Mel-Spectrogram-Only CNN

March 20, 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
```

```
2025-03-20 08:41:34.290348: 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

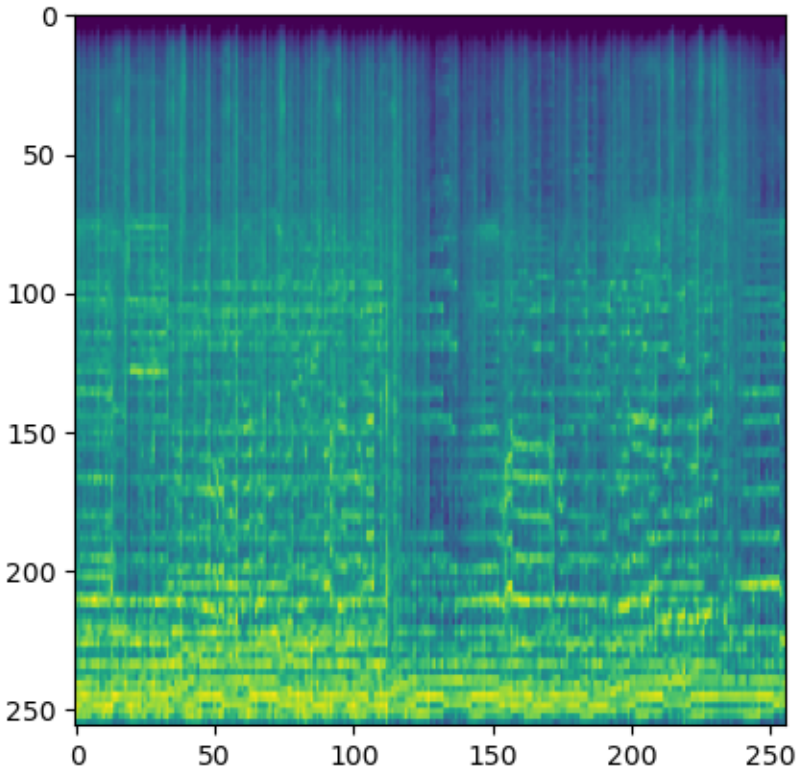
[5]:
```
import os
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
```

```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_128')
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             37s 1s/step -
accuracy: 0.1001 - loss: 2.3039 - val_accuracy: 0.0650 - val_loss: 2.3107 -
learning_rate: 1.0000e-04
Epoch 2/20
25/25             52s 2s/step -
accuracy: 0.1086 - loss: 2.3010 - val_accuracy: 0.0850 - val_loss: 2.3016 -
learning_rate: 1.0000e-04
Epoch 3/20
25/25             70s 3s/step -
accuracy: 0.1150 - loss: 2.2975 - val_accuracy: 0.0950 - val_loss: 2.2837 -
learning_rate: 1.0000e-04
Epoch 4/20
25/25             74s 3s/step -
accuracy: 0.1340 - loss: 2.2817 - val_accuracy: 0.1800 - val_loss: 2.2452 -
learning_rate: 1.0000e-04
Epoch 5/20
25/25             70s 3s/step -
accuracy: 0.1660 - loss: 2.2374 - val_accuracy: 0.1850 - val_loss: 2.1815 -
learning_rate: 1.0000e-04
Epoch 6/20
25/25             72s 3s/step -
accuracy: 0.1976 - loss: 2.1658 - val_accuracy: 0.2850 - val_loss: 2.1015 -
learning_rate: 1.0000e-04
Epoch 7/20
25/25             78s 3s/step -
accuracy: 0.2105 - loss: 2.0898 - val_accuracy: 0.3600 - val_loss: 1.9706 -
learning_rate: 1.0000e-04
Epoch 8/20
25/25             90s 4s/step -
accuracy: 0.2229 - loss: 2.0444 - val_accuracy: 0.3200 - val_loss: 1.8819 -
learning_rate: 1.0000e-04
Epoch 9/20
25/25             102s 4s/step -
accuracy: 0.2821 - loss: 1.9578 - val_accuracy: 0.3150 - val_loss: 1.8832 -
learning_rate: 1.0000e-04
Epoch 10/20
25/25             106s 4s/step -
accuracy: 0.2963 - loss: 1.8723 - val_accuracy: 0.3250 - val_loss: 1.7665 -
learning_rate: 1.0000e-04
Epoch 11/20
25/25             153s 5s/step -
accuracy: 0.2949 - loss: 1.8557 - val_accuracy: 0.3200 - val_loss: 1.7467 -
```

```
learning_rate: 1.0000e-04
Epoch 12/20
25/25             146s 5s/step -
accuracy: 0.2981 - loss: 1.8683 - val_accuracy: 0.4000 - val_loss: 1.6457 -
learning_rate: 1.0000e-04
Epoch 13/20
25/25             120s 5s/step -
accuracy: 0.3337 - loss: 1.7648 - val_accuracy: 0.3750 - val_loss: 1.6515 -
learning_rate: 1.0000e-04
Epoch 14/20
25/25             123s 5s/step -
accuracy: 0.3681 - loss: 1.7380 - val_accuracy: 0.4400 - val_loss: 1.5832 -
learning_rate: 1.0000e-04
Epoch 15/20
25/25             134s 5s/step -
accuracy: 0.3468 - loss: 1.7188 - val_accuracy: 0.4050 - val_loss: 1.5909 -
learning_rate: 1.0000e-04
Epoch 16/20
25/25             120s 5s/step -
accuracy: 0.3766 - loss: 1.6857 - val_accuracy: 0.4650 - val_loss: 1.5177 -
learning_rate: 1.0000e-04
Epoch 17/20
25/25             139s 5s/step -
accuracy: 0.4130 - loss: 1.5750 - val_accuracy: 0.4050 - val_loss: 1.5298 -
learning_rate: 1.0000e-04
Epoch 18/20
25/25             144s 5s/step -
accuracy: 0.4264 - loss: 1.6061 - val_accuracy: 0.4650 - val_loss: 1.4420 -
learning_rate: 1.0000e-04
Epoch 19/20
25/25             140s 5s/step -
accuracy: 0.4308 - loss: 1.5075 - val_accuracy: 0.4600 - val_loss: 1.4393 -
learning_rate: 1.0000e-04
Epoch 20/20
25/25             147s 5s/step -
accuracy: 0.4204 - loss: 1.5798 - val_accuracy: 0.4650 - val_loss: 1.4283 -
learning_rate: 1.0000e-04
```

[8]:
```
<keras.src.callbacks.history.History at 0x7f35c42e7d10>
```

3.3 7 - Check the accuracy

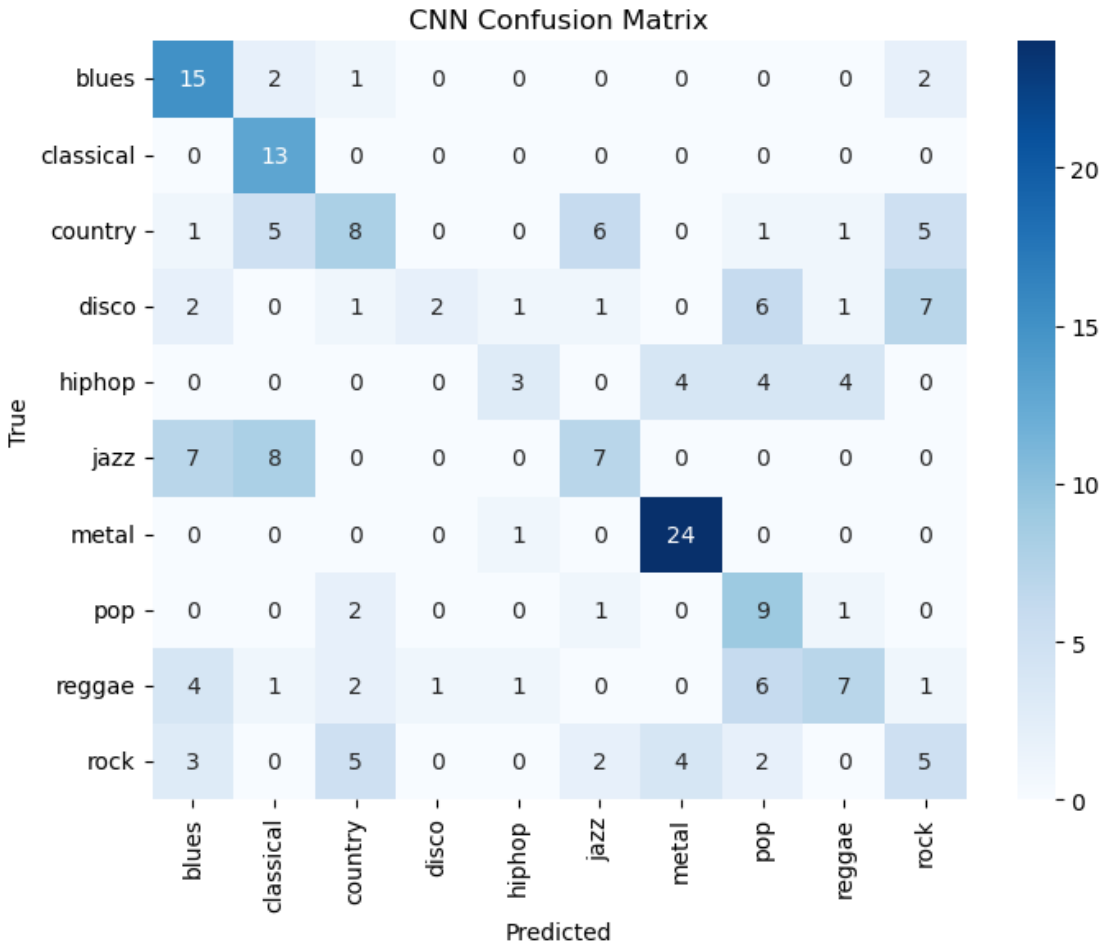
[9]:
```
evaluation = model.evaluate(X_test, y_test)
print(f"Test accuracy: {evaluation[1]:.3f}")
```

```
7/7               7s 1s/step -
accuracy: 0.4524 - loss: 1.3968
Test accuracy: 0.465
```

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                 8s 1s/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_128')
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}")
```

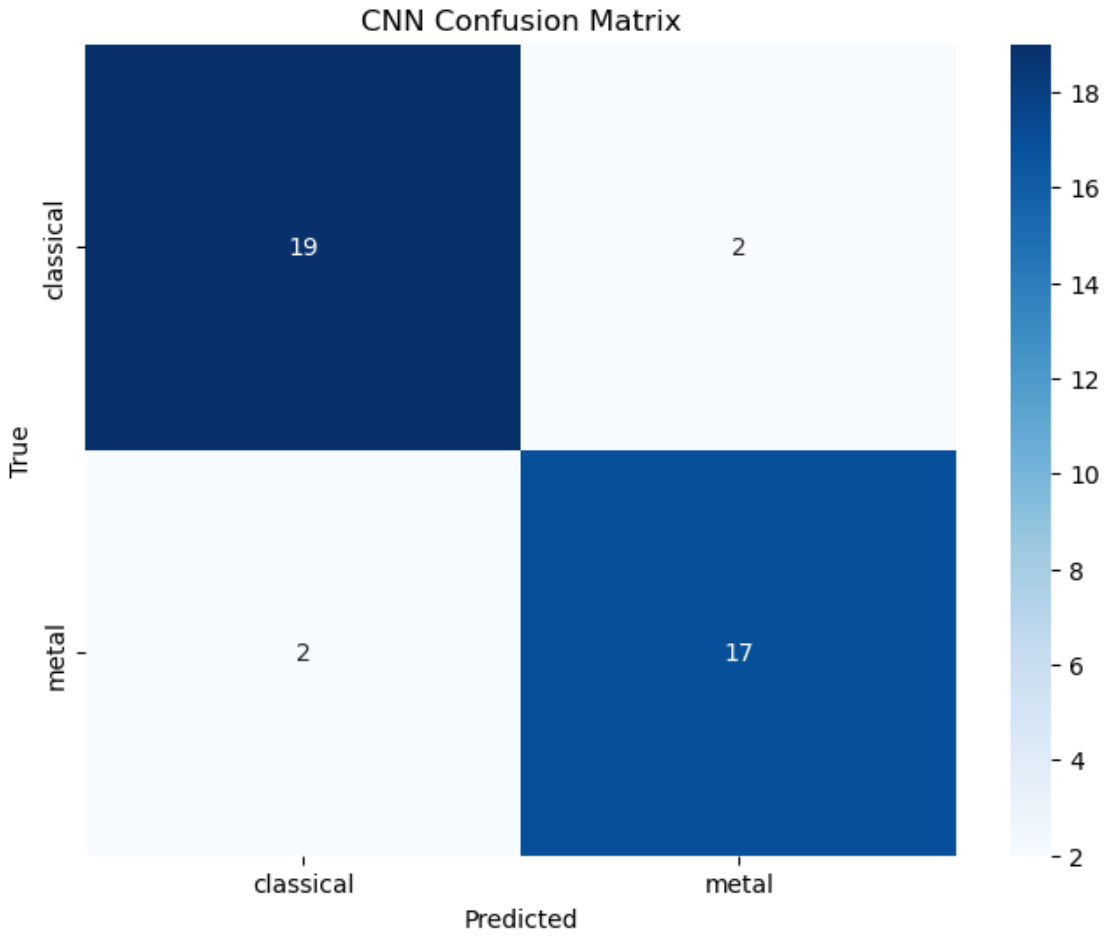
```
Going through classical
Going through metal

/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             39s 5s/step -
accuracy: 0.1680 - loss: 2.2552 - val_accuracy: 0.4750 - val_loss: 1.8325 -
learning_rate: 1.0000e-04
Epoch 2/20
5/5             26s 5s/step -
accuracy: 0.4946 - loss: 1.7795 - val_accuracy: 0.4750 - val_loss: 1.1069 -
learning_rate: 1.0000e-04
Epoch 3/20
5/5             23s 5s/step -
accuracy: 0.5653 - loss: 1.1661 - val_accuracy: 0.4750 - val_loss: 1.0976 -
learning_rate: 1.0000e-04
Epoch 4/20
5/5             43s 5s/step -
accuracy: 0.4843 - loss: 1.2341 - val_accuracy: 0.4750 - val_loss: 0.7797 -
learning_rate: 1.0000e-04
Epoch 5/20
5/5             25s 5s/step -
accuracy: 0.5530 - loss: 1.1913 - val_accuracy: 0.4750 - val_loss: 0.8310 -
learning_rate: 1.0000e-04
Epoch 6/20
5/5             25s 5s/step -
accuracy: 0.5839 - loss: 0.9167 - val_accuracy: 0.5250 - val_loss: 0.7485 -
learning_rate: 1.0000e-04
Epoch 7/20
5/5             41s 5s/step -
accuracy: 0.5998 - loss: 0.8413 - val_accuracy: 0.5500 - val_loss: 0.7014 -
learning_rate: 1.0000e-04
Epoch 8/20
5/5             23s 5s/step -
accuracy: 0.6485 - loss: 0.8210 - val_accuracy: 0.7500 - val_loss: 0.5604 -
learning_rate: 1.0000e-04
Epoch 9/20
5/5             42s 5s/step -
accuracy: 0.6997 - loss: 0.7244 - val_accuracy: 0.8500 - val_loss: 0.4549 -
learning_rate: 1.0000e-04
Epoch 10/20
5/5             43s 5s/step -
accuracy: 0.7780 - loss: 0.5425 - val_accuracy: 0.8500 - val_loss: 0.3613 -
```

```
learning_rate: 1.0000e-04
Epoch 11/20
5/5             40s 5s/step -
accuracy: 0.8111 - loss: 0.4714 - val_accuracy: 0.8750 - val_loss: 0.2593 -
learning_rate: 1.0000e-04
Epoch 12/20
5/5             25s 5s/step -
accuracy: 0.8307 - loss: 0.3986 - val_accuracy: 0.9000 - val_loss: 0.2237 -
learning_rate: 1.0000e-04
Epoch 13/20
5/5             25s 5s/step -
accuracy: 0.8359 - loss: 0.3516 - val_accuracy: 0.9000 - val_loss: 0.2275 -
learning_rate: 1.0000e-04
Epoch 14/20
5/5             41s 5s/step -
accuracy: 0.9151 - loss: 0.2140 - val_accuracy: 0.9000 - val_loss: 0.2538 -
learning_rate: 1.0000e-04
Epoch 15/20
5/5             24s 5s/step -
accuracy: 0.9349 - loss: 0.2258 - val_accuracy: 0.8750 - val_loss: 0.2926 -
learning_rate: 1.0000e-04
Epoch 16/20
5/5             38s 4s/step -
accuracy: 0.9633 - loss: 0.1523 - val_accuracy: 0.8750 - val_loss: 0.2760 -
learning_rate: 5.0000e-05
Epoch 17/20
5/5             42s 5s/step -
accuracy: 0.9378 - loss: 0.1884 - val_accuracy: 0.9000 - val_loss: 0.2599 -
learning_rate: 5.0000e-05
Epoch 18/20
5/5             45s 5s/step -
accuracy: 0.9139 - loss: 0.1833 - val_accuracy: 0.9000 - val_loss: 0.2736 -
learning_rate: 5.0000e-05
Epoch 19/20
5/5             26s 5s/step -
accuracy: 0.9237 - loss: 0.3551 - val_accuracy: 0.9000 - val_loss: 0.2538 -
learning_rate: 2.5000e-05
Epoch 20/20
5/5             26s 5s/step -
accuracy: 0.9633 - loss: 0.1284 - val_accuracy: 0.9000 - val_loss: 0.2663 -
learning_rate: 2.5000e-05
2/2             2s 246ms/step -
accuracy: 0.8917 - loss: 0.2765
Test accuracy: 0.900
```

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                 3s 936ms/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_128')
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}")
```

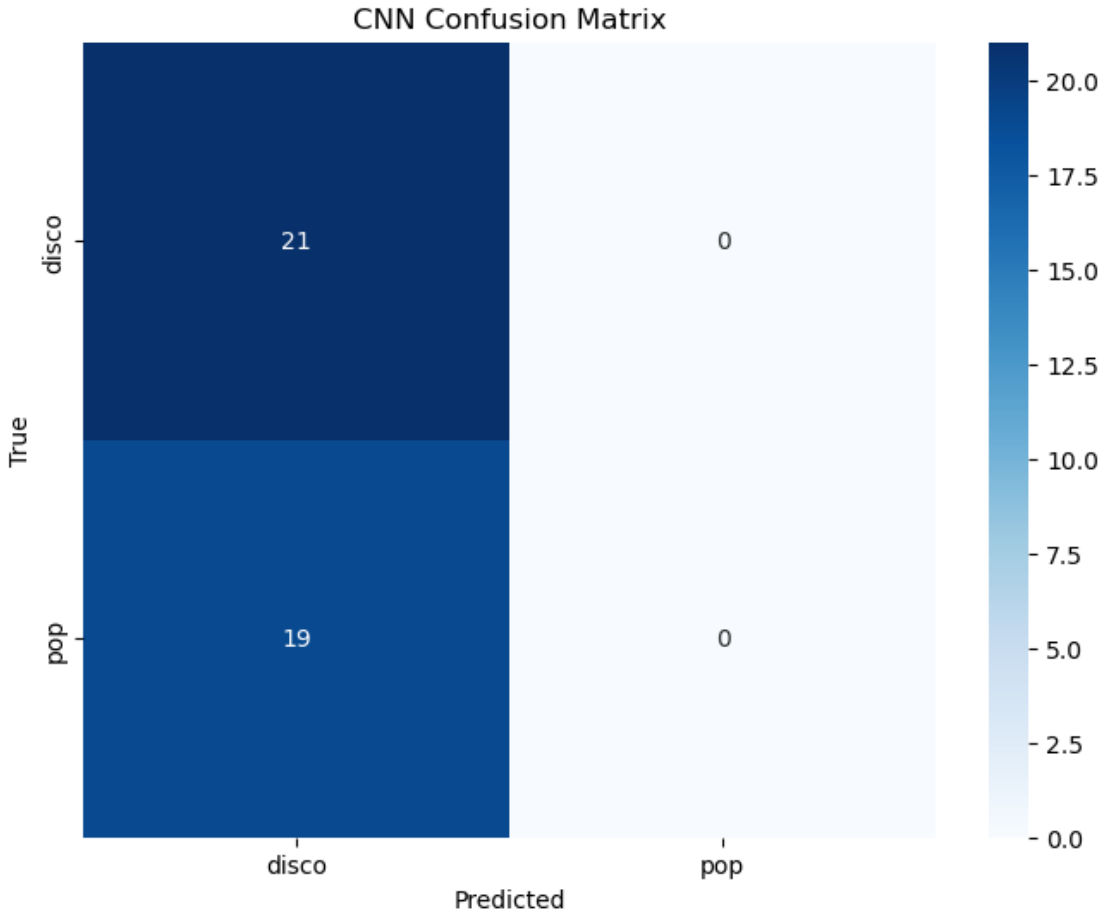
```
Going through disco
Going through pop

/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             35s 5s/step -
accuracy: 0.2016 - loss: 2.2494 - val_accuracy: 0.4750 - val_loss: 1.9092 -
learning_rate: 1.0000e-04
Epoch 2/20
5/5             24s 5s/step -
accuracy: 0.4110 - loss: 1.8692 - val_accuracy: 0.5250 - val_loss: 1.1288 -
learning_rate: 1.0000e-04
Epoch 3/20
5/5             43s 5s/step -
accuracy: 0.5136 - loss: 1.2502 - val_accuracy: 0.5250 - val_loss: 0.7387 -
learning_rate: 1.0000e-04
Epoch 4/20
5/5             26s 5s/step -
accuracy: 0.4853 - loss: 1.0591 - val_accuracy: 0.5250 - val_loss: 0.7007 -
learning_rate: 1.0000e-04
Epoch 5/20
5/5             40s 5s/step -
accuracy: 0.5010 - loss: 1.1153 - val_accuracy: 0.5250 - val_loss: 0.7245 -
learning_rate: 1.0000e-04
Epoch 6/20
5/5             37s 4s/step -
accuracy: 0.5159 - loss: 0.8856 - val_accuracy: 0.5250 - val_loss: 0.7702 -
learning_rate: 1.0000e-04
Epoch 7/20
5/5             25s 5s/step -
accuracy: 0.5098 - loss: 0.9580 - val_accuracy: 0.5250 - val_loss: 0.7899 -
learning_rate: 1.0000e-04
Epoch 8/20
5/5             25s 5s/step -
accuracy: 0.5207 - loss: 0.9621 - val_accuracy: 0.5250 - val_loss: 0.7797 -
learning_rate: 5.0000e-05
Epoch 9/20
5/5             39s 5s/step -
accuracy: 0.5664 - loss: 0.8690 - val_accuracy: 0.5250 - val_loss: 0.7528 -
learning_rate: 5.0000e-05
Epoch 10/20
5/5             23s 5s/step -
accuracy: 0.5079 - loss: 0.8959 - val_accuracy: 0.5250 - val_loss: 0.7298 -
```

```
learning_rate: 5.0000e-05
Epoch 11/20
5/5             24s 5s/step -
accuracy: 0.5138 - loss: 0.8676 - val_accuracy: 0.5250 - val_loss: 0.7247 -
learning_rate: 2.5000e-05
Epoch 12/20
5/5             44s 5s/step -
accuracy: 0.5371 - loss: 0.8682 - val_accuracy: 0.5250 - val_loss: 0.7230 -
learning_rate: 2.5000e-05
Epoch 13/20
5/5             40s 5s/step -
accuracy: 0.5256 - loss: 0.8562 - val_accuracy: 0.5250 - val_loss: 0.7240 -
learning_rate: 2.5000e-05
Epoch 14/20
5/5             40s 5s/step -
accuracy: 0.4043 - loss: 0.8866 - val_accuracy: 0.5250 - val_loss: 0.7244 -
learning_rate: 1.2500e-05
Epoch 15/20
5/5             25s 5s/step -
accuracy: 0.4968 - loss: 0.9110 - val_accuracy: 0.5250 - val_loss: 0.7258 -
learning_rate: 1.2500e-05
Epoch 16/20
5/5             26s 5s/step -
accuracy: 0.5333 - loss: 0.8946 - val_accuracy: 0.5250 - val_loss: 0.7276 -
learning_rate: 1.2500e-05
Epoch 17/20
5/5             26s 5s/step -
accuracy: 0.5398 - loss: 0.8423 - val_accuracy: 0.5250 - val_loss: 0.7286 -
learning_rate: 6.2500e-06
Epoch 18/20
5/5             40s 5s/step -
accuracy: 0.4131 - loss: 1.0000 - val_accuracy: 0.5250 - val_loss: 0.7305 -
learning_rate: 6.2500e-06
Epoch 19/20
5/5             41s 5s/step -
accuracy: 0.5500 - loss: 0.8580 - val_accuracy: 0.5250 - val_loss: 0.7317 -
learning_rate: 6.2500e-06
Epoch 20/20
5/5             42s 5s/step -
accuracy: 0.4912 - loss: 0.9316 - val_accuracy: 0.5250 - val_loss: 0.7321 -
learning_rate: 3.1250e-06
2/2             2s 441ms/step -
accuracy: 0.5375 - loss: 0.7296
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
0x7f354c7c9da0> 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             1s
2s/stepWARNING:tensorflow:6 out of the last 11 calls to <function
TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at
0x7f354c7c9da0> 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             4s 2s/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_128')
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}")
```

```
['rock', 'blues', 'classical', 'reggae', 'jazz']
Going through rock
Going through blues
Going through classical
Going through reggae
Going through jazz
```

```
/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             74s 5s/step -
accuracy: 0.1854 - loss: 2.1934 - val_accuracy: 0.1000 - val_loss: 1.8217 -
```

```
learning_rate: 1.0000e-04
Epoch 2/20
13/13             80s 5s/step -
accuracy: 0.1970 - loss: 1.9210 - val_accuracy: 0.1400 - val_loss: 1.7852 -
learning_rate: 1.0000e-04
Epoch 3/20
13/13             82s 5s/step -
accuracy: 0.2280 - loss: 1.8253 - val_accuracy: 0.1400 - val_loss: 1.7209 -
learning_rate: 1.0000e-04
Epoch 4/20
13/13             77s 4s/step -
accuracy: 0.2296 - loss: 1.7596 - val_accuracy: 0.1400 - val_loss: 1.7137 -
learning_rate: 1.0000e-04
Epoch 5/20
13/13             63s 5s/step -
accuracy: 0.2317 - loss: 1.7760 - val_accuracy: 0.3400 - val_loss: 1.7323 -
learning_rate: 1.0000e-04
Epoch 6/20
13/13             60s 5s/step -
accuracy: 0.2371 - loss: 1.7027 - val_accuracy: 0.3300 - val_loss: 1.6576 -
learning_rate: 1.0000e-04
Epoch 7/20
13/13             54s 4s/step -
accuracy: 0.2420 - loss: 1.7197 - val_accuracy: 0.3300 - val_loss: 1.6012 -
learning_rate: 1.0000e-04
Epoch 8/20
13/13             58s 5s/step -
accuracy: 0.2332 - loss: 1.7134 - val_accuracy: 0.3900 - val_loss: 1.5890 -
learning_rate: 1.0000e-04
Epoch 9/20
13/13             81s 4s/step -
accuracy: 0.2505 - loss: 1.6462 - val_accuracy: 0.3100 - val_loss: 1.6335 -
learning_rate: 1.0000e-04
Epoch 10/20
13/13             61s 5s/step -
accuracy: 0.3143 - loss: 1.6143 - val_accuracy: 0.4000 - val_loss: 1.5923 -
learning_rate: 1.0000e-04
Epoch 11/20
13/13             79s 4s/step -
accuracy: 0.2966 - loss: 1.5992 - val_accuracy: 0.3500 - val_loss: 1.5364 -
learning_rate: 1.0000e-04
Epoch 12/20
13/13             75s 4s/step -
accuracy: 0.3250 - loss: 1.5333 - val_accuracy: 0.4200 - val_loss: 1.5307 -
learning_rate: 1.0000e-04
Epoch 13/20
13/13             89s 4s/step -
accuracy: 0.3075 - loss: 1.5101 - val_accuracy: 0.3500 - val_loss: 1.4983 -
```

```
learning_rate: 1.0000e-04
Epoch 14/20
13/13             52s 4s/step -
accuracy: 0.3218 - loss: 1.4913 - val_accuracy: 0.3600 - val_loss: 1.5197 -
learning_rate: 1.0000e-04
Epoch 15/20
13/13             85s 4s/step -
accuracy: 0.3455 - loss: 1.5316 - val_accuracy: 0.3900 - val_loss: 1.6015 -
learning_rate: 1.0000e-04
Epoch 16/20
13/13             80s 4s/step -
accuracy: 0.3492 - loss: 1.4626 - val_accuracy: 0.5200 - val_loss: 1.3328 -
learning_rate: 1.0000e-04
Epoch 17/20
13/13             57s 4s/step -
accuracy: 0.4954 - loss: 1.3244 - val_accuracy: 0.6000 - val_loss: 1.2204 -
learning_rate: 1.0000e-04
Epoch 18/20
13/13             80s 4s/step -
accuracy: 0.5091 - loss: 1.2173 - val_accuracy: 0.5700 - val_loss: 1.2116 -
learning_rate: 1.0000e-04
Epoch 19/20
13/13             56s 4s/step -
accuracy: 0.4462 - loss: 1.2465 - val_accuracy: 0.5800 - val_loss: 1.1273 -
learning_rate: 1.0000e-04
Epoch 20/20
13/13             53s 4s/step -
accuracy: 0.5256 - loss: 1.1652 - val_accuracy: 0.5600 - val_loss: 1.0863 -
learning_rate: 1.0000e-04
4/4             4s 788ms/step -
accuracy: 0.5167 - loss: 1.1370
Test accuracy: 0.560
```

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                 4s 950ms/step
```