convolutional_neural_network

August 22, 2024

[1]:
```python
# Base
import librosa # alternativa pyAudioAnalysis ali audioFlux
import numpy as np
import os
import h5py
import time
import datetime
from scipy import signal
import matplotlib.pyplot as plt
import sys

# Preprocessing, Metrics
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score

# Keras, Classification
import keras
from keras import models
from keras import layers
from sklearn.svm import SVC
import tensorflow as tf
from keras.callbacks import EarlyStopping, ModelCheckpoint
from sklearn.metrics import confusion_matrix
from keras.utils import to_categorical

# Try using OpenL3 on 5 second part of audio signal
import openl3

# Parameters
genres = np.array('pop rock classical blues country disco metal jazz reggae␣
↪hiphop'.split())
n_genres = len(genres)
n_genres_files = 100
embedding_size = 512
n_windows = 46 # For 5 second signal chunk - number of timestamps
n_parts_sig = 6   # signali so dolgi 30 sekund, usak del mora trajati 5 sekund
```

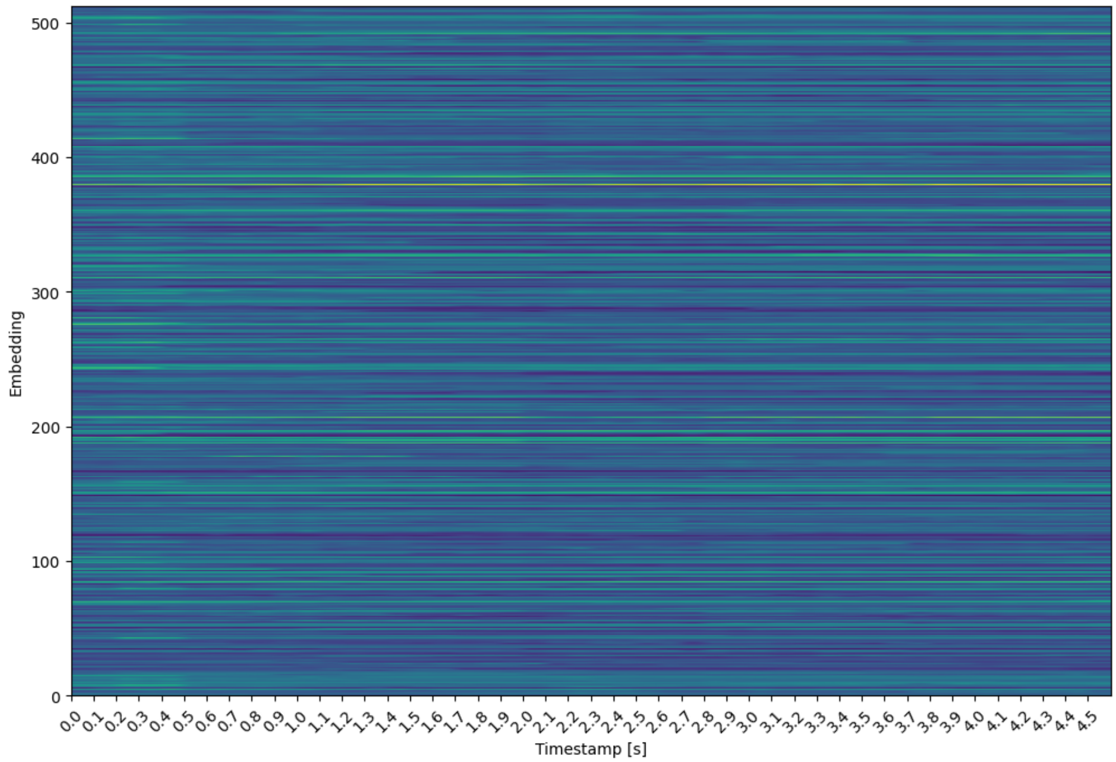
```
# np.set_printoptions(threshold=sys.maxsize)
```

[ ]:
```
# OpenL3
# pip install OpenL3

# - model za izdvajanje značilnic zvoka naučen na AudioSet podatkovni bazi
# - deluje nad melovim spektrogramom s 128 ali 256 območji (filtri), lahko␣
↪uporabimo tudi linearno različico - klasični spektrogram
#   - "linear", "mel128", "mel256"
# - model je naučen nad dvema tipoma podatkov (glasba in okolje)
#   - "env", "music"
# - model vrne 512 ali 6144 značilnic za posamezno obravnavano okno
# - model privzeto uporablja širino okna 1 sekundo (na začetek doda 0.5␣
↪sekundni zero-pad) - časovni trenutki so poravnani na sredino okna
# - korak okna je privzeto 0.1 sekunde

# emb, ts = get_audio_embedding(audio, sr, model=None, input_repr=None,␣
↪content_type="music", embedding_size=6144,
#                              center=True, hop_size=0.1, batch_size=32,␣
↪frontend="kapre", verbose=True)

# emb - polje značilnic za vsa okna
# ts - časovni trenutki oken
```

[2]:
```
# Load file
fn = f'./genres/rock/rock.00000.wav'
sig, sr = librosa.load(fn, mono=True, duration=5)

# Use OpenL3
emb, ts = openl3.get_audio_embedding(sig, sr, content_type="music",␣
↪input_repr="mel256", embedding_size=512, verbose=False)

# Shapes
print('Embedding size:', np.shape(emb))
print('Timestamps size:', np.shape(ts))

# Plot
plt.figure(figsize=(12,8))
plt.imshow(emb.T, extent=[0, len(ts), 0, 512], aspect='auto') # Transpose emb␣
↪for better view (time on X-axis)
plt.xlabel("Timestamp [s]")
plt.ylabel("Embedding")
plt.xticks(np.arange(0, 46, step=1), np.round(ts, 2), rotation=45)
plt.show()
```

```
Embedding size: (46, 512)
```

Timestamps size: (46,)



[ ]:
```
# Parameters
genres = np.array('pop rock classical blues country disco metal jazz reggae␣
↪hiphop'.split())
n_genres = len(genres)
n_genres_files = 100
embedding_size = 512
n_windows = 46 # For 5 second signal chunk - number of timestamps
n_parts_sig = 6   # signali so dolgi 30 sekund, vsak del mora trajati 5 sekund

# [NumberOfSignalParts,NumberOfWindows,NumberOfFeatures]
data = np.zeros((n_genres * n_genres_files * n_parts_sig, n_windows,␣
↪embedding_size))

# [NumberOfSignalParts,1]
data_labels = np.zeros((n_genres * n_genres_files * n_parts_sig, 1))

# Dataset - Will take some time to generate
data_index = 0
for i_genre in range(0, n_genres):
```

```
    for filename in os.listdir(f'C:
↪\\Users\\Viktorija\\Desktop\\ROSIS\\N8\\Data\\genres_original\\{genres[i_genre]}'):
↪
        fn = f'C:
↪\\Users\\Viktorija\\Desktop\\ROSIS\\N8\\Data\\genres_original\\{genres[i_genre]}\\{filename

        # There is one problematic file - format problem (can try ffmpeg␣
↪decoder)
        try:
            # Load file (sig-signal; sr-sampling rate)
            sig, sr = librosa.load(fn, mono=True, duration=30)

            # For demo we will only use first 5 seconds of audio
            # Change this!
            # Read at least 5-6 chunks of certain length - have to modify␣
↪n_windows
            # Maybe read 30 seconds of audio and divide into 6 parts by 5␣
↪seconds
            # Be careful - the size of training data defines later usage

            for i in range(0, len(sig), sr * 5): #sampling rate (of 1 sec) * 5␣
↪seconds

                part = sig[i : (i + sr * 5)]

                emb, _ = openl3.get_audio_embedding(part, sr,␣
↪content_type="music", input_repr="mel256", embedding_size=512, verbose=0)

                # Features - Data
                data[data_index, :] = emb

                # Genre - Label
                data_labels[data_index] = i_genre

                data_index = data_index + 1
        except:
            pass

# Save to h5 file
hf = h5py.File('dataset_openl3.h5', 'w')
hf.create_dataset('data', data=data)
hf.create_dataset('data_labels', data=data_labels)
hf.close()
```

```
WARNING:tensorflow:5 out of the last 9 calls to <function
Model.make_predict_function.<locals>.predict_function at 0x000001901D5AF280>
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.
WARNING:tensorflow:6 out of the last 11 calls to <function
Model.make_predict_function.<locals>.predict_function at 0x000001906CF1ECA0>
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.
```

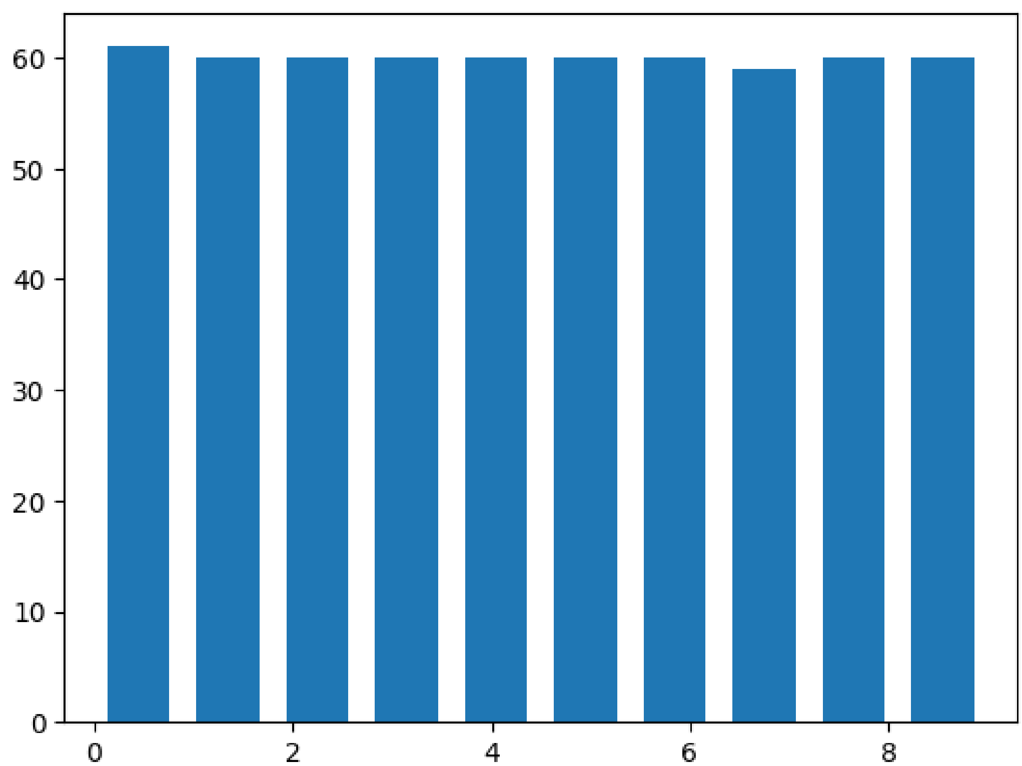
[3]:

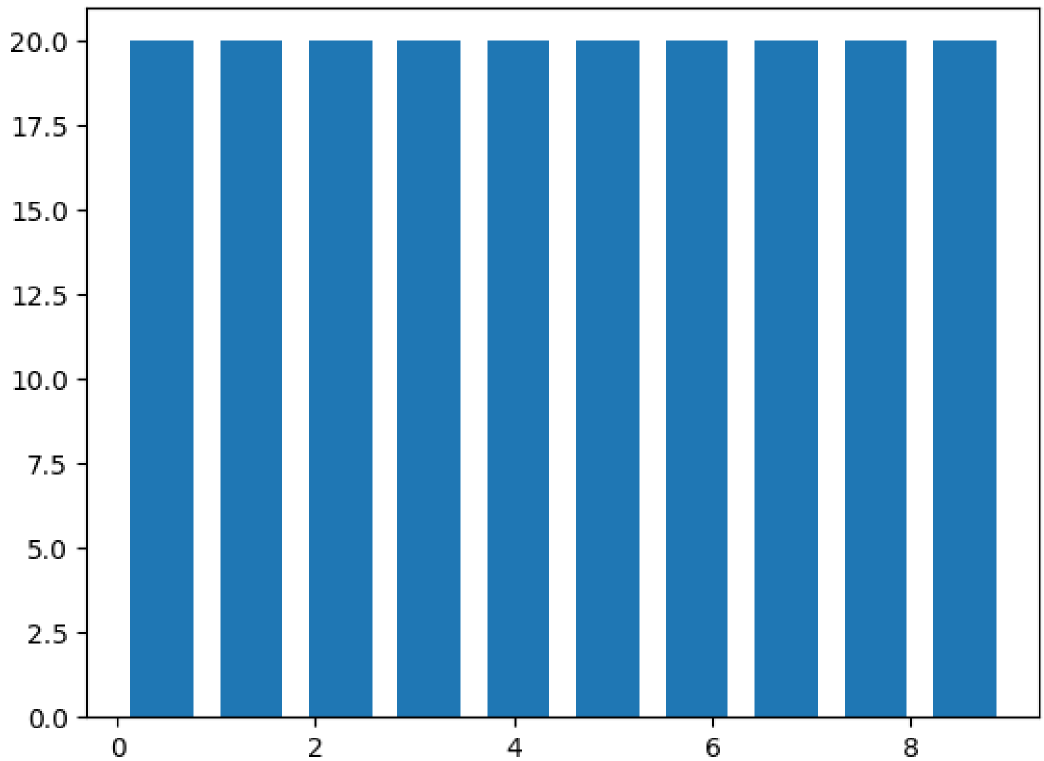
```
# Load dataset from h5 file
hf = h5py.File('dataset_openl3.h5', 'r')

data = hf.get('data')
data = np.array(data)

data_labels = hf.get('data_labels')
data_labels = np.array(data_labels)

hf.close()
```

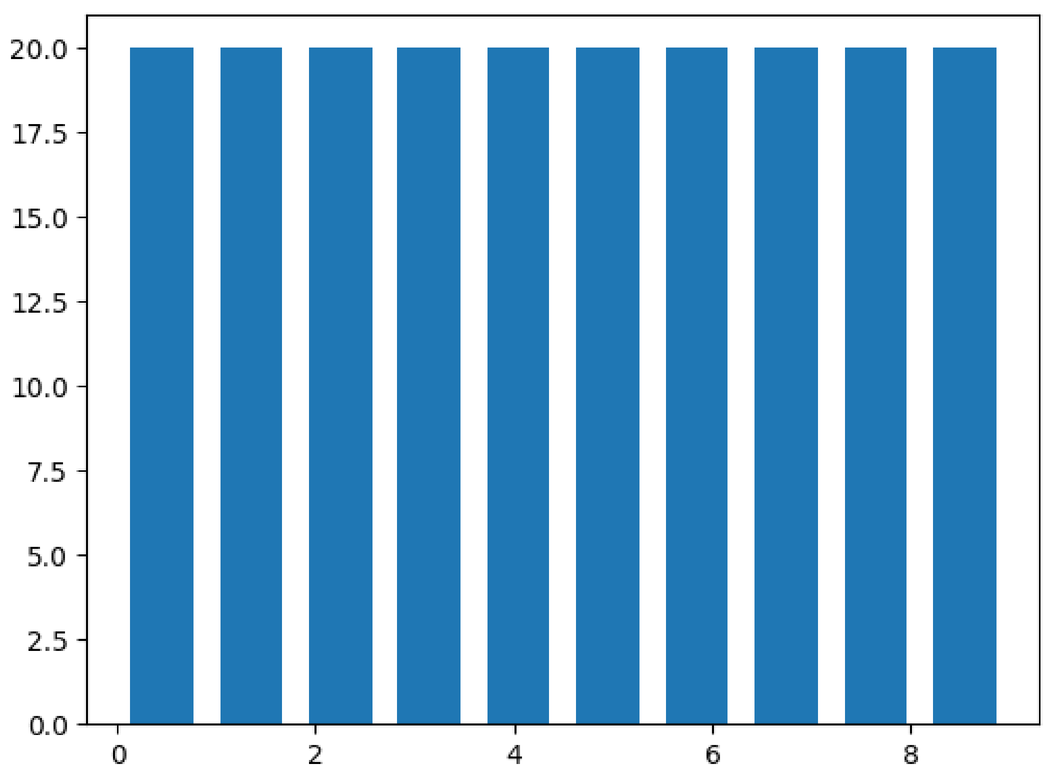
[4]:

```
# Normalize
scaler = StandardScaler()

x = np.reshape(data, newshape=(data.shape[0]*data.shape[1], data.shape[2]))
X = np.reshape(scaler.fit_transform(np.array(x, dtype = float)), newshape=data.
 ↪shape)

# Split into test and train
# Why stratify=data_labels?
# Check the histograms, try removing stratify

X_train, X_test, y_train, y_test = train_test_split(X, data_labels, test_size=0.
 ↪2, stratify=data_labels)

# Split into train and valid
# Why stratify=y_train?
# Check the histograms, try removing stratify
```

```
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train, test_size=0.
    ↪25, stratify=y_train)

X_train = np.expand_dims(X_train, 3)
X_test = np.expand_dims(X_test, 3)
X_val = np.expand_dims(X_val, 3)

plt.hist(y_train, bins=n_genres, rwidth=0.7)
plt.show()
plt.hist(y_test, bins=n_genres, rwidth=0.7)
plt.show()
plt.hist(y_val, bins=n_genres, rwidth=0.7)
plt.show()
```




20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
2
4
6
8


20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
0
2
4
6
8

[7]:
```
# Fix the model - add extra layers, change the number of neurons, number of␣
↪filters, etc...

# NN model
model = models.Sequential()

model.add(layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu',␣
↪input_shape=X_train.shape[1:]))
model.add(layers.Conv2D(filters=32, kernel_size=(3, 3), activation='relu'))
model.add(layers.MaxPool2D(pool_size=(2, 2)))
model.add(layers.Dropout(rate=0.25))

model.add(layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(layers.Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(layers.MaxPool2D(pool_size=(2, 2)))
model.add(layers.Dropout(rate=0.25))

model.add(layers.Conv2D(filters=128, kernel_size=(3, 3), activation='relu'))
model.add(layers.Conv2D(filters=128, kernel_size=(3, 3), activation='relu'))
model.add(layers.MaxPool2D(pool_size=(2, 2)))
model.add(layers.Dropout(rate=0.25))

model.add(layers.Flatten())

model.add(layers.Dense(256, activation='relu'))
model.add(layers.Dropout(rate=0.5))
model.add(layers.Dense(n_genres))
```

[8]:
```
opt = keras.optimizers.Adam(learning_rate=0.0001)
loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True) #␣
↪Computes the crossentropy loss between the labels and predictions
metr = keras.metrics.SparseCategoricalAccuracy() # Calculates how often␣
↪predictions match integer labels
model.compile(optimizer=opt, loss=loss, metrics=metr)

# model.summary()
```

[9]:
```
# Stopping criterion to avoid overfitting
# patience: Number of epochs with no improvement after which training will be␣
↪stopped.
early_stopping = EarlyStopping(monitor='val_loss', patience=5)

# Save best weights
```

```
model_checkpoint = ModelCheckpoint("weights_openl3.h5", save_best_only=True,␣
↪save_weights_only=True)

# Train
t_epochs = 50 # Needs to be tuned
b_size = 32 # Needs to be tuned as well - What is batch_size?
history = model.fit(X_train, y_train, validation_data=(X_val, y_val),␣
↪epochs=t_epochs, batch_size=b_size,
                    callbacks=[early_stopping, model_checkpoint])

# Load best weights
model.load_weights("weights_openl3.h5")
```

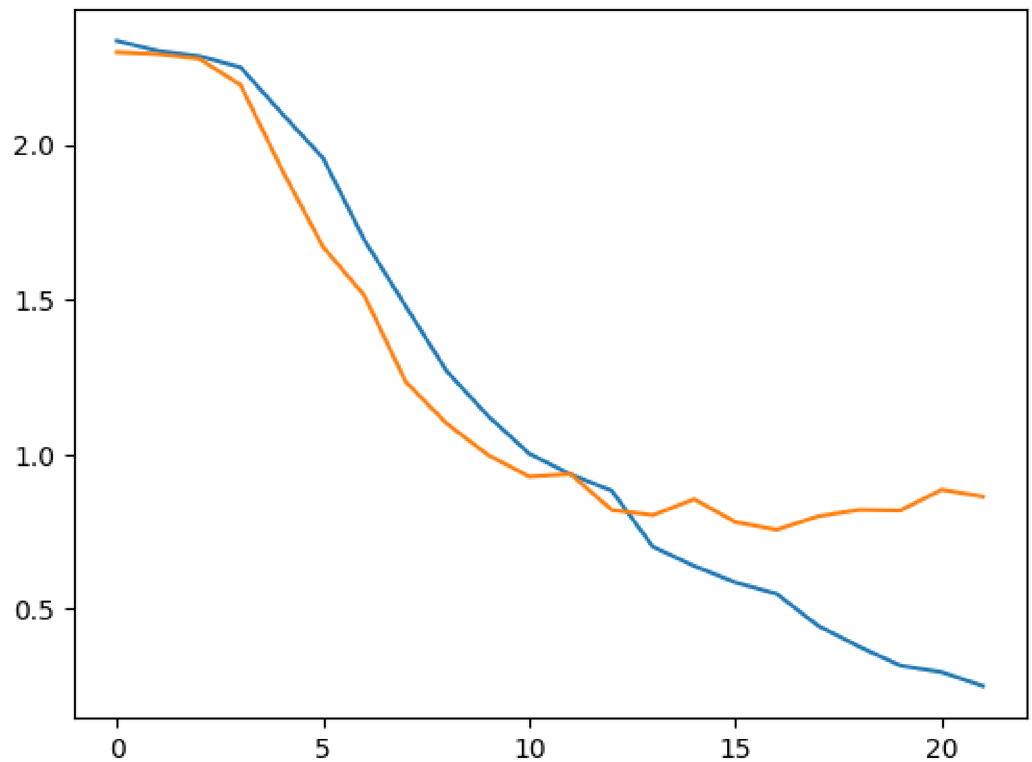
```
Epoch 1/50
19/19 [==============================] - 28s 1s/step - loss: 2.3359 -
sparse_categorical_accuracy: 0.0850 - val_loss: 2.2993 -
val_sparse_categorical_accuracy: 0.1050
Epoch 2/50
19/19 [==============================] - 27s 1s/step - loss: 2.3031 -
sparse_categorical_accuracy: 0.1117 - val_loss: 2.2929 -
val_sparse_categorical_accuracy: 0.1500
Epoch 3/50
19/19 [==============================] - 28s 1s/step - loss: 2.2865 -
sparse_categorical_accuracy: 0.1667 - val_loss: 2.2777 -
val_sparse_categorical_accuracy: 0.2450
Epoch 4/50
19/19 [==============================] - 28s 1s/step - loss: 2.2503 -
sparse_categorical_accuracy: 0.1783 - val_loss: 2.1932 -
val_sparse_categorical_accuracy: 0.3350
Epoch 5/50
19/19 [==============================] - 34s 2s/step - loss: 2.1015 -
sparse_categorical_accuracy: 0.2783 - val_loss: 1.9199 -
val_sparse_categorical_accuracy: 0.4200
Epoch 6/50
19/19 [==============================] - 38s 2s/step - loss: 1.9569 -
sparse_categorical_accuracy: 0.3333 - val_loss: 1.6699 -
val_sparse_categorical_accuracy: 0.5100
Epoch 7/50
19/19 [==============================] - 31s 2s/step - loss: 1.6924 -
sparse_categorical_accuracy: 0.4433 - val_loss: 1.5128 -
val_sparse_categorical_accuracy: 0.4950
Epoch 8/50
19/19 [==============================] - 29s 2s/step - loss: 1.4789 -
sparse_categorical_accuracy: 0.4600 - val_loss: 1.2345 -
val_sparse_categorical_accuracy: 0.6150
Epoch 9/50
19/19 [==============================] - 29s 2s/step - loss: 1.2678 -
```

```
sparse_categorical_accuracy: 0.5567 - val_loss: 1.0985 -
val_sparse_categorical_accuracy: 0.6300
Epoch 10/50
19/19 [==============================] - 30s 2s/step - loss: 1.1234 -
sparse_categorical_accuracy: 0.5983 - val_loss: 0.9973 -
val_sparse_categorical_accuracy: 0.6600
Epoch 11/50
19/19 [==============================] - 29s 2s/step - loss: 1.0015 -
sparse_categorical_accuracy: 0.6600 - val_loss: 0.9275 -
val_sparse_categorical_accuracy: 0.6750
Epoch 12/50
19/19 [==============================] - 28s 1s/step - loss: 0.9344 -
sparse_categorical_accuracy: 0.6817 - val_loss: 0.9357 -
val_sparse_categorical_accuracy: 0.7000
Epoch 13/50
19/19 [==============================] - 29s 2s/step - loss: 0.8819 -
sparse_categorical_accuracy: 0.7100 - val_loss: 0.8198 -
val_sparse_categorical_accuracy: 0.7250
Epoch 14/50
19/19 [==============================] - 29s 2s/step - loss: 0.7005 -
sparse_categorical_accuracy: 0.7750 - val_loss: 0.8032 -
val_sparse_categorical_accuracy: 0.7200
Epoch 15/50
19/19 [==============================] - 28s 1s/step - loss: 0.6383 -
sparse_categorical_accuracy: 0.7783 - val_loss: 0.8541 -
val_sparse_categorical_accuracy: 0.6950
Epoch 16/50
19/19 [==============================] - 29s 2s/step - loss: 0.5855 -
sparse_categorical_accuracy: 0.8050 - val_loss: 0.7807 -
val_sparse_categorical_accuracy: 0.7200
Epoch 17/50
19/19 [==============================] - 29s 2s/step - loss: 0.5484 -
sparse_categorical_accuracy: 0.8200 - val_loss: 0.7562 -
val_sparse_categorical_accuracy: 0.7300
Epoch 18/50
19/19 [==============================] - 29s 2s/step - loss: 0.4457 -
sparse_categorical_accuracy: 0.8433 - val_loss: 0.7988 -
val_sparse_categorical_accuracy: 0.7300
Epoch 19/50
19/19 [==============================] - 32s 2s/step - loss: 0.3776 -
sparse_categorical_accuracy: 0.8700 - val_loss: 0.8203 -
val_sparse_categorical_accuracy: 0.7250
Epoch 20/50
19/19 [==============================] - 37s 2s/step - loss: 0.3165 -
sparse_categorical_accuracy: 0.8850 - val_loss: 0.8182 -
val_sparse_categorical_accuracy: 0.7400
Epoch 21/50
19/19 [==============================] - 36s 2s/step - loss: 0.2953 -
```

```
sparse_categorical_accuracy: 0.8950 - val_loss: 0.8847 -
val_sparse_categorical_accuracy: 0.7350
Epoch 22/50
19/19 [==============================] - 35s 2s/step - loss: 0.2506 -
sparse_categorical_accuracy: 0.9100 - val_loss: 0.8628 -
val_sparse_categorical_accuracy: 0.7250
```

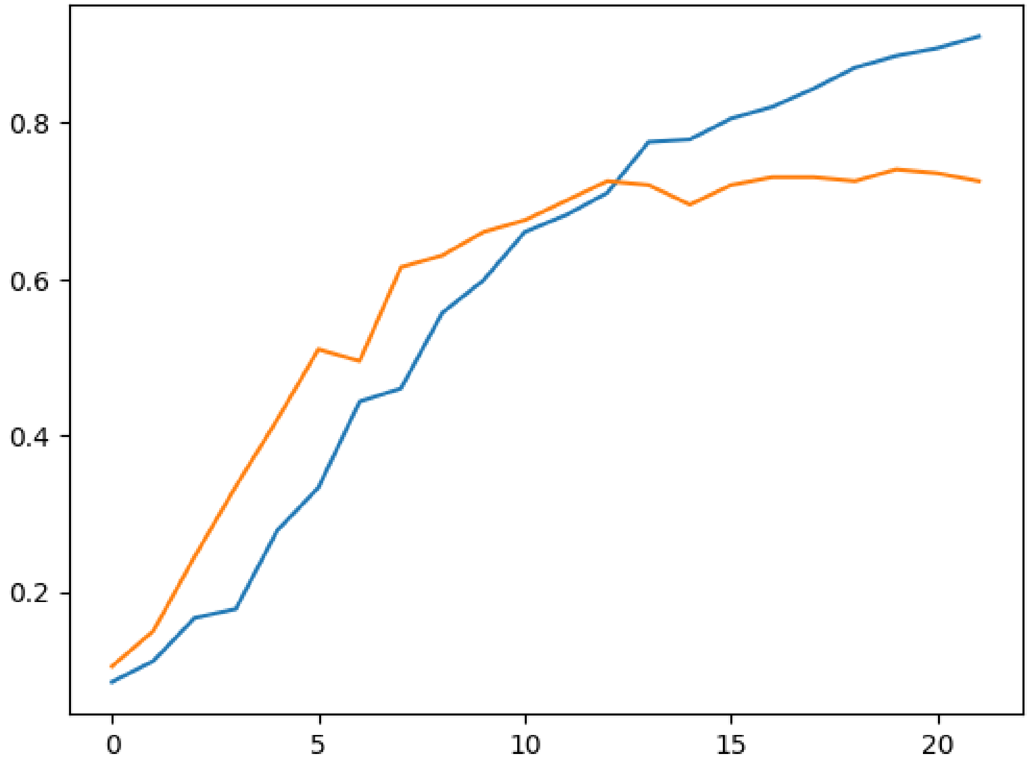
[10]:
```python
# Lets observe the loss metric on both the training (blue) and validation
↪(orange) set
# What do we noice?
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
```

[10]: [<matplotlib.lines.Line2D at 0x22108ac9910>]



[11]:
```python
plt.plot(history.history['sparse_categorical_accuracy'])
plt.plot(history.history['val_sparse_categorical_accuracy'])
```

[11]: [<matplotlib.lines.Line2D at 0x221081ad100>]



[12]:
```
# Now to evaluate our model on train and test data

# Train NN
loss, acc = model.evaluate(X_train, y_train, verbose=0)
print('Acc train NN: %.3f' % acc)

# Test NN
loss, acc = model.evaluate(X_test, y_test, verbose=0)
print('Acc test NN: %.3f' % acc)

# Val NN
loss, acc = model.evaluate(X_val, y_val, verbose=0)
print('Acc val NN: %.3f' % acc)
```

```
Acc train NN: 0.948
Acc test NN: 0.735
Acc val NN: 0.730
```

[13]:
```
# Test NN
# Predictions for additional analysis
predictions = model.predict(X_test)
```

```
# Confusion matrix
predicted_labels = np.argmax(predictions, axis=1)
conf = confusion_matrix(y_test, predicted_labels, normalize="pred") # Normalize␣
 ↪pred! Explain why?

# Visualise confusion matrix
plt.imshow(conf)
plt.ylabel("Actual")
plt.xlabel("Predicted")
plt.yticks(np.arange(n_genres), genres)
plt.xticks(np.arange(n_genres), genres, rotation='vertical')
plt.colorbar()
```

```
7/7 [==============================] - 2s 236ms/step
```

[13]:

```
<matplotlib.colorbar.Colorbar at 0x22106d33df0>
```