

CNN for Chromagram only (3 secs)

1 - All 10

```
In [2]: 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', 'rock', 'soul', 'swing']
FILE_PATH = os.path.join('Data', 'chromagrams (3 secs)', 'chromagram_24')

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, [36, 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(36, (3, 3), activation='relu', input_shape=(36, 256, 1)),
    MaxPooling2D((2, 2)),

    Flatten(),

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

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

    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])

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

# Learning rate adjustment
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3, min_lr=1e-5)

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

# 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 ————— 42s 162ms/step - accuracy: 0.1079 - loss: 2.3649 - val_accuracy: 0.0650 - val_loss: 2.3046 - learning_rate: 1.0000e-04

Epoch 2/20
250/250 ————— 41s 164ms/step - accuracy: 0.1102 - loss: 2.3001 - val_accuracy: 0.1345 - val_loss: 2.2949 - learning_rate: 1.0000e-04

Epoch 3/20
250/250 ————— 42s 169ms/step - accuracy: 0.1266 - loss: 2.2759 - val_accuracy: 0.1615 - val_loss: 2.2078 - learning_rate: 1.0000e-04

Epoch 4/20
250/250 ————— 42s 169ms/step - accuracy: 0.1492 - loss: 2.2123 - val_accuracy: 0.1450 - val_loss: 2.1427 - learning_rate: 1.0000e-04

Epoch 5/20
250/250 ————— 43s 173ms/step - accuracy: 0.1565 - loss: 2.1633 - val_accuracy: 0.1515 - val_loss: 2.1101 - learning_rate: 1.0000e-04

Epoch 6/20
250/250 ————— 43s 171ms/step - accuracy: 0.1719 - loss: 2.1265 - val_accuracy: 0.1605 - val_loss: 2.1077 - learning_rate: 1.0000e-04

Epoch 7/20
250/250 ————— 42s 170ms/step - accuracy: 0.1886 - loss: 2.0928 - val_accuracy: 0.1575 - val_loss: 2.0663 - learning_rate: 1.0000e-04

Epoch 8/20
250/250 ————— 43s 171ms/step - accuracy: 0.1777 - loss: 2.0898 - val_accuracy: 0.1555 - val_loss: 2.0363 - learning_rate: 1.0000e-04

Epoch 9/20
250/250 ————— 43s 170ms/step - accuracy: 0.1956 - loss: 2.0554 - val_accuracy: 0.1565 - val_loss: 2.0229 - learning_rate: 1.0000e-04

Epoch 10/20
250/250 ————— 43s 173ms/step - accuracy: 0.2079 - loss: 2.0360 - val_accuracy: 0.1670 - val_loss: 2.0210 - learning_rate: 1.0000e-04

Epoch 11/20
250/250 ————— 55s 219ms/step - accuracy: 0.2241 - loss: 2.0236 - val_accuracy: 0.1830 - val_loss: 2.0137 - learning_rate: 1.0000e-04

Epoch 12/20
250/250 ————— 69s 275ms/step - accuracy: 0.2184 - loss: 2.0014 - val_accuracy: 0.1525 - val_loss: 2.0175 - learning_rate: 1.0000e-04

Epoch 13/20
250/250 ————— 81s 270ms/step - accuracy: 0.2303 - loss: 1.9731 - val_accuracy: 0.2130 - val_loss: 2.0019 - learning_rate: 1.0000e-04

Epoch 14/20
250/250 ————— 68s 273ms/step - accuracy: 0.2414 - loss: 1.9726 - val_accuracy: 0.2025 - val_loss: 1.9999 - learning_rate: 1.0000e-04

Epoch 15/20
250/250 ————— 68s 272ms/step - accuracy: 0.2372 - loss: 1.9671 - val_accuracy: 0.2090 - val_loss: 1.9907 - learning_rate: 1.0000e-04

Epoch 16/20
250/250 ————— 68s 270ms/step - accuracy: 0.2483 - loss: 1.9308 - val_accuracy: 0.1680 - val_loss: 2.0225 - learning_rate: 1.0000e-04

Epoch 17/20
250/250 ————— 62s 249ms/step - accuracy: 0.2428 - loss: 1.9146 - val_accuracy: 0.1990 - val_loss: 1.9943 - learning_rate: 1.0000e-04

Epoch 18/20
250/250 ————— 67s 267ms/step - accuracy: 0.2420 - loss: 1.9104 - val_accuracy: 0.1730 - val_loss: 2.0178 - learning_rate: 1.0000e-04

Epoch 19/20
250/250 ————— 67s 270ms/step - accuracy: 0.2557 - loss: 1.8979 - val_accuracy: 0.2060 - val_loss: 1.9866 - learning_rate: 5.0000e-05

Epoch 20/20
250/250 ————— 82s 270ms/step - accuracy: 0.2613 - loss: 1.8692 - val_accuracy: 0.2110 - val_loss: 1.9850 - learning_rate: 5.0000e-05

63/63 ————— 3s 41ms/step - accuracy: 0.2530 - loss: 1.9285
Test accuracy: 0.211

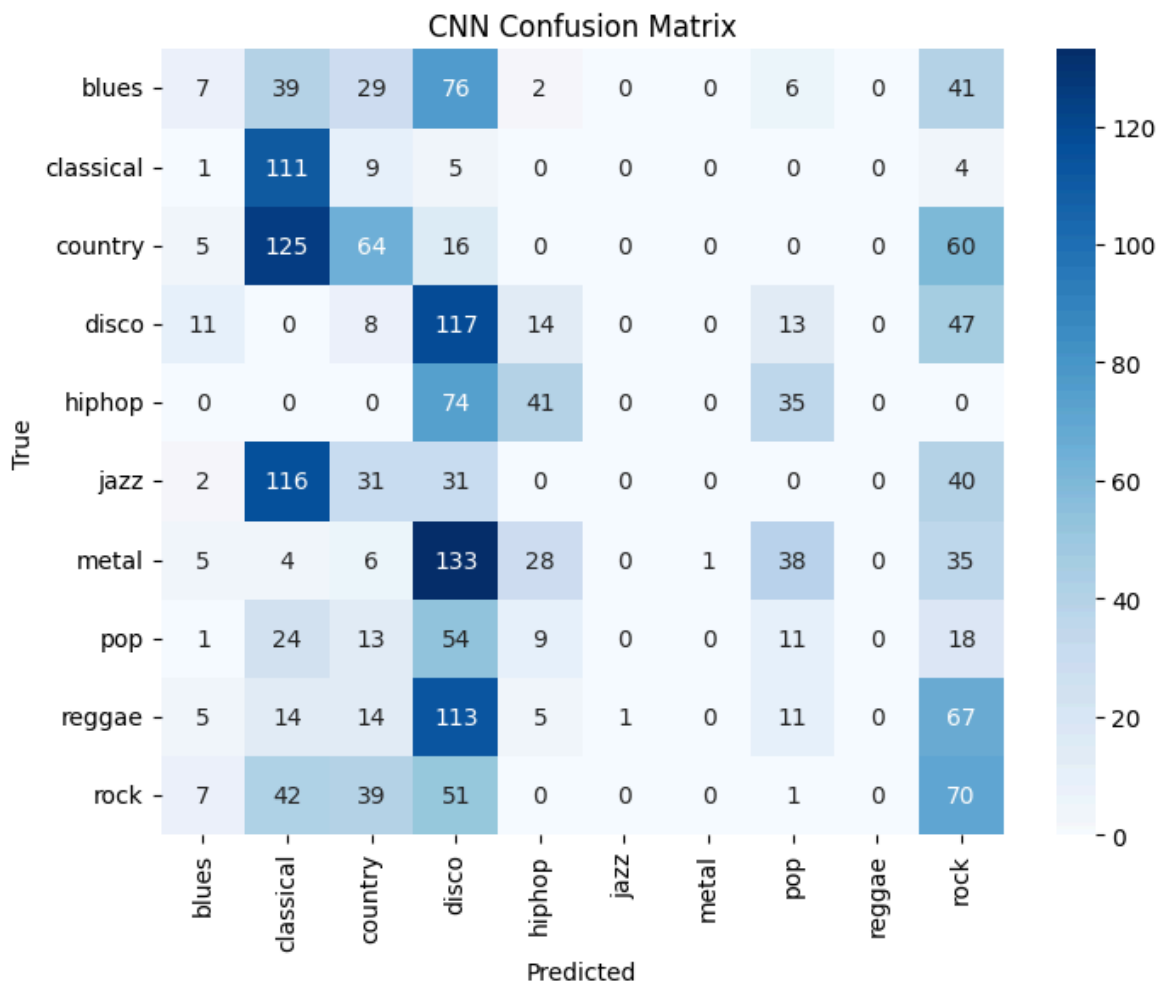
Apply the confusion matrix after the model

```
In [3]: 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 ————— 3s 40ms/step



2 - Limited Genres Easy (metal and classical)

```
In [4]: 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 = ['classical', 'metal']
FILE_PATH = os.path.join('Data', 'chromagrams (3 secs)', 'chromagram_24')

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, [36, 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(36, (3, 3), activation='relu', input_shape=(36, 256, 1)),
    MaxPooling2D((2, 2)),

    Flatten(),

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

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

    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])

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

# Learning rate adjustment
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3, min_lr=0.00001)

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

# 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` or `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  18s 288ms/step - accuracy: 0.4685 - loss: 1.2583 - val
_accuracy: 0.5250 - val_loss: 0.7284 - learning_rate: 1.0000e-04


Epoch 2/20
50/50  19s 251ms/step - accuracy: 0.5052 - loss: 0.9210 - val
_accuracy: 0.5250 - val_loss: 0.6701 - learning_rate: 1.0000e-04


Epoch 3/20
50/50  13s 253ms/step - accuracy: 0.5442 - loss: 0.7863 - val
_accuracy: 0.8325 - val_loss: 0.6324 - learning_rate: 1.0000e-04


Epoch 4/20
50/50  9s 185ms/step - accuracy: 0.6206 - loss: 0.6835 - val
_accuracy: 0.7600 - val_loss: 0.5715 - learning_rate: 1.0000e-04


Epoch 5/20
50/50  10s 196ms/step - accuracy: 0.6305 - loss: 0.6499 - val
_accuracy: 0.8700 - val_loss: 0.4766 - learning_rate: 1.0000e-04


Epoch 6/20
50/50  14s 284ms/step - accuracy: 0.7079 - loss: 0.5750 - val
_accuracy: 0.7875 - val_loss: 0.4645 - learning_rate: 1.0000e-04


Epoch 7/20
50/50  14s 280ms/step - accuracy: 0.7524 - loss: 0.5219 - val
_accuracy: 0.8425 - val_loss: 0.3902 - learning_rate: 1.0000e-04


Epoch 8/20
50/50  14s 278ms/step - accuracy: 0.7869 - loss: 0.4822 - val
_accuracy: 0.8450 - val_loss: 0.3573 - learning_rate: 1.0000e-04


Epoch 9/20
50/50  14s 272ms/step - accuracy: 0.7985 - loss: 0.4485 - val
_accuracy: 0.9125 - val_loss: 0.2661 - learning_rate: 1.0000e-04


Epoch 10/20
50/50  14s 271ms/step - accuracy: 0.8436 - loss: 0.3833 - val
_accuracy: 0.8850 - val_loss: 0.2751 - learning_rate: 1.0000e-04


Epoch 11/20
50/50  21s 270ms/step - accuracy: 0.8797 - loss: 0.3248 - val
_accuracy: 0.9175 - val_loss: 0.2248 - learning_rate: 1.0000e-04


Epoch 12/20
50/50  14s 277ms/step - accuracy: 0.8783 - loss: 0.3136 - val
_accuracy: 0.9525 - val_loss: 0.1697 - learning_rate: 1.0000e-04


Epoch 13/20
50/50  14s 269ms/step - accuracy: 0.8937 - loss: 0.2834 - val
_accuracy: 0.9275 - val_loss: 0.1911 - learning_rate: 1.0000e-04


Epoch 14/20
50/50  14s 274ms/step - accuracy: 0.9227 - loss: 0.2193 - val
_accuracy: 0.9075 - val_loss: 0.2386 - learning_rate: 1.0000e-04


Epoch 15/20
50/50  14s 276ms/step - accuracy: 0.9211 - loss: 0.2364 - val
_accuracy: 0.9550 - val_loss: 0.1361 - learning_rate: 1.0000e-04

Epoch 16/20
50/50  14s 270ms/step - accuracy: 0.9467 - loss: 0.1834 - val
_accuracy: 0.9250 - val_loss: 0.1867 - learning_rate: 1.0000e-04

Epoch 17/20
50/50  21s 272ms/step - accuracy: 0.9550 - loss: 0.1552 - val
_accuracy: 0.9225 - val_loss: 0.1751 - learning_rate: 1.0000e-04

Epoch 18/20
50/50  14s 277ms/step - accuracy: 0.9565 - loss: 0.1577 - val
_accuracy: 0.9575 - val_loss: 0.1191 - learning_rate: 1.0000e-04

Epoch 19/20
50/50  14s 268ms/step - accuracy: 0.9458 - loss: 0.1636 - val
_accuracy: 0.9600 - val_loss: 0.1198 - learning_rate: 1.0000e-04

Epoch 20/20
50/50  14s 274ms/step - accuracy: 0.9579 - loss: 0.1396 - val
_accuracy: 0.9400 - val_loss: 0.1424 - learning_rate: 1.0000e-04

13/13 ————— 1s 42ms/step - accuracy: 0.9692 - loss: 0.0876
Test accuracy: 0.940

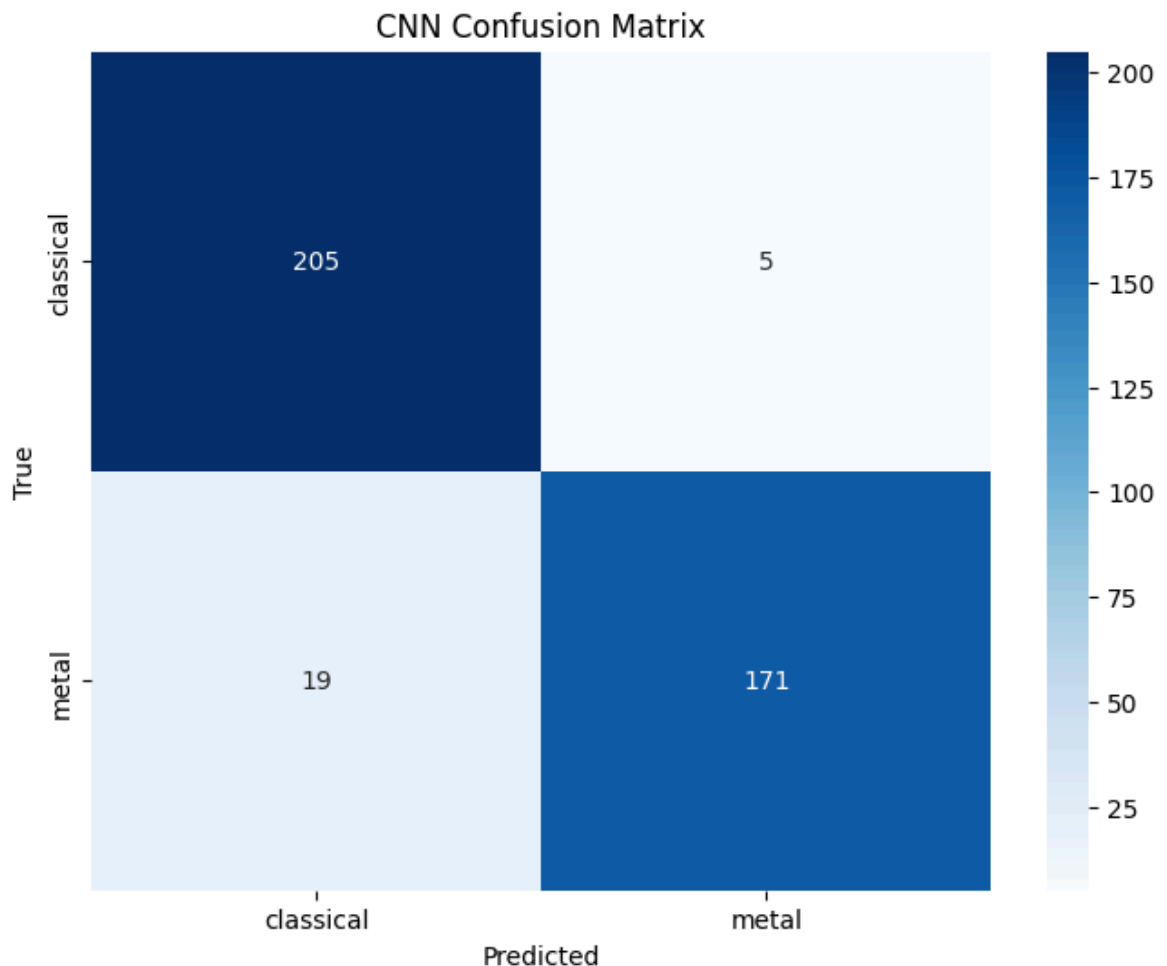
Confusion Matrix Easy (classical and metal)

```
In [5]: 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 ————— 1s 55ms/step



3 - Limited genres Hard (disco and pop)

```
In [6]: 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 = ['disco', 'pop']
FILE_PATH = os.path.join('Data', 'chromagrams (3 secs)', 'chromagram_24')

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, [36, 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(36, (3, 3), activation='relu', input_shape=(36, 256, 1)),
    MaxPooling2D((2, 2)),

    Flatten(),

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

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

    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])

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

# Learning rate adjustment
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3, min_lr=0.00001)

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

# 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  18s 280ms/step - accuracy: 0.4754 - loss: 1.3401 - val
_accuracy: 0.4750 - val_loss: 0.8067 - learning_rate: 1.0000e-04


Epoch 2/20
50/50  13s 259ms/step - accuracy: 0.4958 - loss: 0.9991 - val
_accuracy: 0.4750 - val_loss: 0.7786 - learning_rate: 1.0000e-04


Epoch 3/20
50/50  13s 250ms/step - accuracy: 0.5088 - loss: 0.8907 - val
_accuracy: 0.4750 - val_loss: 0.7690 - learning_rate: 1.0000e-04


Epoch 4/20
50/50  13s 264ms/step - accuracy: 0.5057 - loss: 0.8356 - val
_accuracy: 0.4750 - val_loss: 0.7412 - learning_rate: 1.0000e-04


Epoch 5/20
50/50  14s 269ms/step - accuracy: 0.5017 - loss: 0.8066 - val
_accuracy: 0.4750 - val_loss: 0.7471 - learning_rate: 1.0000e-04


Epoch 6/20
50/50  14s 276ms/step - accuracy: 0.4835 - loss: 0.8043 - val
_accuracy: 0.4750 - val_loss: 0.7376 - learning_rate: 1.0000e-04


Epoch 7/20
50/50  14s 272ms/step - accuracy: 0.5354 - loss: 0.7596 - val
_accuracy: 0.5250 - val_loss: 0.7120 - learning_rate: 1.0000e-04


Epoch 8/20
50/50  14s 272ms/step - accuracy: 0.5141 - loss: 0.7768 - val
_accuracy: 0.5325 - val_loss: 0.7108 - learning_rate: 1.0000e-04


Epoch 9/20
50/50  14s 274ms/step - accuracy: 0.5219 - loss: 0.7474 - val
_accuracy: 0.5250 - val_loss: 0.7150 - learning_rate: 1.0000e-04


Epoch 10/20
50/50  14s 273ms/step - accuracy: 0.5113 - loss: 0.7433 - val
_accuracy: 0.4750 - val_loss: 0.7269 - learning_rate: 1.0000e-04


Epoch 11/20
50/50  14s 278ms/step - accuracy: 0.5004 - loss: 0.7446 - val
_accuracy: 0.5250 - val_loss: 0.7066 - learning_rate: 1.0000e-04


Epoch 12/20
50/50  14s 271ms/step - accuracy: 0.5069 - loss: 0.7349 - val
_accuracy: 0.5250 - val_loss: 0.6997 - learning_rate: 1.0000e-04


Epoch 13/20
50/50  14s 275ms/step - accuracy: 0.5022 - loss: 0.7367 - val
_accuracy: 0.4750 - val_loss: 0.7105 - learning_rate: 1.0000e-04


Epoch 14/20
50/50  13s 267ms/step - accuracy: 0.5070 - loss: 0.7370 - val
_accuracy: 0.5250 - val_loss: 0.7046 - learning_rate: 1.0000e-04


Epoch 15/20
50/50  21s 277ms/step - accuracy: 0.5038 - loss: 0.7335 - val
_accuracy: 0.4750 - val_loss: 0.7074 - learning_rate: 1.0000e-04

Epoch 16/20
50/50  14s 273ms/step - accuracy: 0.5119 - loss: 0.7380 - val
_accuracy: 0.4950 - val_loss: 0.7050 - learning_rate: 5.0000e-05

Epoch 17/20
50/50  14s 276ms/step - accuracy: 0.4874 - loss: 0.7422 - val
_accuracy: 0.4750 - val_loss: 0.7102 - learning_rate: 5.0000e-05

Epoch 18/20
50/50  14s 269ms/step - accuracy: 0.4902 - loss: 0.7233 - val
_accuracy: 0.5250 - val_loss: 0.6977 - learning_rate: 5.0000e-05

Epoch 19/20
50/50  14s 275ms/step - accuracy: 0.4928 - loss: 0.7288 - val
_accuracy: 0.5250 - val_loss: 0.6998 - learning_rate: 5.0000e-05

Epoch 20/20
50/50  14s 275ms/step - accuracy: 0.4915 - loss: 0.7284 - val
_accuracy: 0.5250 - val_loss: 0.7024 - learning_rate: 5.0000e-05

13/13 ————— 1s 45ms/step - accuracy: 0.8098 - loss: 0.6693
Test accuracy: 0.525

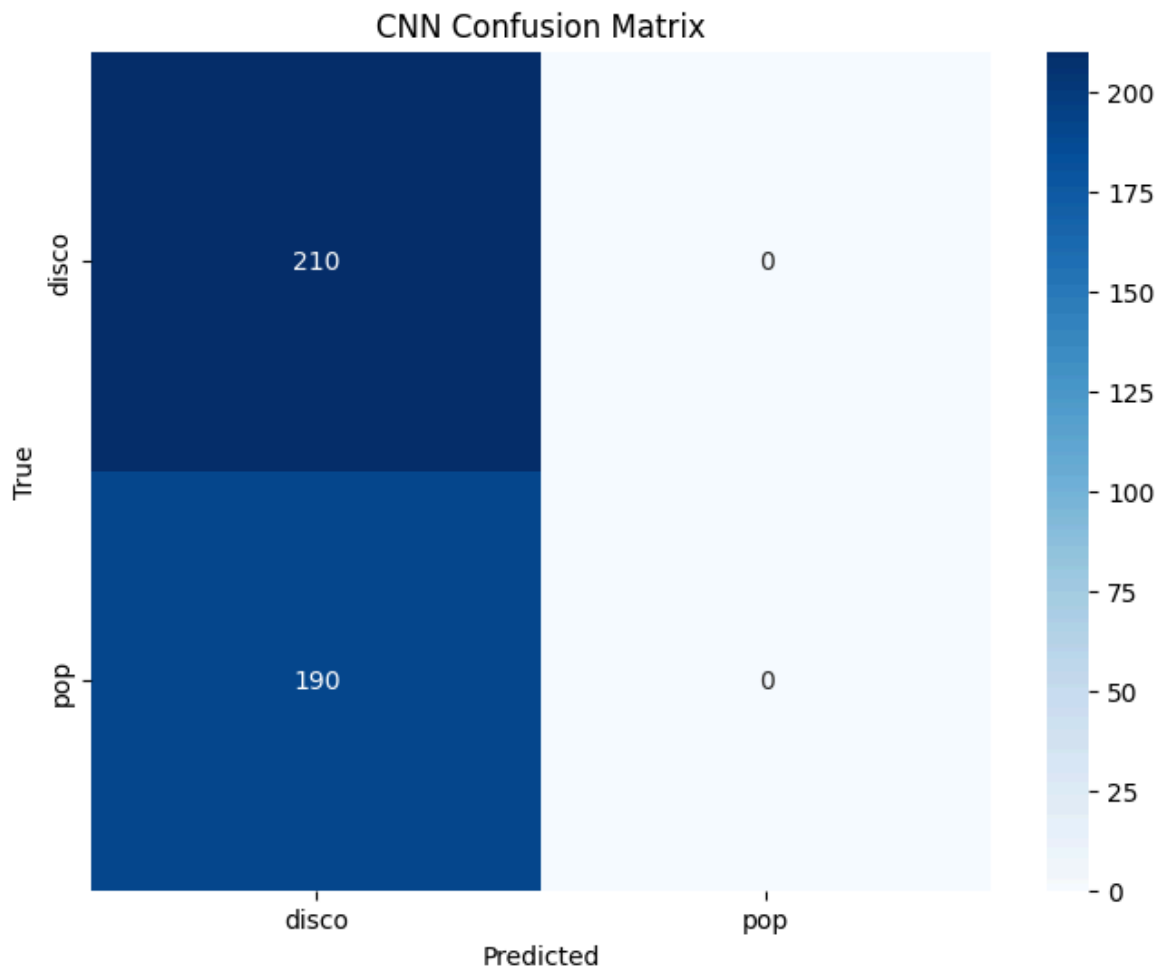
12 - Confusion Matrix Hard (disco and pop)

```
In [7]: 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 ————— 1s 53ms/step



13 - Limited Genres Medium (5 random)

```
In [8]: 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
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', 'rock', 'soul', 'swing']
GENRES = random.sample(GENRES, 5)
print(GENRES)
FILE_PATH = os.path.join('Data', 'chromagrams (3 secs)', 'chromagram_24')

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, [36, 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(36, (3, 3), activation='relu', input_shape=(36, 256, 1)),
    MaxPooling2D((2, 2)),

    Flatten(),

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

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

    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])

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

# Learning rate adjustment
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3, min_lr=1e-5)

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

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

```

```

['hiphop', 'jazz', 'classical', 'metal', 'rock']
Processing genre: hiphop
Processing genre: jazz
Processing genre: classical
Processing genre: metal
Processing genre: rock
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 ————— 39s 272ms/step - accuracy: 0.1908 - loss: 2.0743 - val_accuracy: 0.2820 - val_loss: 1.7062 - learning_rate: 1.0000e-04
Epoch 2/20
125/125 ————— 34s 272ms/step - accuracy: 0.2175 - loss: 1.7769 - val_accuracy: 0.3650 - val_loss: 1.6216 - learning_rate: 1.0000e-04
Epoch 3/20
125/125 ————— 41s 270ms/step - accuracy: 0.2336 - loss: 1.7115 - val_accuracy: 0.3040 - val_loss: 1.5988 - learning_rate: 1.0000e-04
Epoch 4/20
125/125 ————— 34s 272ms/step - accuracy: 0.2686 - loss: 1.6200 - val_accuracy: 0.4140 - val_loss: 1.4314 - learning_rate: 1.0000e-04
Epoch 5/20
125/125 ————— 34s 272ms/step - accuracy: 0.3031 - loss: 1.5191 - val_accuracy: 0.4100 - val_loss: 1.3329 - learning_rate: 1.0000e-04
Epoch 6/20
125/125 ————— 34s 271ms/step - accuracy: 0.3485 - loss: 1.4149 - val_accuracy: 0.4620 - val_loss: 1.2562 - learning_rate: 1.0000e-04
Epoch 7/20
125/125 ————— 34s 271ms/step - accuracy: 0.3729 - loss: 1.3495 - val_accuracy: 0.4590 - val_loss: 1.1917 - learning_rate: 1.0000e-04
Epoch 8/20
125/125 ————— 33s 261ms/step - accuracy: 0.4249 - loss: 1.2639 - val_accuracy: 0.5370 - val_loss: 1.1303 - learning_rate: 1.0000e-04
Epoch 9/20
125/125 ————— 42s 266ms/step - accuracy: 0.4425 - loss: 1.2115 - val_accuracy: 0.5540 - val_loss: 1.0905 - learning_rate: 1.0000e-04
Epoch 10/20
125/125 ————— 34s 271ms/step - accuracy: 0.4462 - loss: 1.1954 - val_accuracy: 0.5160 - val_loss: 1.1033 - learning_rate: 1.0000e-04
Epoch 11/20
125/125 ————— 34s 269ms/step - accuracy: 0.4735 - loss: 1.1514 - val_accuracy: 0.5390 - val_loss: 1.0504 - learning_rate: 1.0000e-04
Epoch 12/20
125/125 ————— 34s 270ms/step - accuracy: 0.4794 - loss: 1.1441 - val_accuracy: 0.5470 - val_loss: 1.0685 - learning_rate: 1.0000e-04
Epoch 13/20
125/125 ————— 34s 269ms/step - accuracy: 0.5001 - loss: 1.1069 - val_accuracy: 0.5650 - val_loss: 1.0612 - learning_rate: 1.0000e-04
Epoch 14/20
125/125 ————— 34s 271ms/step - accuracy: 0.5060 - loss: 1.0825 - val_accuracy: 0.5610 - val_loss: 1.0631 - learning_rate: 1.0000e-04
Epoch 15/20
125/125 ————— 34s 272ms/step - accuracy: 0.5262 - loss: 1.0715 - val_accuracy: 0.5620 - val_loss: 1.0227 - learning_rate: 5.0000e-05
Epoch 16/20
125/125 ————— 34s 267ms/step - accuracy: 0.5422 - loss: 1.0616 - val_accuracy: 0.5670 - val_loss: 1.0359 - learning_rate: 5.0000e-05
Epoch 17/20
125/125 ————— 34s 268ms/step - accuracy: 0.5585 - loss: 1.0175 - val_accuracy: 0.5520 - val_loss: 1.0348 - learning_rate: 5.0000e-05
Epoch 18/20
125/125 ————— 33s 266ms/step - accuracy: 0.5510 - loss: 1.0130 - val_accuracy: 0.5810 - val_loss: 1.0291 - learning_rate: 5.0000e-05
Epoch 19/20
125/125 ————— 34s 267ms/step - accuracy: 0.5654 - loss: 0.9978 - val_accuracy: 0.5760 - val_loss: 1.0427 - learning_rate: 2.5000e-05
Epoch 20/20
125/125 ————— 33s 267ms/step - accuracy: 0.5659 - loss: 0.9990 - val_accuracy: 0.5700 - val_loss: 1.0385 - learning_rate: 2.5000e-05

32/32 ————— 1s 39ms/step - accuracy: 0.6694 - loss: 0.9135
Test accuracy: 0.570

14 - Confusion Matrix Medium (5 random)

```
In [9]: 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 ————— 1s 40ms/step

