Onset HeatMap Clips-Only (3 secs) CNN

March 20, 2025

1 CNN for Onset HeatMap Clip Images (3 secs)

1.1 1 - All the imports

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

```
2025-03-20 08:41:39.459993: 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)

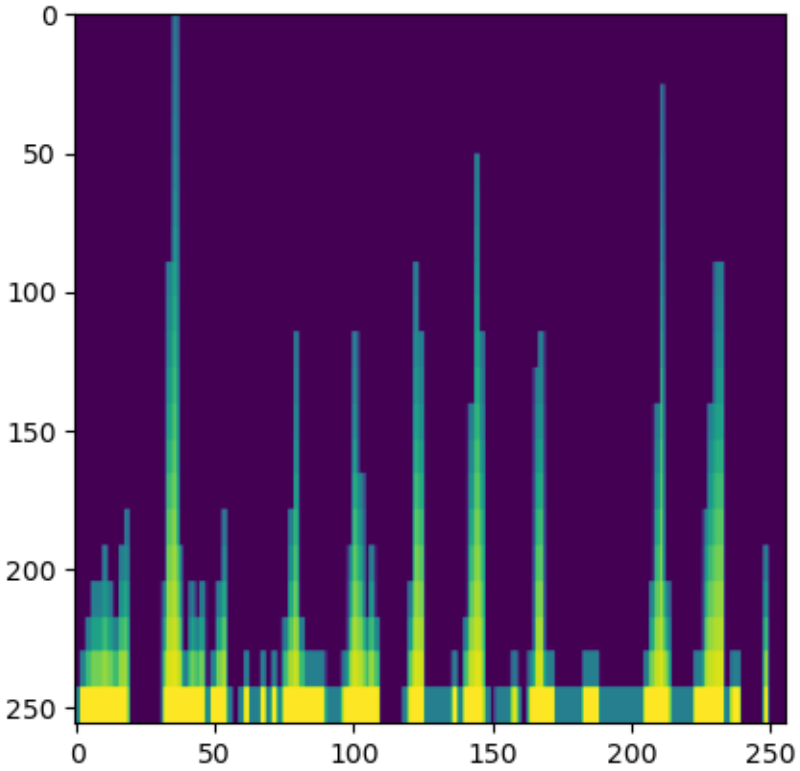
3 4 - Load the images

[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_heatmaps (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]:
```
# 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
250/250             1824s 7s/step -
accuracy: 0.1232 - loss: 3.2460 - val_accuracy: 0.1010 - val_loss: 5.1231 -
learning_rate: 1.0000e-04
Epoch 2/20
250/250             1916s 8s/step -
accuracy: 0.1342 - loss: 2.2658 - val_accuracy: 0.1125 - val_loss: 2.2836 -
learning_rate: 1.0000e-04
Epoch 3/20
250/250             1700s 7s/step -
accuracy: 0.1594 - loss: 2.2341 - val_accuracy: 0.1605 - val_loss: 2.2100 -
learning_rate: 1.0000e-04
Epoch 4/20
250/250             1442s 6s/step -
accuracy: 0.1597 - loss: 2.2294 - val_accuracy: 0.1835 - val_loss: 2.1868 -
learning_rate: 1.0000e-04
Epoch 5/20
250/250             1354s 5s/step -
accuracy: 0.1643 - loss: 2.2126 - val_accuracy: 0.1830 - val_loss: 2.1783 -
learning_rate: 1.0000e-04
Epoch 6/20
250/250             1298s 5s/step -
accuracy: 0.1662 - loss: 2.1974 - val_accuracy: 0.1820 - val_loss: 2.1697 -
learning_rate: 1.0000e-04
Epoch 7/20
250/250             1114s 4s/step -
accuracy: 0.1749 - loss: 2.1746 - val_accuracy: 0.1905 - val_loss: 2.1442 -
learning_rate: 1.0000e-04
Epoch 8/20
250/250             1106s 4s/step -
accuracy: 0.1859 - loss: 2.1552 - val_accuracy: 0.1965 - val_loss: 2.1452 -
learning_rate: 1.0000e-04
Epoch 9/20
250/250             1124s 4s/step -
accuracy: 0.1823 - loss: 2.1420 - val_accuracy: 0.2150 - val_loss: 2.1270 -
learning_rate: 1.0000e-04
Epoch 10/20
250/250             1151s 4s/step -
accuracy: 0.1790 - loss: 2.1550 - val_accuracy: 0.2020 - val_loss: 2.1227 -
learning_rate: 1.0000e-04
Epoch 11/20
250/250             1109s 4s/step -
accuracy: 0.1920 - loss: 2.1323 - val_accuracy: 0.1940 - val_loss: 2.1427 -
learning_rate: 1.0000e-04
Epoch 12/20
```

```
250/250             1139s 4s/step -
accuracy: 0.1919 - loss: 2.1172 - val_accuracy: 0.2070 - val_loss: 2.1026 -
learning_rate: 1.0000e-04
Epoch 13/20
250/250             1121s 4s/step -
accuracy: 0.1988 - loss: 2.0976 - val_accuracy: 0.1890 - val_loss: 2.1514 -
learning_rate: 1.0000e-04
Epoch 14/20
250/250             1026s 4s/step -
accuracy: 0.2057 - loss: 2.0857 - val_accuracy: 0.2165 - val_loss: 2.1089 -
learning_rate: 1.0000e-04
Epoch 15/20
250/250             904s 4s/step -
accuracy: 0.2285 - loss: 2.0483 - val_accuracy: 0.2320 - val_loss: 2.0777 -
learning_rate: 1.0000e-04
Epoch 16/20
250/250             929s 4s/step -
accuracy: 0.2063 - loss: 2.0366 - val_accuracy: 0.2210 - val_loss: 2.0858 -
learning_rate: 1.0000e-04
Epoch 17/20
250/250             883s 4s/step -
accuracy: 0.2126 - loss: 2.0374 - val_accuracy: 0.2280 - val_loss: 2.0685 -
learning_rate: 1.0000e-04
Epoch 18/20
250/250             936s 4s/step -
accuracy: 0.2316 - loss: 1.9973 - val_accuracy: 0.2130 - val_loss: 2.0938 -
learning_rate: 1.0000e-04
Epoch 19/20
250/250             923s 4s/step -
accuracy: 0.2342 - loss: 2.0400 - val_accuracy: 0.2170 - val_loss: 2.0909 -
learning_rate: 1.0000e-04
Epoch 20/20
250/250             930s 4s/step -
accuracy: 0.2437 - loss: 1.9813 - val_accuracy: 0.2220 - val_loss: 2.0828 -
learning_rate: 1.0000e-04
```

[8]:
```
<keras.src.callbacks.history.History at 0x7febb848e4e0>
```

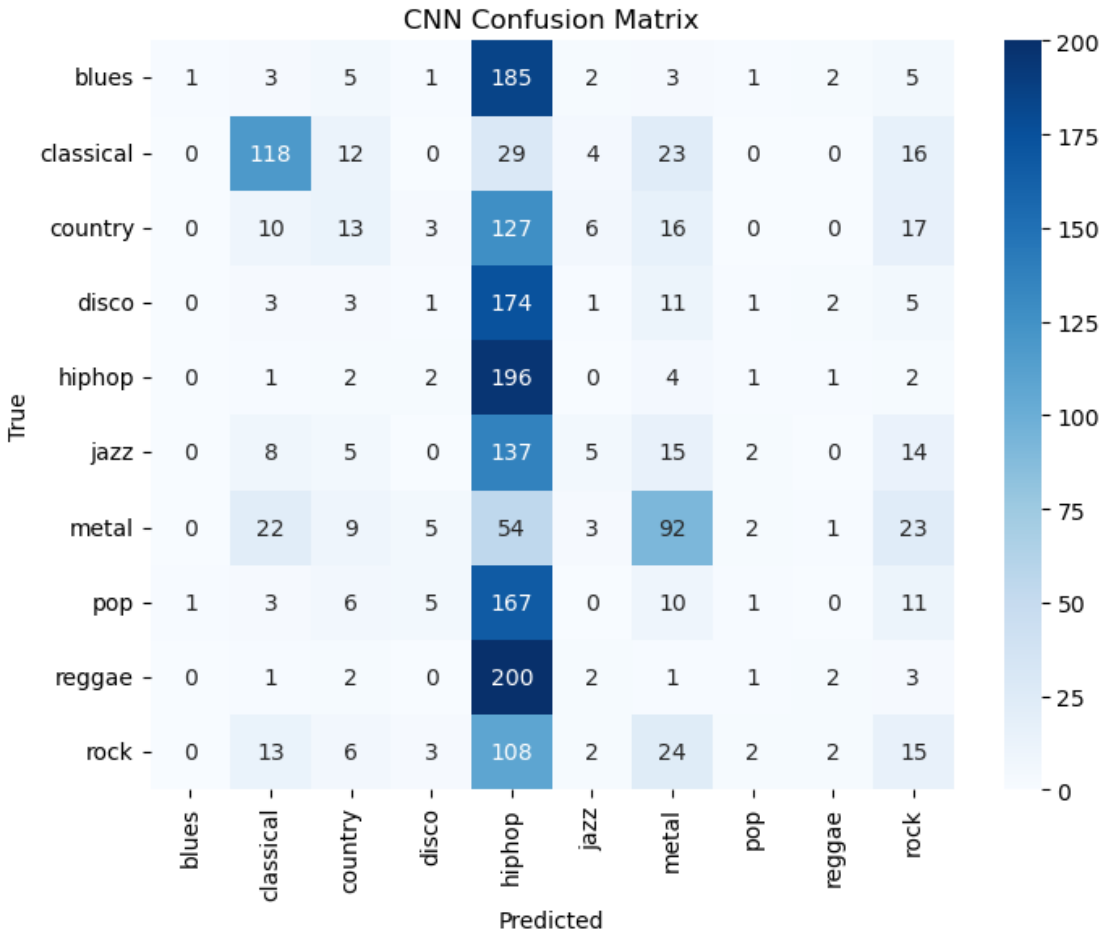
[9]:
```
evaluation = model.evaluate(X_test, y_test)
print(f"Test accuracy: {evaluation[1]:.3f}")
```

```
63/63             29s 462ms/step -
accuracy: 0.2235 - loss: 2.0759
Test accuracy: 0.222
```

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                30s 470ms/step
```



4.1 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_heatmaps (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}")
```

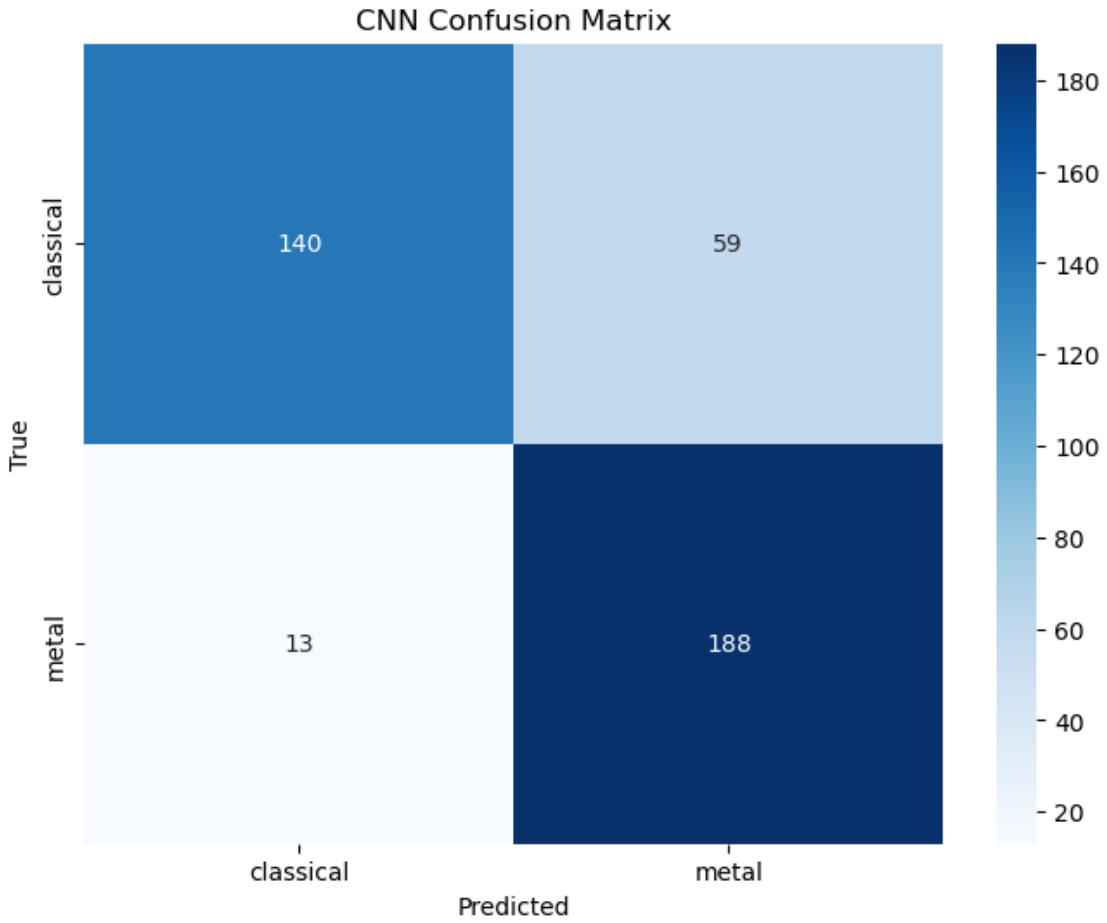
```
Going through classical
Going through metal

/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             114s 2s/step -
accuracy: 0.4191 - loss: 1.4383 - val_accuracy: 0.5025 - val_loss: 0.7113 -
learning_rate: 1.0000e-04
Epoch 2/20
50/50             121s 2s/step -
accuracy: 0.5213 - loss: 0.8330 - val_accuracy: 0.6275 - val_loss: 0.6854 -
learning_rate: 1.0000e-04
Epoch 3/20
50/50             144s 2s/step -
accuracy: 0.5505 - loss: 0.7688 - val_accuracy: 0.6900 - val_loss: 0.6176 -
learning_rate: 1.0000e-04
Epoch 4/20
50/50             133s 2s/step -
accuracy: 0.5957 - loss: 0.7272 - val_accuracy: 0.7250 - val_loss: 0.5794 -
learning_rate: 1.0000e-04
Epoch 5/20
50/50             100s 2s/step -
accuracy: 0.6318 - loss: 0.6545 - val_accuracy: 0.7400 - val_loss: 0.5513 -
learning_rate: 1.0000e-04
Epoch 6/20
50/50             91s 2s/step -
accuracy: 0.6824 - loss: 0.6191 - val_accuracy: 0.7375 - val_loss: 0.5341 -
learning_rate: 1.0000e-04
Epoch 7/20
50/50             97s 2s/step -
accuracy: 0.6768 - loss: 0.6151 - val_accuracy: 0.7650 - val_loss: 0.5092 -
learning_rate: 1.0000e-04
Epoch 8/20
50/50             93s 2s/step -
accuracy: 0.7167 - loss: 0.5675 - val_accuracy: 0.7975 - val_loss: 0.4809 -
learning_rate: 1.0000e-04
Epoch 9/20
50/50             113s 2s/step -
accuracy: 0.7353 - loss: 0.5557 - val_accuracy: 0.8175 - val_loss: 0.4655 -
learning_rate: 1.0000e-04
Epoch 10/20
50/50             106s 2s/step -
accuracy: 0.7639 - loss: 0.5134 - val_accuracy: 0.7975 - val_loss: 0.4780 -
```

```
learning_rate: 1.0000e-04
Epoch 11/20
50/50             108s 2s/step -
accuracy: 0.7722 - loss: 0.5093 - val_accuracy: 0.8100 - val_loss: 0.4452 -
learning_rate: 1.0000e-04
Epoch 12/20
50/50             105s 2s/step -
accuracy: 0.7918 - loss: 0.4801 - val_accuracy: 0.8100 - val_loss: 0.4213 -
learning_rate: 1.0000e-04
Epoch 13/20
50/50             101s 2s/step -
accuracy: 0.8150 - loss: 0.4332 - val_accuracy: 0.8075 - val_loss: 0.4192 -
learning_rate: 1.0000e-04
Epoch 14/20
50/50             97s 2s/step -
accuracy: 0.8057 - loss: 0.4475 - val_accuracy: 0.7650 - val_loss: 0.4965 -
learning_rate: 1.0000e-04
Epoch 15/20
50/50             92s 2s/step -
accuracy: 0.8247 - loss: 0.4221 - val_accuracy: 0.8200 - val_loss: 0.4084 -
learning_rate: 1.0000e-04
Epoch 16/20
50/50             141s 2s/step -
accuracy: 0.8289 - loss: 0.4202 - val_accuracy: 0.8250 - val_loss: 0.4094 -
learning_rate: 1.0000e-04
Epoch 17/20
50/50             97s 2s/step -
accuracy: 0.8302 - loss: 0.4055 - val_accuracy: 0.8200 - val_loss: 0.4061 -
learning_rate: 1.0000e-04
Epoch 18/20
50/50             96s 2s/step -
accuracy: 0.8488 - loss: 0.3693 - val_accuracy: 0.8325 - val_loss: 0.3984 -
learning_rate: 1.0000e-04
Epoch 19/20
50/50             60s 1s/step -
accuracy: 0.8341 - loss: 0.3832 - val_accuracy: 0.8175 - val_loss: 0.4142 -
learning_rate: 1.0000e-04
Epoch 20/20
50/50             80s 2s/step -
accuracy: 0.8538 - loss: 0.3455 - val_accuracy: 0.8200 - val_loss: 0.4221 -
learning_rate: 1.0000e-04
13/13             4s 289ms/step -
accuracy: 0.8363 - loss: 0.4000
Test accuracy: 0.820
```

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                4s 303ms/step
```



4.3 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_heatmaps (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}")
```

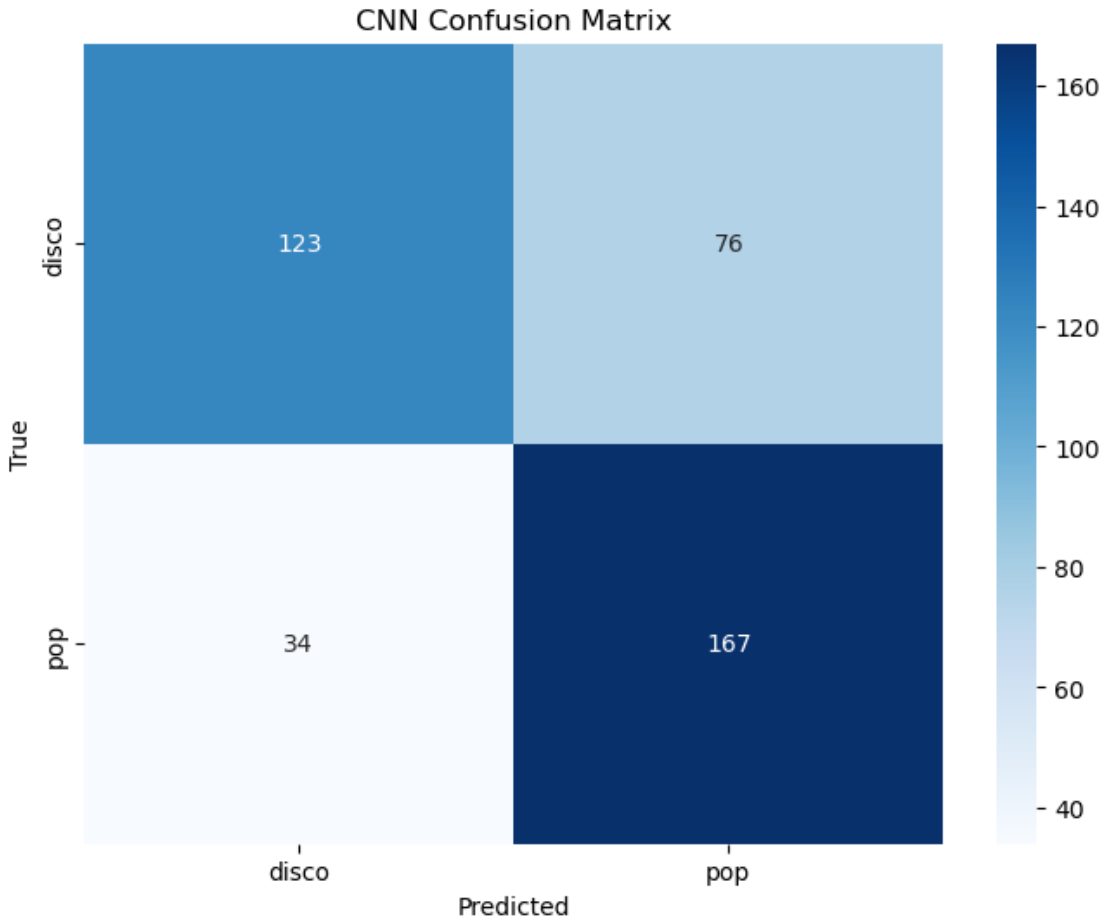
```
Going through disco
Going through pop

/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: 0.4644 - loss: 1.5534 - val_accuracy: 0.4975 - val_loss: 0.7865 -
learning_rate: 1.0000e-04
Epoch 2/20
50/50             84s 2s/step -
accuracy: 0.4805 - loss: 0.9125 - val_accuracy: 0.5025 - val_loss: 0.7245 -
learning_rate: 1.0000e-04
Epoch 3/20
50/50             71s 1s/step -
accuracy: 0.5325 - loss: 0.8011 - val_accuracy: 0.6075 - val_loss: 0.7090 -
learning_rate: 1.0000e-04
Epoch 4/20
50/50             82s 2s/step -
accuracy: 0.5008 - loss: 0.7686 - val_accuracy: 0.5025 - val_loss: 0.7097 -
learning_rate: 1.0000e-04
Epoch 5/20
50/50             84s 2s/step -
accuracy: 0.5288 - loss: 0.7617 - val_accuracy: 0.6375 - val_loss: 0.6920 -
learning_rate: 1.0000e-04
Epoch 6/20
50/50             81s 2s/step -
accuracy: 0.5223 - loss: 0.7416 - val_accuracy: 0.5025 - val_loss: 0.6961 -
learning_rate: 1.0000e-04
Epoch 7/20
50/50             68s 1s/step -
accuracy: 0.5170 - loss: 0.7351 - val_accuracy: 0.5550 - val_loss: 0.6917 -
learning_rate: 1.0000e-04
Epoch 8/20
50/50             74s 1s/step -
accuracy: 0.5277 - loss: 0.7242 - val_accuracy: 0.5025 - val_loss: 0.6915 -
learning_rate: 1.0000e-04
Epoch 9/20
50/50             84s 2s/step -
accuracy: 0.5707 - loss: 0.7021 - val_accuracy: 0.6900 - val_loss: 0.6408 -
learning_rate: 1.0000e-04
Epoch 10/20
50/50             79s 2s/step -
accuracy: 0.5952 - loss: 0.6782 - val_accuracy: 0.7025 - val_loss: 0.6043 -
```

```
learning_rate: 1.0000e-04
Epoch 11/20
50/50             75s 1s/step -
accuracy: 0.6632 - loss: 0.6301 - val_accuracy: 0.7050 - val_loss: 0.5958 -
learning_rate: 1.0000e-04
Epoch 12/20
50/50             84s 2s/step -
accuracy: 0.6614 - loss: 0.6249 - val_accuracy: 0.7150 - val_loss: 0.5827 -
learning_rate: 1.0000e-04
Epoch 13/20
50/50             142s 2s/step -
accuracy: 0.7036 - loss: 0.6023 - val_accuracy: 0.7200 - val_loss: 0.5619 -
learning_rate: 1.0000e-04
Epoch 14/20
50/50             84s 2s/step -
accuracy: 0.7347 - loss: 0.5545 - val_accuracy: 0.6900 - val_loss: 0.6032 -
learning_rate: 1.0000e-04
Epoch 15/20
50/50             85s 2s/step -
accuracy: 0.7299 - loss: 0.5711 - val_accuracy: 0.7050 - val_loss: 0.5768 -
learning_rate: 1.0000e-04
Epoch 16/20
50/50             85s 2s/step -
accuracy: 0.7323 - loss: 0.5592 - val_accuracy: 0.7275 - val_loss: 0.5612 -
learning_rate: 1.0000e-04
Epoch 17/20
50/50             141s 2s/step -
accuracy: 0.7275 - loss: 0.5543 - val_accuracy: 0.7350 - val_loss: 0.5533 -
learning_rate: 1.0000e-04
Epoch 18/20
50/50             142s 2s/step -
accuracy: 0.7436 - loss: 0.5249 - val_accuracy: 0.7275 - val_loss: 0.5556 -
learning_rate: 1.0000e-04
Epoch 19/20
50/50             141s 2s/step -
accuracy: 0.7513 - loss: 0.5255 - val_accuracy: 0.7500 - val_loss: 0.5384 -
learning_rate: 1.0000e-04
Epoch 20/20
50/50             71s 1s/step -
accuracy: 0.7604 - loss: 0.5156 - val_accuracy: 0.7250 - val_loss: 0.5732 -
learning_rate: 1.0000e-04
13/13             4s 273ms/step -
accuracy: 0.7282 - loss: 0.5859
Test accuracy: 0.725
```

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                4s 296ms/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', 'onset_heatmaps (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}")
```

```
['pop', 'blues', 'rock', 'metal', 'jazz']
Going through pop
Going through blues
Going through rock
Going through metal
Going through jazz
```

```
/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             209s 2s/step -
accuracy: 0.2040 - loss: 1.9535 - val_accuracy: 0.1900 - val_loss: 1.6738 -
learning_rate: 1.0000e-04
Epoch 2/20
125/125             177s 1s/step -
accuracy: 0.2039 - loss: 1.7171 - val_accuracy: 0.2680 - val_loss: 1.6175 -
learning_rate: 1.0000e-04
Epoch 3/20
125/125             169s 1s/step -
accuracy: 0.2443 - loss: 1.6528 - val_accuracy: 0.3310 - val_loss: 1.5688 -
learning_rate: 1.0000e-04
Epoch 4/20
125/125             196s 2s/step -
accuracy: 0.2821 - loss: 1.6019 - val_accuracy: 0.3270 - val_loss: 1.5490 -
learning_rate: 1.0000e-04
Epoch 5/20
125/125             216s 2s/step -
accuracy: 0.2955 - loss: 1.5774 - val_accuracy: 0.3580 - val_loss: 1.5250 -
learning_rate: 1.0000e-04
Epoch 6/20
125/125             200s 2s/step -
accuracy: 0.2926 - loss: 1.5497 - val_accuracy: 0.3400 - val_loss: 1.5250 -
learning_rate: 1.0000e-04
Epoch 7/20
125/125             215s 2s/step -
accuracy: 0.3288 - loss: 1.5243 - val_accuracy: 0.3520 - val_loss: 1.4970 -
learning_rate: 1.0000e-04
Epoch 8/20
125/125             212s 2s/step -
accuracy: 0.3539 - loss: 1.4933 - val_accuracy: 0.3810 - val_loss: 1.4574 -
```

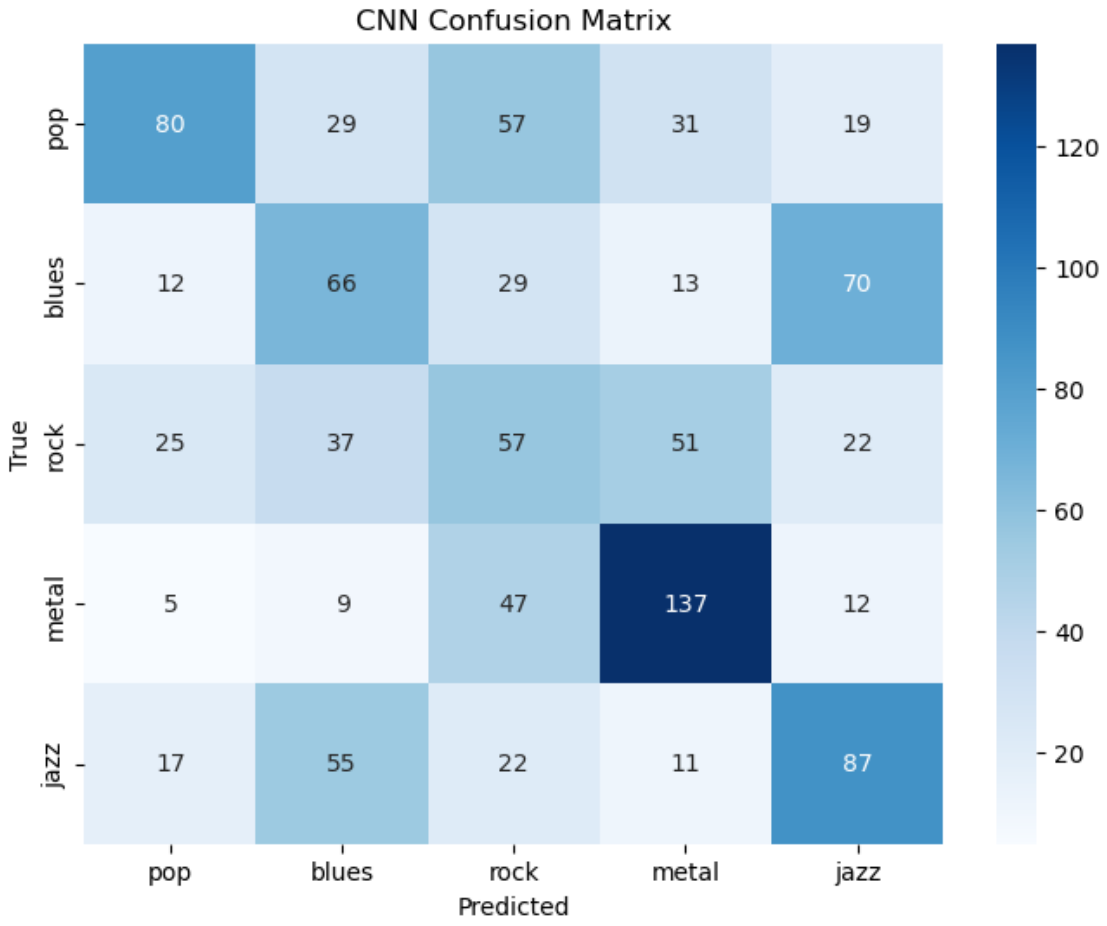
```
learning_rate: 1.0000e-04
Epoch 9/20
125/125             212s 2s/step -
accuracy: 0.3876 - loss: 1.4512 - val_accuracy: 0.4060 - val_loss: 1.4174 -
learning_rate: 1.0000e-04
Epoch 10/20
125/125             246s 2s/step -
accuracy: 0.3939 - loss: 1.4138 - val_accuracy: 0.4010 - val_loss: 1.3704 -
learning_rate: 1.0000e-04
Epoch 11/20
125/125             213s 2s/step -
accuracy: 0.4364 - loss: 1.3467 - val_accuracy: 0.4180 - val_loss: 1.3310 -
learning_rate: 1.0000e-04
Epoch 12/20
125/125             261s 2s/step -
accuracy: 0.4506 - loss: 1.2899 - val_accuracy: 0.4120 - val_loss: 1.3181 -
learning_rate: 1.0000e-04
Epoch 13/20
125/125             220s 2s/step -
accuracy: 0.4713 - loss: 1.2552 - val_accuracy: 0.4240 - val_loss: 1.3256 -
learning_rate: 1.0000e-04
Epoch 14/20
125/125             220s 2s/step -
accuracy: 0.5043 - loss: 1.1947 - val_accuracy: 0.4230 - val_loss: 1.3161 -
learning_rate: 1.0000e-04
Epoch 15/20
125/125             212s 2s/step -
accuracy: 0.5271 - loss: 1.1618 - val_accuracy: 0.4140 - val_loss: 1.3268 -
learning_rate: 1.0000e-04
Epoch 16/20
125/125             221s 2s/step -
accuracy: 0.5575 - loss: 1.0823 - val_accuracy: 0.3890 - val_loss: 1.3805 -
learning_rate: 1.0000e-04
Epoch 17/20
125/125             253s 2s/step -
accuracy: 0.6019 - loss: 1.0037 - val_accuracy: 0.4370 - val_loss: 1.3334 -
learning_rate: 1.0000e-04
Epoch 18/20
125/125             201s 2s/step -
accuracy: 0.6565 - loss: 0.8975 - val_accuracy: 0.4420 - val_loss: 1.4169 -
learning_rate: 5.0000e-05
Epoch 19/20
125/125             211s 2s/step -
accuracy: 0.6860 - loss: 0.8304 - val_accuracy: 0.4320 - val_loss: 1.4507 -
learning_rate: 5.0000e-05
Epoch 20/20
125/125             191s 2s/step -
accuracy: 0.6984 - loss: 0.7857 - val_accuracy: 0.4270 - val_loss: 1.5247 -
```

```
learning_rate: 5.0000e-05
32/32             9s 280ms/step -
accuracy: 0.4012 - loss: 1.5807
Test accuracy: 0.427
```

4.6 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()
```

```
32/32             10s 295ms/step
```


CNN Confusion Matrix
True
Predicted
pop
blues
rock
metal
jazz
80 29 57 31 19
12 66 29 13 70
25 37 57 51 22
5 9 47 137 12
17 55 22 11 87
120
100
80
60
40
20