multilayer_perceptron

August 22, 2024

[1]:

```
# 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

# 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

# Parameters
genres = np.array('pop rock classical blues country disco metal jazz reggae␣
↪hiphop'.split())
n_genres = len(genres)
n_genres_files = 100  # število datotek v posamezni mapi žanra
n_features = 6
n_mfcc_coef = 20
n_parts_sig = 6   # signali so dolgi 30 sekund, vsak del mora trajati 5 sekund
```

[2]:

```
def extract_features(y, sr, n_features, n_mfcc_coef, n_fft=512, hop_length=160,␣
↪window=signal.windows.hamming(512), fmin=300, fmax=8000):
    vect = np.zeros(n_features + n_mfcc_coef)
```

```
    #https://devopedia.org/audio-feature-extraction#qst-ans-4

    #zero_crossing_rate šteje, kako pogosto signal prečka x-os
    #tempo ocenjuje število udarcev na minuto
    #tempogram meri, kako se tempo spreminja skozi čas
    #rms izračuna energijo signala
    #spectral_centroid izračuna frekvenčni pas, v katerem je skoncentrirana␣
↪večina energije
    #spectral_bandwidth podaja varianco od spektralnega_centroida

    #y and sr are sampled data and sampling rate, respectively
    #n_fft is the number of the fft coefficients
    #hop_length is the distance between two frames
    #window is the used window function (Hamming)
    #fmin and fmax are minimum and maximum frequencies

    vect[0] = np.mean(librosa.feature.zero_crossing_rate(y))
    vect[1] = np.mean(librosa.feature.tempo(y=y, sr=sr))
    vect[2] = np.mean(librosa.feature.tempogram(y=y, sr=sr))
    vect[3] = np.mean(librosa.feature.rms(y=y))
    vect[4] = np.mean(librosa.feature.spectral_centroid(y=y, sr=sr))
    vect[5] = np.mean(librosa.feature.spectral_bandwidth(y=y, sr=sr))

    # MFCC
    # Can use Kapre (https://github.com/keunwoochoi/kapre) GPU
    mfcc = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=n_mfcc_coef)

    for i in range(0, n_mfcc_coef):
        vect[i + n_features] = np.mean(mfcc[i])

    return vect
```

[8]:
```
# Dataset - Will take some time to generate
data = np.zeros((n_genres * n_genres_files * n_parts_sig, n_features +␣
↪n_mfcc_coef))
data_labels = np.zeros((n_genres * n_genres_files * n_parts_sig, 1))

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)

            # Split signals into smaller chunks
            for y in np.split(sig, n_parts_sig):

                # Features - Data
                data[data_index, 0:n_features + n_mfcc_coef] =␣
↪extract_features(sig, sr, n_features, n_mfcc_coef)

                # Genre - Label
                data_labels[data_index] = i_genre

                data_index = data_index + 1
        except:
            pass

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

```
C:\Users\Viktorija\AppData\Local\Temp\ipykernel_13656\1276745270.py:13:
UserWarning: PySoundFile failed. Trying audioread instead.
  sig, sr = librosa.load(fn, mono=True, duration=30)
```

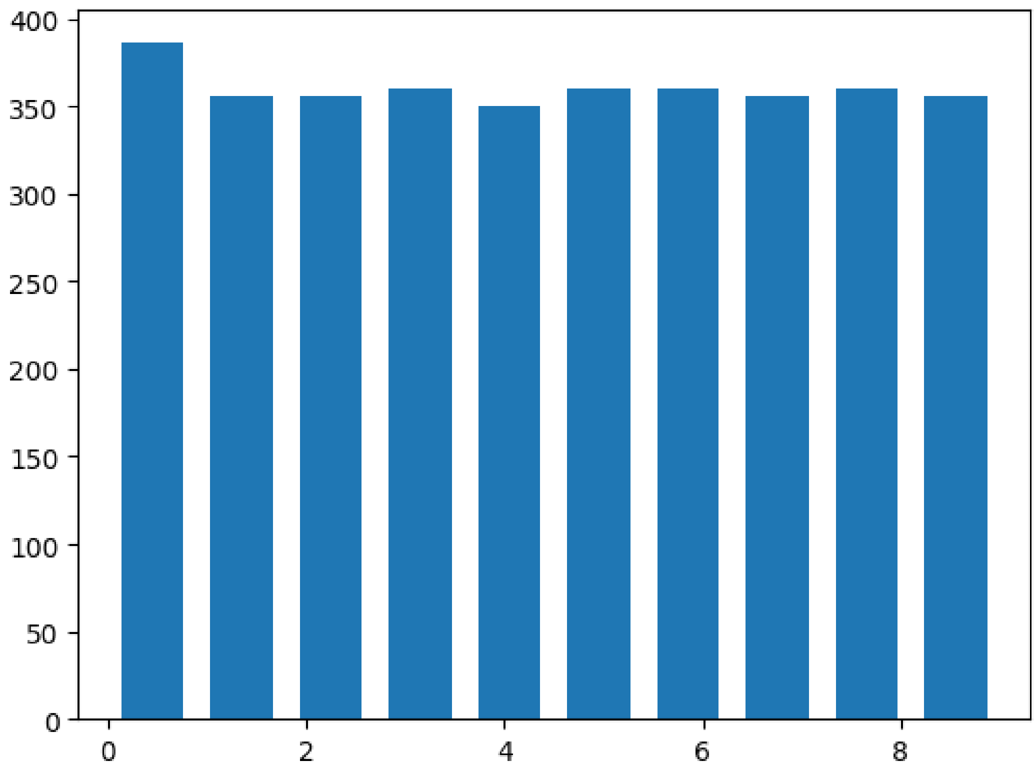
[3]:
```
# Load dataset from h5 file
hf = h5py.File('dataset.h5', 'r')

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

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

print('Data size:', np.shape(data))
print('Data_labels size:', np.shape(data_labels))

hf.close()
```

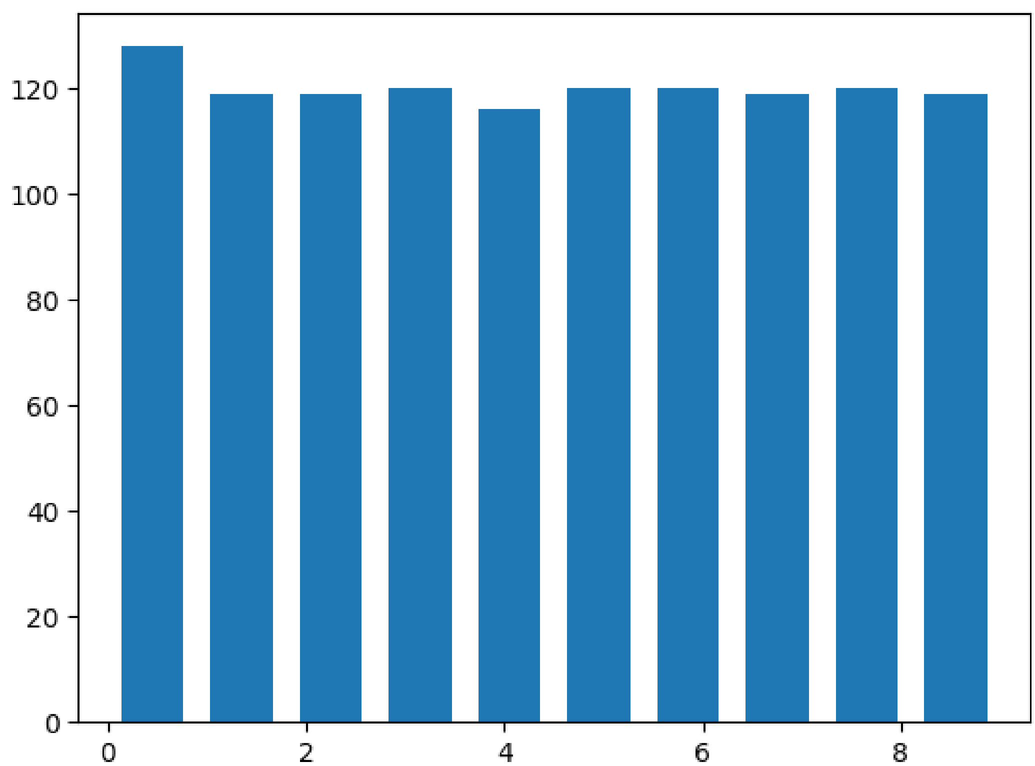
```
Data size: (6000, 26)
Data_labels size: (6000, 1)
```

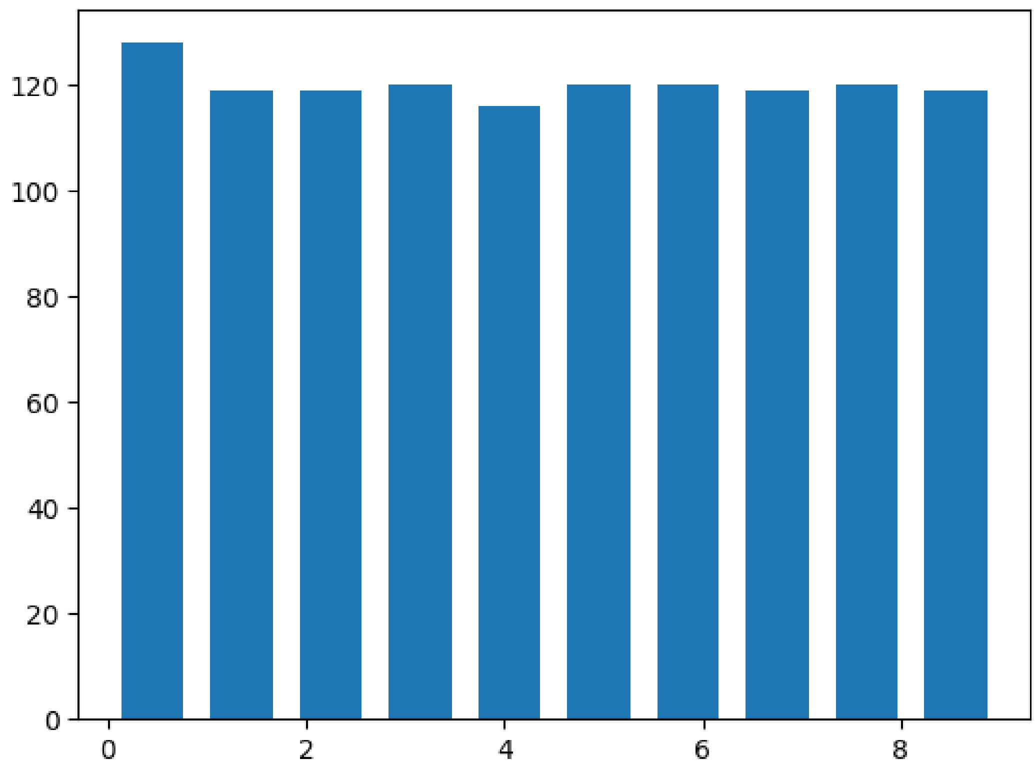
[4]:

```
# Normalize
scaler = StandardScaler()
X = scaler.fit_transform(np.array(data, dtype = float))

# 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)

# Sizes
print('Train:', np.shape(y_train))
print('Test:', np.shape(y_test))
print('Val:', np.shape(y_val))

# The truth is - there is no optimal split percentage
# train 80%; valid 10%; test 10%
# train 70%; valid 15%; test 15%
# tarin 60%; valid 20%; test 20%

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()
```

```
Train: (3600, 1)
Test: (1200, 1)
Val: (1200, 1)
```


400
350
300
250
200
150
100
50
0
0
2
4
6
8


120
100
80
60
40
20
0
0
2
4
6
8



[5]:
```
# NN
#https://medium.com/@sdoshi579/
↪classification-of-music-into-different-genres-using-keras-82ab5339efe0
model = models.Sequential()
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(32, activation='relu'))
model.add(layers.Dense(n_genres, activation='softmax')) # Output layer - 10␣
↪genres

opt = keras.optimizers.Adam(learning_rate = 0.0001)
loss = tf.keras.losses.SparseCategoricalCrossentropy() # 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])
```

[6]:
```
# 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("audio_weights.weights.h5",␣
↪save_best_only=True, save_weights_only=True)

# Train
t_epochs = 100 # Needs to be tuned
b_size = 8 # 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("audio_weights.weights.h5")
```

```
Epoch 1/100
450/450              2s 2ms/step -
loss: 2.3139 - sparse_categorical_accuracy: 0.1435 - val_loss: 1.9932 -
val_sparse_categorical_accuracy: 0.3167
Epoch 2/100
450/450              1s 2ms/step -
loss: 1.9208 - sparse_categorical_accuracy: 0.3392 - val_loss: 1.7378 -
val_sparse_categorical_accuracy: 0.4192
Epoch 3/100
450/450              1s 2ms/step -
loss: 1.6816 - sparse_categorical_accuracy: 0.4235 - val_loss: 1.5724 -
val_sparse_categorical_accuracy: 0.4692
Epoch 4/100
450/450              1s 2ms/step -
loss: 1.5322 - sparse_categorical_accuracy: 0.4690 - val_loss: 1.4500 -
val_sparse_categorical_accuracy: 0.5150
Epoch 5/100
450/450              1s 2ms/step -
loss: 1.4439 - sparse_categorical_accuracy: 0.5086 - val_loss: 1.3534 -
val_sparse_categorical_accuracy: 0.5592
Epoch 6/100
450/450              1s 2ms/step -
loss: 1.3388 - sparse_categorical_accuracy: 0.5534 - val_loss: 1.2740 -
val_sparse_categorical_accuracy: 0.5783
Epoch 7/100
450/450              1s 2ms/step -
loss: 1.2460 - sparse_categorical_accuracy: 0.5894 - val_loss: 1.2094 -
val_sparse_categorical_accuracy: 0.6017
Epoch 8/100
450/450              1s 2ms/step -
loss: 1.1707 - sparse_categorical_accuracy: 0.6145 - val_loss: 1.1568 -
```

```
val_sparse_categorical_accuracy: 0.6133
Epoch 9/100
450/450              1s 2ms/step -
loss: 1.1177 - sparse_categorical_accuracy: 0.6141 - val_loss: 1.1110 -
val_sparse_categorical_accuracy: 0.6283
Epoch 10/100
450/450              1s 2ms/step -
loss: 1.0930 - sparse_categorical_accuracy: 0.6312 - val_loss: 1.0725 -
val_sparse_categorical_accuracy: 0.6483
Epoch 11/100
450/450              1s 1ms/step -
loss: 1.0470 - sparse_categorical_accuracy: 0.6534 - val_loss: 1.0383 -
val_sparse_categorical_accuracy: 0.6667
Epoch 12/100
450/450              1s 2ms/step -
loss: 1.0014 - sparse_categorical_accuracy: 0.6705 - val_loss: 1.0083 -
val_sparse_categorical_accuracy: 0.6842
Epoch 13/100
450/450              1s 2ms/step -
loss: 0.9724 - sparse_categorical_accuracy: 0.6882 - val_loss: 0.9828 -
val_sparse_categorical_accuracy: 0.6833
Epoch 14/100
450/450              1s 2ms/step -
loss: 0.9519 - sparse_categorical_accuracy: 0.6823 - val_loss: 0.9584 -
val_sparse_categorical_accuracy: 0.6933
Epoch 15/100
450/450              1s 1ms/step -
loss: 0.9308 - sparse_categorical_accuracy: 0.6975 - val_loss: 0.9350 -
val_sparse_categorical_accuracy: 0.6975
Epoch 16/100
450/450              1s 2ms/step -
loss: 0.9083 - sparse_categorical_accuracy: 0.7039 - val_loss: 0.9146 -
val_sparse_categorical_accuracy: 0.7058
Epoch 17/100
450/450              1s 2ms/step -
loss: 0.8672 - sparse_categorical_accuracy: 0.7128 - val_loss: 0.8959 -
val_sparse_categorical_accuracy: 0.7075
Epoch 18/100
450/450              1s 1ms/step -
loss: 0.8743 - sparse_categorical_accuracy: 0.7150 - val_loss: 0.8786 -
val_sparse_categorical_accuracy: 0.7067
Epoch 19/100
450/450              1s 2ms/step -
loss: 0.8462 - sparse_categorical_accuracy: 0.7267 - val_loss: 0.8626 -
val_sparse_categorical_accuracy: 0.7225
Epoch 20/100
450/450              1s 2ms/step -
loss: 0.8378 - sparse_categorical_accuracy: 0.7221 - val_loss: 0.8478 -
```

```
val_sparse_categorical_accuracy: 0.7258
Epoch 21/100
450/450            1s 2ms/step -
loss: 0.7818 - sparse_categorical_accuracy: 0.7457 - val_loss: 0.8319 -
val_sparse_categorical_accuracy: 0.7383
Epoch 22/100
450/450            1s 2ms/step -
loss: 0.7722 - sparse_categorical_accuracy: 0.7511 - val_loss: 0.8184 -
val_sparse_categorical_accuracy: 0.7408
Epoch 23/100
450/450            1s 2ms/step -
loss: 0.7546 - sparse_categorical_accuracy: 0.7584 - val_loss: 0.8061 -
val_sparse_categorical_accuracy: 0.7467
Epoch 24/100
450/450            1s 2ms/step -
loss: 0.7391 - sparse_categorical_accuracy: 0.7710 - val_loss: 0.7931 -
val_sparse_categorical_accuracy: 0.7450
Epoch 25/100
450/450            1s 1ms/step -
loss: 0.7398 - sparse_categorical_accuracy: 0.7585 - val_loss: 0.7801 -
val_sparse_categorical_accuracy: 0.7600
Epoch 26/100
450/450            1s 2ms/step -
loss: 0.7405 - sparse_categorical_accuracy: 0.7765 - val_loss: 0.7685 -
val_sparse_categorical_accuracy: 0.7592
Epoch 27/100
450/450            1s 2ms/step -
loss: 0.6931 - sparse_categorical_accuracy: 0.7911 - val_loss: 0.7566 -
val_sparse_categorical_accuracy: 0.7700
Epoch 28/100
450/450            1s 2ms/step -
loss: 0.6959 - sparse_categorical_accuracy: 0.7840 - val_loss: 0.7466 -
val_sparse_categorical_accuracy: 0.7775
Epoch 29/100
450/450            1s 2ms/step -
loss: 0.6792 - sparse_categorical_accuracy: 0.8010 - val_loss: 0.7337 -
val_sparse_categorical_accuracy: 0.7758
Epoch 30/100
450/450            1s 2ms/step -
loss: 0.6735 - sparse_categorical_accuracy: 0.8010 - val_loss: 0.7237 -
val_sparse_categorical_accuracy: 0.7742
Epoch 31/100
450/450            1s 2ms/step -
loss: 0.6507 - sparse_categorical_accuracy: 0.7999 - val_loss: 0.7150 -
val_sparse_categorical_accuracy: 0.7800
Epoch 32/100
450/450            1s 2ms/step -
loss: 0.6328 - sparse_categorical_accuracy: 0.8077 - val_loss: 0.7027 -
```

```
val_sparse_categorical_accuracy: 0.7858
Epoch 33/100
450/450            1s 2ms/step -
loss: 0.6555 - sparse_categorical_accuracy: 0.7950 - val_loss: 0.6940 -
val_sparse_categorical_accuracy: 0.7892
Epoch 34/100
450/450            1s 2ms/step -
loss: 0.5996 - sparse_categorical_accuracy: 0.8300 - val_loss: 0.6817 -
val_sparse_categorical_accuracy: 0.7925
Epoch 35/100
450/450            1s 2ms/step -
loss: 0.5900 - sparse_categorical_accuracy: 0.8258 - val_loss: 0.6724 -
val_sparse_categorical_accuracy: 0.7950
Epoch 36/100
450/450            1s 2ms/step -
loss: 0.6078 - sparse_categorical_accuracy: 0.8208 - val_loss: 0.6645 -
val_sparse_categorical_accuracy: 0.7942
Epoch 37/100
450/450            1s 2ms/step -
loss: 0.5596 - sparse_categorical_accuracy: 0.8375 - val_loss: 0.6568 -
val_sparse_categorical_accuracy: 0.7925
Epoch 38/100
450/450            1s 2ms/step -
loss: 0.5889 - sparse_categorical_accuracy: 0.8202 - val_loss: 0.6449 -
val_sparse_categorical_accuracy: 0.7983
Epoch 39/100
450/450            1s 2ms/step -
loss: 0.5586 - sparse_categorical_accuracy: 0.8346 - val_loss: 0.6371 -
val_sparse_categorical_accuracy: 0.8000
Epoch 40/100
450/450            1s 2ms/step -
loss: 0.5518 - sparse_categorical_accuracy: 0.8372 - val_loss: 0.6255 -
val_sparse_categorical_accuracy: 0.8083
Epoch 41/100
450/450            1s 2ms/step -
loss: 0.5462 - sparse_categorical_accuracy: 0.8473 - val_loss: 0.6191 -
val_sparse_categorical_accuracy: 0.8100
Epoch 42/100
450/450            1s 2ms/step -
loss: 0.5221 - sparse_categorical_accuracy: 0.8515 - val_loss: 0.6086 -
val_sparse_categorical_accuracy: 0.8067
Epoch 43/100
450/450            1s 2ms/step -
loss: 0.5288 - sparse_categorical_accuracy: 0.8481 - val_loss: 0.5988 -
val_sparse_categorical_accuracy: 0.8125
Epoch 44/100
450/450            1s 2ms/step -
loss: 0.5067 - sparse_categorical_accuracy: 0.8446 - val_loss: 0.5926 -
```

```
val_sparse_categorical_accuracy: 0.8158
Epoch 45/100
450/450            1s 2ms/step -
loss: 0.5148 - sparse_categorical_accuracy: 0.8590 - val_loss: 0.5848 -
val_sparse_categorical_accuracy: 0.8133
Epoch 46/100
450/450            1s 2ms/step -
loss: 0.5066 - sparse_categorical_accuracy: 0.8417 - val_loss: 0.5785 -
val_sparse_categorical_accuracy: 0.8142
Epoch 47/100
450/450            1s 2ms/step -
loss: 0.4868 - sparse_categorical_accuracy: 0.8488 - val_loss: 0.5698 -
val_sparse_categorical_accuracy: 0.8183
Epoch 48/100
450/450            1s 2ms/step -
loss: 0.4660 - sparse_categorical_accuracy: 0.8622 - val_loss: 0.5616 -
val_sparse_categorical_accuracy: 0.8167
Epoch 49/100
450/450            1s 2ms/step -
loss: 0.4581 - sparse_categorical_accuracy: 0.8661 - val_loss: 0.5516 -
val_sparse_categorical_accuracy: 0.8250
Epoch 50/100
450/450            1s 2ms/step -
loss: 0.4605 - sparse_categorical_accuracy: 0.8632 - val_loss: 0.5463 -
val_sparse_categorical_accuracy: 0.8192
Epoch 51/100
450/450            1s 2ms/step -
loss: 0.4510 - sparse_categorical_accuracy: 0.8659 - val_loss: 0.5388 -
val_sparse_categorical_accuracy: 0.8200
Epoch 52/100
450/450            1s 2ms/step -
loss: 0.4125 - sparse_categorical_accuracy: 0.8825 - val_loss: 0.5306 -
val_sparse_categorical_accuracy: 0.8317
Epoch 53/100
450/450            1s 2ms/step -
loss: 0.4377 - sparse_categorical_accuracy: 0.8674 - val_loss: 0.5261 -
val_sparse_categorical_accuracy: 0.8358
Epoch 54/100
450/450            1s 2ms/step -
loss: 0.4637 - sparse_categorical_accuracy: 0.8604 - val_loss: 0.5170 -
val_sparse_categorical_accuracy: 0.8375
Epoch 55/100
450/450            1s 2ms/step -
loss: 0.4259 - sparse_categorical_accuracy: 0.8772 - val_loss: 0.5118 -
val_sparse_categorical_accuracy: 0.8400
Epoch 56/100
450/450            1s 2ms/step -
loss: 0.4191 - sparse_categorical_accuracy: 0.8759 - val_loss: 0.5010 -
```

```
val_sparse_categorical_accuracy: 0.8467
Epoch 57/100
450/450              1s 2ms/step -
loss: 0.4349 - sparse_categorical_accuracy: 0.8736 - val_loss: 0.4974 -
val_sparse_categorical_accuracy: 0.8417
Epoch 58/100
450/450              1s 2ms/step -
loss: 0.4022 - sparse_categorical_accuracy: 0.8842 - val_loss: 0.4891 -
val_sparse_categorical_accuracy: 0.8475
Epoch 59/100
450/450              1s 2ms/step -
loss: 0.3902 - sparse_categorical_accuracy: 0.8899 - val_loss: 0.4814 -
val_sparse_categorical_accuracy: 0.8550
Epoch 60/100
450/450              1s 2ms/step -
loss: 0.3886 - sparse_categorical_accuracy: 0.8910 - val_loss: 0.4794 -
val_sparse_categorical_accuracy: 0.8517
Epoch 61/100
450/450              1s 2ms/step -
loss: 0.3944 - sparse_categorical_accuracy: 0.8812 - val_loss: 0.4696 -
val_sparse_categorical_accuracy: 0.8592
Epoch 62/100
450/450              1s 2ms/step -
loss: 0.3744 - sparse_categorical_accuracy: 0.8986 - val_loss: 0.4618 -
val_sparse_categorical_accuracy: 0.8583
Epoch 63/100
450/450              1s 2ms/step -
loss: 0.3790 - sparse_categorical_accuracy: 0.8927 - val_loss: 0.4573 -
val_sparse_categorical_accuracy: 0.8583
Epoch 64/100
450/450              1s 2ms/step -
loss: 0.3432 - sparse_categorical_accuracy: 0.9030 - val_loss: 0.4506 -
val_sparse_categorical_accuracy: 0.8650
Epoch 65/100
450/450              1s 2ms/step -
loss: 0.3667 - sparse_categorical_accuracy: 0.8972 - val_loss: 0.4442 -
val_sparse_categorical_accuracy: 0.8717
Epoch 66/100
450/450              1s 2ms/step -
loss: 0.3527 - sparse_categorical_accuracy: 0.9078 - val_loss: 0.4363 -
val_sparse_categorical_accuracy: 0.8650
Epoch 67/100
450/450              1s 2ms/step -
loss: 0.3310 - sparse_categorical_accuracy: 0.9192 - val_loss: 0.4281 -
val_sparse_categorical_accuracy: 0.8692
Epoch 68/100
450/450              1s 2ms/step -
loss: 0.3213 - sparse_categorical_accuracy: 0.9210 - val_loss: 0.4242 -
```

```
val_sparse_categorical_accuracy: 0.8733
Epoch 69/100
450/450            1s 2ms/step -
loss: 0.3365 - sparse_categorical_accuracy: 0.9194 - val_loss: 0.4179 -
val_sparse_categorical_accuracy: 0.8750
Epoch 70/100
450/450            1s 2ms/step -
loss: 0.3162 - sparse_categorical_accuracy: 0.9183 - val_loss: 0.4142 -
val_sparse_categorical_accuracy: 0.8808
Epoch 71/100
450/450            1s 2ms/step -
loss: 0.3186 - sparse_categorical_accuracy: 0.9188 - val_loss: 0.4089 -
val_sparse_categorical_accuracy: 0.8842
Epoch 72/100
450/450            1s 2ms/step -
loss: 0.3272 - sparse_categorical_accuracy: 0.9169 - val_loss: 0.3998 -
val_sparse_categorical_accuracy: 0.8842
Epoch 73/100
450/450            1s 2ms/step -
loss: 0.3059 - sparse_categorical_accuracy: 0.9239 - val_loss: 0.3951 -
val_sparse_categorical_accuracy: 0.8858
Epoch 74/100
450/450            1s 2ms/step -
loss: 0.3131 - sparse_categorical_accuracy: 0.9214 - val_loss: 0.3888 -
val_sparse_categorical_accuracy: 0.8858
Epoch 75/100
450/450            1s 2ms/step -
loss: 0.3059 - sparse_categorical_accuracy: 0.9250 - val_loss: 0.3817 -
val_sparse_categorical_accuracy: 0.8875
Epoch 76/100
450/450            1s 2ms/step -
loss: 0.2991 - sparse_categorical_accuracy: 0.9266 - val_loss: 0.3786 -
val_sparse_categorical_accuracy: 0.8917
Epoch 77/100
450/450            1s 2ms/step -
loss: 0.2873 - sparse_categorical_accuracy: 0.9307 - val_loss: 0.3749 -
val_sparse_categorical_accuracy: 0.8875
Epoch 78/100
450/450            1s 2ms/step -
loss: 0.2829 - sparse_categorical_accuracy: 0.9325 - val_loss: 0.3684 -
val_sparse_categorical_accuracy: 0.8908
Epoch 79/100
450/450            1s 2ms/step -
loss: 0.2779 - sparse_categorical_accuracy: 0.9383 - val_loss: 0.3601 -
val_sparse_categorical_accuracy: 0.8983
Epoch 80/100
450/450            1s 2ms/step -
loss: 0.2826 - sparse_categorical_accuracy: 0.9305 - val_loss: 0.3581 -
```

```
val_sparse_categorical_accuracy: 0.9017
Epoch 81/100
450/450            1s 3ms/step -
loss: 0.2566 - sparse_categorical_accuracy: 0.9415 - val_loss: 0.3508 -
val_sparse_categorical_accuracy: 0.9033
Epoch 82/100
450/450            1s 2ms/step -
loss: 0.2635 - sparse_categorical_accuracy: 0.9400 - val_loss: 0.3472 -
val_sparse_categorical_accuracy: 0.9067
Epoch 83/100
450/450            1s 2ms/step -
loss: 0.2584 - sparse_categorical_accuracy: 0.9397 - val_loss: 0.3410 -
val_sparse_categorical_accuracy: 0.9067
Epoch 84/100
450/450            1s 2ms/step -
loss: 0.2502 - sparse_categorical_accuracy: 0.9420 - val_loss: 0.3334 -
val_sparse_categorical_accuracy: 0.9033
Epoch 85/100
450/450            1s 2ms/step -
loss: 0.2515 - sparse_categorical_accuracy: 0.9413 - val_loss: 0.3279 -
val_sparse_categorical_accuracy: 0.9117
Epoch 86/100
450/450            1s 2ms/step -
loss: 0.2563 - sparse_categorical_accuracy: 0.9456 - val_loss: 0.3238 -
val_sparse_categorical_accuracy: 0.9133
Epoch 87/100
450/450            1s 2ms/step -
loss: 0.2471 - sparse_categorical_accuracy: 0.9434 - val_loss: 0.3233 -
val_sparse_categorical_accuracy: 0.9117
Epoch 88/100
450/450            1s 2ms/step -
loss: 0.2350 - sparse_categorical_accuracy: 0.9539 - val_loss: 0.3123 -
val_sparse_categorical_accuracy: 0.9108
Epoch 89/100
450/450            1s 2ms/step -
loss: 0.2341 - sparse_categorical_accuracy: 0.9510 - val_loss: 0.3074 -
val_sparse_categorical_accuracy: 0.9150
Epoch 90/100
450/450            1s 2ms/step -
loss: 0.2268 - sparse_categorical_accuracy: 0.9516 - val_loss: 0.3041 -
val_sparse_categorical_accuracy: 0.9183
Epoch 91/100
450/450            1s 2ms/step -
loss: 0.2127 - sparse_categorical_accuracy: 0.9626 - val_loss: 0.2972 -
val_sparse_categorical_accuracy: 0.9133
Epoch 92/100
450/450            1s 2ms/step -
loss: 0.2117 - sparse_categorical_accuracy: 0.9619 - val_loss: 0.2970 -
```

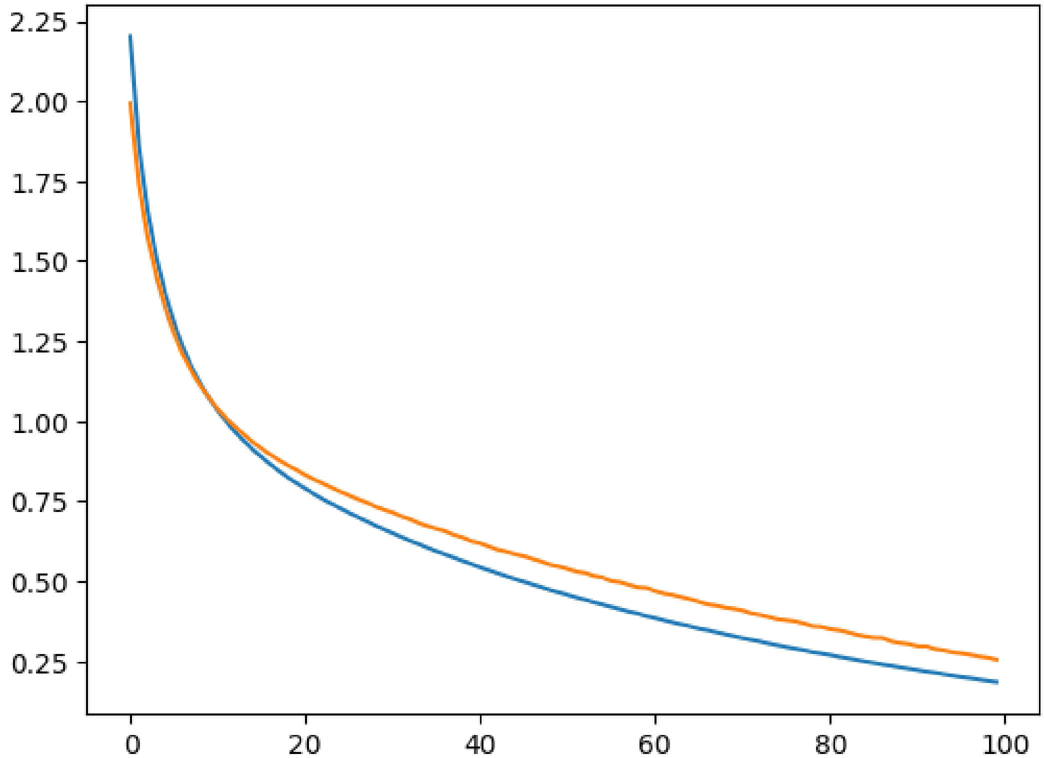
```
val_sparse_categorical_accuracy: 0.9200
Epoch 93/100
450/450              1s 2ms/step -
loss: 0.2165 - sparse_categorical_accuracy: 0.9587 - val_loss: 0.2883 -
val_sparse_categorical_accuracy: 0.9217
Epoch 94/100
450/450              1s 2ms/step -
loss: 0.2068 - sparse_categorical_accuracy: 0.9604 - val_loss: 0.2844 -
val_sparse_categorical_accuracy: 0.9208
Epoch 95/100
450/450              1s 2ms/step -
loss: 0.2196 - sparse_categorical_accuracy: 0.9494 - val_loss: 0.2784 -
val_sparse_categorical_accuracy: 0.9233
Epoch 96/100
450/450              1s 2ms/step -
loss: 0.2019 - sparse_categorical_accuracy: 0.9571 - val_loss: 0.2752 -
val_sparse_categorical_accuracy: 0.9183
Epoch 97/100
450/450              1s 2ms/step -
loss: 0.1960 - sparse_categorical_accuracy: 0.9604 - val_loss: 0.2710 -
val_sparse_categorical_accuracy: 0.9250
Epoch 98/100
450/450              1s 2ms/step -
loss: 0.1944 - sparse_categorical_accuracy: 0.9603 - val_loss: 0.2654 -
val_sparse_categorical_accuracy: 0.9258
Epoch 99/100
450/450              1s 2ms/step -
loss: 0.2044 - sparse_categorical_accuracy: 0.9586 - val_loss: 0.2616 -
val_sparse_categorical_accuracy: 0.9275
Epoch 100/100
450/450              1s 2ms/step -
loss: 0.1817 - sparse_categorical_accuracy: 0.9696 - val_loss: 0.2557 -
val_sparse_categorical_accuracy: 0.9292
```

[7]:

```
# 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'])
```

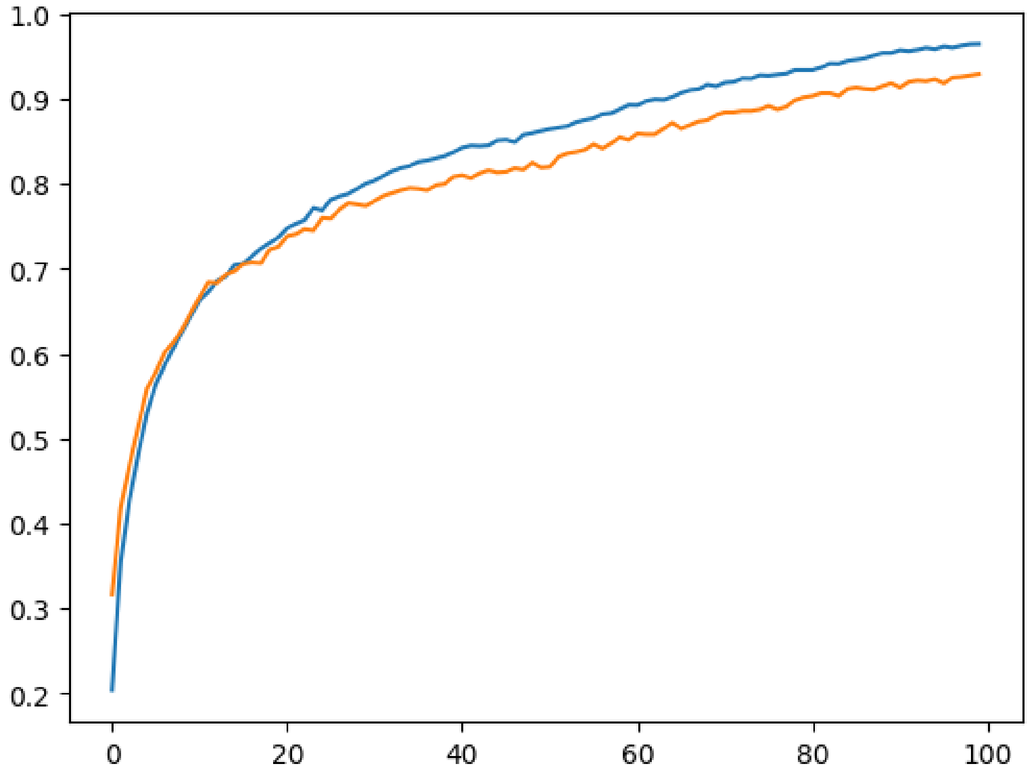
[7]:

```
[<matplotlib.lines.Line2D at 0x2551f83c6a0>]
```



[8]:
```
#what do these graphs mean

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

[8]:
```
[<matplotlib.lines.Line2D at 0x255208552e0>]
```



[9]:
```
# Now to evaluate our model on train and test data

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

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

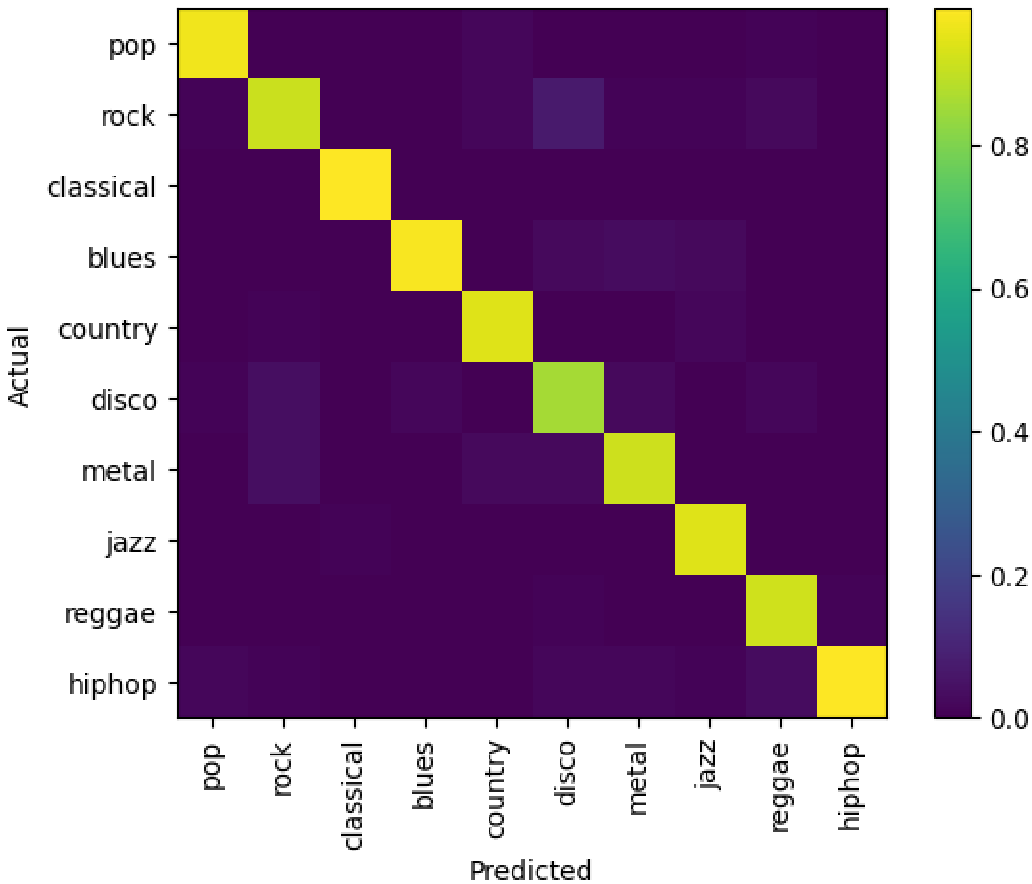
```
Acc train NN: 0.966
Acc test NN: 0.942
```

[10]:
```
# 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")

# 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()
```

```
38/38                 0s 2ms/step
```

[10]: <matplotlib.colorbar.Colorbar at 0x255209ce5e0>



[ ]:
```
import datetime
import miniaudio
from IPython.display import clear_output, display

current_title = ""

def title(client: miniaudio.IceCastClient, title: str):
    global current_title
```

```
    current_title = title

def stream_processing(source):

    predictions = []
    pred_interval = 44100 * 60 # sampling frequency * 60 seconds

    sample_count = 0

    while True:
        # Get frame - Only one channel - The chunk of signal is too small!
        sample_data = np.array(source.send(8192))[0::2]
        sample_count += len(sample_data)

        if len(sample_data) > 0:
            #Features
            feat = extract_features(sample_data, 44100, n_features, n_mfcc_coef)

            # Normalization
            feat_norm = scaler.transform(feat.reshape(1, -1))
            pred_nn = model.predict(feat_norm, verbose=0)
            predictions.append(np.argmax(pred_nn[0]))

            clear_output(wait=True)
            print("Title: " + current_title)
            print(datetime.datetime.now())

            most_common = np.bincount(predictions).argmax()
            print("NN: " + genres[most_common])

        if sample_count >= pred_interval:

            plt.clf()
            plt.hist(predictions, bins=np.arange(len(genres) + 1) - 0.5)
            plt.xticks(np.arange(len(genres)), genres)
            plt.title('Napovedi žanrov')
            plt.xlabel('Žanr')
            plt.ylabel('Število pojavitev')
            # plt.show()
            print("Save fig!")
            plt.savefig(str("C:
↪\\Users\\Viktorija\\Desktop\\ROSIS\\N8\\mlp_output\\histogram_" + datetime.
↪datetime.now().strftime("%H_%M_%S") + ".png"))

            predictions = []
            sample_count = 0
```

```python
        yield sample_data

# Internet radio source - Radio 1
source = miniaudio.IceCastClient("http://live1.radio1.si/Radio1",
 ↪update_stream_title=title)

print("Connected")
print("Station: ", source.station_name)

# Stream
stream_in = miniaudio.stream_any(source, source.audio_format,
 ↪output_format=miniaudio.SampleFormat.FLOAT32)
# Device
device = miniaudio.PlaybackDevice(output_format=miniaudio.SampleFormat.FLOAT32,
 ↪nchannels=1, sample_rate=44100)

stream = stream_processing(stream_in)
next(stream)
device.start(stream)

while True:
    time.sleep(0.1)

#RICK ASTLEY - TOGETHER FOREVER -> disco
#CALVIN HARRIS & ALESSO FEAT. HURTS - UNDER CONTROL -> disco
```

```
Title: CALVIN HARRIS & ALESSO FEAT. HURTS - UNDER CONTROL
2024-08-16 10:55:22.945470
NN: disco
```