Tempogram-Only CNN

March 20, 2025

1 CNN for Tempogram

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 08:42:01.904114: 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, 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 (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

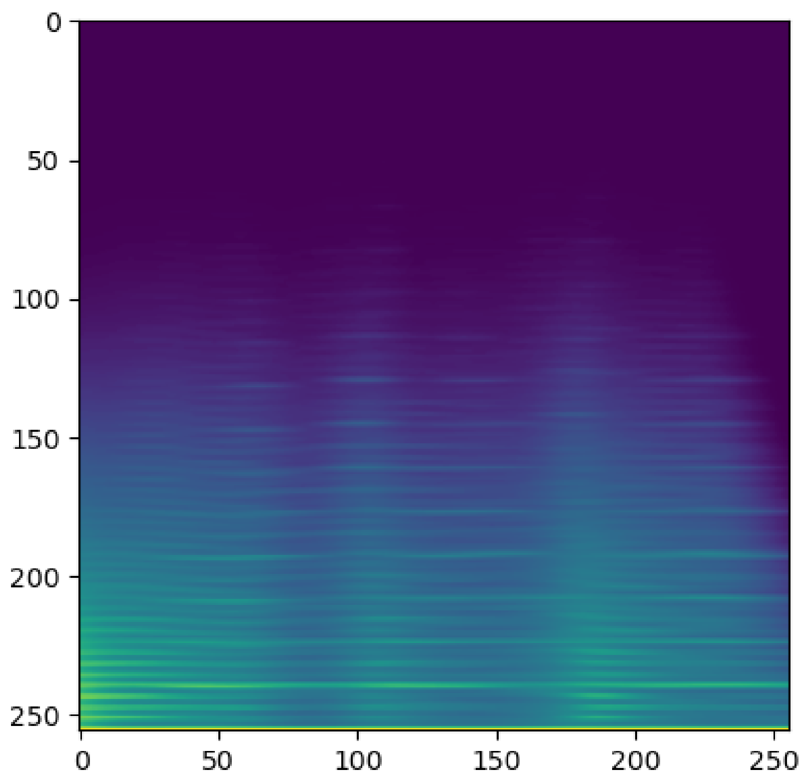
[5]:
```
import tensorflow as tf
import os
import numpy as np
from sklearn.model_selection import train_test_split

GENRES = ['blues', 'classical', 'country', 'disco', 'hiphop', 'jazz', 'metal',␣
↪'pop', 'reggae', 'rock']
FILE_PATH = os.path.join('Data', 'tempograms (30 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
25/25             123s 4s/step -
accuracy: 0.1001 - loss: 4.7744 - val_accuracy: 0.1000 - val_loss: 2.3094 -
learning_rate: 1.0000e-04
Epoch 2/20
25/25             114s 5s/step -
accuracy: 0.1530 - loss: 2.8332 - val_accuracy: 0.1000 - val_loss: 2.3190 -
learning_rate: 1.0000e-04
Epoch 3/20
25/25             147s 5s/step -
accuracy: 0.1295 - loss: 2.4556 - val_accuracy: 0.1000 - val_loss: 2.3364 -
learning_rate: 1.0000e-04
Epoch 4/20
25/25             151s 6s/step -
accuracy: 0.1656 - loss: 2.2950 - val_accuracy: 0.1000 - val_loss: 2.3498 -
learning_rate: 1.0000e-04
Epoch 5/20
25/25             223s 7s/step -
accuracy: 0.1495 - loss: 2.2987 - val_accuracy: 0.1000 - val_loss: 2.3534 -
learning_rate: 5.0000e-05
Epoch 6/20
25/25             182s 7s/step -
accuracy: 0.1519 - loss: 2.2809 - val_accuracy: 0.1000 - val_loss: 2.3668 -
learning_rate: 5.0000e-05
Epoch 7/20
25/25             187s 7s/step -
accuracy: 0.1758 - loss: 2.2086 - val_accuracy: 0.1000 - val_loss: 2.3855 -
learning_rate: 5.0000e-05
Epoch 8/20
25/25             187s 8s/step -
accuracy: 0.1607 - loss: 2.2146 - val_accuracy: 0.1000 - val_loss: 2.3908 -
learning_rate: 2.5000e-05
Epoch 9/20
25/25             203s 8s/step -
accuracy: 0.1890 - loss: 2.2048 - val_accuracy: 0.1000 - val_loss: 2.3936 -
learning_rate: 2.5000e-05
Epoch 10/20
25/25             186s 7s/step -
accuracy: 0.1805 - loss: 2.2220 - val_accuracy: 0.1000 - val_loss: 2.3950 -
learning_rate: 2.5000e-05
Epoch 11/20
25/25             190s 8s/step -
accuracy: 0.1981 - loss: 2.1543 - val_accuracy: 0.1050 - val_loss: 2.3945 -
learning_rate: 1.2500e-05
Epoch 12/20
```

```
25/25                 202s 8s/step -
accuracy: 0.1985 - loss: 2.1479 - val_accuracy: 0.1100 - val_loss: 2.3902 -
learning_rate: 1.2500e-05
Epoch 13/20
25/25                 206s 8s/step -
accuracy: 0.1788 - loss: 2.1575 - val_accuracy: 0.1200 - val_loss: 2.3780 -
learning_rate: 1.2500e-05
Epoch 14/20
25/25                 189s 8s/step -
accuracy: 0.1836 - loss: 2.1996 - val_accuracy: 0.1150 - val_loss: 2.3589 -
learning_rate: 6.2500e-06
Epoch 15/20
25/25                 200s 7s/step -
accuracy: 0.1917 - loss: 2.1710 - val_accuracy: 0.0750 - val_loss: 2.3432 -
learning_rate: 6.2500e-06
Epoch 16/20
25/25                 204s 8s/step -
accuracy: 0.2209 - loss: 2.1503 - val_accuracy: 0.0700 - val_loss: 2.3255 -
learning_rate: 6.2500e-06
Epoch 17/20
25/25                 192s 8s/step -
accuracy: 0.2176 - loss: 2.1260 - val_accuracy: 0.0900 - val_loss: 2.3026 -
learning_rate: 3.1250e-06
Epoch 18/20
25/25                 197s 8s/step -
accuracy: 0.2114 - loss: 2.1631 - val_accuracy: 0.1250 - val_loss: 2.2876 -
learning_rate: 3.1250e-06
Epoch 19/20
25/25                 203s 8s/step -
accuracy: 0.2408 - loss: 2.1193 - val_accuracy: 0.1200 - val_loss: 2.2749 -
learning_rate: 3.1250e-06
Epoch 20/20
25/25                 195s 8s/step -
accuracy: 0.2511 - loss: 2.1153 - val_accuracy: 0.1350 - val_loss: 2.2612 -
learning_rate: 3.1250e-06
```

[8]: <keras.src.callbacks.history.History at 0x7f205046c350>

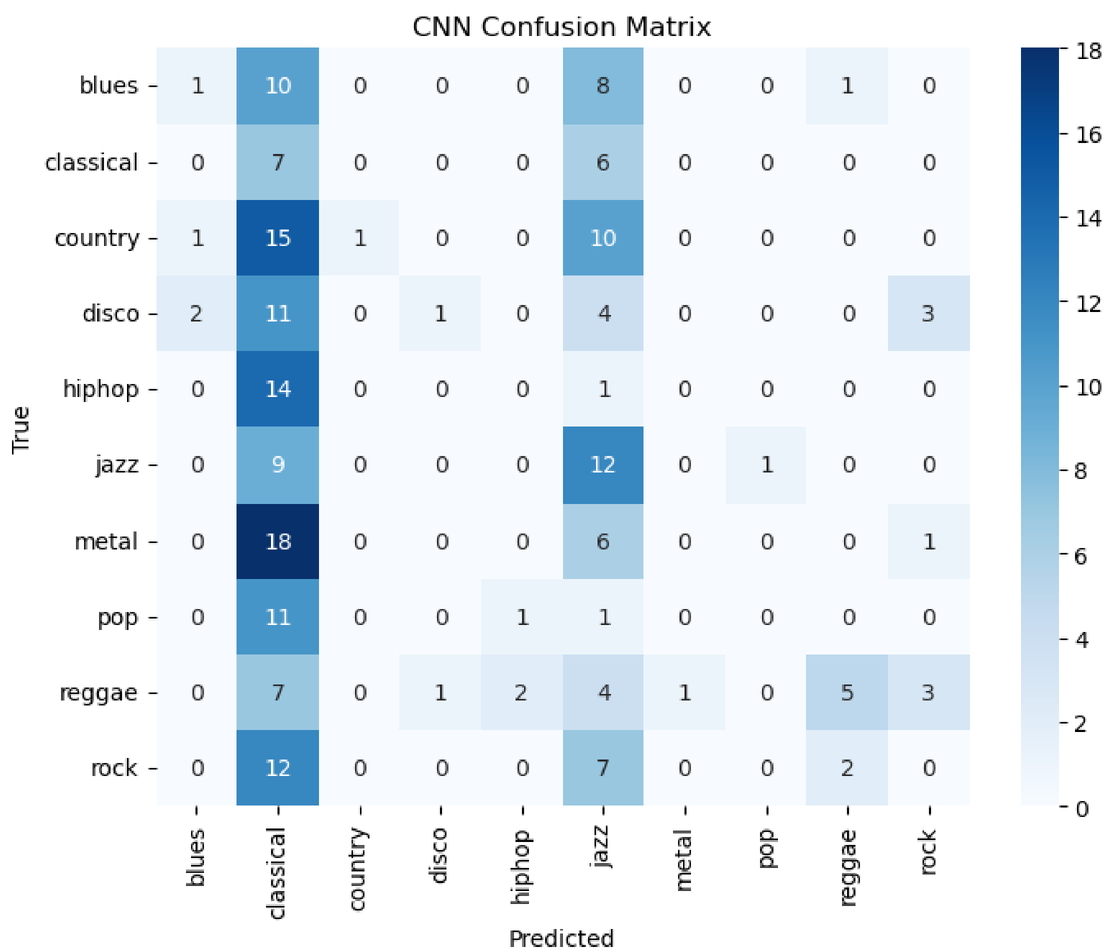
[9]:
```
evaluation = model.evaluate(X_test, y_test)
print(f"Test accuracy: {evaluation[1]:.3f}")
```

```
7/7             7s 952ms/step -
accuracy: 0.1310 - loss: 2.2568
Test accuracy: 0.135
```

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()
```

```
7/7          7s 895ms/step
```



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 (30 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
```

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

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

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

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

    Conv2D(256, (3, 3), activation='relu'),
    Normalization(),
    MaxPooling2D((2, 2)),

    Flatten(),

    Dense(512, activation='relu'),
    Dropout(0.5),

    Dense(256, activation='relu'),
    Dropout(0.5),

    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])

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

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

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

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

Going through classical

```
Going through metal
```

```
/opt/conda/lib/python3.12/site-
packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)
```

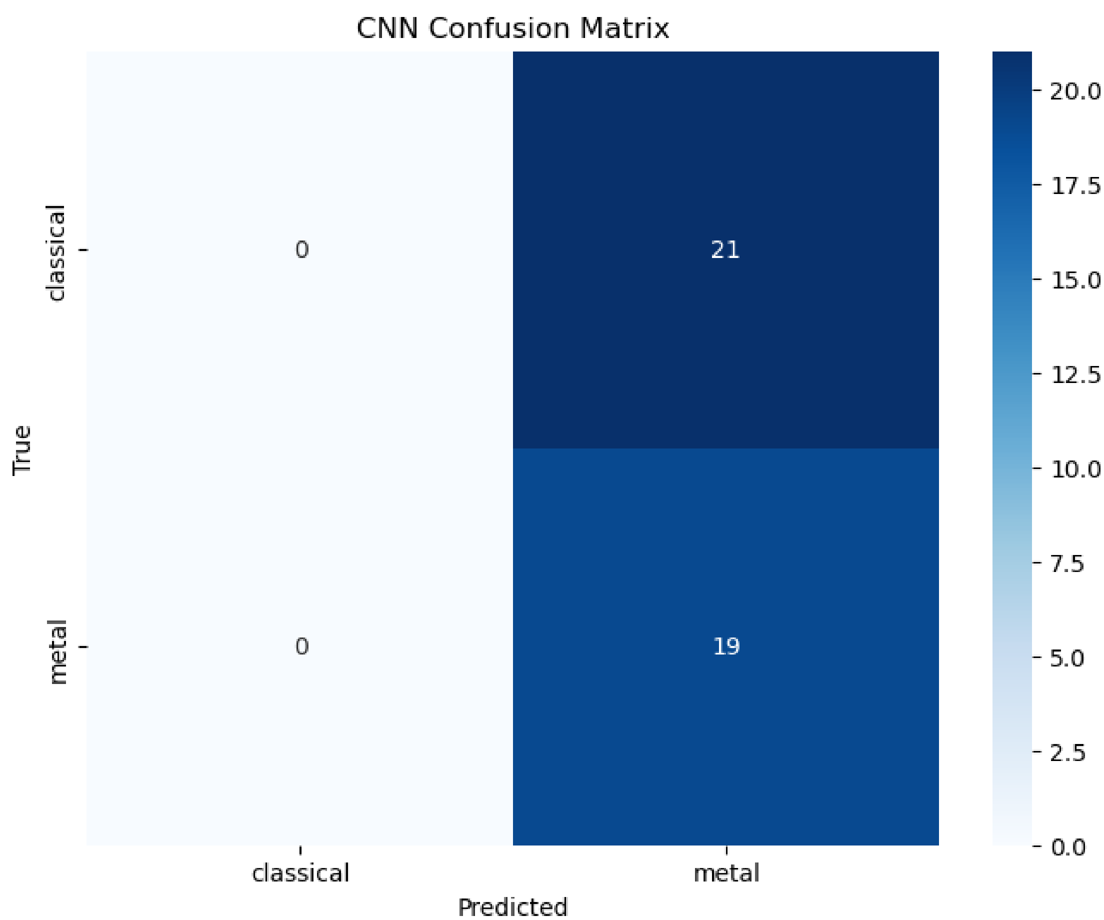
```
Epoch 1/20
5/5            38s 6s/step -
accuracy: 0.3451 - loss: 2.2043 - val_accuracy: 0.5250 - val_loss: 1.7436 -
learning_rate: 1.0000e-04
Epoch 2/20
5/5            32s 4s/step -
accuracy: 0.4463 - loss: 1.6111 - val_accuracy: 0.5250 - val_loss: 0.9217 -
learning_rate: 1.0000e-04
Epoch 3/20
5/5            27s 5s/step -
accuracy: 0.5500 - loss: 0.9538 - val_accuracy: 0.4750 - val_loss: 0.7285 -
learning_rate: 1.0000e-04
Epoch 4/20
5/5            21s 4s/step -
accuracy: 0.4806 - loss: 0.9698 - val_accuracy: 0.4750 - val_loss: 0.7707 -
learning_rate: 1.0000e-04
Epoch 5/20
5/5            27s 6s/step -
accuracy: 0.4419 - loss: 1.0288 - val_accuracy: 0.4750 - val_loss: 0.7600 -
learning_rate: 1.0000e-04
Epoch 6/20
5/5            38s 5s/step -
accuracy: 0.5516 - loss: 0.7950 - val_accuracy: 0.4750 - val_loss: 0.7771 -
learning_rate: 1.0000e-04
Epoch 7/20
5/5            26s 5s/step -
accuracy: 0.4775 - loss: 0.8757 - val_accuracy: 0.4750 - val_loss: 0.7772 -
learning_rate: 5.0000e-05
Epoch 8/20
5/5            45s 6s/step -
accuracy: 0.5210 - loss: 0.8222 - val_accuracy: 0.4750 - val_loss: 0.7645 -
learning_rate: 5.0000e-05
Epoch 9/20
5/5            34s 5s/step -
accuracy: 0.5135 - loss: 0.8598 - val_accuracy: 0.4750 - val_loss: 0.7522 -
learning_rate: 5.0000e-05
Epoch 10/20
5/5            42s 5s/step -
accuracy: 0.5135 - loss: 0.8438 - val_accuracy: 0.4750 - val_loss: 0.7454 -
learning_rate: 2.5000e-05
```

```
Epoch 11/20
5/5            23s 5s/step -
accuracy: 0.5299 - loss: 0.9208 - val_accuracy: 0.4750 - val_loss: 0.7442 -
learning_rate: 2.5000e-05
Epoch 12/20
5/5            22s 4s/step -
accuracy: 0.4938 - loss: 0.8210 - val_accuracy: 0.4750 - val_loss: 0.7457 -
learning_rate: 2.5000e-05
Epoch 13/20
5/5            47s 6s/step -
accuracy: 0.4610 - loss: 0.8990 - val_accuracy: 0.4750 - val_loss: 0.7452 -
learning_rate: 1.2500e-05
Epoch 14/20
5/5            40s 6s/step -
accuracy: 0.5747 - loss: 0.7961 - val_accuracy: 0.4750 - val_loss: 0.7452 -
learning_rate: 1.2500e-05
Epoch 15/20
5/5            28s 6s/step -
accuracy: 0.4355 - loss: 0.9441 - val_accuracy: 0.4750 - val_loss: 0.7439 -
learning_rate: 1.2500e-05
Epoch 16/20
5/5            28s 6s/step -
accuracy: 0.4551 - loss: 0.8932 - val_accuracy: 0.4750 - val_loss: 0.7436 -
learning_rate: 6.2500e-06
Epoch 17/20
5/5            28s 6s/step -
accuracy: 0.5710 - loss: 0.7890 - val_accuracy: 0.4750 - val_loss: 0.7436 -
learning_rate: 6.2500e-06
Epoch 18/20
5/5            26s 5s/step -
accuracy: 0.5205 - loss: 0.8446 - val_accuracy: 0.4750 - val_loss: 0.7431 -
learning_rate: 6.2500e-06
Epoch 19/20
5/5            27s 5s/step -
accuracy: 0.5177 - loss: 0.8032 - val_accuracy: 0.4750 - val_loss: 0.7432 -
learning_rate: 3.1250e-06
Epoch 20/20
5/5            24s 5s/step -
accuracy: 0.4895 - loss: 0.7941 - val_accuracy: 0.4750 - val_loss: 0.7429 -
learning_rate: 3.1250e-06
2/2            2s 398ms/step -
accuracy: 0.4625 - loss: 0.7456
Test accuracy: 0.475
```

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()
```

```
2/2               2s 635ms/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 (30 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
```

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

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

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

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

    Conv2D(256, (3, 3), activation='relu'),
    Normalization(),
    MaxPooling2D((2, 2)),

    Flatten(),

    Dense(512, activation='relu'),
    Dropout(0.5),

    Dense(256, activation='relu'),
    Dropout(0.5),

    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])

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

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

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

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

```
Going through disco
```

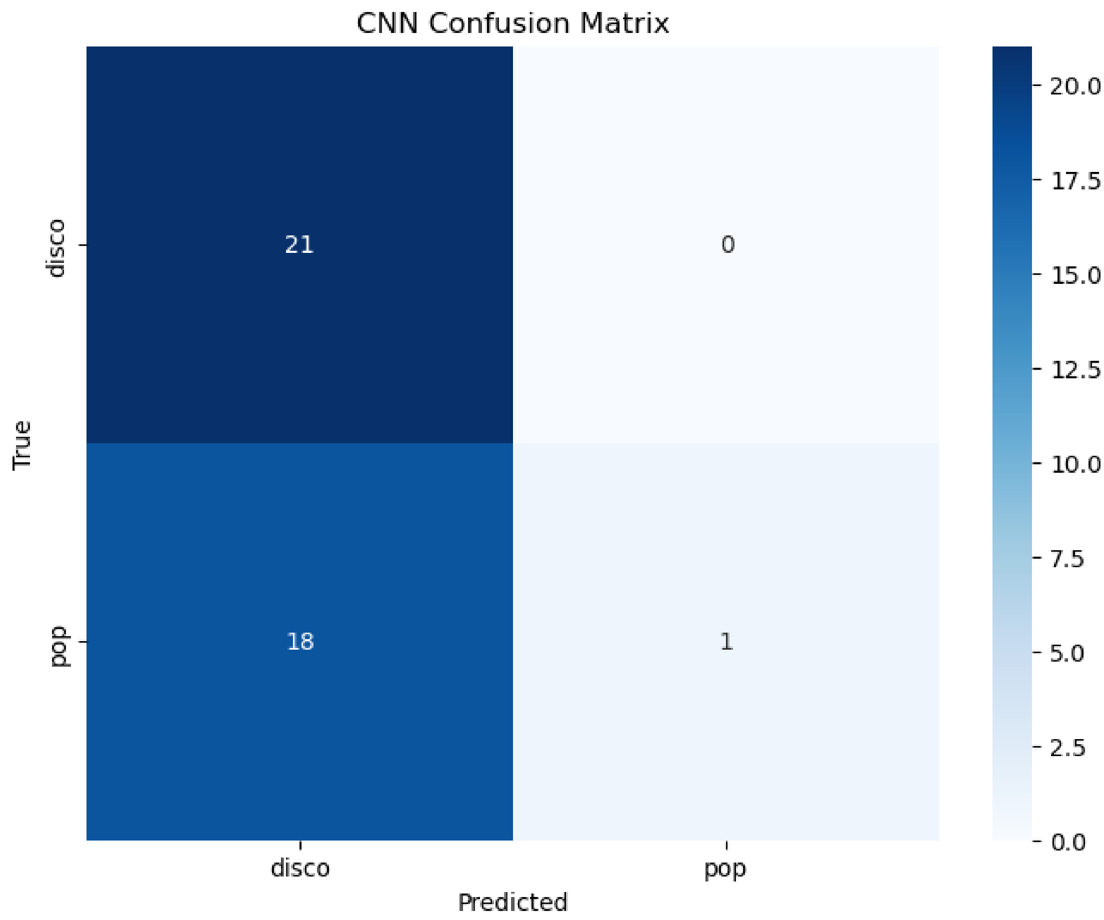
```
Going through pop

/opt/conda/lib/python3.12/site-
packages/keras/src/layers/convolutional/base_conv.py:107: UserWarning: Do not
pass an `input_shape`/`input_dim` argument to a layer. When using Sequential
models, prefer using an `Input(shape)` object as the first layer in the model
instead.
  super().__init__(activity_regularizer=activity_regularizer, **kwargs)

Epoch 1/20
5/5            37s 5s/step -
accuracy: 0.3098 - loss: 2.1813 - val_accuracy: 0.4750 - val_loss: 1.6631 -
learning_rate: 1.0000e-04
Epoch 2/20
5/5            25s 5s/step -
accuracy: 0.4637 - loss: 1.5099 - val_accuracy: 0.4750 - val_loss: 0.9119 -
learning_rate: 1.0000e-04
Epoch 3/20
5/5            25s 5s/step -
accuracy: 0.5122 - loss: 0.9495 - val_accuracy: 0.4750 - val_loss: 0.7157 -
learning_rate: 1.0000e-04
Epoch 4/20
5/5            37s 4s/step -
accuracy: 0.4688 - loss: 0.9638 - val_accuracy: 0.5250 - val_loss: 0.7386 -
learning_rate: 1.0000e-04
Epoch 5/20
5/5            42s 5s/step -
accuracy: 0.5609 - loss: 0.8824 - val_accuracy: 0.5250 - val_loss: 0.7067 -
learning_rate: 1.0000e-04
Epoch 6/20
5/5            23s 5s/step -
accuracy: 0.5774 - loss: 0.8099 - val_accuracy: 0.4750 - val_loss: 0.7248 -
learning_rate: 1.0000e-04
Epoch 7/20
5/5            24s 5s/step -
accuracy: 0.5084 - loss: 0.8160 - val_accuracy: 0.4750 - val_loss: 0.7539 -
learning_rate: 1.0000e-04
Epoch 8/20
5/5            23s 5s/step -
accuracy: 0.4834 - loss: 0.8989 - val_accuracy: 0.4750 - val_loss: 0.7583 -
learning_rate: 1.0000e-04
Epoch 9/20
5/5            23s 5s/step -
accuracy: 0.5031 - loss: 0.8271 - val_accuracy: 0.4750 - val_loss: 0.7454 -
learning_rate: 5.0000e-05
Epoch 10/20
5/5            20s 4s/step -
accuracy: 0.5009 - loss: 0.8087 - val_accuracy: 0.4750 - val_loss: 0.7282 -
learning_rate: 5.0000e-05
```

```
Epoch 11/20
5/5            24s 5s/step -
accuracy: 0.4835 - loss: 0.8420 - val_accuracy: 0.4750 - val_loss: 0.7196 -
learning_rate: 5.0000e-05
Epoch 12/20
5/5            26s 5s/step -
accuracy: 0.4576 - loss: 0.8637 - val_accuracy: 0.5500 - val_loss: 0.7177 -
learning_rate: 2.5000e-05
Epoch 13/20
5/5            23s 5s/step -
accuracy: 0.5949 - loss: 0.7240 - val_accuracy: 0.4750 - val_loss: 0.7143 -
learning_rate: 2.5000e-05
Epoch 14/20
5/5            43s 5s/step -
accuracy: 0.4724 - loss: 0.8047 - val_accuracy: 0.5500 - val_loss: 0.7115 -
learning_rate: 2.5000e-05
Epoch 15/20
5/5            39s 5s/step -
accuracy: 0.4415 - loss: 0.8288 - val_accuracy: 0.5250 - val_loss: 0.7110 -
learning_rate: 1.2500e-05
Epoch 16/20
5/5            24s 5s/step -
accuracy: 0.4888 - loss: 0.8182 - val_accuracy: 0.5250 - val_loss: 0.7104 -
learning_rate: 1.2500e-05
Epoch 17/20
5/5            41s 5s/step -
accuracy: 0.4532 - loss: 0.8122 - val_accuracy: 0.5250 - val_loss: 0.7101 -
learning_rate: 1.2500e-05
Epoch 18/20
5/5            23s 5s/step -
accuracy: 0.4664 - loss: 0.8441 - val_accuracy: 0.5250 - val_loss: 0.7103 -
learning_rate: 6.2500e-06
Epoch 19/20
5/5            21s 4s/step -
accuracy: 0.4898 - loss: 0.8348 - val_accuracy: 0.5250 - val_loss: 0.7105 -
learning_rate: 6.2500e-06
Epoch 20/20
5/5            42s 5s/step -
accuracy: 0.5993 - loss: 0.7472 - val_accuracy: 0.5500 - val_loss: 0.7102 -
learning_rate: 6.2500e-06
2/2            1s 225ms/step -
accuracy: 0.5646 - loss: 0.7095
Test accuracy: 0.550
```

4.4 12 - Confusion Matrix Hard (disco and pop)

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

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

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

```
WARNING:tensorflow:5 out of the last 10 calls to <function
TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at
0x7f1fd436c7c0> triggered tf.function retracing. Tracing is expensive and the
excessive number of tracings could be due to (1) creating @tf.function
repeatedly in a loop, (2) passing tensors with different shapes, (3) passing
Python objects instead of tensors. For (1), please define your @tf.function
outside of the loop. For (2), @tf.function has reduce_retracing=True option that
can avoid unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
1/2           1s
1s/stepWARNING:tensorflow:6 out of the last 11 calls to <function
TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at
0x7f1fd436c7c0> triggered tf.function retracing. Tracing is expensive and the
excessive number of tracings could be due to (1) creating @tf.function
repeatedly in a loop, (2) passing tensors with different shapes, (3) passing
Python objects instead of tensors. For (1), please define your @tf.function
outside of the loop. For (2), @tf.function has reduce_retracing=True option that
can avoid unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
2/2           2s 643ms/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 (30 secs)')
X = []
y = []

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

```python
# 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)),
```

```python
    Conv2D(256, (3, 3), activation='relu'),
    Normalization(),
    MaxPooling2D((2, 2)),

    Flatten(),

    Dense(512, activation='relu'),
    Dropout(0.5),

    Dense(256, activation='relu'),
    Dropout(0.5),

    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])

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

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

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

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

```
['rock', 'disco', 'reggae', 'metal', 'pop']
Going through rock
Going through disco
Going through reggae
Going through metal
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
13/13              63s 4s/step -
accuracy: 0.1030 - loss: 2.2777 - val_accuracy: 0.1000 - val_loss: 2.1451 -
learning_rate: 1.0000e-04
```

```
Epoch 2/20
13/13            83s 4s/step -
accuracy: 0.2136 - loss: 2.0416 - val_accuracy: 0.1000 - val_loss: 1.9178 -
learning_rate: 1.0000e-04
Epoch 3/20
13/13            81s 4s/step -
accuracy: 0.2196 - loss: 1.9410 - val_accuracy: 0.1400 - val_loss: 1.8166 -
learning_rate: 1.0000e-04
Epoch 4/20
13/13            80s 4s/step -
accuracy: 0.2595 - loss: 1.8321 - val_accuracy: 0.1400 - val_loss: 1.7923 -
learning_rate: 1.0000e-04
Epoch 5/20
13/13            53s 4s/step -
accuracy: 0.2032 - loss: 1.8528 - val_accuracy: 0.1400 - val_loss: 1.7592 -
learning_rate: 1.0000e-04
Epoch 6/20
13/13            51s 4s/step -
accuracy: 0.1810 - loss: 1.8780 - val_accuracy: 0.1400 - val_loss: 1.7554 -
learning_rate: 1.0000e-04
Epoch 7/20
13/13            52s 4s/step -
accuracy: 0.2131 - loss: 1.8251 - val_accuracy: 0.1400 - val_loss: 1.7576 -
learning_rate: 1.0000e-04
Epoch 8/20
13/13            81s 4s/step -
accuracy: 0.2358 - loss: 1.7375 - val_accuracy: 0.1400 - val_loss: 1.7339 -
learning_rate: 1.0000e-04
Epoch 9/20
13/13            47s 4s/step -
accuracy: 0.2150 - loss: 1.7343 - val_accuracy: 0.1400 - val_loss: 1.7638 -
learning_rate: 1.0000e-04
Epoch 10/20
13/13            49s 4s/step -
accuracy: 0.1877 - loss: 1.7438 - val_accuracy: 0.1400 - val_loss: 1.7687 -
learning_rate: 1.0000e-04
Epoch 11/20
13/13            48s 4s/step -
accuracy: 0.2219 - loss: 1.7560 - val_accuracy: 0.1400 - val_loss: 1.6918 -
learning_rate: 1.0000e-04
Epoch 12/20
13/13            79s 3s/step -
accuracy: 0.1943 - loss: 1.7281 - val_accuracy: 0.1400 - val_loss: 1.6797 -
learning_rate: 1.0000e-04
Epoch 13/20
13/13            47s 4s/step -
accuracy: 0.2269 - loss: 1.6970 - val_accuracy: 0.1600 - val_loss: 1.6660 -
learning_rate: 1.0000e-04
```

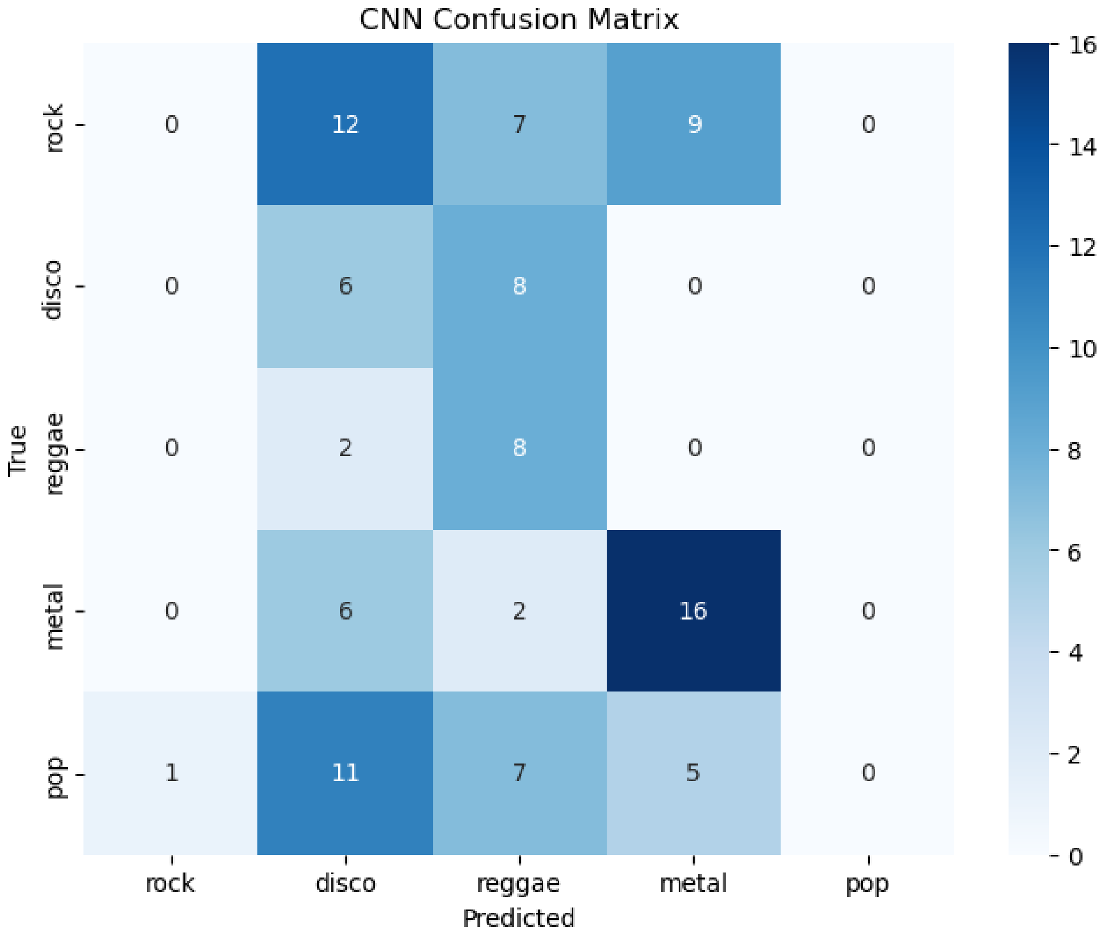
```
Epoch 14/20
13/13            47s 4s/step -
accuracy: 0.2614 - loss: 1.6811 - val_accuracy: 0.3000 - val_loss: 1.6239 -
learning_rate: 1.0000e-04
Epoch 15/20
13/13            47s 4s/step -
accuracy: 0.2378 - loss: 1.6758 - val_accuracy: 0.2900 - val_loss: 1.6162 -
learning_rate: 1.0000e-04
Epoch 16/20
13/13            36s 3s/step -
accuracy: 0.2947 - loss: 1.5697 - val_accuracy: 0.3000 - val_loss: 1.5655 -
learning_rate: 1.0000e-04
Epoch 17/20
13/13            50s 4s/step -
accuracy: 0.3339 - loss: 1.5470 - val_accuracy: 0.3100 - val_loss: 1.5617 -
learning_rate: 1.0000e-04
Epoch 18/20
13/13            77s 3s/step -
accuracy: 0.2929 - loss: 1.5705 - val_accuracy: 0.3200 - val_loss: 1.5785 -
learning_rate: 1.0000e-04
Epoch 19/20
13/13            41s 3s/step -
accuracy: 0.3028 - loss: 1.5406 - val_accuracy: 0.3100 - val_loss: 1.5470 -
learning_rate: 1.0000e-04
Epoch 20/20
13/13            93s 4s/step -
accuracy: 0.3113 - loss: 1.5715 - val_accuracy: 0.3000 - val_loss: 1.6595 -
learning_rate: 1.0000e-04
4/4           3s 601ms/step -
accuracy: 0.3075 - loss: 1.6741
Test accuracy: 0.300
```

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()
```

```
4/4                3s 620ms/step
```




```
[ ]:
```