Mel-Spectrogram-Only CNN

March 23, 2025

1 CNN for Mel-Spectrogram (30 secs)

1.1 1 - All the imports

[1]:
```
import os
import numpy as np
from sklearn.model_selection import train_test_split
import tensorflow as tf
```

```
2025-03-23 00:55:33.570310: I tensorflow/core/platform/cpu_feature_guard.cc:210]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild
TensorFlow with the appropriate compiler flags.
```

1.2 2 - Put the data within the model

[2]:
```
# Import a single image and save it to be read by the model

image = os.path.join('blues.00000.png')

# Load the image
image = tf.io.read_file(image)

# Convert to a numpy array
image = tf.image.decode_png(image, channels=1)
image = tf.image.convert_image_dtype(image, tf.float32)
image = tf.image.resize(image, [256, 256])
image = image.numpy()
```

2 3 - Create the model

[3]:
```
from tensorflow.keras import models
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
 ↪Dropout, Normalization

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(10, activation='softmax')
])
```

```
/opt/conda/lib/python3.12/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)
```

[4]:
```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 254, 254, 32)	320
normalization (Normalization)	(None, 254, 254, 32)	65
max_pooling2d (MaxPooling2D)	(None, 127, 127, 32)	0
conv2d_1 (Conv2D)	(None, 125, 125, 64)	18,496

normalization_1 (Normalization)	(None, 125, 125, 64)	129
max_pooling2d_1 (MaxPooling2D)	(None, 62, 62, 64)	0
conv2d_2 (Conv2D)	(None, 60, 60, 128)	73,856
normalization_2 (Normalization)	(None, 60, 60, 128)	257
max_pooling2d_2 (MaxPooling2D)	(None, 30, 30, 128)	0
conv2d_3 (Conv2D)	(None, 28, 28, 256)	295,168
normalization_3 (Normalization)	(None, 28, 28, 256)	513
max_pooling2d_3 (MaxPooling2D)	(None, 14, 14, 256)	0
flatten (Flatten)	(None, 50176)	0
dense (Dense)	(None, 512)	25,690,624
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 256)	131,328
dropout_1 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 128)	32,896
dense_3 (Dense)	(None, 10)	1,290

Total params: 26,244,942 (100.12 MB)

Trainable params: 26,243,978 (100.11 MB)

Non-trainable params: 964 (3.78 KB)

3 4 - Load the images

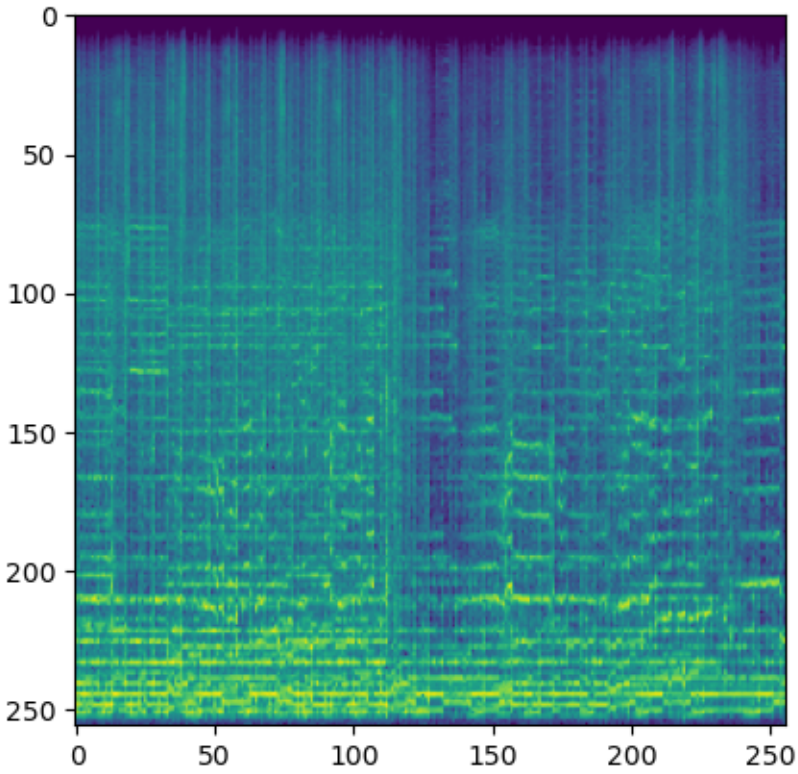
[5]:
```
import os
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
```

```python
# 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', 'mel_spectrograms', 'mel_spectrogram_512')
X = []
y = []

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

# Loop through the genres and load the images with augmentation
for genre in GENRES:
    genre_dir = os.path.join(FILE_PATH, genre)
    print(f"Going through {genre}")
    for file in os.listdir(genre_dir):
        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
        X.append(image)
        y.append(GENRE_TO_INDEX[genre])

# Convert lists to numpy arrays
X = np.array(X)
y = np.array(y)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

```
Going through blues
Going through classical
Going through country
Going through disco
Going through hiphop
Going through jazz
Going through metal
Going through pop
Going through reggae
```

```
Going through rock
```

[6]:
```
# Show image as a sanity check
import matplotlib.pyplot as plt
plt.imshow(X_train[22].reshape(256, 256))
plt.show()
```



3.1 5 - Compile the model

[7]:
```
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau

model.compile(optimizer=Adam(learning_rate=0.0001),␣
 ↪loss='sparse_categorical_crossentropy', metrics=['accuracy'])
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3,␣
 ↪min_lr=1e-6)
```

3.2 6 - Fit the model

[8]:
```
model.fit(X_train, y_train, epochs=20, validation_data=(X_test, y_test),␣
↪batch_size=32, callbacks=[reduce_lr])
```

```
Epoch 1/20
25/25             32s 1s/step -
accuracy: 0.1009 - loss: 2.3071 - val_accuracy: 0.1050 - val_loss: 2.3032 -
learning_rate: 1.0000e-04
Epoch 2/20
25/25             45s 2s/step -
accuracy: 0.1131 - loss: 2.3045 - val_accuracy: 0.0650 - val_loss: 2.2942 -
learning_rate: 1.0000e-04
Epoch 3/20
25/25             55s 2s/step -
accuracy: 0.1077 - loss: 2.2757 - val_accuracy: 0.1200 - val_loss: 2.2674 -
learning_rate: 1.0000e-04
Epoch 4/20
25/25             48s 2s/step -
accuracy: 0.1655 - loss: 2.2419 - val_accuracy: 0.1150 - val_loss: 2.2195 -
learning_rate: 1.0000e-04
Epoch 5/20
25/25             81s 2s/step -
accuracy: 0.1746 - loss: 2.1960 - val_accuracy: 0.1250 - val_loss: 2.1825 -
learning_rate: 1.0000e-04
Epoch 6/20
25/25             48s 2s/step -
accuracy: 0.2115 - loss: 2.1497 - val_accuracy: 0.1350 - val_loss: 2.1622 -
learning_rate: 1.0000e-04
Epoch 7/20
25/25             53s 2s/step -
accuracy: 0.1883 - loss: 2.1048 - val_accuracy: 0.1650 - val_loss: 2.1223 -
learning_rate: 1.0000e-04
Epoch 8/20
25/25             58s 2s/step -
accuracy: 0.1987 - loss: 2.1070 - val_accuracy: 0.2000 - val_loss: 2.0434 -
learning_rate: 1.0000e-04
Epoch 9/20
25/25             68s 2s/step -
accuracy: 0.2206 - loss: 1.9999 - val_accuracy: 0.2150 - val_loss: 1.9882 -
learning_rate: 1.0000e-04
Epoch 10/20
25/25             93s 2s/step -
accuracy: 0.2194 - loss: 1.9881 - val_accuracy: 0.3050 - val_loss: 2.0044 -
learning_rate: 1.0000e-04
Epoch 11/20
25/25             84s 2s/step -
accuracy: 0.2297 - loss: 2.0183 - val_accuracy: 0.2950 - val_loss: 1.9488 -
```

```
learning_rate: 1.0000e-04
Epoch 12/20
25/25             48s 2s/step -
accuracy: 0.2491 - loss: 1.9434 - val_accuracy: 0.3300 - val_loss: 1.9078 -
learning_rate: 1.0000e-04
Epoch 13/20
25/25             48s 2s/step -
accuracy: 0.2793 - loss: 1.9322 - val_accuracy: 0.2800 - val_loss: 1.9159 -
learning_rate: 1.0000e-04
Epoch 14/20
25/25             49s 2s/step -
accuracy: 0.2881 - loss: 1.9570 - val_accuracy: 0.3450 - val_loss: 1.8023 -
learning_rate: 1.0000e-04
Epoch 15/20
25/25             81s 2s/step -
accuracy: 0.3074 - loss: 1.8703 - val_accuracy: 0.3050 - val_loss: 1.8407 -
learning_rate: 1.0000e-04
Epoch 16/20
25/25             49s 2s/step -
accuracy: 0.2956 - loss: 1.8775 - val_accuracy: 0.4100 - val_loss: 1.7067 -
learning_rate: 1.0000e-04
Epoch 17/20
25/25             61s 2s/step -
accuracy: 0.3531 - loss: 1.7673 - val_accuracy: 0.4600 - val_loss: 1.5606 -
learning_rate: 1.0000e-04
Epoch 18/20
25/25             82s 2s/step -
accuracy: 0.3407 - loss: 1.7063 - val_accuracy: 0.4100 - val_loss: 1.6352 -
learning_rate: 1.0000e-04
Epoch 19/20
25/25             80s 2s/step -
accuracy: 0.3800 - loss: 1.6886 - val_accuracy: 0.4700 - val_loss: 1.5111 -
learning_rate: 1.0000e-04
Epoch 20/20
25/25             60s 2s/step -
accuracy: 0.4093 - loss: 1.6223 - val_accuracy: 0.4600 - val_loss: 1.4881 -
learning_rate: 1.0000e-04
```

[8]:
```
<keras.src.callbacks.history.History at 0x7f4f384a9f70>
```

3.3 7 - Check the accuracy

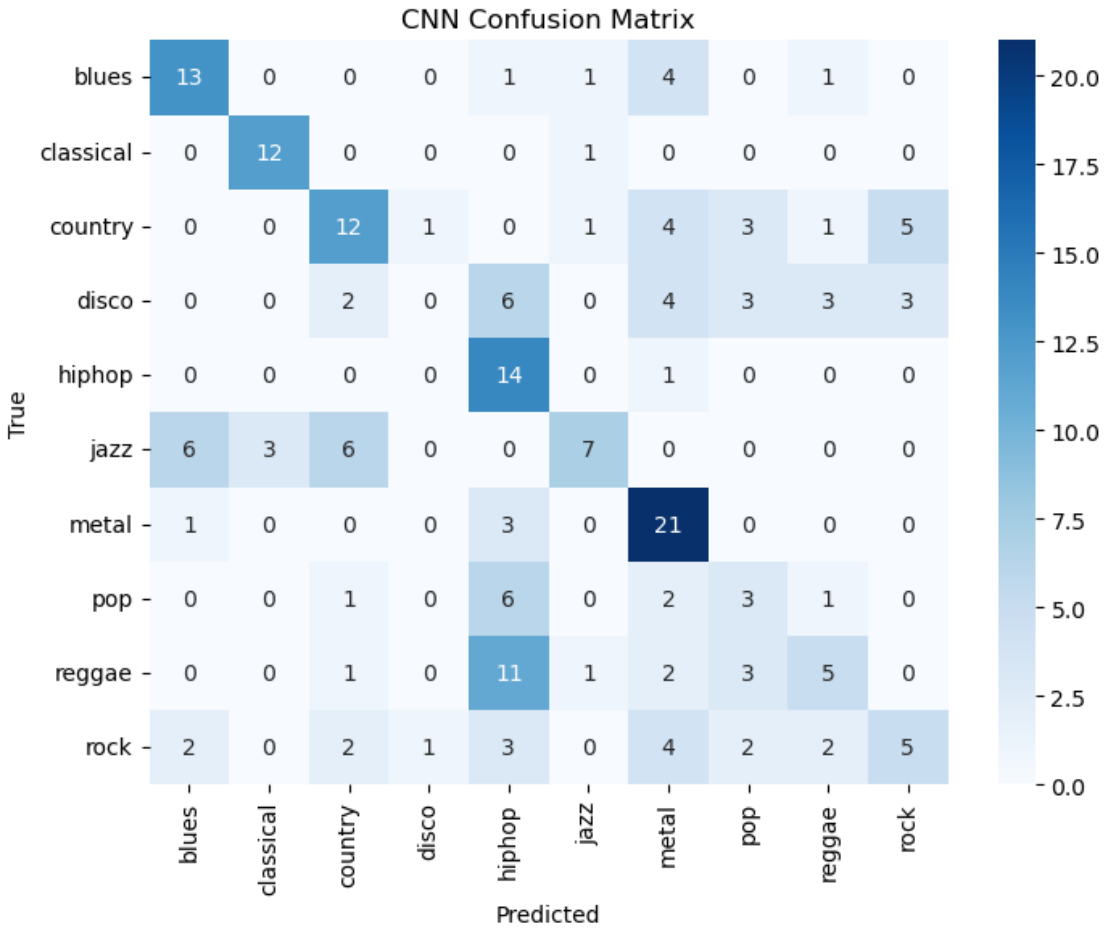
[9]:
```
evaluation = model.evaluate(X_test, y_test)
print(f"Test accuracy: {evaluation[1]:.3f}")
```

```
7/7             3s 474ms/step -
accuracy: 0.4671 - loss: 1.4759
Test accuracy: 0.460
```

4 8 - Apply the confusion matrix after the model

[10]:
```
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                 4s 519ms/step
```



4.1 9 - Limited Genres Easy (metal and classical)

[11]:
```
import os
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split

# 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', 'mel_spectrogram_512')
X = []
y = []

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

# Loop through the genres and load the images with augmentation
for genre in GENRES:
    genre_dir = os.path.join(FILE_PATH, genre)
    print(f"Going through {genre}")
    for file in os.listdir(genre_dir):
        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
        X.append(image)
        y.append(GENRE_TO_INDEX[genre])

# Convert lists to numpy arrays
X = np.array(X)
y = np.array(y)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

```
from tensorflow.keras import models
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
 ↪Dropout, Normalization

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(10, activation='softmax')
])

from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau

model.compile(optimizer=Adam(learning_rate=0.0001),␣
 ↪loss='sparse_categorical_crossentropy', metrics=['accuracy'])
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3,␣
 ↪min_lr=1e-6)

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

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

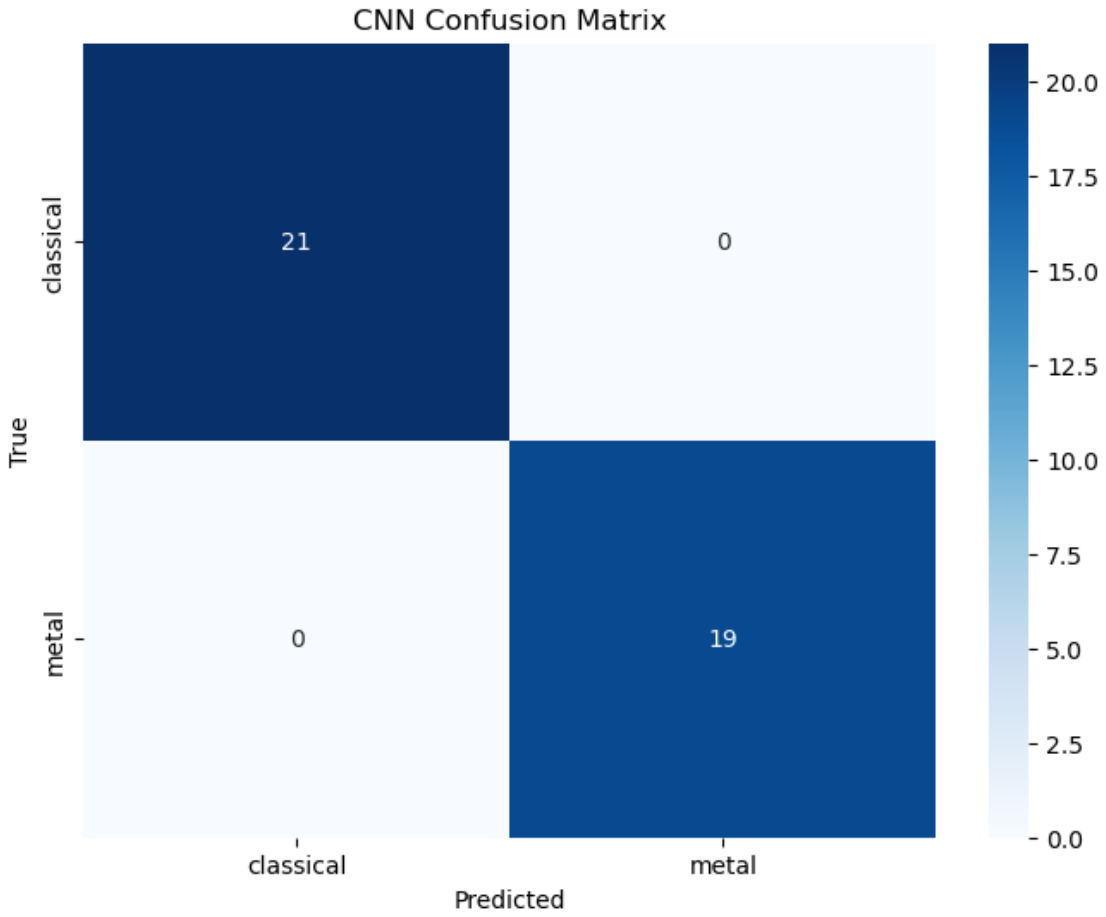
```
Going through classical
Going through metal

/opt/conda/lib/python3.12/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
5/5            21s 3s/step -
accuracy: 0.3814 - loss: 2.1715 - val_accuracy: 0.4750 - val_loss: 1.5851 -
learning_rate: 1.0000e-04
Epoch 2/20
5/5            18s 2s/step -
accuracy: 0.5536 - loss: 1.4823 - val_accuracy: 0.4750 - val_loss: 0.8452 -
learning_rate: 1.0000e-04
Epoch 3/20
5/5            9s 2s/step -
accuracy: 0.5367 - loss: 0.9851 - val_accuracy: 0.8750 - val_loss: 0.6405 -
learning_rate: 1.0000e-04
Epoch 4/20
5/5            11s 2s/step -
accuracy: 0.5702 - loss: 0.9281 - val_accuracy: 0.6500 - val_loss: 0.5933 -
learning_rate: 1.0000e-04
Epoch 5/20
5/5            11s 2s/step -
accuracy: 0.5595 - loss: 0.8349 - val_accuracy: 1.0000 - val_loss: 0.5213 -
learning_rate: 1.0000e-04
Epoch 6/20
5/5            9s 2s/step -
accuracy: 0.6051 - loss: 0.7666 - val_accuracy: 0.9500 - val_loss: 0.4772 -
learning_rate: 1.0000e-04
Epoch 7/20
5/5            12s 2s/step -
accuracy: 0.6544 - loss: 0.7249 - val_accuracy: 1.0000 - val_loss: 0.4232 -
learning_rate: 1.0000e-04
Epoch 8/20
5/5            13s 3s/step -
accuracy: 0.7857 - loss: 0.5776 - val_accuracy: 1.0000 - val_loss: 0.2940 -
learning_rate: 1.0000e-04
Epoch 9/20
5/5            12s 3s/step -
accuracy: 0.8736 - loss: 0.4341 - val_accuracy: 1.0000 - val_loss: 0.1949 -
learning_rate: 1.0000e-04
Epoch 10/20
5/5            12s 2s/step -
accuracy: 0.8816 - loss: 0.3539 - val_accuracy: 1.0000 - val_loss: 0.0828 -
```

```
learning_rate: 1.0000e-04
Epoch 11/20
5/5             12s 2s/step -
accuracy: 0.9264 - loss: 0.2630 - val_accuracy: 1.0000 - val_loss: 0.0487 -
learning_rate: 1.0000e-04
Epoch 12/20
5/5             11s 2s/step -
accuracy: 0.9379 - loss: 0.2221 - val_accuracy: 1.0000 - val_loss: 0.0517 -
learning_rate: 1.0000e-04
Epoch 13/20
5/5             12s 2s/step -
accuracy: 0.9032 - loss: 0.2736 - val_accuracy: 1.0000 - val_loss: 0.0176 -
learning_rate: 1.0000e-04
Epoch 14/20
5/5             20s 2s/step -
accuracy: 0.9328 - loss: 0.2625 - val_accuracy: 1.0000 - val_loss: 0.0539 -
learning_rate: 1.0000e-04
Epoch 15/20
5/5             12s 2s/step -
accuracy: 0.9535 - loss: 0.1512 - val_accuracy: 1.0000 - val_loss: 0.0172 -
learning_rate: 1.0000e-04
Epoch 16/20
5/5             12s 2s/step -
accuracy: 0.9732 - loss: 0.1117 - val_accuracy: 1.0000 - val_loss: 0.0191 -
learning_rate: 1.0000e-04
Epoch 17/20
5/5             21s 2s/step -
accuracy: 0.9271 - loss: 0.1641 - val_accuracy: 1.0000 - val_loss: 0.0211 -
learning_rate: 1.0000e-04
Epoch 18/20
5/5             12s 3s/step -
accuracy: 0.9593 - loss: 0.1287 - val_accuracy: 1.0000 - val_loss: 0.0136 -
learning_rate: 1.0000e-04
Epoch 19/20
5/5             12s 2s/step -
accuracy: 0.9538 - loss: 0.1272 - val_accuracy: 1.0000 - val_loss: 0.0194 -
learning_rate: 1.0000e-04
Epoch 20/20
5/5             12s 2s/step -
accuracy: 0.9541 - loss: 0.1200 - val_accuracy: 1.0000 - val_loss: 0.0093 -
learning_rate: 1.0000e-04
2/2             1s 154ms/step -
accuracy: 1.0000 - loss: 0.0094
Test accuracy: 1.000
```

4.2 10 - Confusion Matrix Easy (metal and classical)

[12]:
```
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 592ms/step
```



4.3 11 - Limited genres Hard (disco and pop)

[13]:
```
import os
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split

# 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', 'mel_spectrogram_512')
X = []
y = []

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

# Loop through the genres and load the images with augmentation
for genre in GENRES:
    genre_dir = os.path.join(FILE_PATH, genre)
    print(f"Going through {genre}")
    for file in os.listdir(genre_dir):
        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
        X.append(image)
        y.append(GENRE_TO_INDEX[genre])

# Convert lists to numpy arrays
X = np.array(X)
y = np.array(y)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

```
from tensorflow.keras import models
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
 ↪Dropout, Normalization

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(10, activation='softmax')
])

from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau

model.compile(optimizer=Adam(learning_rate=0.0001),␣
 ↪loss='sparse_categorical_crossentropy', metrics=['accuracy'])
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3,␣
 ↪min_lr=1e-6)

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

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

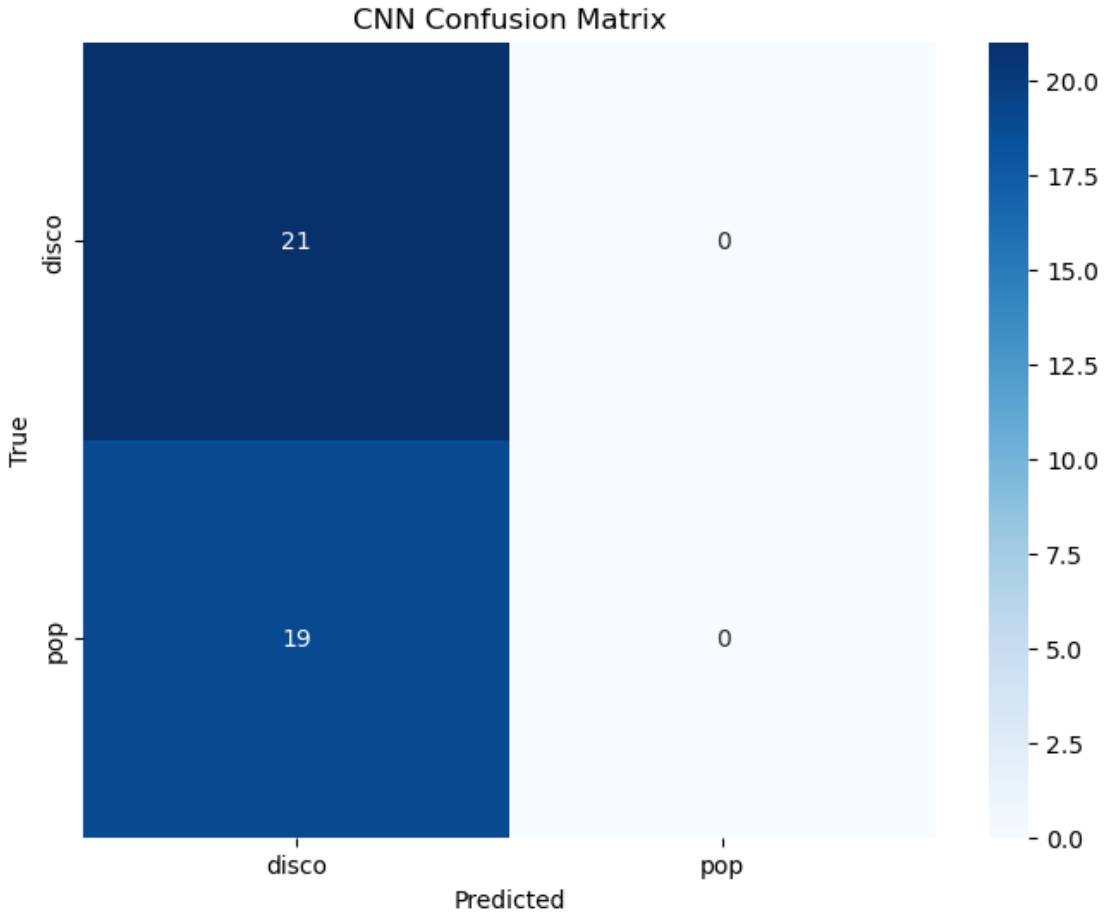
```
Going through disco
Going through pop

/opt/conda/lib/python3.12/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
5/5             20s 3s/step -
accuracy: 0.2543 - loss: 2.2439 - val_accuracy: 0.5250 - val_loss: 1.8512 -
learning_rate: 1.0000e-04
Epoch 2/20
5/5             12s 2s/step -
accuracy: 0.4489 - loss: 1.8033 - val_accuracy: 0.5250 - val_loss: 1.0793 -
learning_rate: 1.0000e-04
Epoch 3/20
5/5             20s 2s/step -
accuracy: 0.4817 - loss: 1.2467 - val_accuracy: 0.5250 - val_loss: 0.7399 -
learning_rate: 1.0000e-04
Epoch 4/20
5/5             21s 2s/step -
accuracy: 0.4322 - loss: 1.1953 - val_accuracy: 0.5250 - val_loss: 0.7234 -
learning_rate: 1.0000e-04
Epoch 5/20
5/5             20s 2s/step -
accuracy: 0.4735 - loss: 1.0931 - val_accuracy: 0.5250 - val_loss: 0.7446 -
learning_rate: 1.0000e-04
Epoch 6/20
5/5             21s 2s/step -
accuracy: 0.4704 - loss: 1.0034 - val_accuracy: 0.4750 - val_loss: 0.7971 -
learning_rate: 1.0000e-04
Epoch 7/20
5/5             20s 2s/step -
accuracy: 0.5594 - loss: 0.8497 - val_accuracy: 0.4750 - val_loss: 0.7784 -
learning_rate: 1.0000e-04
Epoch 8/20
5/5             20s 2s/step -
accuracy: 0.6276 - loss: 0.8849 - val_accuracy: 0.5250 - val_loss: 0.7482 -
learning_rate: 5.0000e-05
Epoch 9/20
5/5             9s 2s/step -
accuracy: 0.5175 - loss: 0.9417 - val_accuracy: 0.5250 - val_loss: 0.7358 -
learning_rate: 5.0000e-05
Epoch 10/20
5/5             9s 2s/step -
accuracy: 0.5569 - loss: 0.9137 - val_accuracy: 0.5250 - val_loss: 0.7334 -
```

```
learning_rate: 5.0000e-05
Epoch 11/20
5/5             9s 2s/step -
accuracy: 0.5324 - loss: 0.9449 - val_accuracy: 0.5250 - val_loss: 0.7375 -
learning_rate: 2.5000e-05
Epoch 12/20
5/5             9s 2s/step -
accuracy: 0.4923 - loss: 0.9043 - val_accuracy: 0.5250 - val_loss: 0.7445 -
learning_rate: 2.5000e-05
Epoch 13/20
5/5             9s 2s/step -
accuracy: 0.4416 - loss: 0.9265 - val_accuracy: 0.5250 - val_loss: 0.7475 -
learning_rate: 2.5000e-05
Epoch 14/20
5/5             11s 2s/step -
accuracy: 0.4013 - loss: 0.9905 - val_accuracy: 0.5250 - val_loss: 0.7471 -
learning_rate: 1.2500e-05
Epoch 15/20
5/5             12s 2s/step -
accuracy: 0.4714 - loss: 0.9219 - val_accuracy: 0.5250 - val_loss: 0.7454 -
learning_rate: 1.2500e-05
Epoch 16/20
5/5             21s 3s/step -
accuracy: 0.5437 - loss: 0.8973 - val_accuracy: 0.5250 - val_loss: 0.7434 -
learning_rate: 1.2500e-05
Epoch 17/20
5/5             20s 3s/step -
accuracy: 0.4978 - loss: 0.8638 - val_accuracy: 0.5250 - val_loss: 0.7418 -
learning_rate: 6.2500e-06
Epoch 18/20
5/5             12s 2s/step -
accuracy: 0.4437 - loss: 0.9025 - val_accuracy: 0.5250 - val_loss: 0.7393 -
learning_rate: 6.2500e-06
Epoch 19/20
5/5             21s 3s/step -
accuracy: 0.5152 - loss: 0.8571 - val_accuracy: 0.5250 - val_loss: 0.7370 -
learning_rate: 6.2500e-06
Epoch 20/20
5/5             11s 2s/step -
accuracy: 0.5513 - loss: 0.8578 - val_accuracy: 0.5250 - val_loss: 0.7362 -
learning_rate: 3.1250e-06
2/2             1s 221ms/step -
accuracy: 0.5375 - loss: 0.7290
Test accuracy: 0.525
```

4.4 12 - Confusion Matrix Hard (disco and pop)

[14]:
```
import seaborn as sns
# from sklearn.metrics import confusion
import numpy as NP
from sklearn.metrics import confusion_matrix

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

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

```
WARNING:tensorflow:5 out of the last 10 calls to <function
TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at
0x7f4e9c1d5e40> triggered tf.function retracing. Tracing is expensive and the
excessive number of tracings could be due to (1) creating @tf.function
repeatedly in a loop, (2) passing tensors with different shapes, (3) passing
Python objects instead of tensors. For (1), please define your @tf.function
outside of the loop. For (2), @tf.function has reduce_retracing=True option that
can avoid unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
1/2             0s
904ms/stepWARNING:tensorflow:6 out of the last 11 calls to <function
TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at
0x7f4e9c1d5e40> triggered tf.function retracing. Tracing is expensive and the
excessive number of tracings could be due to (1) creating @tf.function
repeatedly in a loop, (2) passing tensors with different shapes, (3) passing
Python objects instead of tensors. For (1), please define your @tf.function
outside of the loop. For (2), @tf.function has reduce_retracing=True option that
can avoid unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
2/2             2s 992ms/step
```



4.5 13 - Limited Genres Medium (5 random)

[15]:
```
import os
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
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',␣
 ↪'pop', 'reggae', 'rock']
```

```
GENRES = random.sample(GENRES, 5)
print(GENRES)

FILE_PATH = os.path.join('Data', 'mel_spectrograms', 'mel_spectrogram_512')
X = []
y = []

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

# Loop through the genres and load the images with augmentation
for genre in GENRES:
    genre_dir = os.path.join(FILE_PATH, genre)
    print(f"Going through {genre}")
    for file in os.listdir(genre_dir):
        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
        X.append(image)
        y.append(GENRE_TO_INDEX[genre])

# Convert lists to numpy arrays
X = np.array(X)
y = np.array(y)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

from tensorflow.keras import models
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
 ↪Dropout, Normalization

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(10, activation='softmax')
])

from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau

model.compile(optimizer=Adam(learning_rate=0.0001),␣
 ↪loss='sparse_categorical_crossentropy', metrics=['accuracy'])
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3,␣
 ↪min_lr=1e-6)

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

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

```
['reggae', 'hiphop', 'blues', 'classical', 'country']
Going through reggae
Going through hiphop
Going through blues
Going through classical
Going through country
```

```
/opt/conda/lib/python3.12/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
13/13             39s 2s/step -
accuracy: 0.1869 - loss: 2.1951 - val_accuracy: 0.2400 - val_loss: 1.7762 -
```

```
learning_rate: 1.0000e-04
Epoch 2/20
13/13             30s 2s/step -
accuracy: 0.2043 - loss: 1.8963 - val_accuracy: 0.1000 - val_loss: 1.7304 -
learning_rate: 1.0000e-04
Epoch 3/20
13/13             31s 2s/step -
accuracy: 0.1741 - loss: 1.8959 - val_accuracy: 0.1700 - val_loss: 1.7146 -
learning_rate: 1.0000e-04
Epoch 4/20
13/13             30s 2s/step -
accuracy: 0.2243 - loss: 1.7900 - val_accuracy: 0.3200 - val_loss: 1.6562 -
learning_rate: 1.0000e-04
Epoch 5/20
13/13             31s 2s/step -
accuracy: 0.2608 - loss: 1.7602 - val_accuracy: 0.2800 - val_loss: 1.6638 -
learning_rate: 1.0000e-04
Epoch 6/20
13/13             28s 2s/step -
accuracy: 0.2333 - loss: 1.7370 - val_accuracy: 0.3200 - val_loss: 1.6010 -
learning_rate: 1.0000e-04
Epoch 7/20
13/13             43s 2s/step -
accuracy: 0.2703 - loss: 1.6960 - val_accuracy: 0.3600 - val_loss: 1.5903 -
learning_rate: 1.0000e-04
Epoch 8/20
13/13             31s 2s/step -
accuracy: 0.2497 - loss: 1.6806 - val_accuracy: 0.3600 - val_loss: 1.4941 -
learning_rate: 1.0000e-04
Epoch 9/20
13/13             33s 2s/step -
accuracy: 0.2943 - loss: 1.6338 - val_accuracy: 0.3600 - val_loss: 1.4932 -
learning_rate: 1.0000e-04
Epoch 10/20
13/13             41s 2s/step -
accuracy: 0.3211 - loss: 1.5119 - val_accuracy: 0.3600 - val_loss: 1.4654 -
learning_rate: 1.0000e-04
Epoch 11/20
13/13             22s 2s/step -
accuracy: 0.3393 - loss: 1.5433 - val_accuracy: 0.4100 - val_loss: 1.3599 -
learning_rate: 1.0000e-04
Epoch 12/20
13/13             24s 2s/step -
accuracy: 0.3467 - loss: 1.4869 - val_accuracy: 0.3600 - val_loss: 1.3401 -
learning_rate: 1.0000e-04
Epoch 13/20
13/13             46s 2s/step -
accuracy: 0.3615 - loss: 1.4385 - val_accuracy: 0.4200 - val_loss: 1.2349 -
```

```
learning_rate: 1.0000e-04
Epoch 14/20
13/13             30s 2s/step -
accuracy: 0.4414 - loss: 1.3432 - val_accuracy: 0.5000 - val_loss: 1.1201 -
learning_rate: 1.0000e-04
Epoch 15/20
13/13             39s 2s/step -
accuracy: 0.4437 - loss: 1.2782 - val_accuracy: 0.3900 - val_loss: 1.2936 -
learning_rate: 1.0000e-04
Epoch 16/20
13/13             25s 2s/step -
accuracy: 0.3924 - loss: 1.3264 - val_accuracy: 0.4500 - val_loss: 1.1846 -
learning_rate: 1.0000e-04
Epoch 17/20
13/13             43s 2s/step -
accuracy: 0.5172 - loss: 1.1864 - val_accuracy: 0.5500 - val_loss: 1.0036 -
learning_rate: 1.0000e-04
Epoch 18/20
13/13             26s 2s/step -
accuracy: 0.4990 - loss: 1.1514 - val_accuracy: 0.3700 - val_loss: 1.2977 -
learning_rate: 1.0000e-04
Epoch 19/20
13/13             41s 2s/step -
accuracy: 0.4729 - loss: 1.2041 - val_accuracy: 0.6700 - val_loss: 0.9468 -
learning_rate: 1.0000e-04
Epoch 20/20
13/13             44s 2s/step -
accuracy: 0.5690 - loss: 1.0341 - val_accuracy: 0.6700 - val_loss: 0.8439 -
learning_rate: 1.0000e-04
4/4             2s 388ms/step -
accuracy: 0.6878 - loss: 0.8492
Test accuracy: 0.670
```

4.6 14 - Confusion Matrix Medium (5 random)

[16]:
```python
import seaborn as sns
# from sklearn.metrics import confusion
import numpy as NP
from sklearn.metrics import confusion_matrix

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

# Plot the confusion matrix
plt.figure(figsize=(8, 6))
sns.heatmap(cnn_cm, annot=True, fmt="d", cmap="Blues", xticklabels=GENRES,␣
 ↪yticklabels=GENRES)
```

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

```
4/4                 2s 528ms/step
```