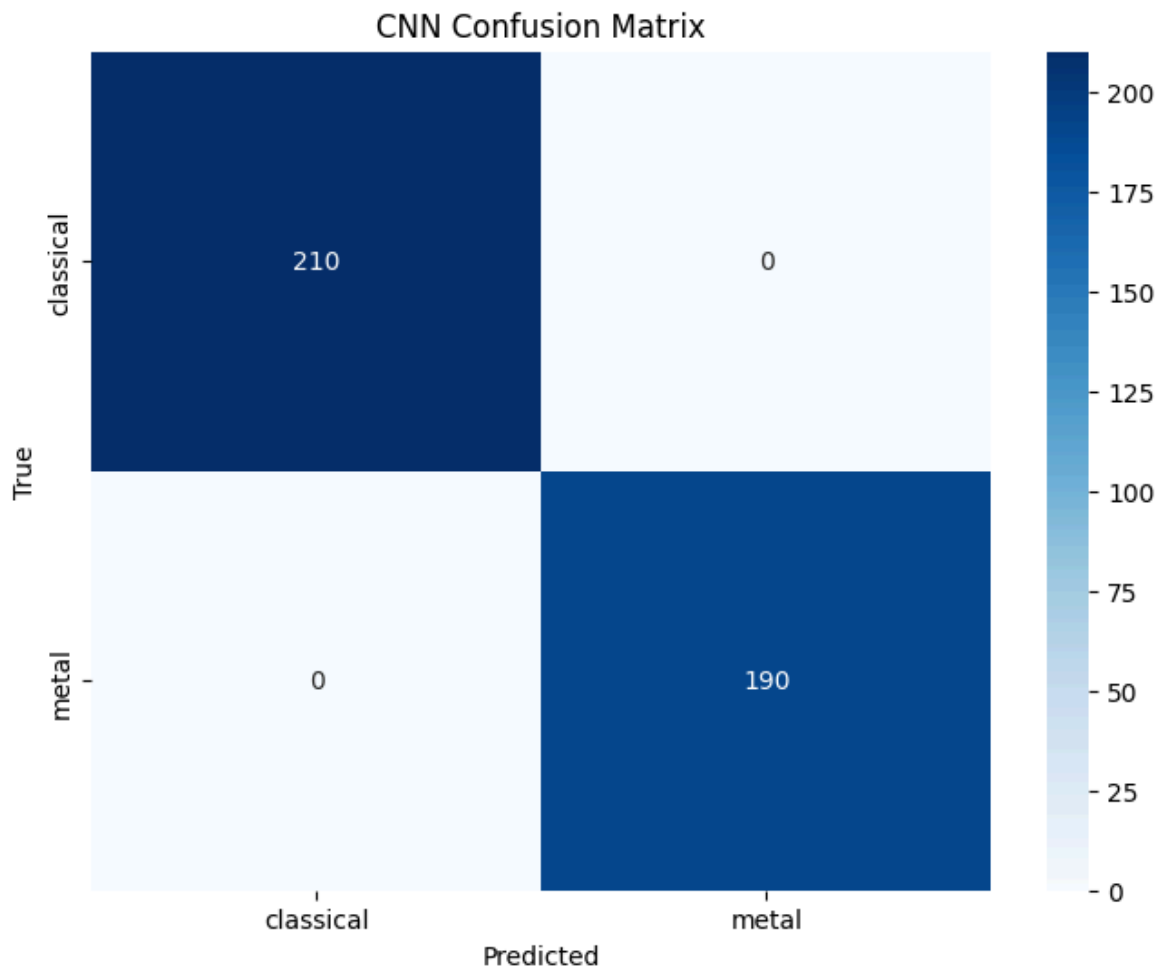
Confusion Matrix Easy (classical and metal)

```
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 437ms/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
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau

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

# Define the genres and file paths
GENRES = ['disco', 'pop']
FILE_PATH = os.path.join('Data', 'mel_spectrograms (3 secs)', 'mel_spectrogram_1')

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  **88s** 2s/step - accuracy: 0.4844 - loss: 0.6934 - val_accuracy: 0.5675 - val_loss: 0.6915 - learning_rate: 1.0000e-04


Epoch 2/20
50/50  **76s** 2s/step - accuracy: 0.5514 - loss: 0.6890 - val_accuracy: 0.6550 - val_loss: 0.6121 - learning_rate: 1.0000e-04


Epoch 3/20
50/50  **75s** 2s/step - accuracy: 0.6925 - loss: 0.5935 - val_accuracy: 0.9125 - val_loss: 0.2759 - learning_rate: 1.0000e-04


Epoch 4/20
50/50  **86s** 2s/step - accuracy: 0.7930 - loss: 0.4327 - val_accuracy: 0.9125 - val_loss: 0.2768 - learning_rate: 1.0000e-04


Epoch 5/20
50/50  **77s** 2s/step - accuracy: 0.8288 - loss: 0.3578 - val_accuracy: 0.9100 - val_loss: 0.2331 - learning_rate: 1.0000e-04


Epoch 6/20
50/50  **77s** 2s/step - accuracy: 0.8373 - loss: 0.3384 - val_accuracy: 0.9125 - val_loss: 0.1929 - learning_rate: 1.0000e-04


Epoch 7/20
50/50  **76s** 2s/step - accuracy: 0.8591 - loss: 0.3109 - val_accuracy: 0.9200 - val_loss: 0.1861 - learning_rate: 1.0000e-04


Epoch 8/20
50/50  **79s** 2s/step - accuracy: 0.8635 - loss: 0.2901 - val_accuracy: 0.9125 - val_loss: 0.1882 - learning_rate: 1.0000e-04


Epoch 9/20
50/50  **78s** 2s/step - accuracy: 0.8821 - loss: 0.2648 - val_accuracy: 0.9225 - val_loss: 0.1823 - learning_rate: 1.0000e-04


Epoch 10/20
50/50  **77s** 2s/step - accuracy: 0.8941 - loss: 0.2460 - val_accuracy: 0.9250 - val_loss: 0.1738 - learning_rate: 1.0000e-04


Epoch 11/20
50/50  **77s** 2s/step - accuracy: 0.8900 - loss: 0.2424 - val_accuracy: 0.9200 - val_loss: 0.1686 - learning_rate: 1.0000e-04


Epoch 12/20
50/50  **76s** 2s/step - accuracy: 0.9047 - loss: 0.2279 - val_accuracy: 0.9175 - val_loss: 0.1729 - learning_rate: 1.0000e-04


Epoch 13/20
50/50  **73s** 1s/step - accuracy: 0.9181 - loss: 0.2045 - val_accuracy: 0.9150 - val_loss: 0.1756 - learning_rate: 1.0000e-04


Epoch 14/20
50/50  **75s** 2s/step - accuracy: 0.9158 - loss: 0.1881 - val_accuracy: 0.9150 - val_loss: 0.1730 - learning_rate: 1.0000e-04


Epoch 15/20
50/50  **76s** 2s/step - accuracy: 0.9277 - loss: 0.1712 - val_accuracy: 0.9200 - val_loss: 0.1743 - learning_rate: 5.0000e-05

Epoch 16/20
50/50  **77s** 2s/step - accuracy: 0.9419 - loss: 0.1558 - val_accuracy: 0.9250 - val_loss: 0.1858 - learning_rate: 5.0000e-05

Epoch 17/20
50/50  **83s** 2s/step - accuracy: 0.9393 - loss: 0.1667 - val_accuracy: 0.9250 - val_loss: 0.1860 - learning_rate: 5.0000e-05

Epoch 18/20
50/50  **75s** 1s/step - accuracy: 0.9511 - loss: 0.1341 - val_accuracy: 0.9250 - val_loss: 0.1847 - learning_rate: 2.5000e-05

Epoch 19/20
50/50  **78s** 2s/step - accuracy: 0.9524 - loss: 0.1277 - val_accuracy: 0.9250 - val_loss: 0.1929 - learning_rate: 2.5000e-05

Epoch 20/20
50/50  **83s** 2s/step - accuracy: 0.9345 - loss: 0.1472 - val_accuracy: 0.9175 - val_loss: 0.2055 - learning_rate: 2.5000e-05

13/13 ————— 6s 430ms/step - accuracy: 0.8990 - loss: 0.2947
Test accuracy: 0.918

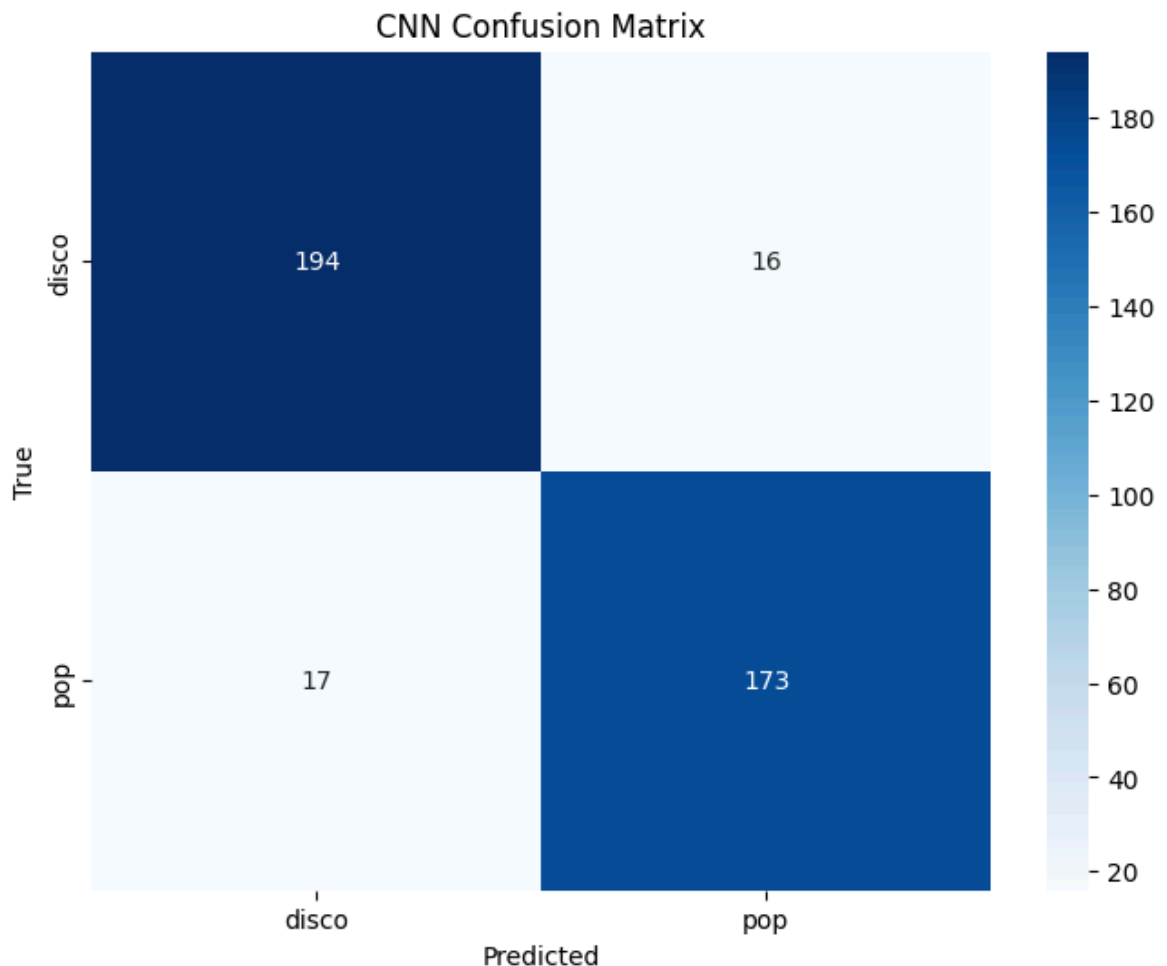
Confusion Matrix Hard (disco and pop)

```
In [6]: 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 440ms/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
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau

import random

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

# Define the genres and file paths
GENRES = ['blues', 'classical', 'country', 'disco', 'hiphop', 'jazz', 'metal', 'rock', 'soul', 'swing']
GENRES = random.sample(GENRES, 5)
print(GENRES)
FILE_PATH = os.path.join('Data', 'mel_spectrograms (3 secs)', 'mel_spectrogram_1')

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}")

```

```
['country', 'rock', 'blues', 'reggae', 'classical']
```

```
Processing genre: country
```

```
Processing genre: rock
```

```
Processing genre: blues
```

```
Processing genre: reggae
```

```
Processing genre: classical
```

```
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 ————— 217s 2s/step - accuracy: 0.2217 - loss: 1.6090 - val
_accuracy: 0.1500 - val_loss: 1.6336 - learning_rate: 1.0000e-04
Epoch 2/20
125/125 ————— 201s 2s/step - accuracy: 0.3284 - loss: 1.5121 - val
_accuracy: 0.5510 - val_loss: 1.0512 - learning_rate: 1.0000e-04
Epoch 3/20
125/125 ————— 183s 1s/step - accuracy: 0.5012 - loss: 1.1681 - val
_accuracy: 0.6570 - val_loss: 0.9023 - learning_rate: 1.0000e-04
Epoch 4/20
125/125 ————— 198s 2s/step - accuracy: 0.5817 - loss: 1.0469 - val
_accuracy: 0.6320 - val_loss: 0.9175 - learning_rate: 1.0000e-04
Epoch 5/20
125/125 ————— 179s 1s/step - accuracy: 0.6120 - loss: 0.9537 - val
_accuracy: 0.6640 - val_loss: 0.8756 - learning_rate: 1.0000e-04
Epoch 6/20
125/125 ————— 226s 2s/step - accuracy: 0.6273 - loss: 0.9364 - val
_accuracy: 0.6810 - val_loss: 0.8127 - learning_rate: 1.0000e-04
Epoch 7/20
125/125 ————— 201s 2s/step - accuracy: 0.6710 - loss: 0.7964 - val
_accuracy: 0.7290 - val_loss: 0.7299 - learning_rate: 1.0000e-04
Epoch 8/20
125/125 ————— 202s 2s/step - accuracy: 0.6816 - loss: 0.8052 - val
_accuracy: 0.7080 - val_loss: 0.7739 - learning_rate: 1.0000e-04
Epoch 9/20
125/125 ————— 153s 1s/step - accuracy: 0.7209 - loss: 0.7303 - val
_accuracy: 0.7190 - val_loss: 0.7588 - learning_rate: 1.0000e-04
Epoch 10/20
125/125 ————— 134s 1s/step - accuracy: 0.7298 - loss: 0.6701 - val
_accuracy: 0.7330 - val_loss: 0.7346 - learning_rate: 1.0000e-04
Epoch 11/20
125/125 ————— 184s 1s/step - accuracy: 0.7663 - loss: 0.6077 - val
_accuracy: 0.7220 - val_loss: 0.7251 - learning_rate: 5.0000e-05
Epoch 12/20
125/125 ————— 195s 2s/step - accuracy: 0.7741 - loss: 0.5651 - val
_accuracy: 0.7550 - val_loss: 0.6591 - learning_rate: 5.0000e-05
Epoch 13/20
125/125 ————— 194s 2s/step - accuracy: 0.7984 - loss: 0.5376 - val
_accuracy: 0.7670 - val_loss: 0.6512 - learning_rate: 5.0000e-05
Epoch 14/20
125/125 ————— 195s 2s/step - accuracy: 0.7941 - loss: 0.5328 - val
_accuracy: 0.7380 - val_loss: 0.7073 - learning_rate: 5.0000e-05
Epoch 15/20
125/125 ————— 184s 1s/step - accuracy: 0.8148 - loss: 0.4878 - val
_accuracy: 0.7100 - val_loss: 0.7696 - learning_rate: 5.0000e-05
Epoch 16/20
125/125 ————— 218s 2s/step - accuracy: 0.8203 - loss: 0.4797 - val
_accuracy: 0.7600 - val_loss: 0.6550 - learning_rate: 5.0000e-05
Epoch 17/20
125/125 ————— 184s 1s/step - accuracy: 0.8392 - loss: 0.4371 - val
_accuracy: 0.7530 - val_loss: 0.6345 - learning_rate: 2.5000e-05
Epoch 18/20
125/125 ————— 130s 1s/step - accuracy: 0.8335 - loss: 0.4379 - val
_accuracy: 0.7560 - val_loss: 0.6773 - learning_rate: 2.5000e-05
Epoch 19/20
125/125 ————— 186s 1s/step - accuracy: 0.8540 - loss: 0.3892 - val
_accuracy: 0.7460 - val_loss: 0.6767 - learning_rate: 2.5000e-05
Epoch 20/20
125/125 ————— 173s 1s/step - accuracy: 0.8503 - loss: 0.3932 - val
_accuracy: 0.7600 - val_loss: 0.6402 - learning_rate: 2.5000e-05

32/32 ————— 45s 1s/step - accuracy: 0.6618 - loss: 0.8637
Test accuracy: 0.760

Confusion Matrix Medium (5 random)

```
In [8]: import seaborn as sns
# from sklearn.metrics import confusion
import numpy as NP
from sklearn.metrics import confusion_matrix

cnn_preds = np.argmax(model.predict(X_test), axis=1)
cnn_cm = confusion_matrix(y_test, cnn_preds)

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cnn_cm, annot=True, fmt="d", cmap="Blues", xticklabels=GENRES, ytick
plt.title("CNN Confusion Matrix")
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()
```

32/32 ————— 38s 1s/step

