Mel-Spectrogram-Only CNN

March 21, 2025

1 CNN for Mel-Spectrogram (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
```

```
2025-03-21 00:51:40.647571: I tensorflow/core/platform/cpu_feature_guard.cc:210]
This TensorFlow binary is optimized to use available CPU instructions in
performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild
TensorFlow with the appropriate compiler flags.
```

1.2 2 - Put the data within the model

[2]:
```
# Import a single image and save it to be read by the model

image = os.path.join('blues.00000.png')

# Load the image
image = tf.io.read_file(image)

# Convert to a numpy array
image = tf.image.decode_png(image, channels=1)
image = tf.image.convert_image_dtype(image, tf.float32)
image = tf.image.resize(image, [256, 256])
image = image.numpy()
```

2 3 - Create the model

[3]:
```
from tensorflow.keras import models
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,␣
 ↪Dropout, Normalization

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

```
    Normalization(),
    MaxPooling2D((2, 2)),

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

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

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

    Flatten(),

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

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

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

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

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

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 254, 254, 32)	320
normalization (Normalization)	(None, 254, 254, 32)	65
max_pooling2d (MaxPooling2D)	(None, 127, 127, 32)	0
conv2d_1 (Conv2D)	(None, 125, 125, 64)	18,496

normalization_1 (Normalization)	(None, 125, 125, 64)	129
max_pooling2d_1 (MaxPooling2D)	(None, 62, 62, 64)	0
conv2d_2 (Conv2D)	(None, 60, 60, 128)	73,856
normalization_2 (Normalization)	(None, 60, 60, 128)	257
max_pooling2d_2 (MaxPooling2D)	(None, 30, 30, 128)	0
conv2d_3 (Conv2D)	(None, 28, 28, 256)	295,168
normalization_3 (Normalization)	(None, 28, 28, 256)	513
max_pooling2d_3 (MaxPooling2D)	(None, 14, 14, 256)	0
flatten (Flatten)	(None, 50176)	0
dense (Dense)	(None, 512)	25,690,624
dropout (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 256)	131,328
dropout_1 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 128)	32,896
dense_3 (Dense)	(None, 10)	1,290

Total params: 26,244,942 (100.12 MB)

Trainable params: 26,243,978 (100.11 MB)

Non-trainable params: 964 (3.78 KB)

3 4 - Load the images

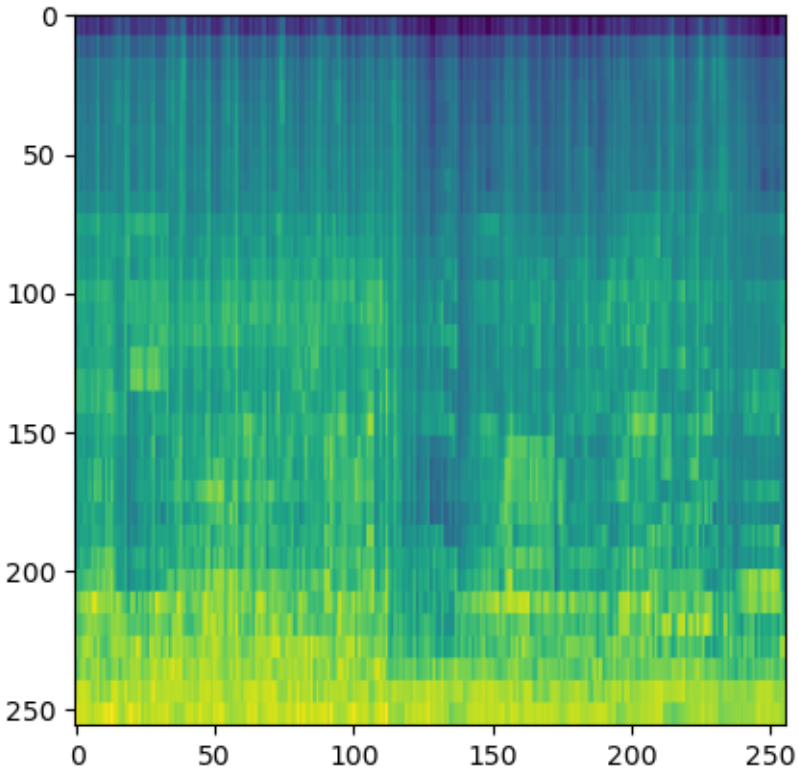
[5]:
```
import os
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split
```

```python
# Augmentation function
def augment_image(image):
    image = tf.image.random_flip_left_right(image)
    image = tf.image.random_brightness(image, max_delta=0.1)
    image = tf.image.random_contrast(image, 0.8, 1.2)
    return image

# Define the genres and file paths
GENRES = ['blues', 'classical', 'country', 'disco', 'hiphop', 'jazz', 'metal',␣
 ↪'pop', 'reggae', 'rock']
FILE_PATH = os.path.join('Data', 'mel_spectrograms', 'mel_spectrogram_32')
X = []
y = []

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

# Loop through the genres and load the images with augmentation
for genre in GENRES:
    genre_dir = os.path.join(FILE_PATH, genre)
    print(f"Going through {genre}")
    for file in os.listdir(genre_dir):
        image = tf.io.read_file(os.path.join(genre_dir, file))
        image = tf.image.decode_png(image, channels=1)
        image = tf.image.convert_image_dtype(image, tf.float32)
        image = tf.image.resize(image, [256, 256])  # Resize to 256x256
        image = augment_image(image)  # Apply augmentation
        image = image.numpy()  # Convert to numpy array
        X.append(image)
        y.append(GENRE_TO_INDEX[genre])

# Convert lists to numpy arrays
X = np.array(X)
y = np.array(y)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

```
Going through blues
Going through classical
Going through country
Going through disco
Going through hiphop
Going through jazz
Going through metal
Going through pop
Going through reggae
```

```
Going through rock
```

[6]:
```
# Show image as a sanity check
import matplotlib.pyplot as plt
plt.imshow(X_train[22].reshape(256, 256))
plt.show()
```



3.1 5 - Compile the model

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

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

3.2 6 - Fit the model

[8]:
```
model.fit(X_train, y_train, epochs=20, validation_data=(X_test, y_test),␣
↪batch_size=32, callbacks=[reduce_lr])
```

```
Epoch 1/20
25/25             65s 2s/step -
accuracy: 0.0953 - loss: 2.3052 - val_accuracy: 0.0650 - val_loss: 2.3025 -
learning_rate: 1.0000e-04
Epoch 2/20
25/25             50s 2s/step -
accuracy: 0.1109 - loss: 2.2979 - val_accuracy: 0.2150 - val_loss: 2.2986 -
learning_rate: 1.0000e-04
Epoch 3/20
25/25             37s 1s/step -
accuracy: 0.1290 - loss: 2.2930 - val_accuracy: 0.1750 - val_loss: 2.2816 -
learning_rate: 1.0000e-04
Epoch 4/20
25/25             52s 2s/step -
accuracy: 0.1985 - loss: 2.2460 - val_accuracy: 0.2150 - val_loss: 2.1861 -
learning_rate: 1.0000e-04
Epoch 5/20
25/25             82s 2s/step -
accuracy: 0.2158 - loss: 2.1599 - val_accuracy: 0.2950 - val_loss: 2.0833 -
learning_rate: 1.0000e-04
Epoch 6/20
25/25             60s 2s/step -
accuracy: 0.2469 - loss: 2.0954 - val_accuracy: 0.2800 - val_loss: 1.9965 -
learning_rate: 1.0000e-04
Epoch 7/20
25/25             101s 3s/step -
accuracy: 0.3163 - loss: 1.9898 - val_accuracy: 0.2850 - val_loss: 1.9485 -
learning_rate: 1.0000e-04
Epoch 8/20
25/25             76s 3s/step -
accuracy: 0.3015 - loss: 1.8769 - val_accuracy: 0.2900 - val_loss: 1.8989 -
learning_rate: 1.0000e-04
Epoch 9/20
25/25             77s 3s/step -
accuracy: 0.3454 - loss: 1.7936 - val_accuracy: 0.3000 - val_loss: 1.8894 -
learning_rate: 1.0000e-04
Epoch 10/20
25/25             74s 3s/step -
accuracy: 0.3017 - loss: 1.8817 - val_accuracy: 0.3050 - val_loss: 1.8379 -
learning_rate: 1.0000e-04
Epoch 11/20
25/25             84s 3s/step -
accuracy: 0.3426 - loss: 1.7582 - val_accuracy: 0.3400 - val_loss: 1.8200 -
```

```
learning_rate: 1.0000e-04
Epoch 12/20
25/25             101s 4s/step -
accuracy: 0.3505 - loss: 1.7707 - val_accuracy: 0.3100 - val_loss: 1.8360 -
learning_rate: 1.0000e-04
Epoch 13/20
25/25             85s 3s/step -
accuracy: 0.3476 - loss: 1.7588 - val_accuracy: 0.3150 - val_loss: 1.8088 -
learning_rate: 1.0000e-04
Epoch 14/20
25/25             147s 4s/step -
accuracy: 0.3952 - loss: 1.6645 - val_accuracy: 0.3650 - val_loss: 1.7219 -
learning_rate: 1.0000e-04
Epoch 15/20
25/25             67s 3s/step -
accuracy: 0.3605 - loss: 1.7098 - val_accuracy: 0.3550 - val_loss: 1.7054 -
learning_rate: 1.0000e-04
Epoch 16/20
25/25             66s 3s/step -
accuracy: 0.3802 - loss: 1.6301 - val_accuracy: 0.3500 - val_loss: 1.7528 -
learning_rate: 1.0000e-04
Epoch 17/20
25/25             76s 3s/step -
accuracy: 0.3605 - loss: 1.7336 - val_accuracy: 0.3850 - val_loss: 1.6834 -
learning_rate: 1.0000e-04
Epoch 18/20
25/25             72s 3s/step -
accuracy: 0.4093 - loss: 1.6555 - val_accuracy: 0.4350 - val_loss: 1.6226 -
learning_rate: 1.0000e-04
Epoch 19/20
25/25             81s 3s/step -
accuracy: 0.4258 - loss: 1.5318 - val_accuracy: 0.4100 - val_loss: 1.6149 -
learning_rate: 1.0000e-04
Epoch 20/20
25/25             73s 3s/step -
accuracy: 0.4750 - loss: 1.4560 - val_accuracy: 0.3900 - val_loss: 1.6673 -
learning_rate: 1.0000e-04
```

[8]:
```
<keras.src.callbacks.history.History at 0x7fcab0128ec0>
```

3.3 7 - Check the accuracy

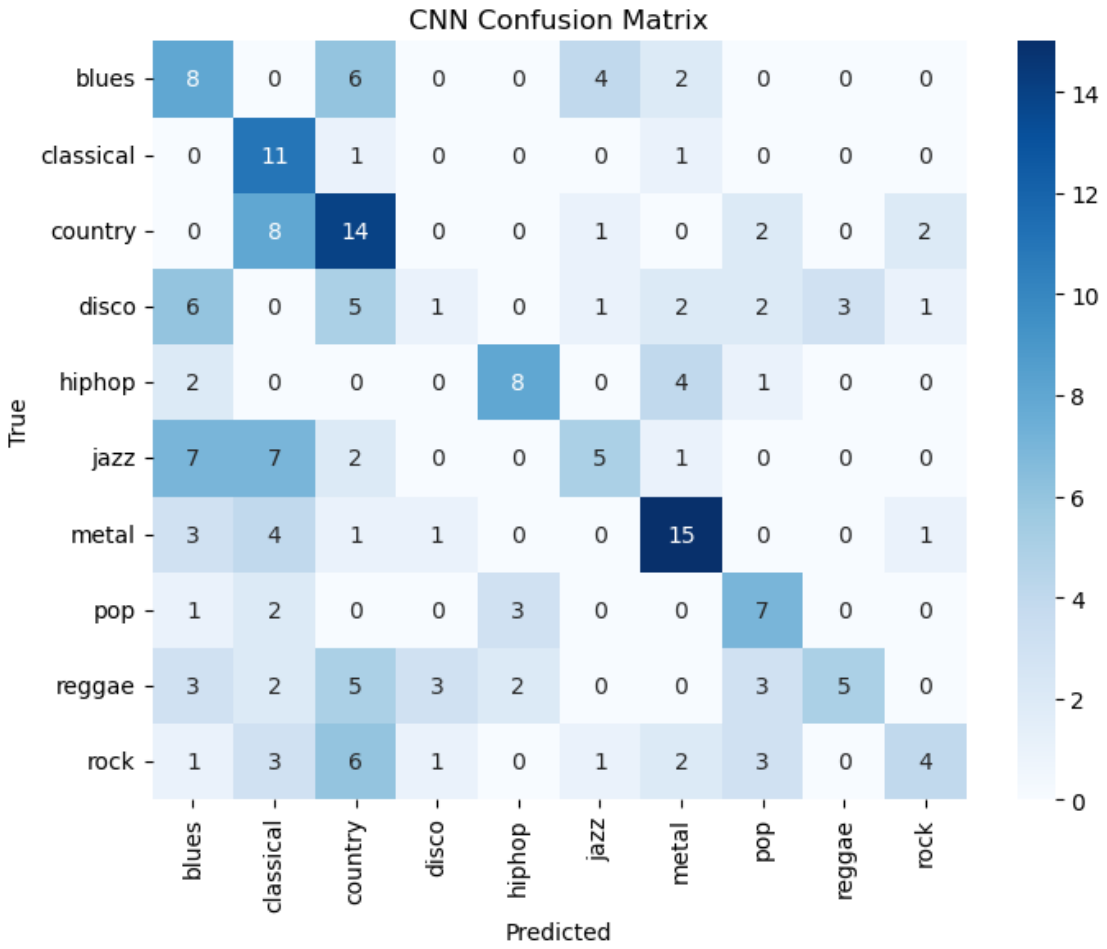
[9]:
```
evaluation = model.evaluate(X_test, y_test)
print(f"Test accuracy: {evaluation[1]:.3f}")
```

```
7/7             4s 574ms/step -
accuracy: 0.3926 - loss: 1.6304
Test accuracy: 0.390
```

4 8 - Apply the confusion matrix after the model

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

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

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

```
7/7                 5s 658ms/step
```



4.1 9 - Limited Genres Easy (metal and classical)

[11]:
```
import os
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split

# Augmentation function
def augment_image(image):
    image = tf.image.random_flip_left_right(image)
    image = tf.image.random_brightness(image, max_delta=0.1)
    image = tf.image.random_contrast(image, 0.8, 1.2)
    return image

# Define the genres and file paths
GENRES = ['classical', 'metal']
FILE_PATH = os.path.join('Data', 'mel_spectrograms', 'mel_spectrogram_32')
X = []
y = []

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

# Loop through the genres and load the images with augmentation
for genre in GENRES:
    genre_dir = os.path.join(FILE_PATH, genre)
    print(f"Going through {genre}")
    for file in os.listdir(genre_dir):
        image = tf.io.read_file(os.path.join(genre_dir, file))
        image = tf.image.decode_png(image, channels=1)
        image = tf.image.convert_image_dtype(image, tf.float32)
        image = tf.image.resize(image, [256, 256])  # Resize to 256x256
        image = augment_image(image)  # Apply augmentation
        image = image.numpy()  # Convert to numpy array
        X.append(image)
        y.append(GENRE_TO_INDEX[genre])

# Convert lists to numpy arrays
X = np.array(X)
y = np.array(y)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

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

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

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

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

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

    Flatten(),

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

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

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

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

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

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

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

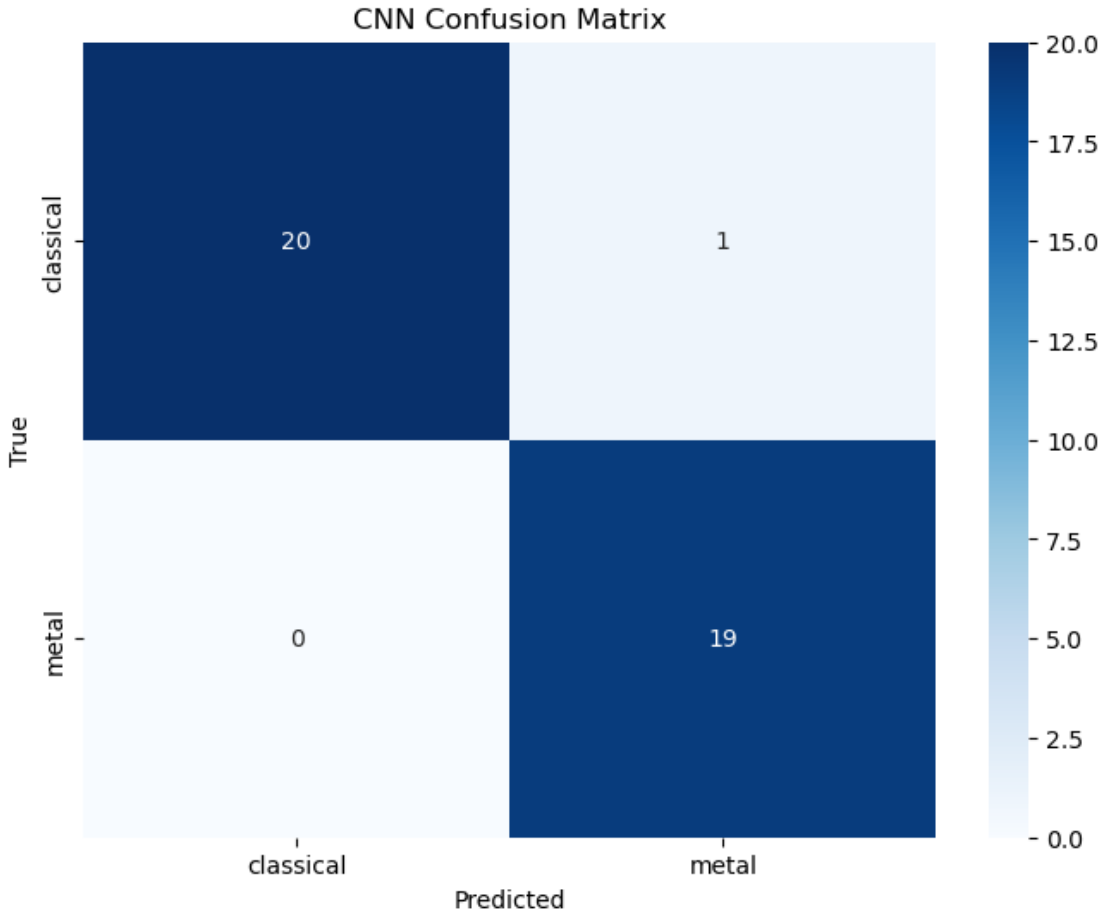
```
Going through classical
Going through metal

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

Epoch 1/20
5/5             23s 3s/step -
accuracy: 0.1472 - loss: 2.2249 - val_accuracy: 0.5250 - val_loss: 1.7616 -
learning_rate: 1.0000e-04
Epoch 2/20
5/5             20s 3s/step -
accuracy: 0.4185 - loss: 1.6901 - val_accuracy: 0.5250 - val_loss: 0.9549 -
learning_rate: 1.0000e-04
Epoch 3/20
5/5             21s 3s/step -
accuracy: 0.4997 - loss: 1.1220 - val_accuracy: 0.9500 - val_loss: 0.6474 -
learning_rate: 1.0000e-04
Epoch 4/20
5/5             15s 3s/step -
accuracy: 0.5453 - loss: 1.1086 - val_accuracy: 0.5500 - val_loss: 0.6328 -
learning_rate: 1.0000e-04
Epoch 5/20
5/5             20s 3s/step -
accuracy: 0.5001 - loss: 1.0571 - val_accuracy: 0.9500 - val_loss: 0.6300 -
learning_rate: 1.0000e-04
Epoch 6/20
5/5             21s 3s/step -
accuracy: 0.5686 - loss: 0.9307 - val_accuracy: 0.8500 - val_loss: 0.6722 -
learning_rate: 1.0000e-04
Epoch 7/20
5/5             21s 3s/step -
accuracy: 0.6301 - loss: 0.7912 - val_accuracy: 0.9500 - val_loss: 0.6510 -
learning_rate: 1.0000e-04
Epoch 8/20
5/5             14s 3s/step -
accuracy: 0.8193 - loss: 0.4390 - val_accuracy: 0.8250 - val_loss: 0.3592 -
learning_rate: 1.0000e-04
Epoch 15/20
5/5             20s 3s/step -
accuracy: 0.8481 - loss: 0.3629 - val_accuracy: 0.9500 - val_loss: 0.2301 -
learning_rate: 1.0000e-04
Epoch 16/20
5/5             12s 2s/step -
accuracy: 0.8777 - loss: 0.3015 - val_accuracy: 0.7750 - val_loss: 0.4160 -
```

```
learning_rate: 1.0000e-04
Epoch 17/20
5/5             23s 3s/step -
accuracy: 0.7872 - loss: 0.4212 - val_accuracy: 0.9250 - val_loss: 0.2505 -
learning_rate: 1.0000e-04
Epoch 18/20
5/5             14s 3s/step -
accuracy: 0.8219 - loss: 0.3761 - val_accuracy: 0.8250 - val_loss: 0.2870 -
learning_rate: 1.0000e-04
Epoch 19/20
5/5             14s 3s/step -
accuracy: 0.9102 - loss: 0.2578 - val_accuracy: 0.9500 - val_loss: 0.2083 -
learning_rate: 5.0000e-05
Epoch 20/20
5/5             14s 3s/step -
accuracy: 0.8877 - loss: 0.3558 - val_accuracy: 0.9750 - val_loss: 0.1807 -
learning_rate: 5.0000e-05
2/2             1s 185ms/step -
accuracy: 0.9729 - loss: 0.1903
Test accuracy: 0.975
```

4.2 10 - Confusion Matrix Easy (metal and classical)

[12]:
```python
import seaborn as sns
# from sklearn.metrics import confusion
import numpy as NP
from sklearn.metrics import confusion_matrix

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

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

```
2/2             1s 533ms/step
```



4.3 11 - Limited genres Hard (disco and pop)

[13]:
```
import os
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split

# Augmentation function
def augment_image(image):
    image = tf.image.random_flip_left_right(image)
    image = tf.image.random_brightness(image, max_delta=0.1)
    image = tf.image.random_contrast(image, 0.8, 1.2)
    return image

# Define the genres and file paths

GENRES = ['disco', 'pop']
FILE_PATH = os.path.join('Data', 'mel_spectrograms', 'mel_spectrogram_32')
```

```
X = []
y = []

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

# Loop through the genres and load the images with augmentation
for genre in GENRES:
    genre_dir = os.path.join(FILE_PATH, genre)
    print(f"Going through {genre}")
    for file in os.listdir(genre_dir):
        image = tf.io.read_file(os.path.join(genre_dir, file))
        image = tf.image.decode_png(image, channels=1)
        image = tf.image.convert_image_dtype(image, tf.float32)
        image = tf.image.resize(image, [256, 256])  # Resize to 256x256
        image = augment_image(image)  # Apply augmentation
        image = image.numpy()  # Convert to numpy array
        X.append(image)
        y.append(GENRE_TO_INDEX[genre])

# Convert lists to numpy arrays
X = np.array(X)
y = np.array(y)

# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

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

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

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

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

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

```
    Flatten(),

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

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

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

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

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

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

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

```
Going through disco
Going through pop
```

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

```
Epoch 1/20
5/5             22s 3s/step -
accuracy: 0.0952 - loss: 2.2424 - val_accuracy: 0.4750 - val_loss: 1.8354 -
learning_rate: 1.0000e-04
Epoch 2/20
5/5             20s 3s/step -
accuracy: 0.3153 - loss: 1.7927 - val_accuracy: 0.5250 - val_loss: 1.0843 -
learning_rate: 1.0000e-04
Epoch 3/20
5/5             21s 3s/step -
accuracy: 0.3611 - loss: 1.3989 - val_accuracy: 0.5250 - val_loss: 0.7397 -
```

```
learning_rate: 1.0000e-04
Epoch 4/20
5/5             14s 3s/step -
accuracy: 0.4973 - loss: 1.1316 - val_accuracy: 0.5250 - val_loss: 0.7082 -
learning_rate: 1.0000e-04
Epoch 5/20
5/5             14s 3s/step -
accuracy: 0.5055 - loss: 1.0738 - val_accuracy: 0.5250 - val_loss: 0.7242 -
learning_rate: 1.0000e-04
Epoch 6/20
5/5             20s 3s/step -
accuracy: 0.4804 - loss: 1.1975 - val_accuracy: 0.5250 - val_loss: 0.7851 -
learning_rate: 1.0000e-04
Epoch 7/20
5/5             21s 3s/step -
accuracy: 0.4980 - loss: 0.9551 - val_accuracy: 0.5250 - val_loss: 0.7897 -
learning_rate: 1.0000e-04
Epoch 8/20
5/5             15s 3s/step -
accuracy: 0.4862 - loss: 0.9490 - val_accuracy: 0.5250 - val_loss: 0.7683 -
learning_rate: 5.0000e-05
Epoch 9/20
5/5             20s 3s/step -
accuracy: 0.5511 - loss: 0.9116 - val_accuracy: 0.5250 - val_loss: 0.7425 -
learning_rate: 5.0000e-05
Epoch 10/20
5/5             11s 2s/step -
accuracy: 0.5001 - loss: 0.9749 - val_accuracy: 0.5250 - val_loss: 0.7227 -
learning_rate: 5.0000e-05
Epoch 11/20
5/5             24s 3s/step -
accuracy: 0.4311 - loss: 0.9582 - val_accuracy: 0.5250 - val_loss: 0.7189 -
learning_rate: 2.5000e-05
Epoch 12/20
5/5             14s 3s/step -
accuracy: 0.5336 - loss: 0.8589 - val_accuracy: 0.5250 - val_loss: 0.7185 -
learning_rate: 2.5000e-05
Epoch 13/20
5/5             14s 3s/step -
accuracy: 0.5051 - loss: 0.8689 - val_accuracy: 0.5250 - val_loss: 0.7165 -
learning_rate: 2.5000e-05
Epoch 14/20
5/5             14s 3s/step -
accuracy: 0.5437 - loss: 0.8528 - val_accuracy: 0.5250 - val_loss: 0.7151 -
learning_rate: 1.2500e-05
Epoch 15/20
5/5             12s 2s/step -
accuracy: 0.5268 - loss: 0.8150 - val_accuracy: 0.5250 - val_loss: 0.7147 -
```

```
learning_rate: 1.2500e-05
Epoch 16/20
5/5             22s 3s/step -
accuracy: 0.4775 - loss: 0.9822 - val_accuracy: 0.5250 - val_loss: 0.7161 -
learning_rate: 1.2500e-05
Epoch 17/20
5/5             13s 3s/step -
accuracy: 0.5513 - loss: 0.8773 - val_accuracy: 0.5250 - val_loss: 0.7167 -
learning_rate: 6.2500e-06
Epoch 18/20
5/5             20s 3s/step -
accuracy: 0.5411 - loss: 0.8141 - val_accuracy: 0.5250 - val_loss: 0.7173 -
learning_rate: 6.2500e-06
Epoch 19/20
5/5             11s 2s/step -
accuracy: 0.6277 - loss: 0.7813 - val_accuracy: 0.5250 - val_loss: 0.7172 -
learning_rate: 6.2500e-06
Epoch 20/20
5/5             11s 2s/step -
accuracy: 0.4694 - loss: 0.9221 - val_accuracy: 0.5250 - val_loss: 0.7173 -
learning_rate: 3.1250e-06
2/2             1s 237ms/step -
accuracy: 0.5375 - loss: 0.7127
Test accuracy: 0.525
```

4.4 12 - Confusion Matrix Hard (disco and pop)

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

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

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

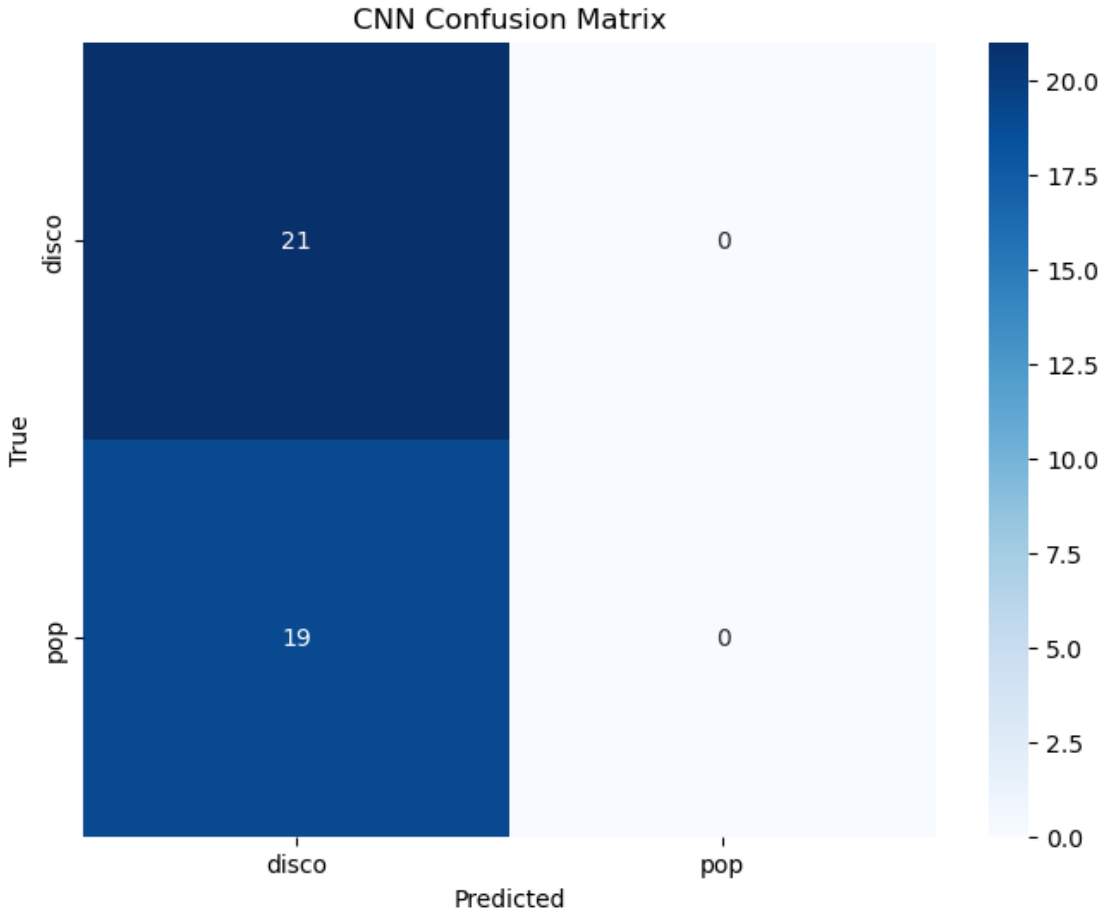
```
WARNING:tensorflow:5 out of the last 10 calls to <function
TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_distributed at
0x7fca884a6340> triggered tf.function retracing. Tracing is expensive and the
excessive number of tracings could be due to (1) creating @tf.function
```

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 0x7fca884a6340> 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 990ms/step
```



4.5 13 - Limited Genres Medium (5 random)

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

# Augmentation function
def augment_image(image):
    image = tf.image.random_flip_left_right(image)
    image = tf.image.random_brightness(image, max_delta=0.1)
    image = tf.image.random_contrast(image, 0.8, 1.2)
    return image

# Define the genres and file paths
GENRES = ['blues', 'classical', 'country', 'disco', 'hiphop', 'jazz', 'metal',␣
 ↪'pop', 'reggae', 'rock']
GENRES = random.sample(GENRES, 5)
print(GENRES)

FILE_PATH = os.path.join('Data', 'mel_spectrograms', 'mel_spectrogram_32')
X = []
y = []

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

# Loop through the genres and load the images with augmentation
for genre in GENRES:
    genre_dir = os.path.join(FILE_PATH, genre)
    print(f"Going through {genre}")
    for file in os.listdir(genre_dir):
        image = tf.io.read_file(os.path.join(genre_dir, file))
        image = tf.image.decode_png(image, channels=1)
        image = tf.image.convert_image_dtype(image, tf.float32)
        image = tf.image.resize(image, [256, 256])  # Resize to 256x256
        image = augment_image(image)  # Apply augmentation
        image = image.numpy()  # Convert to numpy array
        X.append(image)
        y.append(GENRE_TO_INDEX[genre])

# Convert lists to numpy arrays
X = np.array(X)
y = np.array(y)

# Split the data into training and testing sets
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

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

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

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

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

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

    Flatten(),

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

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

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

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

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

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

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

```
['hiphop', 'blues', 'classical', 'jazz', 'pop']
Going through hiphop
Going through blues
Going through classical
Going through jazz
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             44s 3s/step -
accuracy: 0.2146 - loss: 2.1531 - val_accuracy: 0.1400 - val_loss: 1.8342 -
learning_rate: 1.0000e-04
Epoch 2/20
13/13             34s 2s/step -
accuracy: 0.1739 - loss: 1.9244 - val_accuracy: 0.1400 - val_loss: 1.7228 -
learning_rate: 1.0000e-04
Epoch 3/20
13/13             29s 2s/step -
accuracy: 0.2617 - loss: 1.8506 - val_accuracy: 0.1700 - val_loss: 1.7450 -
learning_rate: 1.0000e-04
Epoch 4/20
13/13             28s 2s/step -
accuracy: 0.2629 - loss: 1.8225 - val_accuracy: 0.1400 - val_loss: 1.6690 -
learning_rate: 1.0000e-04
Epoch 5/20
13/13             33s 3s/step -
accuracy: 0.2886 - loss: 1.7340 - val_accuracy: 0.1400 - val_loss: 1.6657 -
learning_rate: 1.0000e-04
Epoch 6/20
13/13             37s 2s/step -
accuracy: 0.2870 - loss: 1.6985 - val_accuracy: 0.1500 - val_loss: 1.6433 -
learning_rate: 1.0000e-04
Epoch 7/20
13/13             32s 2s/step -
accuracy: 0.2648 - loss: 1.6549 - val_accuracy: 0.1500 - val_loss: 1.6541 -
learning_rate: 1.0000e-04
Epoch 8/20
13/13             43s 3s/step -
accuracy: 0.2876 - loss: 1.7098 - val_accuracy: 0.2500 - val_loss: 1.6407 -
learning_rate: 1.0000e-04
```

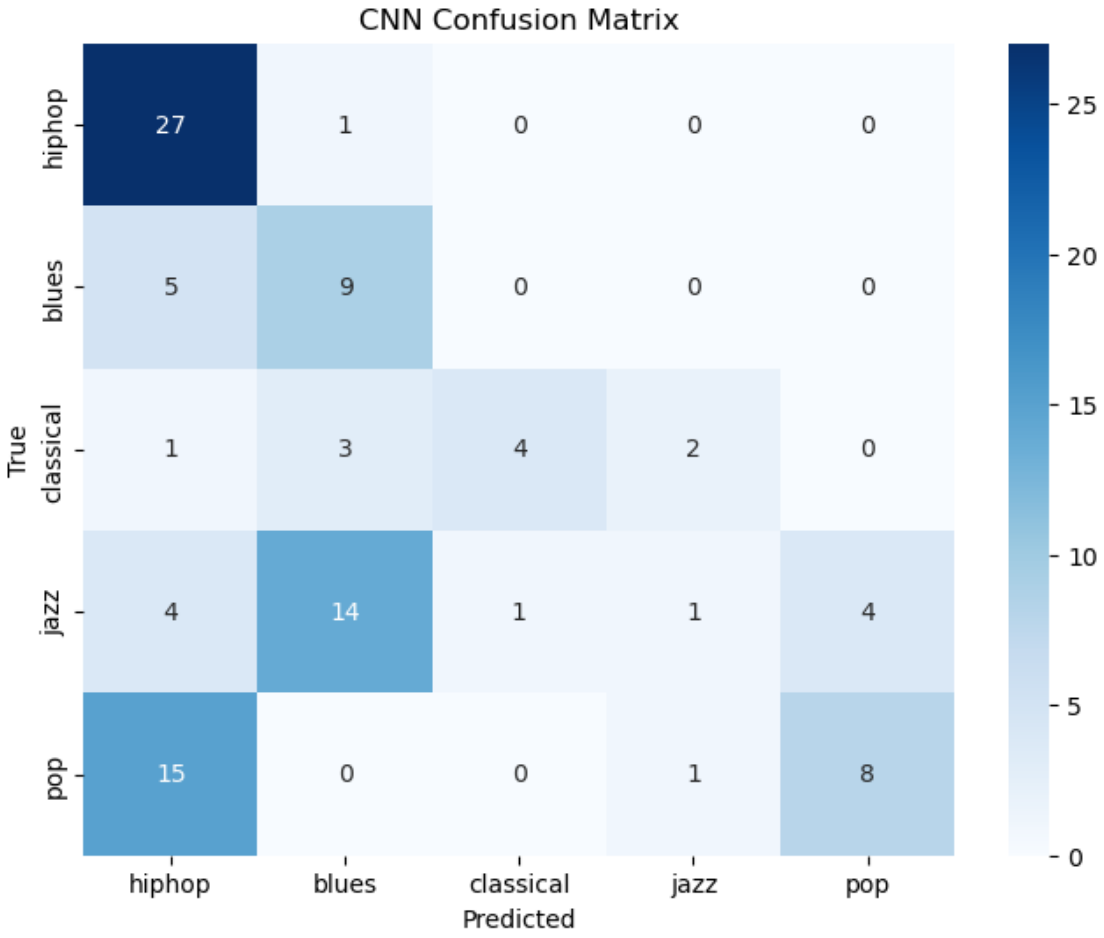
```
Epoch 9/20
13/13             40s 3s/step -
accuracy: 0.2835 - loss: 1.6349 - val_accuracy: 0.4000 - val_loss: 1.5025 -
learning_rate: 1.0000e-04
Epoch 10/20
13/13             46s 3s/step -
accuracy: 0.2688 - loss: 1.5492 - val_accuracy: 0.4000 - val_loss: 1.5402 -
learning_rate: 1.0000e-04
Epoch 11/20
13/13             41s 3s/step -
accuracy: 0.3915 - loss: 1.4347 - val_accuracy: 0.4400 - val_loss: 1.3902 -
learning_rate: 1.0000e-04
Epoch 12/20
13/13             40s 3s/step -
accuracy: 0.3615 - loss: 1.4412 - val_accuracy: 0.4700 - val_loss: 1.4123 -
learning_rate: 1.0000e-04
Epoch 13/20
13/13             34s 3s/step -
accuracy: 0.3856 - loss: 1.3626 - val_accuracy: 0.4600 - val_loss: 1.2885 -
learning_rate: 1.0000e-04
Epoch 14/20
13/13             46s 3s/step -
accuracy: 0.4719 - loss: 1.2948 - val_accuracy: 0.5800 - val_loss: 1.2279 -
learning_rate: 1.0000e-04
Epoch 15/20
13/13             39s 3s/step -
accuracy: 0.4676 - loss: 1.2074 - val_accuracy: 0.5100 - val_loss: 1.1995 -
learning_rate: 1.0000e-04
Epoch 16/20
13/13             32s 2s/step -
accuracy: 0.5400 - loss: 1.1320 - val_accuracy: 0.5300 - val_loss: 1.1308 -
learning_rate: 1.0000e-04
Epoch 17/20
13/13             35s 3s/step -
accuracy: 0.5496 - loss: 1.0961 - val_accuracy: 0.5900 - val_loss: 1.0614 -
learning_rate: 1.0000e-04
Epoch 18/20
13/13             43s 3s/step -
accuracy: 0.5570 - loss: 1.0951 - val_accuracy: 0.6500 - val_loss: 1.0640 -
learning_rate: 1.0000e-04
Epoch 19/20
13/13             32s 2s/step -
accuracy: 0.6150 - loss: 0.9955 - val_accuracy: 0.5600 - val_loss: 1.0584 -
learning_rate: 1.0000e-04
Epoch 20/20
13/13             48s 3s/step -
accuracy: 0.6067 - loss: 1.0018 - val_accuracy: 0.4900 - val_loss: 1.1657 -
learning_rate: 1.0000e-04
```

```
4/4             2s 416ms/step -
accuracy: 0.4887 - loss: 1.1907
Test accuracy: 0.490
```

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()
```

```
4/4             2s 460ms/step
```


CNN Confusion Matrix
True
Predicted
hiphop
blues
classical
jazz
pop
27 1 0 0 0
5 9 0 0 0
1 3 4 2 0
4 14 1 1 4
15 0 0 1 8
25
20
15
10
5
0