Mel-Spectrogram-Only (3 secs) CNN

March 20, 2025

1 CNN for Mel-Spectrogram (3 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 08:41:28.574835: 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)

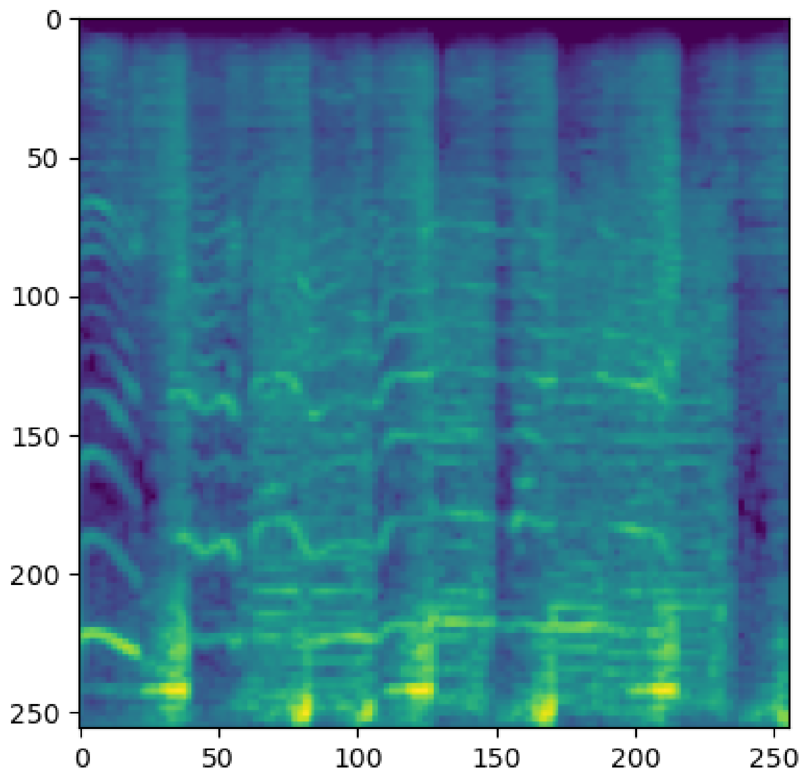
3 4 - Load the images

[5]:
```python
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 = ['blues', 'classical', 'country', 'disco', 'hiphop', 'jazz', 'metal',␣
↪'pop', 'reggae', 'rock']
FILE_PATH = os.path.join('Data', 'mel_spectrograms (3 secs)',␣
↪'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]:
```python
# 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]:
```python
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
250/250            1148s 5s/step -
accuracy: 0.1189 - loss: 2.2847 - val_accuracy: 0.2580 - val_loss: 2.0443 -
learning_rate: 1.0000e-04
Epoch 2/20
250/250            1167s 5s/step -
accuracy: 0.2462 - loss: 2.0294 - val_accuracy: 0.4210 - val_loss: 1.6541 -
learning_rate: 1.0000e-04
Epoch 3/20
250/250            1166s 5s/step -
accuracy: 0.3581 - loss: 1.7286 - val_accuracy: 0.5185 - val_loss: 1.4254 -
learning_rate: 1.0000e-04
Epoch 4/20
250/250            1031s 4s/step -
accuracy: 0.4385 - loss: 1.5383 - val_accuracy: 0.5630 - val_loss: 1.2900 -
learning_rate: 1.0000e-04
Epoch 5/20
250/250            937s 4s/step -
accuracy: 0.4903 - loss: 1.4072 - val_accuracy: 0.5580 - val_loss: 1.2294 -
learning_rate: 1.0000e-04
Epoch 6/20
250/250            793s 3s/step -
accuracy: 0.5408 - loss: 1.2939 - val_accuracy: 0.6240 - val_loss: 1.0986 -
learning_rate: 1.0000e-04
Epoch 7/20
250/250            782s 3s/step -
accuracy: 0.5569 - loss: 1.2344 - val_accuracy: 0.6625 - val_loss: 1.0026 -
learning_rate: 1.0000e-04
Epoch 8/20
250/250            734s 3s/step -
accuracy: 0.6229 - loss: 1.0805 - val_accuracy: 0.6720 - val_loss: 0.9456 -
learning_rate: 1.0000e-04
Epoch 9/20
250/250            745s 3s/step -
accuracy: 0.6538 - loss: 1.0211 - val_accuracy: 0.6980 - val_loss: 0.8964 -
learning_rate: 1.0000e-04
Epoch 10/20
250/250            793s 3s/step -
accuracy: 0.6855 - loss: 0.9251 - val_accuracy: 0.6840 - val_loss: 0.9330 -
learning_rate: 1.0000e-04
```

```
Epoch 11/20
250/250              624s 2s/step -
accuracy: 0.7078 - loss: 0.8654 - val_accuracy: 0.7385 - val_loss: 0.8064 -
learning_rate: 1.0000e-04
Epoch 12/20
250/250              622s 2s/step -
accuracy: 0.7193 - loss: 0.8166 - val_accuracy: 0.7455 - val_loss: 0.7936 -
learning_rate: 1.0000e-04
Epoch 13/20
250/250              621s 2s/step -
accuracy: 0.7519 - loss: 0.7267 - val_accuracy: 0.7555 - val_loss: 0.7257 -
learning_rate: 1.0000e-04
Epoch 14/20
250/250              597s 2s/step -
accuracy: 0.7797 - loss: 0.6572 - val_accuracy: 0.7725 - val_loss: 0.6967 -
learning_rate: 1.0000e-04
Epoch 15/20
250/250              601s 2s/step -
accuracy: 0.7887 - loss: 0.6051 - val_accuracy: 0.7800 - val_loss: 0.7025 -
learning_rate: 1.0000e-04
Epoch 16/20
250/250              624s 2s/step -
accuracy: 0.8151 - loss: 0.5531 - val_accuracy: 0.7675 - val_loss: 0.7415 -
learning_rate: 1.0000e-04
Epoch 17/20
250/250              635s 2s/step -
accuracy: 0.8322 - loss: 0.4943 - val_accuracy: 0.7735 - val_loss: 0.6919 -
learning_rate: 1.0000e-04
Epoch 18/20
250/250              627s 2s/step -
accuracy: 0.8425 - loss: 0.4535 - val_accuracy: 0.7880 - val_loss: 0.6802 -
learning_rate: 1.0000e-04
Epoch 19/20
250/250              623s 2s/step -
accuracy: 0.8744 - loss: 0.3792 - val_accuracy: 0.7880 - val_loss: 0.6655 -
learning_rate: 1.0000e-04
Epoch 20/20
250/250              598s 2s/step -
accuracy: 0.8923 - loss: 0.3428 - val_accuracy: 0.8010 - val_loss: 0.6395 -
learning_rate: 1.0000e-04
```

```
[8]: <keras.src.callbacks.history.History at 0x7fd9b83c67e0>
```

3.3 7 - Check the accuracy

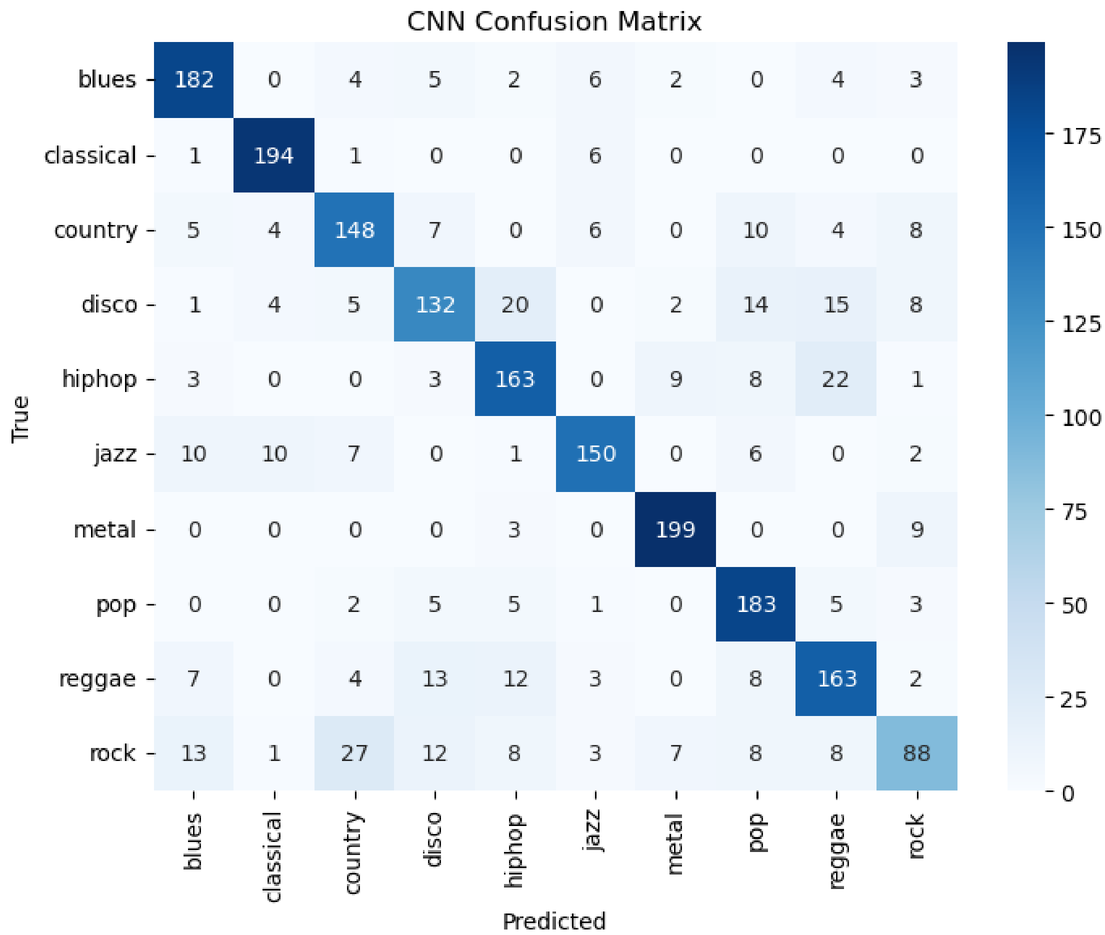
[9]:
```
evaluation = model.evaluate(X_test, y_test)
print(f"Test accuracy: {evaluation[1]:.3f}")
```

```
63/63              35s 562ms/step -
accuracy: 0.7960 - loss: 0.6598
Test accuracy: 0.801
```

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()
```

```
63/63              38s 586ms/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']
```

```python
FILE_PATH = os.path.join('Data', 'mel_spectrograms (3 secs)',␣
 ↪'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
50/50              142s 3s/step -
accuracy: 0.5391 - loss: 1.2428 - val_accuracy: 0.9225 - val_loss: 0.3014 -
learning_rate: 1.0000e-04
Epoch 2/20
50/50              127s 3s/step -
accuracy: 0.8315 - loss: 0.3835 - val_accuracy: 0.9550 - val_loss: 0.1265 -
learning_rate: 1.0000e-04
Epoch 3/20
```

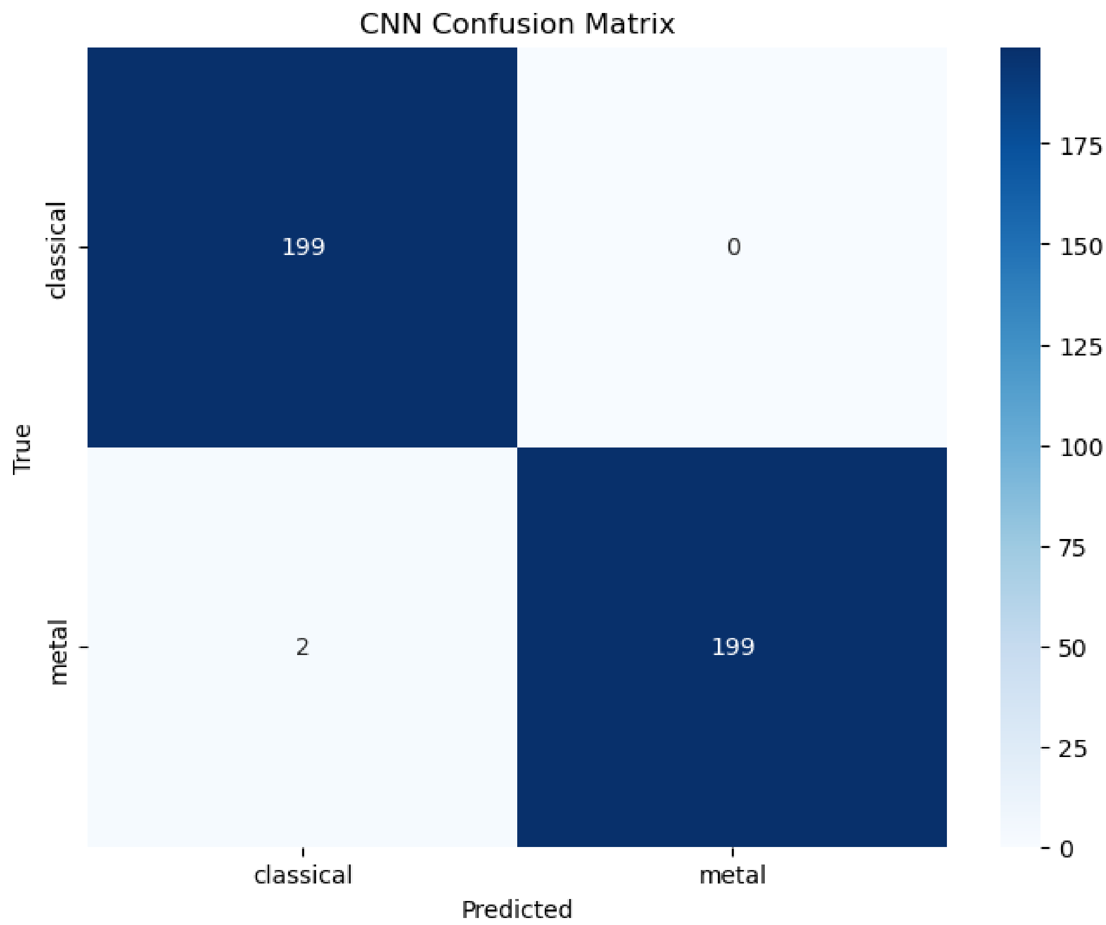
```
50/50            128s 3s/step -
accuracy: 0.9540 - loss: 0.1479 - val_accuracy: 0.9800 - val_loss: 0.0713 -
learning_rate: 1.0000e-04
Epoch 4/20
50/50            127s 3s/step -
accuracy: 0.9701 - loss: 0.0931 - val_accuracy: 0.9850 - val_loss: 0.0317 -
learning_rate: 1.0000e-04
Epoch 5/20
50/50            129s 3s/step -
accuracy: 0.9887 - loss: 0.0446 - val_accuracy: 0.9900 - val_loss: 0.0290 -
learning_rate: 1.0000e-04
Epoch 6/20
50/50            144s 3s/step -
accuracy: 0.9851 - loss: 0.0356 - val_accuracy: 0.9875 - val_loss: 0.0415 -
learning_rate: 1.0000e-04
Epoch 7/20
50/50            127s 3s/step -
accuracy: 0.9871 - loss: 0.0425 - val_accuracy: 0.9950 - val_loss: 0.0251 -
learning_rate: 1.0000e-04
Epoch 8/20
50/50            143s 3s/step -
accuracy: 0.9886 - loss: 0.0304 - val_accuracy: 0.9975 - val_loss: 0.0156 -
learning_rate: 1.0000e-04
Epoch 9/20
50/50            127s 3s/step -
accuracy: 0.9906 - loss: 0.0239 - val_accuracy: 0.9950 - val_loss: 0.0145 -
learning_rate: 1.0000e-04
Epoch 10/20
50/50            147s 3s/step -
accuracy: 0.9957 - loss: 0.0166 - val_accuracy: 0.9900 - val_loss: 0.0192 -
learning_rate: 1.0000e-04
Epoch 11/20
50/50            135s 2s/step -
accuracy: 0.9937 - loss: 0.0230 - val_accuracy: 0.9900 - val_loss: 0.0216 -
learning_rate: 1.0000e-04
Epoch 12/20
50/50            128s 3s/step -
accuracy: 0.9934 - loss: 0.0208 - val_accuracy: 0.9975 - val_loss: 0.0102 -
learning_rate: 1.0000e-04
Epoch 13/20
50/50            139s 3s/step -
accuracy: 0.9979 - loss: 0.0133 - val_accuracy: 0.9975 - val_loss: 0.0095 -
learning_rate: 1.0000e-04
Epoch 14/20
50/50            123s 2s/step -
accuracy: 0.9983 - loss: 0.0121 - val_accuracy: 0.9925 - val_loss: 0.0238 -
learning_rate: 1.0000e-04
Epoch 15/20
```

```
50/50              145s 3s/step -
accuracy: 0.9945 - loss: 0.0206 - val_accuracy: 0.9950 - val_loss: 0.0156 -
learning_rate: 1.0000e-04
Epoch 16/20
50/50              127s 3s/step -
accuracy: 0.9989 - loss: 0.0044 - val_accuracy: 0.9975 - val_loss: 0.0090 -
learning_rate: 1.0000e-04
Epoch 17/20
50/50              122s 2s/step -
accuracy: 0.9922 - loss: 0.0313 - val_accuracy: 0.9950 - val_loss: 0.0185 -
learning_rate: 1.0000e-04
Epoch 18/20
50/50              106s 2s/step -
accuracy: 0.9980 - loss: 0.0083 - val_accuracy: 0.9975 - val_loss: 0.0094 -
learning_rate: 1.0000e-04
Epoch 19/20
50/50              104s 2s/step -
accuracy: 0.9998 - loss: 0.0054 - val_accuracy: 0.9975 - val_loss: 0.0089 -
learning_rate: 1.0000e-04
Epoch 20/20
50/50              104s 2s/step -
accuracy: 0.9998 - loss: 0.0026 - val_accuracy: 0.9950 - val_loss: 0.0173 -
learning_rate: 1.0000e-04
13/13              6s 468ms/step -
accuracy: 0.9955 - loss: 0.0108
Test accuracy: 0.995
```

4.2 10 - Confusion Matrix Easy (classical and metal)

[12]:
```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              7s 477ms/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']
```

```python
FILE_PATH = os.path.join('Data', 'mel_spectrograms (3 secs)',␣
↪'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
50/50                112s 2s/step -
accuracy: 0.4823 - loss: 1.5432 - val_accuracy: 0.5025 - val_loss: 0.8406 -
learning_rate: 1.0000e-04
Epoch 2/20
50/50                103s 2s/step -
accuracy: 0.4917 - loss: 0.9068 - val_accuracy: 0.5025 - val_loss: 0.7451 -
learning_rate: 1.0000e-04
Epoch 3/20
```

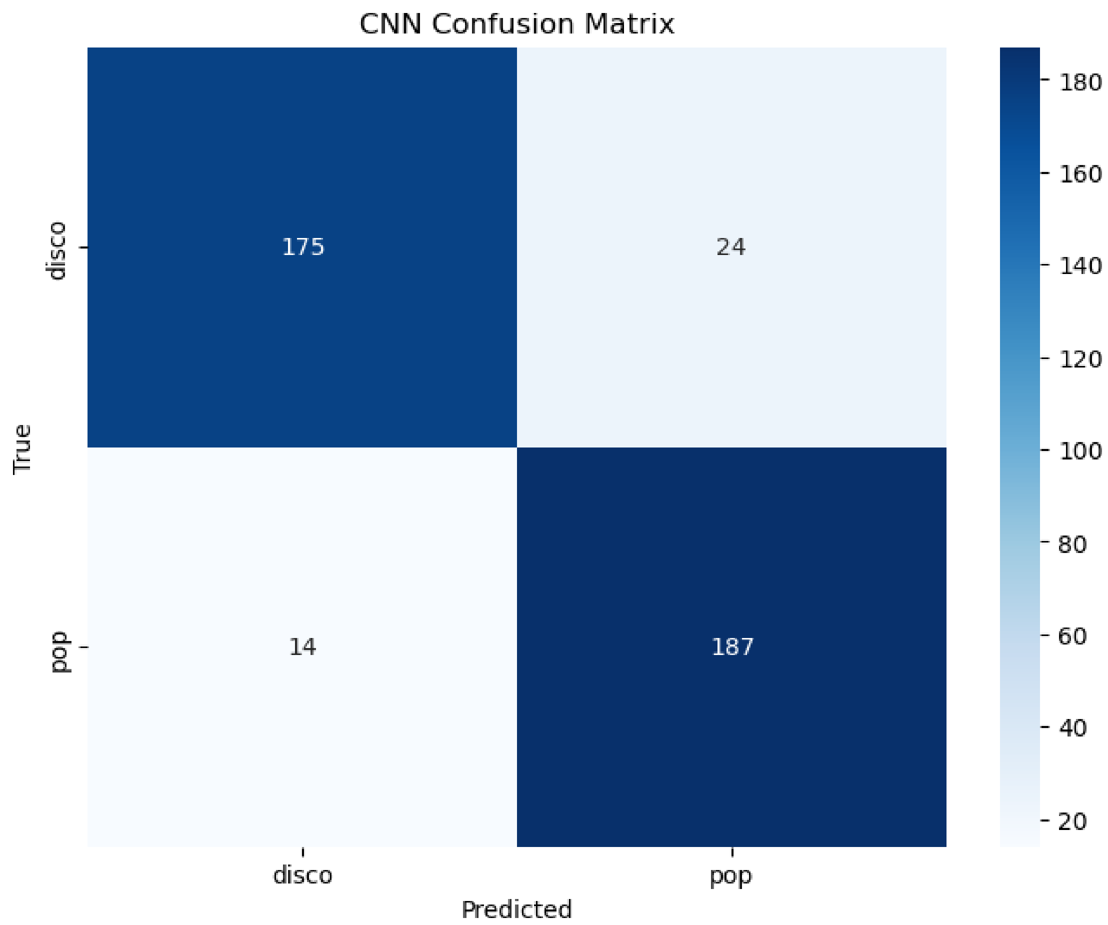
```
50/50              104s 2s/step -
accuracy: 0.5234 - loss: 0.7951 - val_accuracy: 0.5975 - val_loss: 0.6506 -
learning_rate: 1.0000e-04
Epoch 4/20
50/50              104s 2s/step -
accuracy: 0.6342 - loss: 0.6504 - val_accuracy: 0.7975 - val_loss: 0.4540 -
learning_rate: 1.0000e-04
Epoch 5/20
50/50              104s 2s/step -
accuracy: 0.7804 - loss: 0.4948 - val_accuracy: 0.7975 - val_loss: 0.4077 -
learning_rate: 1.0000e-04
Epoch 6/20
50/50              104s 2s/step -
accuracy: 0.8038 - loss: 0.4387 - val_accuracy: 0.8125 - val_loss: 0.3727 -
learning_rate: 1.0000e-04
Epoch 7/20
50/50              142s 2s/step -
accuracy: 0.8029 - loss: 0.4106 - val_accuracy: 0.8175 - val_loss: 0.3476 -
learning_rate: 1.0000e-04
Epoch 8/20
50/50              139s 2s/step -
accuracy: 0.8339 - loss: 0.3631 - val_accuracy: 0.8325 - val_loss: 0.3033 -
learning_rate: 1.0000e-04
Epoch 9/20
50/50              101s 2s/step -
accuracy: 0.8152 - loss: 0.3635 - val_accuracy: 0.8425 - val_loss: 0.3034 -
learning_rate: 1.0000e-04
Epoch 10/20
50/50              107s 2s/step -
accuracy: 0.8359 - loss: 0.3441 - val_accuracy: 0.8600 - val_loss: 0.2781 -
learning_rate: 1.0000e-04
Epoch 11/20
50/50              105s 2s/step -
accuracy: 0.8388 - loss: 0.3331 - val_accuracy: 0.8550 - val_loss: 0.2753 -
learning_rate: 1.0000e-04
Epoch 12/20
50/50              102s 2s/step -
accuracy: 0.8819 - loss: 0.2711 - val_accuracy: 0.8625 - val_loss: 0.2569 -
learning_rate: 1.0000e-04
Epoch 13/20
50/50              101s 2s/step -
accuracy: 0.8490 - loss: 0.3158 - val_accuracy: 0.8725 - val_loss: 0.2816 -
learning_rate: 1.0000e-04
Epoch 14/20
50/50              144s 2s/step -
accuracy: 0.8790 - loss: 0.2737 - val_accuracy: 0.8700 - val_loss: 0.2544 -
learning_rate: 1.0000e-04
Epoch 15/20
```

```
50/50                102s 2s/step -
accuracy: 0.8912 - loss: 0.2553 - val_accuracy: 0.8625 - val_loss: 0.2457 -
learning_rate: 1.0000e-04
Epoch 16/20
50/50                104s 2s/step -
accuracy: 0.8820 - loss: 0.2635 - val_accuracy: 0.8875 - val_loss: 0.2379 -
learning_rate: 1.0000e-04
Epoch 17/20
50/50                140s 2s/step -
accuracy: 0.8972 - loss: 0.2342 - val_accuracy: 0.8950 - val_loss: 0.2388 -
learning_rate: 1.0000e-04
Epoch 18/20
50/50                103s 2s/step -
accuracy: 0.8979 - loss: 0.2548 - val_accuracy: 0.8825 - val_loss: 0.2874 -
learning_rate: 1.0000e-04
Epoch 19/20
50/50                103s 2s/step -
accuracy: 0.8868 - loss: 0.2602 - val_accuracy: 0.9000 - val_loss: 0.2397 -
learning_rate: 1.0000e-04
Epoch 20/20
50/50                102s 2s/step -
accuracy: 0.8984 - loss: 0.2308 - val_accuracy: 0.9050 - val_loss: 0.2460 -
learning_rate: 5.0000e-05
13/13                6s 435ms/step -
accuracy: 0.9098 - loss: 0.2490
Test accuracy: 0.905
```

4.4 12 - Confusion Matrix Hard (disco and pop)

[14]:
```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                7s 461ms/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 (3 secs)',␣
↪'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}")
```

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

```
125/125 268s 2s/step -
accuracy: 0.2023 - loss: 1.9665 - val_accuracy: 0.1920 - val_loss: 1.6578 -
learning_rate: 1.0000e-04
Epoch 2/20
125/125 259s 2s/step -
accuracy: 0.2381 - loss: 1.6973 - val_accuracy: 0.3700 - val_loss: 1.5228 -
learning_rate: 1.0000e-04
Epoch 3/20
125/125 258s 2s/step -
accuracy: 0.3272 - loss: 1.5411 - val_accuracy: 0.4480 - val_loss: 1.3061 -
learning_rate: 1.0000e-04
Epoch 4/20
125/125 259s 2s/step -
accuracy: 0.4059 - loss: 1.3973 - val_accuracy: 0.5410 - val_loss: 1.2085 -
learning_rate: 1.0000e-04
Epoch 5/20
125/125 262s 2s/step -
accuracy: 0.4607 - loss: 1.2693 - val_accuracy: 0.5790 - val_loss: 1.1131 -
learning_rate: 1.0000e-04
Epoch 6/20
125/125 256s 2s/step -
accuracy: 0.5165 - loss: 1.1971 - val_accuracy: 0.5570 - val_loss: 1.1562 -
learning_rate: 1.0000e-04
Epoch 7/20
125/125 259s 2s/step -
accuracy: 0.5419 - loss: 1.1400 - val_accuracy: 0.5970 - val_loss: 1.0141 -
learning_rate: 1.0000e-04
Epoch 8/20
125/125 259s 2s/step -
accuracy: 0.5516 - loss: 1.1023 - val_accuracy: 0.6090 - val_loss: 0.9652 -
learning_rate: 1.0000e-04
Epoch 9/20
125/125 262s 2s/step -
accuracy: 0.5967 - loss: 0.9989 - val_accuracy: 0.6250 - val_loss: 0.9531 -
learning_rate: 1.0000e-04
Epoch 10/20
125/125 258s 2s/step -
accuracy: 0.6304 - loss: 0.9560 - val_accuracy: 0.6540 - val_loss: 0.8696 -
learning_rate: 1.0000e-04
Epoch 11/20
125/125 256s 2s/step -
accuracy: 0.6663 - loss: 0.8965 - val_accuracy: 0.6690 - val_loss: 0.8236 -
learning_rate: 1.0000e-04
Epoch 12/20
125/125 265s 2s/step -
accuracy: 0.6783 - loss: 0.8314 - val_accuracy: 0.6710 - val_loss: 0.8520 -
learning_rate: 1.0000e-04
Epoch 13/20
```

```
125/125 225s 2s/step -
accuracy: 0.7070 - loss: 0.7839 - val_accuracy: 0.7140 - val_loss: 0.8016 -
learning_rate: 1.0000e-04
Epoch 14/20
125/125 204s 2s/step -
accuracy: 0.7225 - loss: 0.7320 - val_accuracy: 0.7180 - val_loss: 0.7440 -
learning_rate: 1.0000e-04
Epoch 15/20
125/125 224s 2s/step -
accuracy: 0.7521 - loss: 0.6408 - val_accuracy: 0.7260 - val_loss: 0.7145 -
learning_rate: 1.0000e-04
Epoch 16/20
125/125 283s 2s/step -
accuracy: 0.7738 - loss: 0.6081 - val_accuracy: 0.7380 - val_loss: 0.6929 -
learning_rate: 1.0000e-04
Epoch 17/20
125/125 285s 2s/step -
accuracy: 0.7770 - loss: 0.5765 - val_accuracy: 0.7460 - val_loss: 0.7053 -
learning_rate: 1.0000e-04
Epoch 18/20
125/125 255s 2s/step -
accuracy: 0.8107 - loss: 0.5148 - val_accuracy: 0.7650 - val_loss: 0.6530 -
learning_rate: 1.0000e-04
Epoch 19/20
125/125 237s 2s/step -
accuracy: 0.8276 - loss: 0.4753 - val_accuracy: 0.7420 - val_loss: 0.7406 -
learning_rate: 1.0000e-04
Epoch 20/20
125/125 240s 2s/step -
accuracy: 0.8366 - loss: 0.4403 - val_accuracy: 0.7750 - val_loss: 0.6317 -
learning_rate: 1.0000e-04
32/32 15s 467ms/step -
accuracy: 0.7772 - loss: 0.5912
Test accuracy: 0.775
```

4.6 14 - Confusion Matrix Medium (5 random)

[16]:
```
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()
```

```
32/32                16s 480ms/step
```