Spectrogram-Only (3 secs) CNN

March 21, 2025

[1]:
```
!pip install librosa
```

```
Requirement already satisfied: librosa in /opt/conda/lib/python3.12/site-
packages (0.11.0)
Requirement already satisfied: audioread>=2.1.9 in
/opt/conda/lib/python3.12/site-packages (from librosa) (3.0.1)
Requirement already satisfied: numba>=0.51.0 in /opt/conda/lib/python3.12/site-
packages (from librosa) (0.60.0)
Requirement already satisfied: numpy>=1.22.3 in /opt/conda/lib/python3.12/site-
packages (from librosa) (2.0.2)
Requirement already satisfied: scipy>=1.6.0 in /opt/conda/lib/python3.12/site-
packages (from librosa) (1.14.1)
Requirement already satisfied: scikit-learn>=1.1.0 in
/opt/conda/lib/python3.12/site-packages (from librosa) (1.5.2)
Requirement already satisfied: joblib>=1.0 in /opt/conda/lib/python3.12/site-
packages (from librosa) (1.4.2)
Requirement already satisfied: decorator>=4.3.0 in
/opt/conda/lib/python3.12/site-packages (from librosa) (5.1.1)
Requirement already satisfied: soundfile>=0.12.1 in
/opt/conda/lib/python3.12/site-packages (from librosa) (0.13.1)
Requirement already satisfied: pooch>=1.1 in /opt/conda/lib/python3.12/site-
packages (from librosa) (1.8.2)
Requirement already satisfied: soxr>=0.3.2 in /opt/conda/lib/python3.12/site-
packages (from librosa) (0.5.0.post1)
Requirement already satisfied: typing_extensions>=4.1.1 in
/opt/conda/lib/python3.12/site-packages (from librosa) (4.12.2)
Requirement already satisfied: lazy_loader>=0.1 in
/opt/conda/lib/python3.12/site-packages (from librosa) (0.4)
Requirement already satisfied: msgpack>=1.0 in /opt/conda/lib/python3.12/site-
packages (from librosa) (1.1.0)
Requirement already satisfied: packaging in /opt/conda/lib/python3.12/site-
packages (from lazy_loader>=0.1->librosa) (24.1)
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in
/opt/conda/lib/python3.12/site-packages (from numba>=0.51.0->librosa) (0.43.0)
Requirement already satisfied: platformdirs>=2.5.0 in
/opt/conda/lib/python3.12/site-packages (from pooch>=1.1->librosa) (4.3.6)
Requirement already satisfied: requests>=2.19.0 in
/opt/conda/lib/python3.12/site-packages (from pooch>=1.1->librosa) (2.32.3)
```

```
Requirement already satisfied: threadpoolctl>=3.1.0 in
/opt/conda/lib/python3.12/site-packages (from scikit-learn>=1.1.0->librosa)
(3.5.0)
Requirement already satisfied: cffi>=1.0 in /opt/conda/lib/python3.12/site-
packages (from soundfile>=0.12.1->librosa) (1.17.1)
Requirement already satisfied: pycparser in /opt/conda/lib/python3.12/site-
packages (from cffi>=1.0->soundfile>=0.12.1->librosa) (2.22)
Requirement already satisfied: charset-normalizer<4,>=2 in
/opt/conda/lib/python3.12/site-packages (from
requests>=2.19.0->pooch>=1.1->librosa) (3.4.0)
Requirement already satisfied: idna<4,>=2.5 in /opt/conda/lib/python3.12/site-
packages (from requests>=2.19.0->pooch>=1.1->librosa) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/opt/conda/lib/python3.12/site-packages (from
requests>=2.19.0->pooch>=1.1->librosa) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in
/opt/conda/lib/python3.12/site-packages (from
requests>=2.19.0->pooch>=1.1->librosa) (2024.8.30)
```

1 CNN for Spectrogram (3 secs)

1.1 1 - All the imports

[2]:
```
import os
import numpy as np
from sklearn.model_selection import train_test_split
import tensorflow as tf
```

```
2025-03-21 00:44:38.874410: 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

[3]:
```
# 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

[4]:
```
from tensorflow.keras import models
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
 ↪Dropout, BatchNormalization

model = models.Sequential([
    Conv2D(32, (3, 3), activation='relu', input_shape=(256, 256, 1)),
    BatchNormalization(),
    MaxPooling2D((2, 2)),

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

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

    Conv2D(256, (3, 3), activation='relu'),
    BatchNormalization(),
    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)
```

[5]:
```
model.summary()
```

```
Model: "sequential"
```

Layer (type)	Output Shape	Param #

conv2d (Conv2D)	(None, 254, 254, 32)	320
batch_normalization (BatchNormalization)	(None, 254, 254, 32)	128
max_pooling2d (MaxPooling2D)	(None, 127, 127, 32)	0
conv2d_1 (Conv2D)	(None, 125, 125, 64)	18,496
batch_normalization_1 (BatchNormalization)	(None, 125, 125, 64)	256
max_pooling2d_1 (MaxPooling2D)	(None, 62, 62, 64)	0
conv2d_2 (Conv2D)	(None, 60, 60, 128)	73,856
batch_normalization_2 (BatchNormalization)	(None, 60, 60, 128)	512
max_pooling2d_2 (MaxPooling2D)	(None, 30, 30, 128)	0
conv2d_3 (Conv2D)	(None, 28, 28, 256)	295,168
batch_normalization_3 (BatchNormalization)	(None, 28, 28, 256)	1,024
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: 26,245,898 (100.12 MB)

Trainable params: 26,244,938 (100.12 MB)

```
Non-trainable params: 960 (3.75 KB)
```

3 4 - Load the images

[6]:
```
import tensorflow as tf
import os
import numpy as np
from sklearn.model_selection import train_test_split

GENRES = ['blues', 'classical', 'country', 'disco', 'hiphop', 'jazz', 'metal',␣
 ↪'pop', 'reggae', 'rock']
FILE_PATH = os.path.join('Data', 'spectrograms (3 secs)', 'spectrogram_256')
X = []
y = []

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

# Define the 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

for genre in GENRES:
    genre_dir = os.path.join(FILE_PATH, genre)
    print(f"Going through {genre}")
    for file in os.listdir(genre_dir):
        try:
            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])

            # Apply the augmentation
            image = augment_image(image)

            image = image.numpy()  # Convert to numpy array for further␣
 ↪processing
            X.append(image)
            y.append(GENRE_TO_INDEX[genre])
        except:
            continue
```

```
X = np.array(X)
y = np.array(y)

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

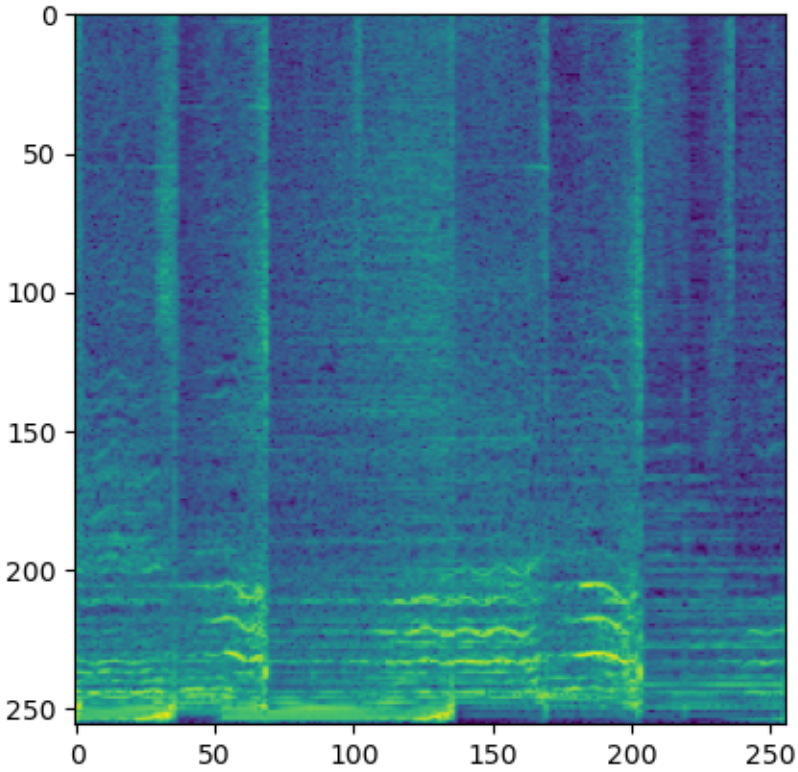
```
Going through blues
Going through classical
Going through country
Going through disco
Going through hiphop
Going through jazz
Going through metal
Going through pop
Going through reggae
Going through rock
```

```
2025-03-21 00:57:56.983206: W tensorflow/core/framework/op_kernel.cc:1841]
OP_REQUIRES failed at whole_file_read_ops.cc:116 : FAILED_PRECONDITION:
Data/spectrograms (3 secs)/spectrogram_256/rock/.ipynb_checkpoints; Is a
directory
2025-03-21 00:57:56.988155: I tensorflow/core/framework/local_rendezvous.cc:405]
Local rendezvous is aborting with status: FAILED_PRECONDITION: Data/spectrograms
(3 secs)/spectrogram_256/rock/.ipynb_checkpoints; Is a directory
```

[7]:
```
X.shape, y.shape
```

[7]:
```
((10000, 256, 256, 1), (10000,))
```

[8]:
```
# Show image as a sanity check
import matplotlib.pyplot as plt
plt.imshow(X_train[7999].reshape(256, 256))
plt.show()
```



[9]:
```
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)
```

[10]:
```
model.fit(X_train, y_train, epochs=20, validation_data=(X_test, y_test),␣
 ↪batch_size=32, callbacks=[reduce_lr])
```

```
Epoch 1/20
250/250             1381s 5s/step -
accuracy: 0.1601 - loss: 3.1593 - val_accuracy: 0.1010 - val_loss: 5.0143 -
learning_rate: 1.0000e-04
Epoch 2/20
250/250             1189s 5s/step -
accuracy: 0.2633 - loss: 2.0170 - val_accuracy: 0.2210 - val_loss: 2.5034 -
learning_rate: 1.0000e-04
Epoch 3/20
250/250             964s 4s/step -
```

```
accuracy: 0.2974 - loss: 1.9189 - val_accuracy: 0.3580 - val_loss: 1.7114 -
learning_rate: 1.0000e-04
Epoch 4/20
250/250            784s 3s/step -
accuracy: 0.3523 - loss: 1.7837 - val_accuracy: 0.4620 - val_loss: 1.4956 -
learning_rate: 1.0000e-04
Epoch 5/20
250/250            801s 3s/step -
accuracy: 0.3829 - loss: 1.6866 - val_accuracy: 0.4935 - val_loss: 1.4203 -
learning_rate: 1.0000e-04
Epoch 6/20
250/250            791s 3s/step -
accuracy: 0.4342 - loss: 1.5779 - val_accuracy: 0.5180 - val_loss: 1.3586 -
learning_rate: 1.0000e-04
Epoch 7/20
250/250            795s 3s/step -
accuracy: 0.4765 - loss: 1.4738 - val_accuracy: 0.6180 - val_loss: 1.2474 -
learning_rate: 1.0000e-04
Epoch 8/20
250/250            637s 3s/step -
accuracy: 0.5191 - loss: 1.3491 - val_accuracy: 0.6615 - val_loss: 1.1249 -
learning_rate: 1.0000e-04
Epoch 9/20
250/250            631s 3s/step -
accuracy: 0.5544 - loss: 1.2650 - val_accuracy: 0.6140 - val_loss: 1.1917 -
learning_rate: 1.0000e-04
Epoch 10/20
250/250            641s 3s/step -
accuracy: 0.5804 - loss: 1.1998 - val_accuracy: 0.6580 - val_loss: 1.0853 -
learning_rate: 1.0000e-04
Epoch 11/20
250/250            639s 3s/step -
accuracy: 0.6206 - loss: 1.0845 - val_accuracy: 0.6965 - val_loss: 1.0171 -
learning_rate: 1.0000e-04
Epoch 12/20
250/250            621s 2s/step -
accuracy: 0.6481 - loss: 1.0130 - val_accuracy: 0.6905 - val_loss: 1.0017 -
learning_rate: 1.0000e-04
Epoch 13/20
250/250            629s 3s/step -
accuracy: 0.6793 - loss: 0.9440 - val_accuracy: 0.6965 - val_loss: 1.0273 -
learning_rate: 1.0000e-04
Epoch 14/20
250/250            621s 2s/step -
accuracy: 0.7040 - loss: 0.8534 - val_accuracy: 0.6575 - val_loss: 1.1556 -
learning_rate: 1.0000e-04
Epoch 15/20
250/250            634s 3s/step -
```

```
accuracy: 0.7099 - loss: 0.8308 - val_accuracy: 0.5790 - val_loss: 1.4206 -
learning_rate: 1.0000e-04
Epoch 16/20
250/250             618s 2s/step -
accuracy: 0.7318 - loss: 0.7707 - val_accuracy: 0.7540 - val_loss: 0.8052 -
learning_rate: 5.0000e-05
Epoch 17/20
250/250             615s 2s/step -
accuracy: 0.7702 - loss: 0.6519 - val_accuracy: 0.7765 - val_loss: 0.7779 -
learning_rate: 5.0000e-05
Epoch 18/20
250/250             618s 2s/step -
accuracy: 0.7737 - loss: 0.6390 - val_accuracy: 0.7645 - val_loss: 0.7963 -
learning_rate: 5.0000e-05
Epoch 19/20
250/250             627s 3s/step -
accuracy: 0.7821 - loss: 0.6070 - val_accuracy: 0.7655 - val_loss: 0.7982 -
learning_rate: 5.0000e-05
Epoch 20/20
250/250             620s 2s/step -
accuracy: 0.7976 - loss: 0.5731 - val_accuracy: 0.7640 - val_loss: 0.8039 -
learning_rate: 5.0000e-05
```

[10]:
```
<keras.src.callbacks.history.History at 0x7f4d60618b90>
```

[11]:
```
evaluation = model.evaluate(X_test, y_test)
print(f"Test accuracy: {evaluation[1]:.3f}")
```

```
63/63             19s 304ms/step -
accuracy: 0.7480 - loss: 0.8153
Test accuracy: 0.764
```

4 Apply the confusion matrix after the model

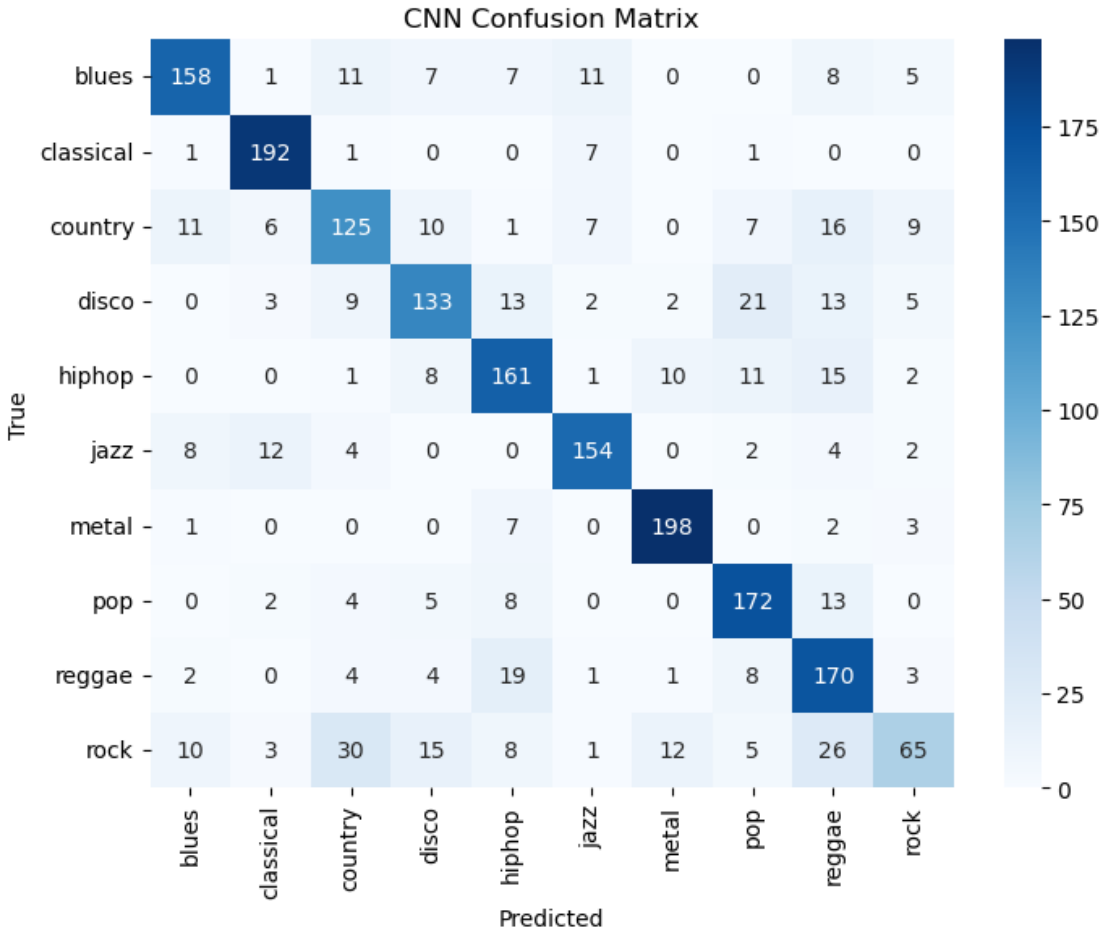
[12]:
```
import seaborn as sns
# from sklearn.metrics import confusion
import numpy as NP
from sklearn.metrics import confusion_matrix

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()
```

```
63/63                20s 309ms/step
```



[13]:
```
X.shape, y.shape
```

[13]:
```
((10000, 256, 256, 1), (10000,))
```

4.1 9 - Limited Genres Easy (metal and classical)

[14]:
```
import tensorflow as tf
import os
import numpy as np
from sklearn.model_selection import train_test_split

GENRES = ['classical', 'metal']
```

```
FILE_PATH = os.path.join('Data', 'spectrograms (3 secs)', 'spectrogram_256')
X = []
y = []

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

# Define the 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

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])

        # Apply the augmentation
        image = augment_image(image)

        image = image.numpy()  # Convert to numpy array for further processing
        X.append(image)
        y.append(GENRE_TO_INDEX[genre])

X = np.array(X)
y = np.array(y)

# Split the data
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
50/50             76s 1s/step -
accuracy: 0.4420 - loss: 1.6141 - val_accuracy: 0.8700 - val_loss: 0.3231 -
```

```
learning_rate: 1.0000e-04
Epoch 2/20
50/50             70s 1s/step -
accuracy: 0.8854 - loss: 0.3115 - val_accuracy: 0.9500 - val_loss: 0.1281 -
learning_rate: 1.0000e-04
Epoch 3/20
50/50             70s 1s/step -
accuracy: 0.9582 - loss: 0.1412 - val_accuracy: 0.9625 - val_loss: 0.1082 -
learning_rate: 1.0000e-04
Epoch 4/20
50/50             69s 1s/step -
accuracy: 0.9631 - loss: 0.1169 - val_accuracy: 0.9475 - val_loss: 0.1402 -
learning_rate: 1.0000e-04
Epoch 5/20
50/50             69s 1s/step -
accuracy: 0.9697 - loss: 0.0994 - val_accuracy: 0.9875 - val_loss: 0.0373 -
learning_rate: 1.0000e-04
Epoch 6/20
50/50             71s 1s/step -
accuracy: 0.9785 - loss: 0.0791 - val_accuracy: 0.9900 - val_loss: 0.0332 -
learning_rate: 1.0000e-04
Epoch 7/20
50/50             71s 1s/step -
accuracy: 0.9821 - loss: 0.0587 - val_accuracy: 0.9750 - val_loss: 0.0527 -
learning_rate: 1.0000e-04
Epoch 8/20
50/50             71s 1s/step -
accuracy: 0.9730 - loss: 0.0675 - val_accuracy: 0.9950 - val_loss: 0.0226 -
learning_rate: 1.0000e-04
Epoch 9/20
50/50             75s 1s/step -
accuracy: 0.9894 - loss: 0.0408 - val_accuracy: 0.9950 - val_loss: 0.0201 -
learning_rate: 1.0000e-04
Epoch 10/20
50/50             71s 1s/step -
accuracy: 0.9921 - loss: 0.0352 - val_accuracy: 0.9925 - val_loss: 0.0176 -
learning_rate: 1.0000e-04
Epoch 11/20
50/50             72s 1s/step -
accuracy: 0.9888 - loss: 0.0411 - val_accuracy: 0.9950 - val_loss: 0.0168 -
learning_rate: 1.0000e-04
Epoch 12/20
50/50             71s 1s/step -
accuracy: 0.9886 - loss: 0.0326 - val_accuracy: 0.9925 - val_loss: 0.0193 -
learning_rate: 1.0000e-04
Epoch 13/20
50/50             72s 1s/step -
accuracy: 0.9943 - loss: 0.0199 - val_accuracy: 0.9975 - val_loss: 0.0071 -
```

```
learning_rate: 1.0000e-04
Epoch 14/20
50/50             72s 1s/step -
accuracy: 0.9981 - loss: 0.0122 - val_accuracy: 0.9975 - val_loss: 0.0073 -
learning_rate: 1.0000e-04
Epoch 15/20
50/50             57s 1s/step -
accuracy: 0.9953 - loss: 0.0206 - val_accuracy: 1.0000 - val_loss: 0.0042 -
learning_rate: 1.0000e-04
Epoch 16/20
50/50             57s 1s/step -
accuracy: 0.9930 - loss: 0.0226 - val_accuracy: 0.9925 - val_loss: 0.0180 -
learning_rate: 1.0000e-04
Epoch 17/20
50/50             57s 1s/step -
accuracy: 0.9937 - loss: 0.0228 - val_accuracy: 1.0000 - val_loss: 0.0033 -
learning_rate: 1.0000e-04
Epoch 18/20
50/50             57s 1s/step -
accuracy: 0.9971 - loss: 0.0107 - val_accuracy: 0.9975 - val_loss: 0.0068 -
learning_rate: 1.0000e-04
Epoch 19/20
50/50             57s 1s/step -
accuracy: 0.9972 - loss: 0.0191 - val_accuracy: 0.9950 - val_loss: 0.0074 -
learning_rate: 1.0000e-04
Epoch 20/20
50/50             57s 1s/step -
accuracy: 0.9983 - loss: 0.0128 - val_accuracy: 0.9950 - val_loss: 0.0119 -
learning_rate: 1.0000e-04
13/13             3s 223ms/step -
accuracy: 0.9955 - loss: 0.0133
Test accuracy: 0.995
```

4.2 10 - Confusion Matrix Easy (classical and metal)

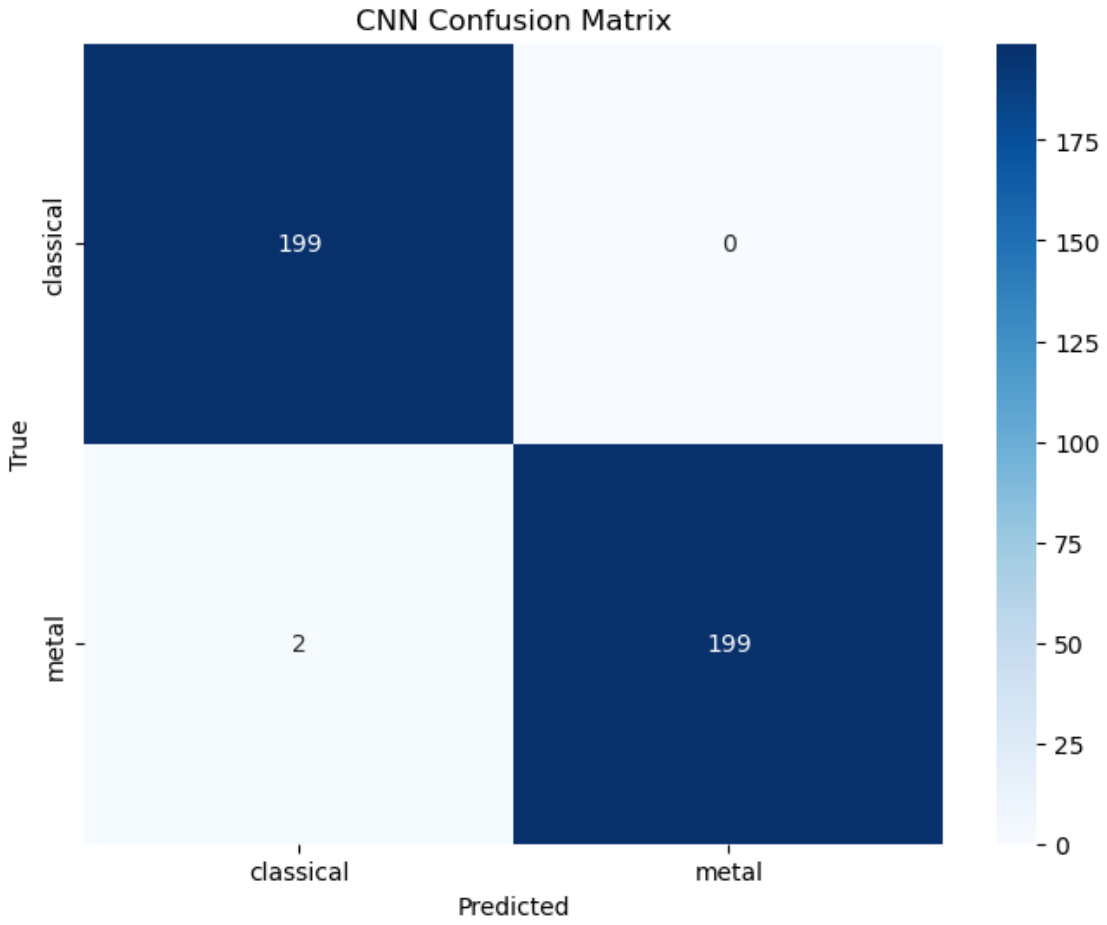
[15]:
```
import seaborn as sns
# from sklearn.metrics import confusion
import numpy as NP
from sklearn.metrics import confusion_matrix

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()
```

```
13/13                3s 240ms/step
```



4.3 11 - Limited genres Hard (disco and pop)

[16]:
```
import tensorflow as tf
import os
import numpy as np
from sklearn.model_selection import train_test_split

GENRES = ['disco', 'pop']
FILE_PATH = os.path.join('Data', 'spectrograms (3 secs)', 'spectrogram_256')
X = []
y = []
```

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

# Define the 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

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])

        # Apply the augmentation
        image = augment_image(image)

        image = image.numpy()  # Convert to numpy array for further processing
        X.append(image)
        y.append(GENRE_TO_INDEX[genre])

X = np.array(X)
y = np.array(y)

# Split the data
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
50/50             59s 1s/step -
accuracy: 0.4197 - loss: 1.4163 - val_accuracy: 0.5025 - val_loss: 0.7680 -
learning_rate: 1.0000e-04
Epoch 2/20
50/50             55s 1s/step -
```

```
accuracy: 0.5129 - loss: 0.8277 - val_accuracy: 0.7900 - val_loss: 0.4794 -
learning_rate: 1.0000e-04
Epoch 3/20
50/50            55s 1s/step -
accuracy: 0.7707 - loss: 0.5066 - val_accuracy: 0.8050 - val_loss: 0.4059 -
learning_rate: 1.0000e-04
Epoch 4/20
50/50            55s 1s/step -
accuracy: 0.7905 - loss: 0.4676 - val_accuracy: 0.8050 - val_loss: 0.3956 -
learning_rate: 1.0000e-04
Epoch 5/20
50/50            55s 1s/step -
accuracy: 0.8020 - loss: 0.4264 - val_accuracy: 0.8100 - val_loss: 0.3481 -
learning_rate: 1.0000e-04
Epoch 6/20
50/50            55s 1s/step -
accuracy: 0.8079 - loss: 0.4242 - val_accuracy: 0.8125 - val_loss: 0.3571 -
learning_rate: 1.0000e-04
Epoch 7/20
50/50            54s 1s/step -
accuracy: 0.8150 - loss: 0.4271 - val_accuracy: 0.8075 - val_loss: 0.3491 -
learning_rate: 1.0000e-04
Epoch 8/20
50/50            54s 1s/step -
accuracy: 0.8082 - loss: 0.3994 - val_accuracy: 0.8275 - val_loss: 0.3267 -
learning_rate: 1.0000e-04
Epoch 9/20
50/50            55s 1s/step -
accuracy: 0.8205 - loss: 0.4061 - val_accuracy: 0.8300 - val_loss: 0.3107 -
learning_rate: 1.0000e-04
Epoch 10/20
50/50            55s 1s/step -
accuracy: 0.8326 - loss: 0.3448 - val_accuracy: 0.8325 - val_loss: 0.3277 -
learning_rate: 1.0000e-04
Epoch 11/20
50/50            55s 1s/step -
accuracy: 0.8143 - loss: 0.3992 - val_accuracy: 0.8200 - val_loss: 0.3459 -
learning_rate: 1.0000e-04
Epoch 12/20
50/50            54s 1s/step -
accuracy: 0.8269 - loss: 0.3619 - val_accuracy: 0.8375 - val_loss: 0.2814 -
learning_rate: 1.0000e-04
Epoch 13/20
50/50            55s 1s/step -
accuracy: 0.8195 - loss: 0.3675 - val_accuracy: 0.8500 - val_loss: 0.2907 -
learning_rate: 1.0000e-04
Epoch 14/20
50/50            55s 1s/step -
```

```
accuracy: 0.8445 - loss: 0.3248 - val_accuracy: 0.8550 - val_loss: 0.2823 -
learning_rate: 1.0000e-04
Epoch 15/20
50/50             54s 1s/step -
accuracy: 0.8583 - loss: 0.3247 - val_accuracy: 0.8600 - val_loss: 0.2657 -
learning_rate: 1.0000e-04
Epoch 16/20
50/50             54s 1s/step -
accuracy: 0.8476 - loss: 0.3249 - val_accuracy: 0.8775 - val_loss: 0.2565 -
learning_rate: 1.0000e-04
Epoch 17/20
50/50             56s 1s/step -
accuracy: 0.8702 - loss: 0.3034 - val_accuracy: 0.8825 - val_loss: 0.2449 -
learning_rate: 1.0000e-04
Epoch 18/20
50/50             55s 1s/step -
accuracy: 0.8727 - loss: 0.2783 - val_accuracy: 0.8825 - val_loss: 0.2508 -
learning_rate: 1.0000e-04
Epoch 19/20
50/50             55s 1s/step -
accuracy: 0.8603 - loss: 0.2923 - val_accuracy: 0.9100 - val_loss: 0.2228 -
learning_rate: 1.0000e-04
Epoch 20/20
50/50             56s 1s/step -
accuracy: 0.8915 - loss: 0.2603 - val_accuracy: 0.9125 - val_loss: 0.2111 -
learning_rate: 1.0000e-04
13/13             3s 221ms/step -
accuracy: 0.9032 - loss: 0.2188
Test accuracy: 0.913
```

4.4 12 - Confusion Matrix Hard (disco and pop)

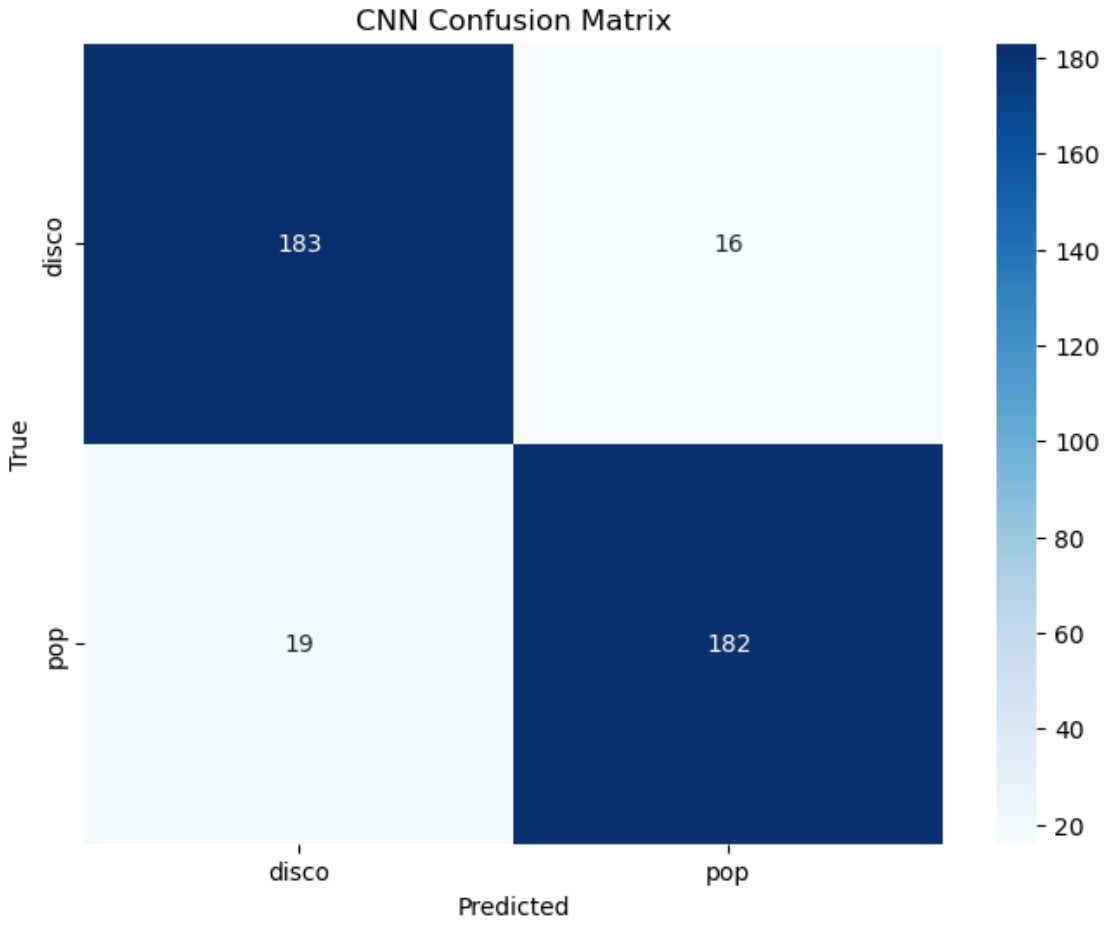
[17]:
```python
import seaborn as sns
# from sklearn.metrics import confusion
import numpy as NP
from sklearn.metrics import confusion_matrix

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()
```

```
13/13                3s 233ms/step
```



4.5 13 - Limited Genres Medium (5 random)

[18]:
```
import tensorflow as tf
import os
import numpy as np
from sklearn.model_selection import train_test_split
import random

GENRES = ['disco', 'pop']
GENRES = random.sample(GENRES, 5)
print(GENRES)
FILE_PATH = os.path.join('Data', 'spectrograms (3 secs)', 'spectrogram_256')
X = []
y = []
```

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

# Define the 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

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])

        # Apply the augmentation
        image = augment_image(image)

        image = image.numpy()  # Convert to numpy array for further processing
        X.append(image)
        y.append(GENRE_TO_INDEX[genre])

X = np.array(X)
y = np.array(y)

# Split the data
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}")
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[18], line 8
      5 import random
      7 GENRES = ['disco', 'pop']
----> 8 GENRES = random.sample(GENRES, 5)
      9 print(GENRES)
     10 FILE_PATH = os.path.join('Data', 'spectrograms (3 secs)',␣
 ↪'spectrogram_256')

File /opt/conda/lib/python3.12/random.py:430, in Random.sample(self, population␣
 ↪k, counts)
    428 randbelow = self._randbelow
    429 if not 0 <= k <= n:
```

```
--> 430     raise ValueError("Sample larger than population or is negative")
    431 result = [None] * k
    432 setsize = 21        # size of a small set minus size of an empty list

ValueError: Sample larger than population or is negative
```

4.6 14 - Confusion Matrix Medium (5 random)

[ ]:
```
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()
```