Tempogram-Only (3 secs) CNN

March 20, 2025

1 CNN for Tempogram (3 secs)

1.1 1 - All the imports

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

```
2025-03-20 08:41:59.031852: 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]:
```python
# 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]:
```python
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)
```

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

```
Model: "sequential"


 Layer (type)                     Output Shape                  Param #

 conv2d (Conv2D)                  (None, 254, 254, 32)              320

 batch_normalization              (None, 254, 254, 32)              128
 (BatchNormalization)

 max_pooling2d (MaxPooling2D)     (None, 127, 127, 32)                0
```

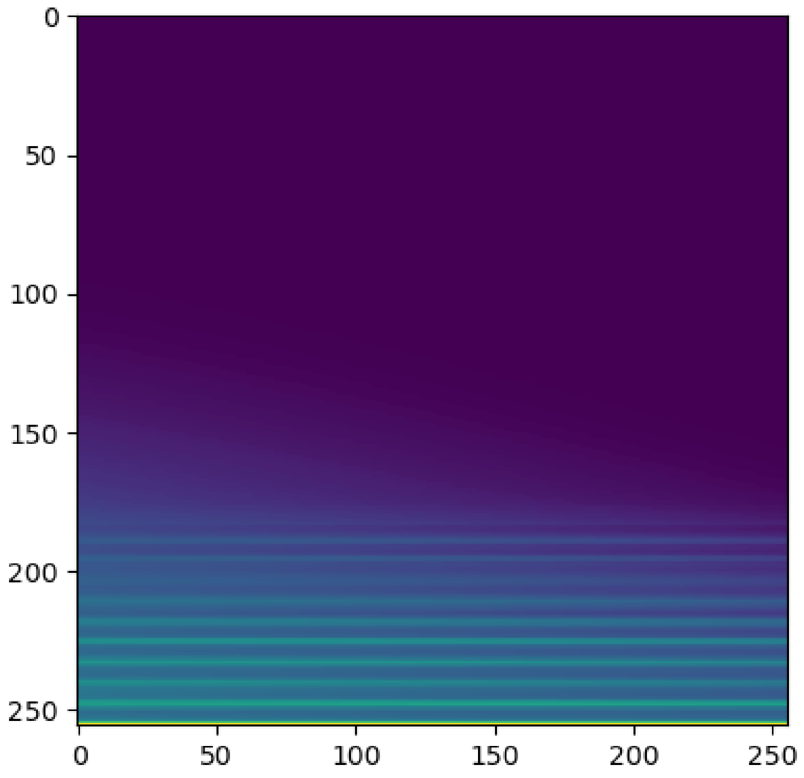
```
conv2d_1 (Conv2D)                (None, 125, 125, 64)           18,496

batch_normalization_1            (None, 125, 125, 64)              256
(BatchNormalization)

max_pooling2d_1 (MaxPooling2D)   (None, 62, 62, 64)                  0

conv2d_2 (Conv2D)                (None, 60, 60, 128)            73,856

batch_normalization_2            (None, 60, 60, 128)               512
(BatchNormalization)

max_pooling2d_2 (MaxPooling2D)   (None, 30, 30, 128)                 0

conv2d_3 (Conv2D)                (None, 28, 28, 256)           295,168

batch_normalization_3            (None, 28, 28, 256)             1,024
(BatchNormalization)

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

[5]:
```python
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', 'tempograms (3 secs)')
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)
```

```
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()
```



[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)
```

```
[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              1846s 7s/step -
accuracy: 0.1107 - loss: 3.2723 - val_accuracy: 0.0975 - val_loss: 2.3432 -
learning_rate: 1.0000e-04
Epoch 2/20
250/250              1956s 8s/step -
accuracy: 0.0971 - loss: 2.3040 - val_accuracy: 0.1055 - val_loss: 2.3580 -
learning_rate: 1.0000e-04
Epoch 3/20
250/250              1642s 7s/step -
accuracy: 0.1029 - loss: 2.3108 - val_accuracy: 0.0920 - val_loss: 2.3029 -
learning_rate: 1.0000e-04
Epoch 4/20
250/250              1378s 6s/step -
accuracy: 0.1089 - loss: 2.3031 - val_accuracy: 0.0930 - val_loss: 2.3029 -
learning_rate: 1.0000e-04
Epoch 5/20
250/250              1347s 5s/step -
accuracy: 0.1057 - loss: 2.3047 - val_accuracy: 0.0930 - val_loss: 2.3030 -
learning_rate: 1.0000e-04
Epoch 6/20
250/250              1301s 5s/step -
accuracy: 0.1010 - loss: 2.3026 - val_accuracy: 0.0920 - val_loss: 2.3033 -
learning_rate: 1.0000e-04
Epoch 7/20
250/250              1123s 4s/step -
accuracy: 0.0993 - loss: 2.3022 - val_accuracy: 0.0930 - val_loss: 2.3031 -
learning_rate: 5.0000e-05
Epoch 8/20
250/250              1120s 4s/step -
accuracy: 0.1059 - loss: 2.3020 - val_accuracy: 0.0915 - val_loss: 2.3031 -
learning_rate: 5.0000e-05
Epoch 9/20
250/250              1138s 4s/step -
accuracy: 0.0996 - loss: 2.3033 - val_accuracy: 0.0930 - val_loss: 2.3031 -
learning_rate: 5.0000e-05
Epoch 10/20
250/250              1137s 5s/step -
accuracy: 0.0987 - loss: 2.3031 - val_accuracy: 0.0930 - val_loss: 2.3031 -
learning_rate: 2.5000e-05
Epoch 11/20
250/250              1105s 4s/step -
accuracy: 0.1025 - loss: 2.3023 - val_accuracy: 0.0925 - val_loss: 2.3031 -
learning_rate: 2.5000e-05
Epoch 12/20
```

```
250/250              1099s 4s/step -
accuracy: 0.0992 - loss: 2.3025 - val_accuracy: 0.0930 - val_loss: 2.3031 -
learning_rate: 2.5000e-05
Epoch 13/20
250/250              1127s 4s/step -
accuracy: 0.1038 - loss: 2.3024 - val_accuracy: 0.0930 - val_loss: 2.3031 -
learning_rate: 1.2500e-05
Epoch 14/20
250/250              1022s 4s/step -
accuracy: 0.1034 - loss: 2.3023 - val_accuracy: 0.0975 - val_loss: 2.3026 -
learning_rate: 1.2500e-05
Epoch 15/20
250/250              938s 4s/step -
accuracy: 0.1067 - loss: 2.3024 - val_accuracy: 0.0950 - val_loss: 2.3026 -
learning_rate: 1.2500e-05
Epoch 16/20
250/250              930s 4s/step -
accuracy: 0.1042 - loss: 2.3006 - val_accuracy: 0.1125 - val_loss: 2.2951 -
learning_rate: 1.2500e-05
Epoch 17/20
250/250              904s 4s/step -
accuracy: 0.1118 - loss: 2.2995 - val_accuracy: 0.1530 - val_loss: 2.2610 -
learning_rate: 1.2500e-05
Epoch 18/20
250/250              944s 4s/step -
accuracy: 0.1241 - loss: 2.2888 - val_accuracy: 0.1415 - val_loss: 2.2728 -
learning_rate: 1.2500e-05
Epoch 19/20
250/250              984s 4s/step -
accuracy: 0.1328 - loss: 2.2760 - val_accuracy: 0.1465 - val_loss: 2.2554 -
learning_rate: 1.2500e-05
Epoch 20/20
250/250              927s 4s/step -
accuracy: 0.1410 - loss: 2.2687 - val_accuracy: 0.1705 - val_loss: 2.2497 -
learning_rate: 1.2500e-05
```

[8]:
```
<keras.src.callbacks.history.History at 0x7f074c10af90>
```

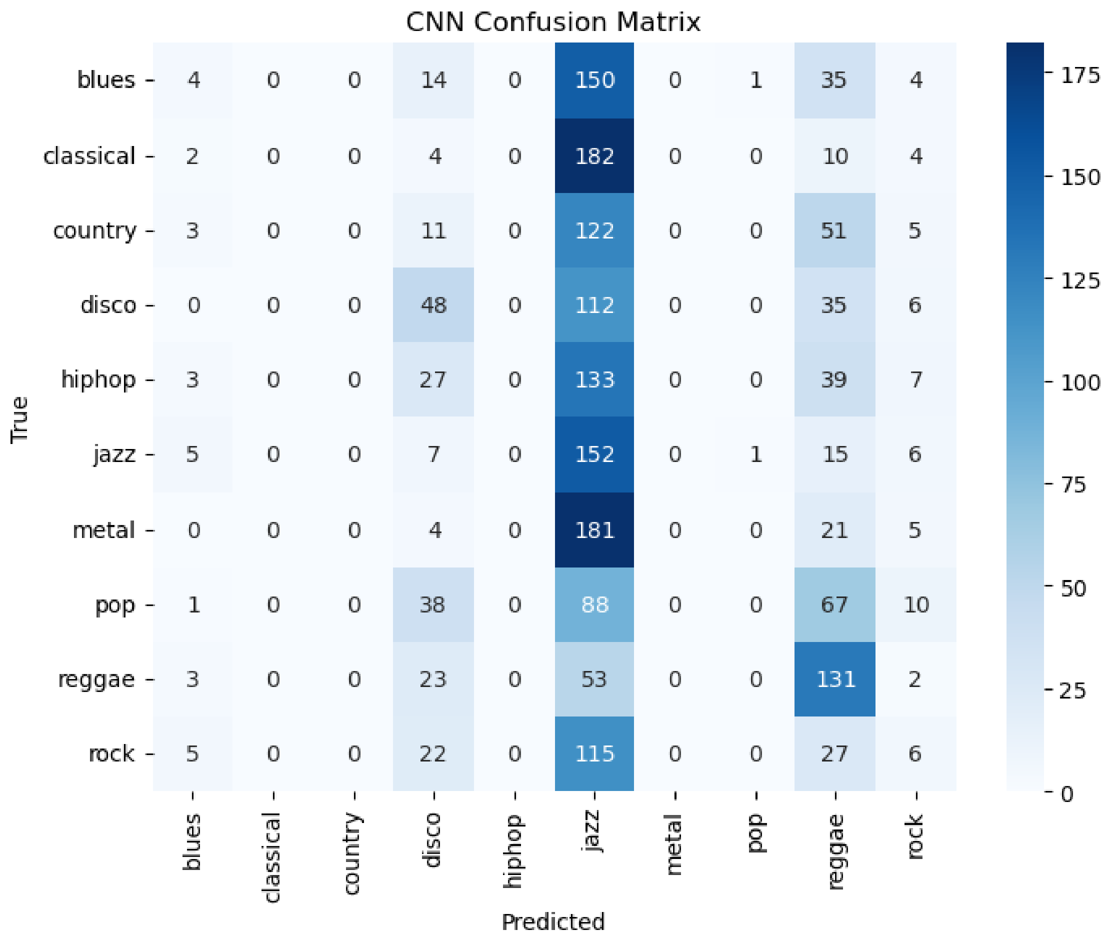
[9]:
```
evaluation = model.evaluate(X_test, y_test)
print(f"Test accuracy: {evaluation[1]:.3f}")
```

```
63/63              29s 466ms/step -
accuracy: 0.1732 - loss: 2.2505
Test accuracy: 0.170
```

4 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                  28s 443ms/step
```



	blues	classical	country	disco	hiphop	jazz	metal	pop	reggae	rock
blues	4	0	0	14	0	150	0	1	35	4
classical	2	0	0	4	0	182	0	0	10	4
country	3	0	0	11	0	122	0	0	51	5
disco	0	0	0	48	0	112	0	0	35	6
hiphop	3	0	0	27	0	133	0	0	39	7
jazz	5	0	0	7	0	152	0	1	15	6
metal	0	0	0	4	0	181	0	0	21	5
pop	1	0	0	38	0	88	0	0	67	10
reggae	3	0	0	23	0	53	0	0	131	2
rock	5	0	0	22	0	115	0	0	27	6

4.1 9 - Limited Genres Easy (metal and classical)

[11]:
```python
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', 'tempograms (3 secs)')
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)
```

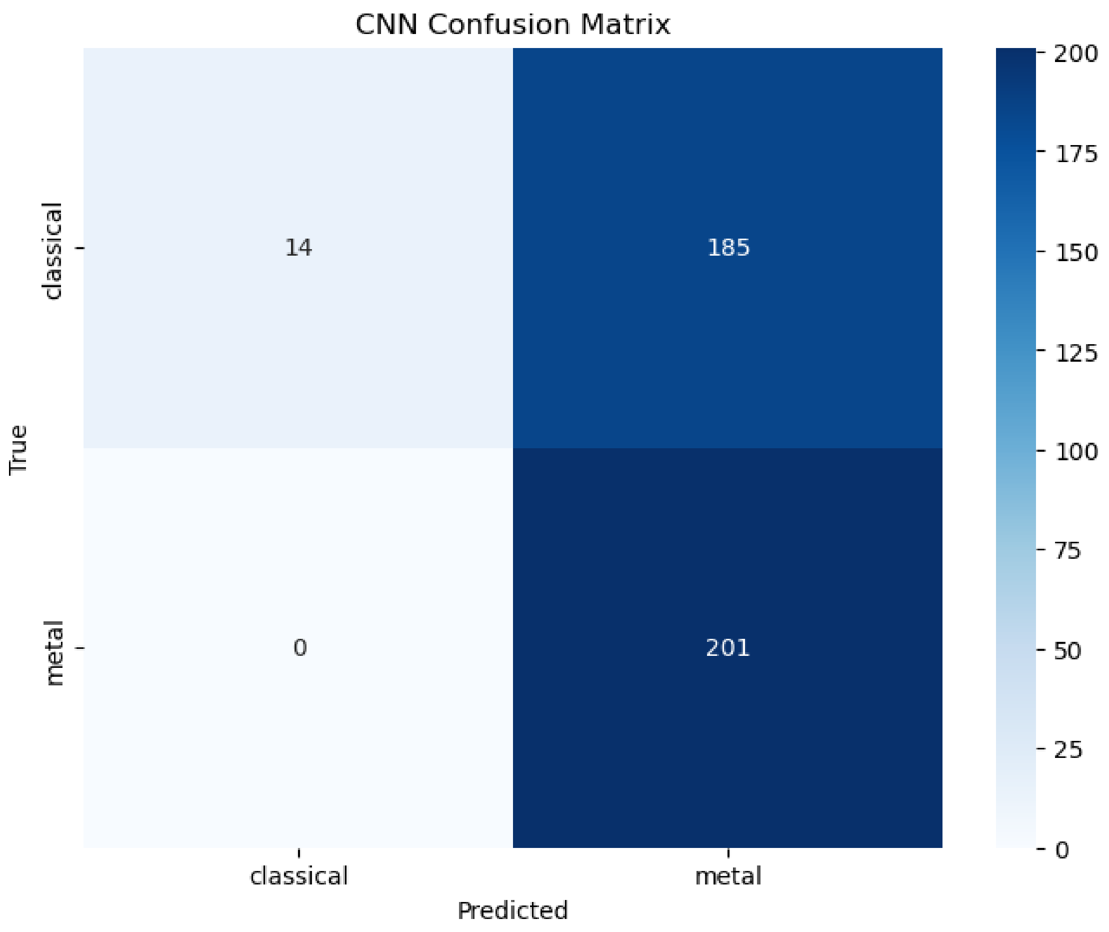
```
Epoch 1/20
50/50            87s 2s/step -
accuracy: 0.4488 - loss: 1.6158 - val_accuracy: 0.4875 - val_loss: 0.7366 -
learning_rate: 1.0000e-04
Epoch 2/20
50/50            86s 2s/step -
accuracy: 0.4632 - loss: 0.8819 - val_accuracy: 0.5025 - val_loss: 0.7149 -
learning_rate: 1.0000e-04
Epoch 3/20
50/50            92s 2s/step -
accuracy: 0.5114 - loss: 0.7963 - val_accuracy: 0.5025 - val_loss: 0.7058 -
learning_rate: 1.0000e-04
Epoch 4/20
50/50            151s 2s/step -
accuracy: 0.5119 - loss: 0.7515 - val_accuracy: 0.5025 - val_loss: 0.7045 -
learning_rate: 1.0000e-04
Epoch 5/20
50/50            94s 2s/step -
accuracy: 0.4782 - loss: 0.7688 - val_accuracy: 0.4975 - val_loss: 0.6977 -
learning_rate: 1.0000e-04
Epoch 6/20
50/50            161s 2s/step -
accuracy: 0.4965 - loss: 0.7432 - val_accuracy: 0.4975 - val_loss: 0.6988 -
learning_rate: 1.0000e-04
Epoch 7/20
50/50            97s 2s/step -
accuracy: 0.5427 - loss: 0.7220 - val_accuracy: 0.4975 - val_loss: 0.6972 -
learning_rate: 1.0000e-04
Epoch 8/20
50/50            107s 2s/step -
accuracy: 0.5343 - loss: 0.7180 - val_accuracy: 0.4975 - val_loss: 0.6979 -
learning_rate: 1.0000e-04
Epoch 9/20
50/50            108s 2s/step -
accuracy: 0.5194 - loss: 0.7140 - val_accuracy: 0.5025 - val_loss: 0.6963 -
learning_rate: 1.0000e-04
Epoch 10/20
50/50            107s 2s/step -
accuracy: 0.4949 - loss: 0.7308 - val_accuracy: 0.4950 - val_loss: 0.6948 -
learning_rate: 1.0000e-04
```

```
Epoch 11/20
50/50            104s 2s/step -
accuracy: 0.5241 - loss: 0.7153 - val_accuracy: 0.5025 - val_loss: 0.7000 -
learning_rate: 1.0000e-04
Epoch 12/20
50/50            106s 2s/step -
accuracy: 0.4997 - loss: 0.7124 - val_accuracy: 0.5425 - val_loss: 0.6928 -
learning_rate: 1.0000e-04
Epoch 13/20
50/50            129s 2s/step -
accuracy: 0.4742 - loss: 0.7269 - val_accuracy: 0.5025 - val_loss: 0.6970 -
learning_rate: 1.0000e-04
Epoch 14/20
50/50            94s 2s/step -
accuracy: 0.5180 - loss: 0.7123 - val_accuracy: 0.4975 - val_loss: 0.6932 -
learning_rate: 1.0000e-04
Epoch 15/20
50/50            100s 2s/step -
accuracy: 0.4656 - loss: 0.7232 - val_accuracy: 0.5175 - val_loss: 0.6935 -
learning_rate: 1.0000e-04
Epoch 16/20
50/50            136s 2s/step -
accuracy: 0.5044 - loss: 0.7168 - val_accuracy: 0.5025 - val_loss: 0.6926 -
learning_rate: 5.0000e-05
Epoch 17/20
50/50            129s 2s/step -
accuracy: 0.5048 - loss: 0.7062 - val_accuracy: 0.6400 - val_loss: 0.6936 -
learning_rate: 5.0000e-05
Epoch 18/20
50/50            60s 1s/step -
accuracy: 0.5059 - loss: 0.7114 - val_accuracy: 0.6400 - val_loss: 0.6929 -
learning_rate: 5.0000e-05
Epoch 19/20
50/50            85s 2s/step -
accuracy: 0.5209 - loss: 0.7084 - val_accuracy: 0.5375 - val_loss: 0.6937 -
learning_rate: 5.0000e-05
Epoch 20/20
50/50            87s 2s/step -
accuracy: 0.4852 - loss: 0.7141 - val_accuracy: 0.5375 - val_loss: 0.6924 -
learning_rate: 2.5000e-05
13/13            5s 381ms/step -
accuracy: 0.5281 - loss: 0.6925
Test accuracy: 0.538
```

4.2 10 - Confusion Matrix Easy (classical and metal)

```
[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()
```

```
13/13              5s 391ms/step
```



4.3 11 - Limited genres Hard (disco and pop)

[13]:
```python
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', 'tempograms (3 secs)')
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)
```

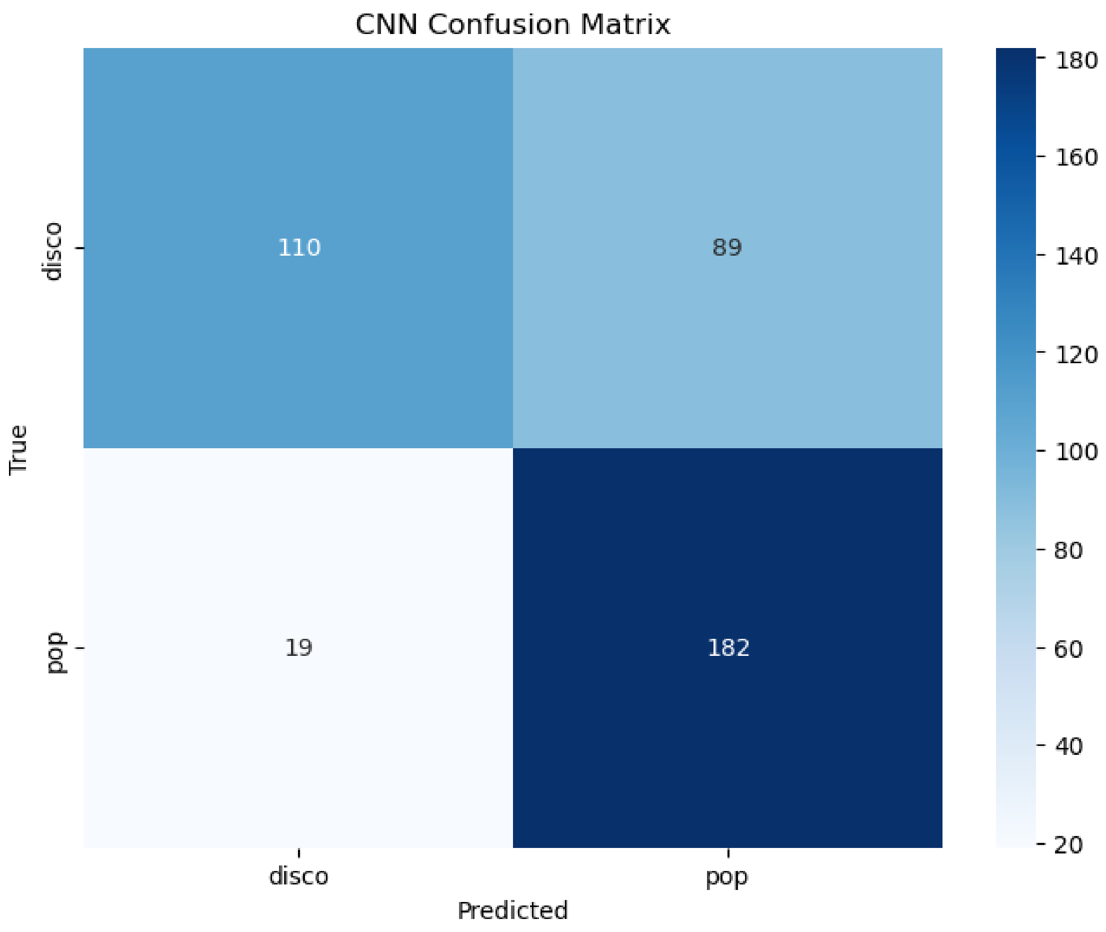
```
Epoch 1/20
50/50            93s 2s/step -
accuracy: 0.4358 - loss: 1.5702 - val_accuracy: 0.5025 - val_loss: 0.7514 -
learning_rate: 1.0000e-04
Epoch 2/20
50/50            85s 2s/step -
accuracy: 0.5055 - loss: 0.8618 - val_accuracy: 0.4975 - val_loss: 0.7315 -
learning_rate: 1.0000e-04
Epoch 3/20
50/50            79s 2s/step -
accuracy: 0.5094 - loss: 0.8087 - val_accuracy: 0.6000 - val_loss: 0.7177 -
learning_rate: 1.0000e-04
Epoch 4/20
50/50            68s 1s/step -
accuracy: 0.5013 - loss: 0.7872 - val_accuracy: 0.5025 - val_loss: 0.7353 -
learning_rate: 1.0000e-04
Epoch 5/20
50/50            76s 2s/step -
accuracy: 0.5387 - loss: 0.7382 - val_accuracy: 0.5025 - val_loss: 0.7043 -
learning_rate: 1.0000e-04
Epoch 6/20
50/50            85s 2s/step -
accuracy: 0.4790 - loss: 0.7619 - val_accuracy: 0.4975 - val_loss: 0.7018 -
learning_rate: 1.0000e-04
Epoch 7/20
50/50            77s 2s/step -
accuracy: 0.4961 - loss: 0.7465 - val_accuracy: 0.4975 - val_loss: 0.7105 -
learning_rate: 1.0000e-04
Epoch 8/20
50/50            78s 2s/step -
accuracy: 0.4984 - loss: 0.7297 - val_accuracy: 0.4975 - val_loss: 0.6993 -
learning_rate: 1.0000e-04
Epoch 9/20
50/50            82s 2s/step -
accuracy: 0.5192 - loss: 0.7251 - val_accuracy: 0.5900 - val_loss: 0.6970 -
learning_rate: 1.0000e-04
Epoch 10/20
50/50            74s 1s/step -
accuracy: 0.5024 - loss: 0.7182 - val_accuracy: 0.4975 - val_loss: 0.6926 -
```

```
learning_rate: 1.0000e-04
Epoch 11/20
50/50            86s 2s/step -
accuracy: 0.5147 - loss: 0.7292 - val_accuracy: 0.5825 - val_loss: 0.6795 -
learning_rate: 1.0000e-04
Epoch 12/20
50/50            86s 2s/step -
accuracy: 0.5181 - loss: 0.7168 - val_accuracy: 0.6025 - val_loss: 0.6741 -
learning_rate: 1.0000e-04
Epoch 13/20
50/50            85s 2s/step -
accuracy: 0.5525 - loss: 0.7013 - val_accuracy: 0.6425 - val_loss: 0.6603 -
learning_rate: 1.0000e-04
Epoch 14/20
50/50            78s 2s/step -
accuracy: 0.5737 - loss: 0.6865 - val_accuracy: 0.6050 - val_loss: 0.6577 -
learning_rate: 1.0000e-04
Epoch 15/20
50/50            77s 2s/step -
accuracy: 0.6078 - loss: 0.6641 - val_accuracy: 0.6725 - val_loss: 0.6176 -
learning_rate: 1.0000e-04
Epoch 16/20
50/50            76s 2s/step -
accuracy: 0.6471 - loss: 0.6437 - val_accuracy: 0.7100 - val_loss: 0.6101 -
learning_rate: 1.0000e-04
Epoch 17/20
50/50            80s 2s/step -
accuracy: 0.6867 - loss: 0.6134 - val_accuracy: 0.7350 - val_loss: 0.5834 -
learning_rate: 1.0000e-04
Epoch 18/20
50/50            73s 1s/step -
accuracy: 0.7021 - loss: 0.5804 - val_accuracy: 0.6925 - val_loss: 0.6476 -
learning_rate: 1.0000e-04
Epoch 19/20
50/50            82s 2s/step -
accuracy: 0.7163 - loss: 0.5865 - val_accuracy: 0.7525 - val_loss: 0.5558 -
learning_rate: 1.0000e-04
Epoch 20/20
50/50            74s 1s/step -
accuracy: 0.7337 - loss: 0.5382 - val_accuracy: 0.7300 - val_loss: 0.5693 -
learning_rate: 1.0000e-04
13/13            4s 283ms/step -
accuracy: 0.7387 - loss: 0.5738
Test accuracy: 0.730
```

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()
```

```
13/13                 5s 379ms/step
```



4.5 13 - Limited Genres Medium (5 random)

[15]:
```
import tensorflow as tf
import os
import numpy as np
from sklearn.model_selection import train_test_split
import random

GENRES = ['blues', 'classical', 'country', 'disco', 'hiphop', 'jazz', 'metal',␣
 ↪'pop', 'reggae', 'rock']
GENRES = random.sample(GENRES, 5)
print(GENRES)
FILE_PATH = os.path.join('Data', 'tempograms (3 secs)')
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])
```

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

```
['metal', 'disco', 'jazz', 'classical', 'rock']
Going through metal
Going through disco
Going through jazz
Going through classical
Going through rock
```

```
/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              215s 2s/step -
accuracy: 0.1996 - loss: 1.9834 - val_accuracy: 0.1920 - val_loss: 1.6781 -
learning_rate: 1.0000e-04
Epoch 2/20
125/125              229s 1s/step -
accuracy: 0.2443 - loss: 1.6729 - val_accuracy: 0.3200 - val_loss: 1.5157 -
learning_rate: 1.0000e-04
Epoch 3/20
125/125              170s 1s/step -
accuracy: 0.3162 - loss: 1.5440 - val_accuracy: 0.4110 - val_loss: 1.4048 -
learning_rate: 1.0000e-04
Epoch 4/20
125/125              208s 2s/step -
accuracy: 0.3493 - loss: 1.4838 - val_accuracy: 0.4010 - val_loss: 1.4004 -
learning_rate: 1.0000e-04
Epoch 5/20
125/125              212s 2s/step -
accuracy: 0.3532 - loss: 1.4689 - val_accuracy: 0.4090 - val_loss: 1.3775 -
learning_rate: 1.0000e-04
Epoch 6/20
125/125              202s 2s/step -
accuracy: 0.3750 - loss: 1.4490 - val_accuracy: 0.4240 - val_loss: 1.3526 -
learning_rate: 1.0000e-04
Epoch 7/20
125/125              213s 2s/step -
accuracy: 0.3818 - loss: 1.4062 - val_accuracy: 0.4440 - val_loss: 1.3735 -
learning_rate: 1.0000e-04
Epoch 8/20
125/125              214s 2s/step -
accuracy: 0.3957 - loss: 1.3938 - val_accuracy: 0.4540 - val_loss: 1.3376 -
```

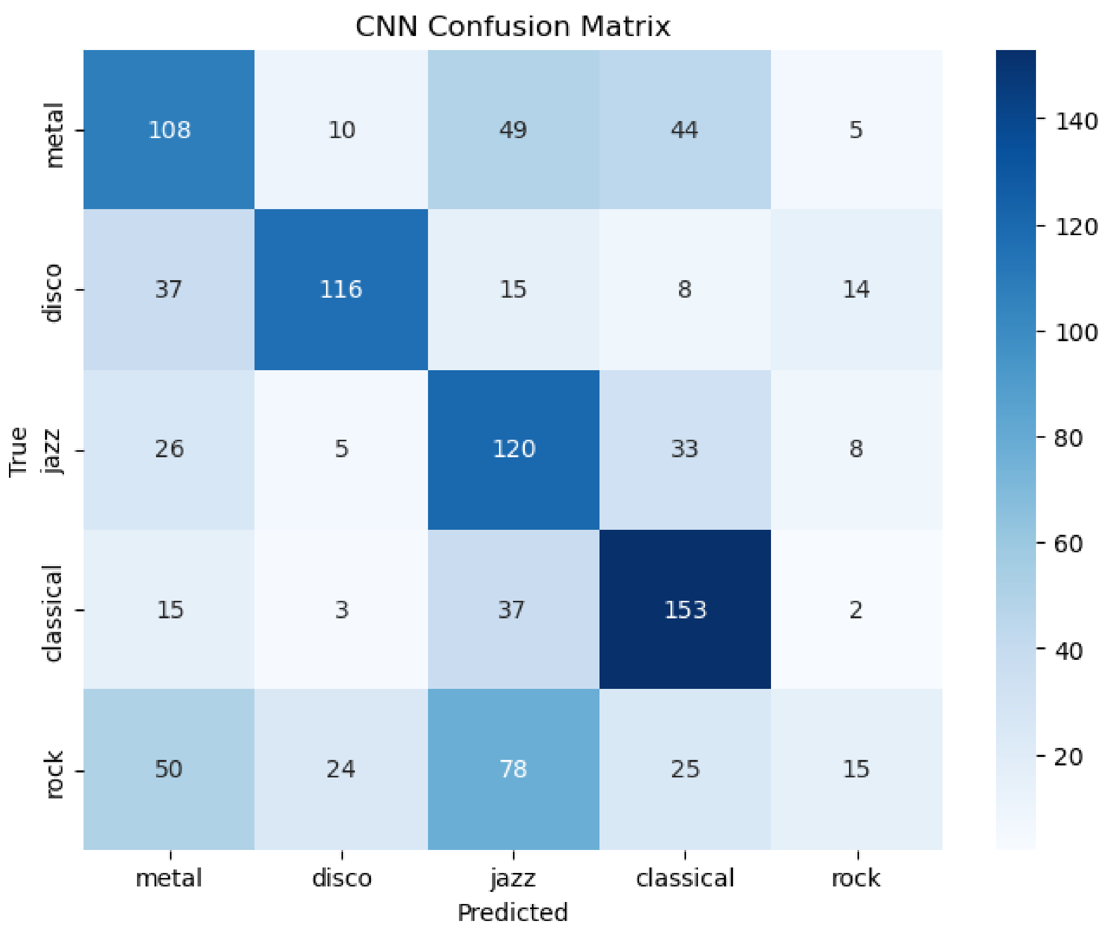
```
learning_rate: 1.0000e-04
Epoch 9/20
125/125            203s 2s/step -
accuracy: 0.4173 - loss: 1.3556 - val_accuracy: 0.4480 - val_loss: 1.3074 -
learning_rate: 1.0000e-04
Epoch 10/20
125/125            195s 2s/step -
accuracy: 0.4193 - loss: 1.3393 - val_accuracy: 0.4600 - val_loss: 1.2939 -
learning_rate: 1.0000e-04
Epoch 11/20
125/125            210s 2s/step -
accuracy: 0.4387 - loss: 1.3117 - val_accuracy: 0.4470 - val_loss: 1.3248 -
learning_rate: 1.0000e-04
Epoch 12/20
125/125            203s 2s/step -
accuracy: 0.4431 - loss: 1.3166 - val_accuracy: 0.4390 - val_loss: 1.3498 -
learning_rate: 1.0000e-04
Epoch 13/20
125/125            214s 2s/step -
accuracy: 0.4466 - loss: 1.3065 - val_accuracy: 0.4780 - val_loss: 1.2788 -
learning_rate: 1.0000e-04
Epoch 14/20
125/125            211s 2s/step -
accuracy: 0.4672 - loss: 1.2686 - val_accuracy: 0.4790 - val_loss: 1.2579 -
learning_rate: 1.0000e-04
Epoch 15/20
125/125            213s 2s/step -
accuracy: 0.4530 - loss: 1.2769 - val_accuracy: 0.4520 - val_loss: 1.3249 -
learning_rate: 1.0000e-04
Epoch 16/20
125/125            214s 2s/step -
accuracy: 0.4750 - loss: 1.2655 - val_accuracy: 0.4910 - val_loss: 1.2537 -
learning_rate: 1.0000e-04
Epoch 17/20
125/125            200s 2s/step -
accuracy: 0.4596 - loss: 1.2684 - val_accuracy: 0.4950 - val_loss: 1.2662 -
learning_rate: 1.0000e-04
Epoch 18/20
125/125            213s 2s/step -
accuracy: 0.4943 - loss: 1.2298 - val_accuracy: 0.5020 - val_loss: 1.2408 -
learning_rate: 1.0000e-04
Epoch 19/20
125/125            259s 2s/step -
accuracy: 0.4795 - loss: 1.2475 - val_accuracy: 0.5160 - val_loss: 1.2006 -
learning_rate: 1.0000e-04
Epoch 20/20
125/125            201s 2s/step -
accuracy: 0.4924 - loss: 1.2366 - val_accuracy: 0.5120 - val_loss: 1.2230 -
```

```
learning_rate: 1.0000e-04
32/32              12s 372ms/step -
accuracy: 0.5205 - loss: 1.1956
Test accuracy: 0.512
```

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              13s 390ms/step
```


CNN Confusion Matrix
True
Predicted
metal
disco
jazz
classical
rock
108 10 49 44 5
37 116 15 8 14
26 5 120 33 8
15 3 37 153 2
50 24 78 25 15
140
120
100
80
60
40
20

```
[ ]:
```