Onset Images-Only CNN

March 20, 2025

1 CNN for Onset Images (30 secs)

1.1 1 - All the imports

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

```
2025-03-20 08:41:47.709046: 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)

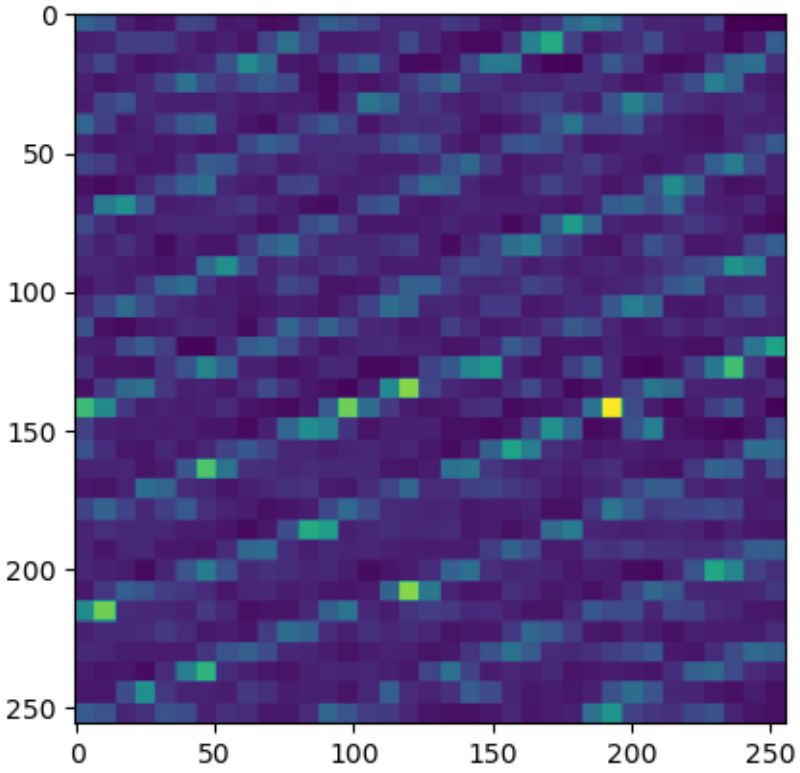
3 4 - Load the images

[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', 'onset_images_full')
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]:
```
# Show image as a sanity check
import matplotlib.pyplot as plt
plt.imshow(X_train[22].reshape(256, 256))
plt.show()
```



[7]:
```
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau

model.compile(optimizer=Adam(learning_rate=0.0001),␣
 ↪loss='sparse_categorical_crossentropy', metrics=['accuracy'])

reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3,␣
 ↪min_lr=1e-6)
```

[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             100s 4s/step -
accuracy: 0.1060 - loss: 4.8772 - val_accuracy: 0.1050 - val_loss: 3.0589 -
learning_rate: 1.0000e-04
Epoch 2/20
25/25             127s 5s/step -
accuracy: 0.1501 - loss: 3.1744 - val_accuracy: 0.1050 - val_loss: 4.4529 -
learning_rate: 1.0000e-04
Epoch 3/20
25/25             124s 5s/step -
accuracy: 0.1781 - loss: 2.3713 - val_accuracy: 0.0650 - val_loss: 5.3940 -
learning_rate: 1.0000e-04
Epoch 4/20
25/25             137s 5s/step -
accuracy: 0.1431 - loss: 2.3143 - val_accuracy: 0.0650 - val_loss: 8.5445 -
learning_rate: 1.0000e-04
Epoch 5/20
25/25             173s 7s/step -
accuracy: 0.1659 - loss: 2.2034 - val_accuracy: 0.0650 - val_loss: 10.1188 -
learning_rate: 5.0000e-05
Epoch 6/20
25/25             224s 8s/step -
accuracy: 0.2263 - loss: 2.1325 - val_accuracy: 0.0650 - val_loss: 12.7187 -
learning_rate: 5.0000e-05
Epoch 7/20
25/25             204s 8s/step -
accuracy: 0.2323 - loss: 2.1223 - val_accuracy: 0.0650 - val_loss: 14.2778 -
learning_rate: 5.0000e-05
Epoch 8/20
25/25             200s 8s/step -
accuracy: 0.2241 - loss: 2.1054 - val_accuracy: 0.0650 - val_loss: 15.0378 -
learning_rate: 2.5000e-05
Epoch 9/20
25/25             192s 8s/step -
accuracy: 0.2499 - loss: 2.0418 - val_accuracy: 0.0650 - val_loss: 16.2305 -
learning_rate: 2.5000e-05
Epoch 10/20
25/25             190s 8s/step -
accuracy: 0.2635 - loss: 2.0668 - val_accuracy: 0.0650 - val_loss: 17.5233 -
learning_rate: 2.5000e-05
Epoch 11/20
25/25             201s 8s/step -
accuracy: 0.2807 - loss: 1.9793 - val_accuracy: 0.0650 - val_loss: 18.1739 -
learning_rate: 1.2500e-05
Epoch 12/20
```

```
25/25             210s 8s/step -
accuracy: 0.2948 - loss: 1.9576 - val_accuracy: 0.0650 - val_loss: 18.6447 -
learning_rate: 1.2500e-05
Epoch 13/20
25/25             188s 7s/step -
accuracy: 0.3079 - loss: 1.9234 - val_accuracy: 0.0650 - val_loss: 19.0384 -
learning_rate: 1.2500e-05
Epoch 14/20
25/25             191s 8s/step -
accuracy: 0.2976 - loss: 1.9706 - val_accuracy: 0.0650 - val_loss: 18.8497 -
learning_rate: 6.2500e-06
Epoch 15/20
25/25             196s 8s/step -
accuracy: 0.3384 - loss: 1.8957 - val_accuracy: 0.0650 - val_loss: 18.4785 -
learning_rate: 6.2500e-06
Epoch 16/20
25/25             195s 8s/step -
accuracy: 0.3276 - loss: 1.9050 - val_accuracy: 0.0650 - val_loss: 17.5852 -
learning_rate: 6.2500e-06
Epoch 17/20
25/25             194s 8s/step -
accuracy: 0.3297 - loss: 1.8839 - val_accuracy: 0.0650 - val_loss: 16.0211 -
learning_rate: 3.1250e-06
Epoch 18/20
25/25             203s 8s/step -
accuracy: 0.3117 - loss: 1.8696 - val_accuracy: 0.0650 - val_loss: 14.1039 -
learning_rate: 3.1250e-06
Epoch 19/20
25/25             207s 8s/step -
accuracy: 0.3496 - loss: 1.8795 - val_accuracy: 0.0650 - val_loss: 11.7919 -
learning_rate: 3.1250e-06
Epoch 20/20
25/25             199s 8s/step -
accuracy: 0.2993 - loss: 1.9069 - val_accuracy: 0.0650 - val_loss: 9.6080 -
learning_rate: 1.5625e-06
```

[8]:
```
<keras.src.callbacks.history.History at 0x7f28259c56a0>
```

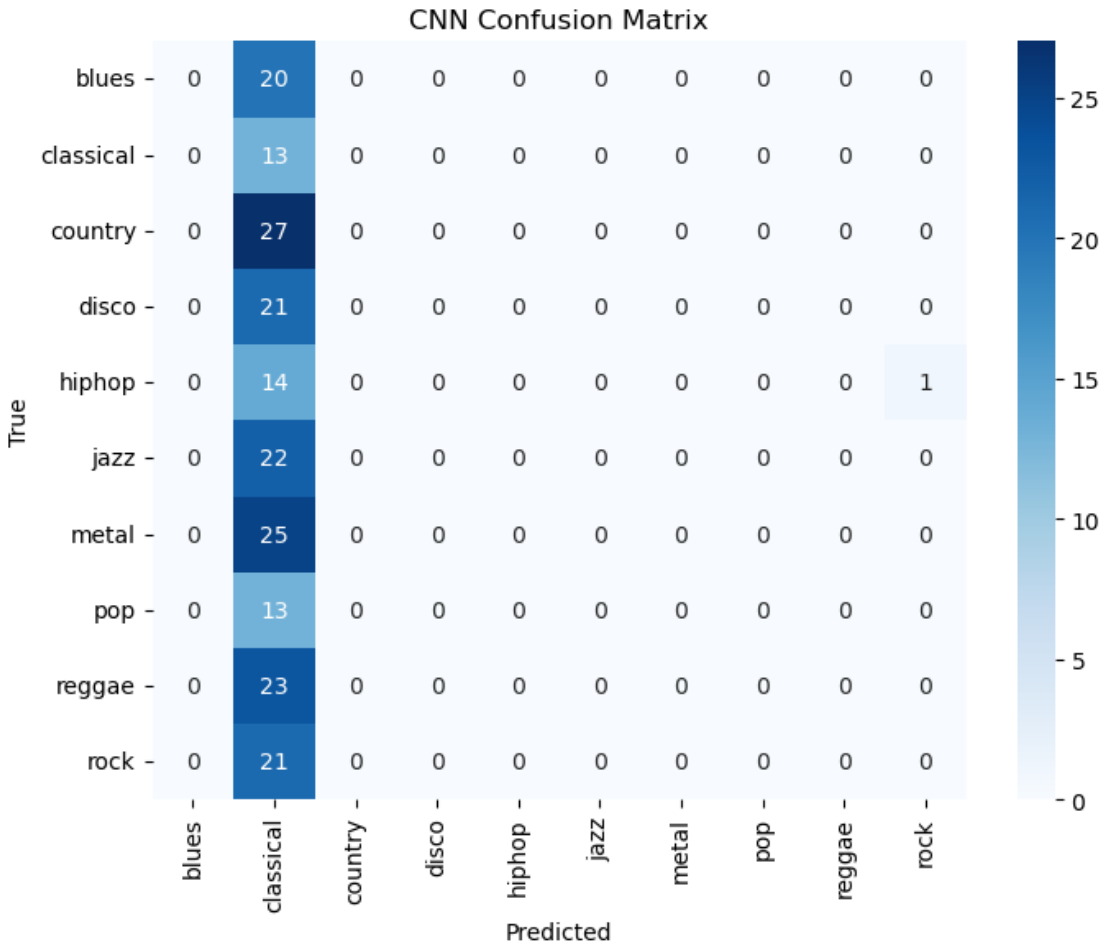
[9]:
```
evaluation = model.evaluate(X_test, y_test)
print(f"Test accuracy: {evaluation[1]:.3f}")
```

```
7/7               7s 884ms/step -
accuracy: 0.0729 - loss: 9.4844
Test accuracy: 0.065
```

[10]:
```
import seaborn as sns
# from sklearn.metrics import confusion
import numpy as NP
```

```
from sklearn.metrics import confusion_matrix

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

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

```
7/7                 8s 1s/step
```



3.1 9 - Limited Genres Easy (metal and classical)

[11]:
```
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', 'onset_images_full')
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
5/5            40s 6s/step -
accuracy: 0.2817 - loss: 2.2470 - val_accuracy: 0.4750 - val_loss: 1.9061 -
learning_rate: 1.0000e-04
Epoch 2/20
5/5            36s 4s/step -
accuracy: 0.4025 - loss: 1.8338 - val_accuracy: 0.4750 - val_loss: 1.0949 -
learning_rate: 1.0000e-04
Epoch 3/20
5/5            26s 6s/step -
accuracy: 0.5014 - loss: 1.2554 - val_accuracy: 0.5250 - val_loss: 0.7320 -
learning_rate: 1.0000e-04
Epoch 4/20
5/5            26s 5s/step -
accuracy: 0.4840 - loss: 0.9605 - val_accuracy: 0.5250 - val_loss: 0.8294 -
learning_rate: 1.0000e-04
Epoch 5/20
5/5            39s 5s/step -
accuracy: 0.5095 - loss: 1.1973 - val_accuracy: 0.5250 - val_loss: 0.7341 -
learning_rate: 1.0000e-04
Epoch 6/20
5/5            44s 5s/step -
accuracy: 0.5048 - loss: 1.0509 - val_accuracy: 0.4750 - val_loss: 0.7363 -
learning_rate: 1.0000e-04
Epoch 7/20
5/5            26s 5s/step -
accuracy: 0.4998 - loss: 0.9083 - val_accuracy: 0.4750 - val_loss: 0.7640 -
learning_rate: 5.0000e-05
Epoch 8/20
5/5            41s 5s/step -
accuracy: 0.5699 - loss: 0.9621 - val_accuracy: 0.4750 - val_loss: 0.7719 -
learning_rate: 5.0000e-05
Epoch 9/20
5/5            23s 5s/step -
accuracy: 0.4553 - loss: 0.9701 - val_accuracy: 0.4750 - val_loss: 0.7579 -
learning_rate: 5.0000e-05
Epoch 10/20
5/5            41s 4s/step -
accuracy: 0.4686 - loss: 0.9588 - val_accuracy: 0.4750 - val_loss: 0.7500 -
learning_rate: 2.5000e-05
Epoch 11/20
5/5            24s 5s/step -
```

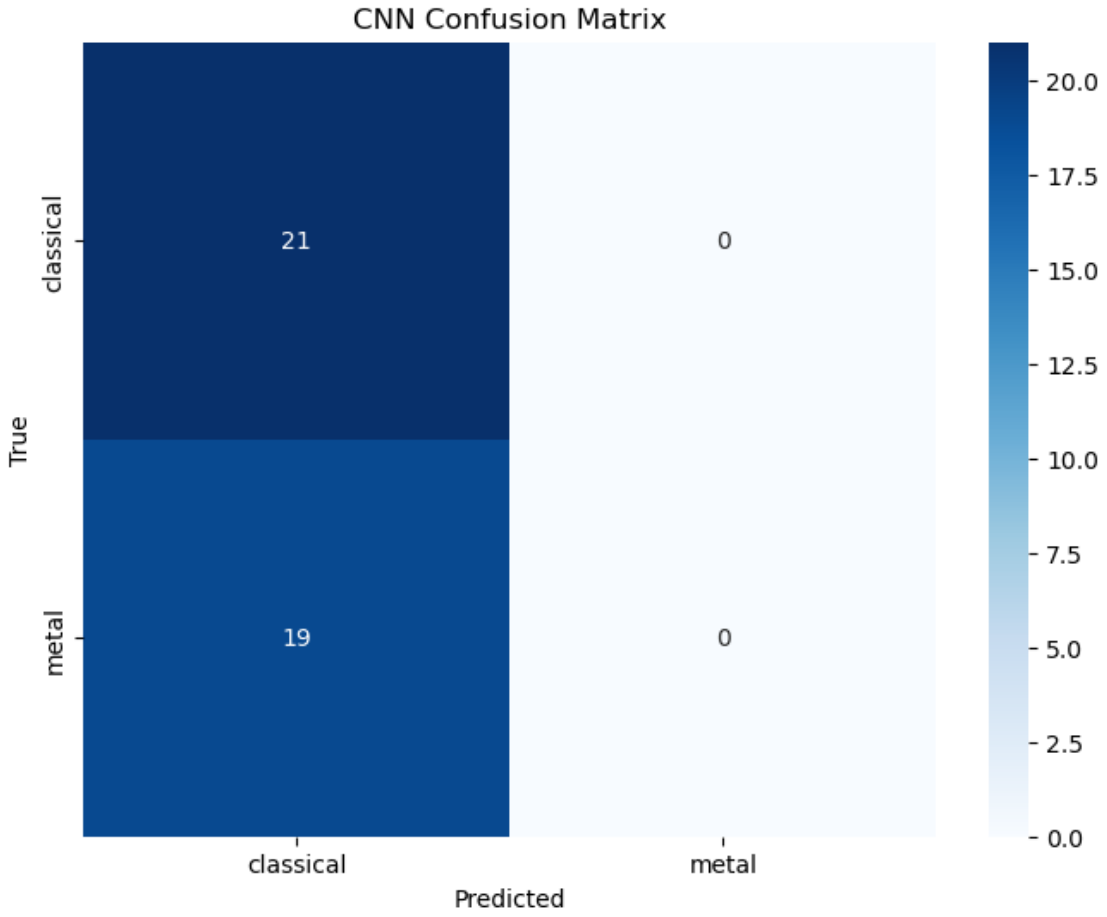
```
accuracy: 0.4386 - loss: 0.9296 - val_accuracy: 0.4750 - val_loss: 0.7423 -
learning_rate: 2.5000e-05
Epoch 12/20
5/5             43s 5s/step -
accuracy: 0.4989 - loss: 0.8281 - val_accuracy: 0.5250 - val_loss: 0.7335 -
learning_rate: 2.5000e-05
Epoch 13/20
5/5             37s 4s/step -
accuracy: 0.4707 - loss: 0.8849 - val_accuracy: 0.5250 - val_loss: 0.7308 -
learning_rate: 1.2500e-05
Epoch 14/20
5/5             46s 5s/step -
accuracy: 0.4557 - loss: 0.9188 - val_accuracy: 0.5250 - val_loss: 0.7292 -
learning_rate: 1.2500e-05
Epoch 15/20
5/5             41s 5s/step -
accuracy: 0.4659 - loss: 0.9689 - val_accuracy: 0.5250 - val_loss: 0.7296 -
learning_rate: 1.2500e-05
Epoch 16/20
5/5             26s 5s/step -
accuracy: 0.5461 - loss: 0.8619 - val_accuracy: 0.5250 - val_loss: 0.7304 -
learning_rate: 1.2500e-05
Epoch 17/20
5/5             41s 5s/step -
accuracy: 0.5532 - loss: 0.8227 - val_accuracy: 0.5250 - val_loss: 0.7316 -
learning_rate: 1.2500e-05
Epoch 18/20
5/5             40s 5s/step -
accuracy: 0.4614 - loss: 0.9113 - val_accuracy: 0.5250 - val_loss: 0.7317 -
learning_rate: 6.2500e-06
Epoch 19/20
5/5             40s 5s/step -
accuracy: 0.4641 - loss: 0.9260 - val_accuracy: 0.5250 - val_loss: 0.7314 -
learning_rate: 6.2500e-06
Epoch 20/20
5/5             41s 5s/step -
accuracy: 0.4490 - loss: 0.8964 - val_accuracy: 0.5250 - val_loss: 0.7314 -
learning_rate: 6.2500e-06
2/2             1s 358ms/step -
accuracy: 0.5375 - loss: 0.7309
Test accuracy: 0.525
```

3.2 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 973ms/step
```



3.3 11 - Limited genres Hard (disco and pop)

[13]:
```
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', 'onset_images_full')
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
5/5             40s 5s/step -
accuracy: 0.1058 - loss: 2.2774 - val_accuracy: 0.5250 - val_loss: 2.0813 -
learning_rate: 1.0000e-04
Epoch 2/20
5/5             24s 5s/step -
accuracy: 0.3490 - loss: 2.0234 - val_accuracy: 0.5250 - val_loss: 1.5609 -
learning_rate: 1.0000e-04
Epoch 3/20
5/5             22s 5s/step -
accuracy: 0.4325 - loss: 1.5591 - val_accuracy: 0.5250 - val_loss: 0.9463 -
learning_rate: 1.0000e-04
Epoch 4/20
5/5             21s 4s/step -
accuracy: 0.4491 - loss: 1.2920 - val_accuracy: 0.4750 - val_loss: 0.7717 -
learning_rate: 1.0000e-04
Epoch 5/20
5/5             19s 4s/step -
accuracy: 0.4501 - loss: 1.2005 - val_accuracy: 0.4750 - val_loss: 0.7664 -
learning_rate: 1.0000e-04
Epoch 6/20
5/5             23s 4s/step -
accuracy: 0.5547 - loss: 1.0866 - val_accuracy: 0.4750 - val_loss: 0.8325 -
learning_rate: 1.0000e-04
Epoch 7/20
5/5             44s 5s/step -
accuracy: 0.5269 - loss: 1.0640 - val_accuracy: 0.4750 - val_loss: 0.8341 -
learning_rate: 1.0000e-04
Epoch 8/20
5/5             39s 5s/step -
accuracy: 0.5547 - loss: 0.9634 - val_accuracy: 0.4750 - val_loss: 0.8078 -
learning_rate: 1.0000e-04
Epoch 9/20
5/5             23s 5s/step -
accuracy: 0.5927 - loss: 0.9369 - val_accuracy: 0.4750 - val_loss: 0.7874 -
learning_rate: 5.0000e-05
Epoch 10/20
5/5             41s 5s/step -
accuracy: 0.4474 - loss: 0.9898 - val_accuracy: 0.4750 - val_loss: 0.7688 -
learning_rate: 5.0000e-05
Epoch 11/20
5/5             23s 5s/step -
```

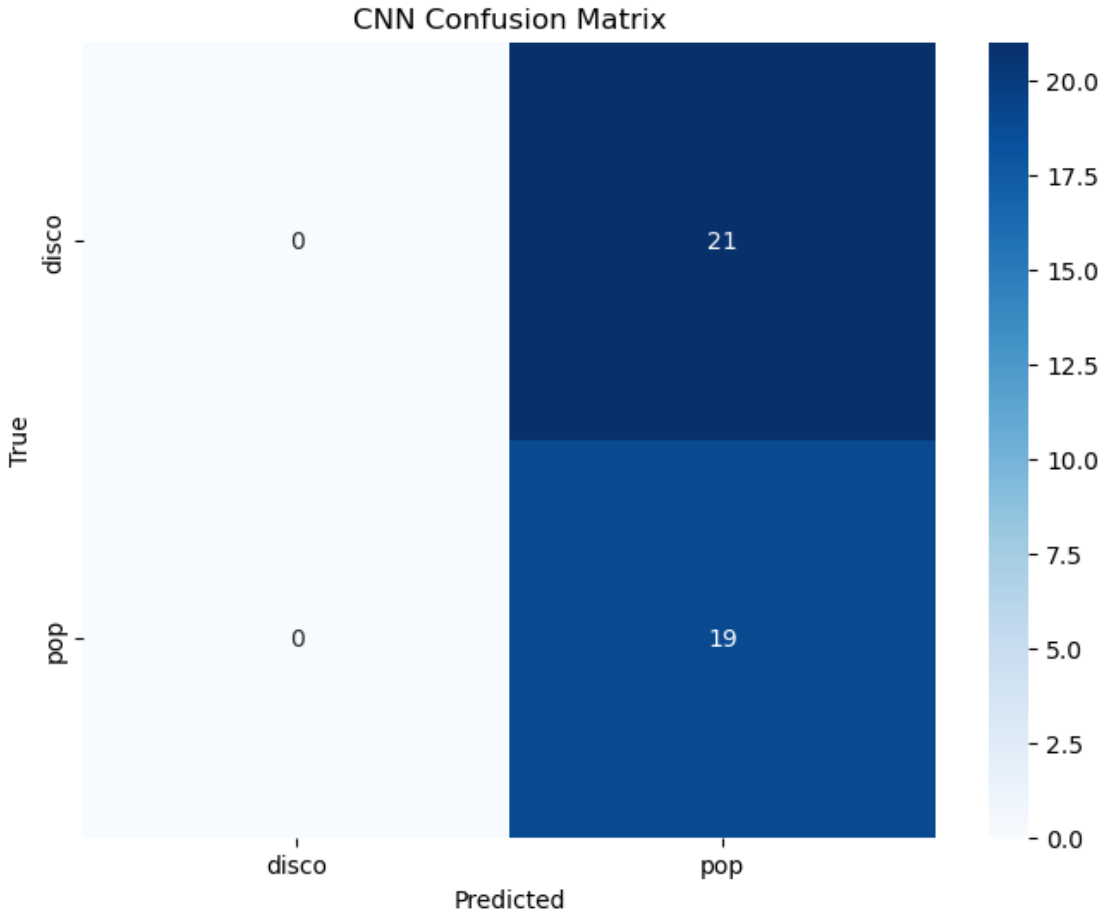
```
accuracy: 0.4774 - loss: 0.9142 - val_accuracy: 0.4750 - val_loss: 0.7616 -
learning_rate: 5.0000e-05
Epoch 12/20
5/5             21s 4s/step -
accuracy: 0.5635 - loss: 0.8698 - val_accuracy: 0.4750 - val_loss: 0.7603 -
learning_rate: 5.0000e-05
Epoch 13/20
5/5             20s 4s/step -
accuracy: 0.5196 - loss: 0.9525 - val_accuracy: 0.4750 - val_loss: 0.7640 -
learning_rate: 5.0000e-05
Epoch 14/20
5/5             22s 4s/step -
accuracy: 0.5024 - loss: 0.8983 - val_accuracy: 0.4750 - val_loss: 0.7730 -
learning_rate: 5.0000e-05
Epoch 15/20
5/5             43s 5s/step -
accuracy: 0.4977 - loss: 0.9399 - val_accuracy: 0.4750 - val_loss: 0.7753 -
learning_rate: 5.0000e-05
Epoch 16/20
5/5             22s 4s/step -
accuracy: 0.5037 - loss: 0.9242 - val_accuracy: 0.4750 - val_loss: 0.7777 -
learning_rate: 2.5000e-05
Epoch 17/20
5/5             22s 5s/step -
accuracy: 0.5283 - loss: 0.8614 - val_accuracy: 0.4750 - val_loss: 0.7792 -
learning_rate: 2.5000e-05
Epoch 18/20
5/5             23s 5s/step -
accuracy: 0.4916 - loss: 0.8689 - val_accuracy: 0.4750 - val_loss: 0.7835 -
learning_rate: 2.5000e-05
Epoch 19/20
5/5             21s 4s/step -
accuracy: 0.6105 - loss: 0.8300 - val_accuracy: 0.4750 - val_loss: 0.7876 -
learning_rate: 1.2500e-05
Epoch 20/20
5/5             42s 4s/step -
accuracy: 0.4676 - loss: 0.9888 - val_accuracy: 0.4750 - val_loss: 0.7885 -
learning_rate: 1.2500e-05
2/2             2s 421ms/step -
accuracy: 0.4625 - loss: 0.7954
Test accuracy: 0.475
```

3.4 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
0x7f277c635440> triggered tf.function retracing. Tracing is expensive and the
excessive number of tracings could be due to (1) creating @tf.function
repeatedly in a loop, (2) passing tensors with different shapes, (3) passing
Python objects instead of tensors. For (1), please define your @tf.function
outside of the loop. For (2), @tf.function has reduce_retracing=True option that
can avoid unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more details.
1/2             1s
2s/stepWARNING:tensorflow:6 out of the last 11 calls to <function
TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at
0x7f277c635440> 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             3s 974ms/step
```



3.5 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', 'onset_images_full')
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}")
```

```
['pop', 'disco', 'blues', 'hiphop', 'classical']
Going through pop
Going through disco
Going through blues
Going through hiphop
Going through classical
```

```
/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             59s 4s/step -
accuracy: 0.1730 - loss: 2.1929 - val_accuracy: 0.2800 - val_loss: 1.7812 -
learning_rate: 1.0000e-04
```

```
Epoch 2/20
13/13             80s 4s/step -
accuracy: 0.1954 - loss: 2.0156 - val_accuracy: 0.2800 - val_loss: 1.8130 -
learning_rate: 1.0000e-04
Epoch 3/20
13/13             50s 4s/step -
accuracy: 0.1886 - loss: 1.9124 - val_accuracy: 0.1200 - val_loss: 1.7419 -
learning_rate: 1.0000e-04
Epoch 4/20
13/13             52s 4s/step -
accuracy: 0.2469 - loss: 1.7632 - val_accuracy: 0.2800 - val_loss: 1.7076 -
learning_rate: 1.0000e-04
Epoch 5/20
13/13             48s 4s/step -
accuracy: 0.2089 - loss: 1.8389 - val_accuracy: 0.2400 - val_loss: 1.6840 -
learning_rate: 1.0000e-04
Epoch 6/20
13/13             86s 4s/step -
accuracy: 0.2441 - loss: 1.7770 - val_accuracy: 0.2400 - val_loss: 1.6297 -
learning_rate: 1.0000e-04
Epoch 7/20
13/13             78s 4s/step -
accuracy: 0.2397 - loss: 1.7255 - val_accuracy: 0.2900 - val_loss: 1.6316 -
learning_rate: 1.0000e-04
Epoch 8/20
13/13             48s 4s/step -
accuracy: 0.2112 - loss: 1.7319 - val_accuracy: 0.2100 - val_loss: 1.6467 -
learning_rate: 1.0000e-04
Epoch 9/20
13/13             47s 4s/step -
accuracy: 0.2383 - loss: 1.7080 - val_accuracy: 0.3700 - val_loss: 1.5495 -
learning_rate: 1.0000e-04
Epoch 10/20
13/13             44s 3s/step -
accuracy: 0.2631 - loss: 1.6567 - val_accuracy: 0.3600 - val_loss: 1.5354 -
learning_rate: 1.0000e-04
Epoch 11/20
13/13             49s 4s/step -
accuracy: 0.2527 - loss: 1.6793 - val_accuracy: 0.3800 - val_loss: 1.5181 -
learning_rate: 1.0000e-04
Epoch 12/20
13/13             75s 3s/step -
accuracy: 0.2461 - loss: 1.6809 - val_accuracy: 0.4300 - val_loss: 1.4819 -
learning_rate: 1.0000e-04
Epoch 13/20
13/13             42s 3s/step -
accuracy: 0.2830 - loss: 1.5821 - val_accuracy: 0.3400 - val_loss: 1.4643 -
learning_rate: 1.0000e-04
```

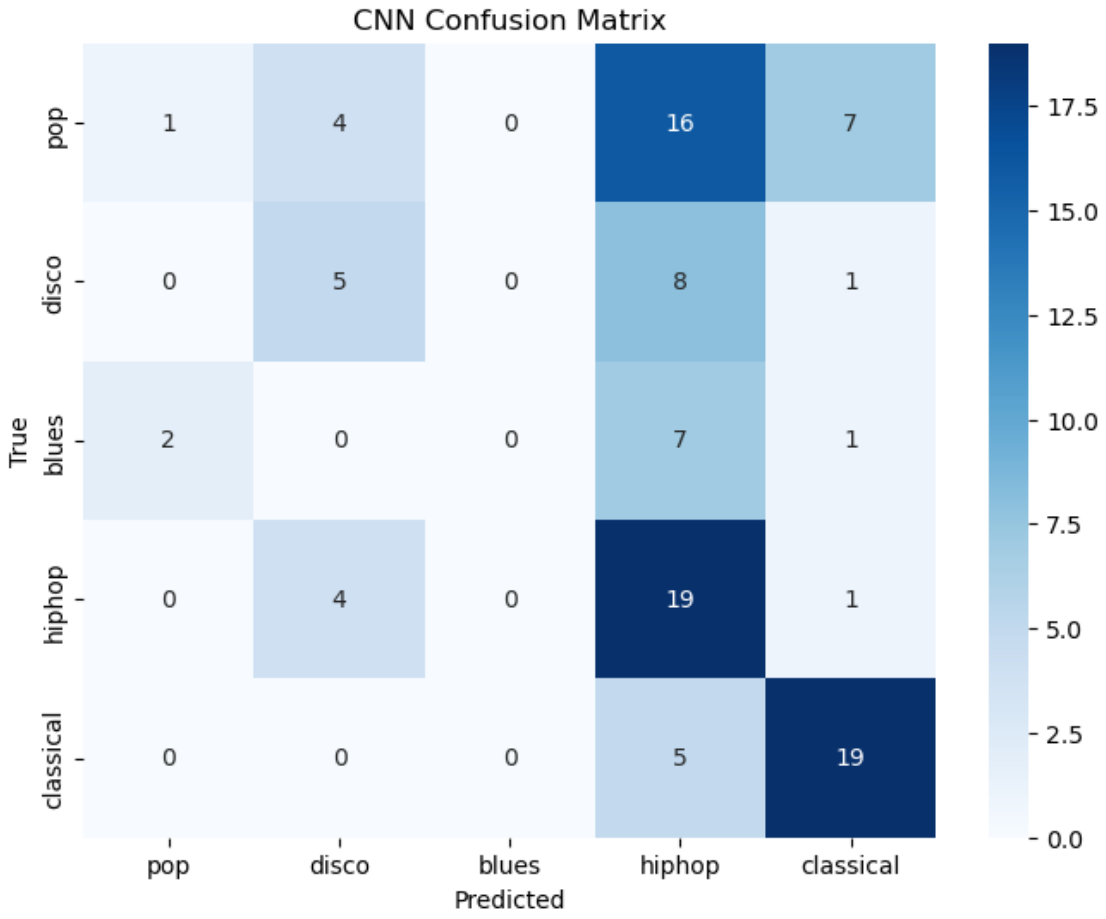
```
Epoch 14/20
13/13             86s 4s/step -
accuracy: 0.3375 - loss: 1.5726 - val_accuracy: 0.3300 - val_loss: 1.5455 -
learning_rate: 1.0000e-04
Epoch 15/20
13/13             50s 4s/step -
accuracy: 0.2957 - loss: 1.6420 - val_accuracy: 0.3200 - val_loss: 1.5639 -
learning_rate: 1.0000e-04
Epoch 16/20
13/13             72s 3s/step -
accuracy: 0.2752 - loss: 1.6421 - val_accuracy: 0.4000 - val_loss: 1.4665 -
learning_rate: 1.0000e-04
Epoch 17/20
13/13             45s 3s/step -
accuracy: 0.3312 - loss: 1.5285 - val_accuracy: 0.4000 - val_loss: 1.4297 -
learning_rate: 5.0000e-05
Epoch 18/20
13/13             47s 4s/step -
accuracy: 0.3402 - loss: 1.5993 - val_accuracy: 0.4200 - val_loss: 1.4438 -
learning_rate: 5.0000e-05
Epoch 19/20
13/13             77s 3s/step -
accuracy: 0.3198 - loss: 1.5287 - val_accuracy: 0.4100 - val_loss: 1.4163 -
learning_rate: 5.0000e-05
Epoch 20/20
13/13             48s 4s/step -
accuracy: 0.3116 - loss: 1.5526 - val_accuracy: 0.4400 - val_loss: 1.4099 -
learning_rate: 5.0000e-05
4/4             3s 740ms/step -
accuracy: 0.4635 - loss: 1.4057
Test accuracy: 0.440
```

3.6 14 - Confusion Matrix Medium (5 random)

[16]:
```python
import seaborn as sns
# from sklearn.metrics import confusion
import numpy as NP
from sklearn.metrics import confusion_matrix

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

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

```
plt.xlabel("Predicted")
plt.ylabel("True")
plt.show()
```

```
4/4                 4s 830ms/step
```



4 Apply the confusion matrix after the model

[ ]: