Ministerul Educaţiei, al Culturii și Cercetării al Republicii Moldova

Universitatea Tehnică a Moldovei

Departamentul Informatică și Ingineria Sistemelor

**RAPORT**

Lucrarea de laborator nr.5

Inteligenta Artificiala

A efectuat:

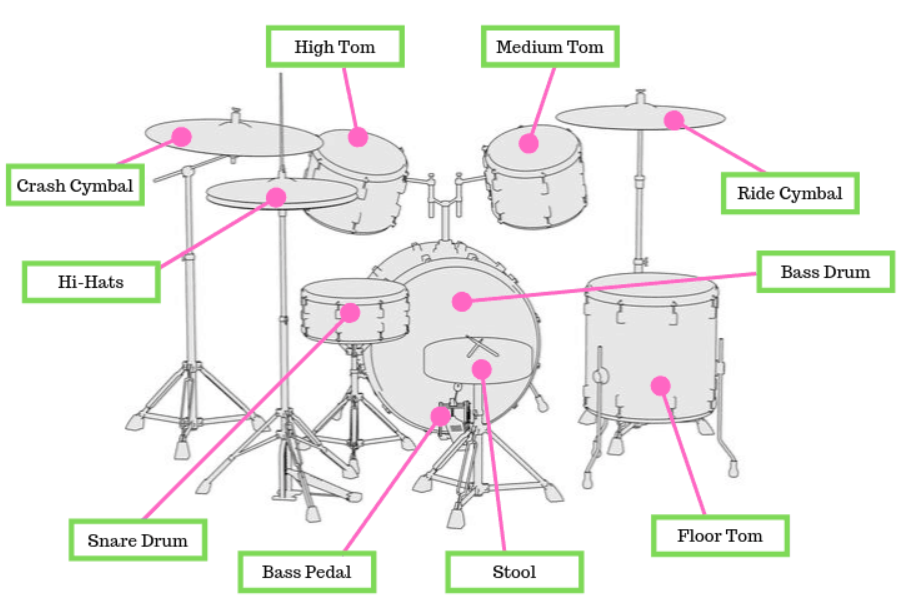
st. gr. C-171 D. Melniciuc

A verificat:

dr., conf.univ. T. Bumbu

Chişinău 2020

Problema alesa:

Sortarea pe categorii a partilor componente de tobe/baterie. Programul trebuie sa sorteze mici secvente audio cu parti componente de tobe la categoria respectiva(E.g. : snare la snare, ride cymbal la ride cymbal).

Fisierele audio se extrag din directorii deja clasificate apoi se proceseaza. Clasificarea se face dupa coeficientul mfcc(Mel-frequency cepstrum).

Codul:

# -\*- coding: utf-8 -\*-

"""Untitled5.ipynb

Automatically generated by Colaboratory.

Original file is located at

https://colab.research.google.com/drive/1zJz2fTzM4BbP0f2FEEDIewO95-d7jfRx

"""

from google.colab import drive

drive.mount('/content/drive')

import array

import os

!pip install pydub

from pydub import AudioSegment

from pydub.playback import play

import numpy as np

import tensorflow as tf

from tensorflow.keras import regularizers

import re

import functools

print = functools.partial(print, flush=True)

import matplotlib.pyplot as plt

from numpy import asarray

from numpy import save

from numpy import load

import librosa, librosa.display

class\_names = ["kick","snare","clap","hats"]

# train\_path\_list = ["samples\\Kicks\\","samples\\Snares\\","samples\\Claps\\","samples\\Hihats - Closed\\"]

# test\_path\_list = ["samples\\test\\Kicks\\","samples\\test\\Snares\\","samples\\test\\Claps\\","samples\\test\\Hihats - Closed\\"]

train\_path\_list = ["/content/drive/MyDrive/samples/Kicks/","/content/drive/MyDrive/samples/Snares/","/content/drive/MyDrive/samples/Claps/","/content/drive/MyDrive/samples/Hihats/"]

test\_path\_list = ["/content/drive/MyDrive/samples/Kicks/","/content/drive/MyDrive/samples/Snares/","/content/drive/MyDrive/samples/Claps/","/content/drive/MyDrive/samples/Hihats/"]

# file = "/content/drive/MyDrive/samples/Kicks/1.wav"

# signal, sr = librosa.load(file, sr=22050)

# librosa.display.waveplot(signal, sr=22050)

# plt.xlabel("time")

# plt.ylabel("amplitude")

# plt.show()

arr = []

for a in train\_path\_list:

for i in os.listdir(a):

print(a, i)

arr.append(a + i)

print("\n")

print(len(arr), " files found\n")

sample\_list = []

n\_fft = 2048

hop\_length = 512

def pitchSample(octaves, sound):

new\_sample\_rate = int(sound.frame\_rate \* (2.0 \*\* octaves))

hipitch\_sound = sound.\_spawn(sound.raw\_data, overrides={'frame\_rate': new\_sample\_rate})

hipitch\_sound = hipitch\_sound.set\_frame\_rate(44100)

return hipitch\_sound

def bassBoostSample(cutoff, sound):

#no actually parametric eq so i layer a lp over

print("cutoff: ", cutoff\*10)

lowpassed = AudioSegment.low\_pass\_filter(sound, (cutoff\*10) + 100)

augmented\_sound = sound + lowpassed

return augmented\_sound

def ms\_samples(sample\_length):

return int((44100 / 1000) \* sample\_length)

def augmentor(sound, aug):

if aug == 1:

pitched\_sounds.append(sound)

else:

for i in range(aug):

aug = float(aug)

spread = ((aug/100) - (aug\*2)/100) + (aug/100)\*i

pitched\_sounds.append(pitchSample(spread,sound))

# pitched\_sounds.append(bassBoostSample(spread,sound))

return pitched\_sounds

sample\_length = 100 #this is ms!

aug = 9

amount\_entries = len(arr)\*aug

np\_mfcc = np.empty((amount\_entries, 9, 13))

#if you already have mfcc's saved it will just load them / if you want to create new ones delete them from the file

# if 'np\_mfcc.npy' in os.listdir():

# print('loading saved data')

# np\_mfcc = load('np\_mfcc.npy')

# sample\_list = load('sample\_list.npy')

# else:

for i in range(len(arr)):

#create mfcc

sound = AudioSegment.from\_file(arr[i], format="wav", channels=1)

sound = sound.set\_channels(1)

pitched\_sounds = []

augmentor(sound,aug)

for x in range(len(pitched\_sounds)):

if re.search("kicks",arr[i]):

sample\_list.append(0)

elif re.search("snares",arr[i]):

sample\_list.append(1)

elif re.search("clap",arr[i]):

sample\_list.append(2)

else:

sample\_list.append(3)

sound = pitched\_sounds[x][:sample\_length]

samples = sound.get\_array\_of\_samples()

if len(samples) < ms\_samples(sample\_length):

padding\_samples = ms\_samples(sample\_length) - len(samples)

for dumi in range(padding\_samples):

samples.append(0)

#turned audio segment into mfcc

samples = np.array(samples)

samples = samples.astype(float)

mfcc = librosa.feature.mfcc(samples, n\_fft=n\_fft, hop\_length=hop\_length, n\_mfcc=13)

mfcc = mfcc.T

# librosa.display.specshow(mfcc, sr=44100, hop\_length=hop\_length)

# plt.show()

mfcc = np.expand\_dims(mfcc, axis=0)

# np\_mfcc = np.append(np\_mfcc, mfcc, axis=2)

np\_mfcc[i\*aug+x] = mfcc

if i % sample\_length == 0:

print(np.floor((i\*sample\_length)/len(arr)))

# save('np\_mfcc.npy', np\_mfcc)

# sample\_list = np.array(sample\_list)

# save('sample\_list.npy', sample\_list)

#3d shape for conv2D layer, 4d if you include batch size

np\_mfcc = np.expand\_dims(np\_mfcc, axis=3)

#shuffle ONCE

seed = 10

np.random.seed(seed)

np.random.shuffle(np\_mfcc)

np.random.seed(seed)

np.random.shuffle(sample\_list)

np.random.seed()

# new test/validation samples

arr = []

for a in test\_path\_list:

for i in os.listdir(a):

# print("\ttesting with ", a + i)

arr.append(a + i)

test\_sample\_list = []

# test\_np\_samples = np.empty((len(arr),ms\_samples(sample\_length)))

test\_np\_mfcc = np.empty((len(arr), 9, 13))

for i in range(len(arr)):

# cant figure out how to play 32bit file

sound = AudioSegment.from\_file(arr[i], format="wav", channels=1)

sound = sound.set\_channels(1)

sound = sound[:sample\_length]

samples = sound.get\_array\_of\_samples()

if re.search("kicks",arr[i]):

test\_sample\_list.append(0)

elif re.search("snares",arr[i]):

test\_sample\_list.append(1)

elif re.search("claps",arr[i]):

test\_sample\_list.append(2)

else:

test\_sample\_list.append(3)

if len(samples) < ms\_samples(sample\_length):

padding\_samples = ms\_samples(sample\_length) - len(samples)

for padno in range(padding\_samples):

samples.append(0)

samples = np.array(samples)

samples = samples.astype(float)

mfcc = librosa.feature.mfcc(samples, n\_fft=n\_fft, hop\_length=hop\_length, n\_mfcc=13)

mfcc = mfcc.T

mfcc = np.expand\_dims(mfcc, axis=0)

test\_np\_mfcc[i] = mfcc

test\_sample\_list = np.array(test\_sample\_list)

test\_np\_mfcc = np.expand\_dims(test\_np\_mfcc, axis=3)

# test\_np\_samples = test\_np\_samples.astype(float) / (2\*\*15)

#shuffle ONCE

seed = 10

np.random.seed(seed)

np.random.shuffle(test\_sample\_list)

np.random.seed(seed)

np.random.shuffle(test\_np\_mfcc)

np.random.seed(seed)

np.random.shuffle(arr)

np.random.seed()

train\_size = int(amount\_entries \* 0.9)

val\_size = amount\_entries - train\_size

training\_ds = tf.data.Dataset.from\_tensor\_slices((np\_mfcc,sample\_list))

# dataset = training\_ds.shuffle(train\_size + val\_size)

val\_ds = training\_ds.skip(train\_size).take(val\_size)

training\_ds = training\_ds.take(train\_size)

print(val\_ds)

print(training\_ds)

batch\_size = 320

STEPS\_PER\_EPOCH = train\_size//batch\_size

def get\_callbacks():

return [

# tfdocs.modeling.EpochDots(),

tf.keras.callbacks.EarlyStopping(monitor='val\_accuracy', patience=6),

# tf.keras.callbacks.TensorBoard(logdir/name),

]

lr\_schedule = tf.keras.optimizers.schedules.InverseTimeDecay(

0.001,

decay\_steps=STEPS\_PER\_EPOCH\*1000,

decay\_rate=1,

staircase=False)

def get\_optimizer():

return tf.keras.optimizers.Adam(lr\_schedule)

inputShape = (9, 13, 1)

model = tf.keras.models.Sequential([

#1st conv layer

tf.keras.layers.Conv2D(32, (3,3), activation='relu', input\_shape=(9, 13, 1), kernel\_regularizer=regularizers.l2(0.001)),

tf.keras.layers.MaxPool2D((3,3), strides=(2,2), padding='same'),

tf.keras.layers.BatchNormalization(),

#flatten & dense & output

tf.keras.layers.Flatten(),

tf.keras.layers.Dense(64, activation='relu', kernel\_regularizer=regularizers.l2(0.001)),

tf.keras.layers.Dropout(0.3),

tf.keras.layers.Dense(len(train\_path\_list))

])

loss\_fn = tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True)

model.compile(optimizer='adam',

loss=loss\_fn,

# loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True),

metrics=['accuracy'])

history = model.fit(training\_ds.shuffle(train\_size).batch(batch\_size),

epochs=100,

validation\_data=val\_ds.batch(batch\_size),

validation\_steps=val\_size,

callbacks=get\_callbacks(),

verbose=1,

# batch\_size=32)

)

model.evaluate(test\_np\_mfcc, test\_sample\_list, verbose=2)

probability\_model = tf.keras.Sequential([

model,

tf.keras.layers.Softmax()

])

for i in range(len(test\_np\_mfcc)):

# because it needs to be a list in a list, [i,4410] just returns a 1d list

result = probability\_model(test\_np\_mfcc[i:i+1])

# print("\n\t\tresult[0]",np.argmax(result[0]) )

# print("\n")

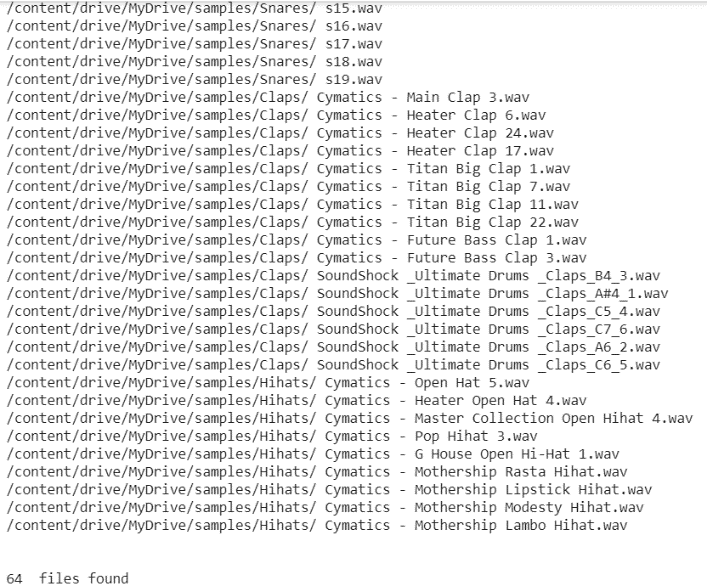
# print("\n\t\t",result)

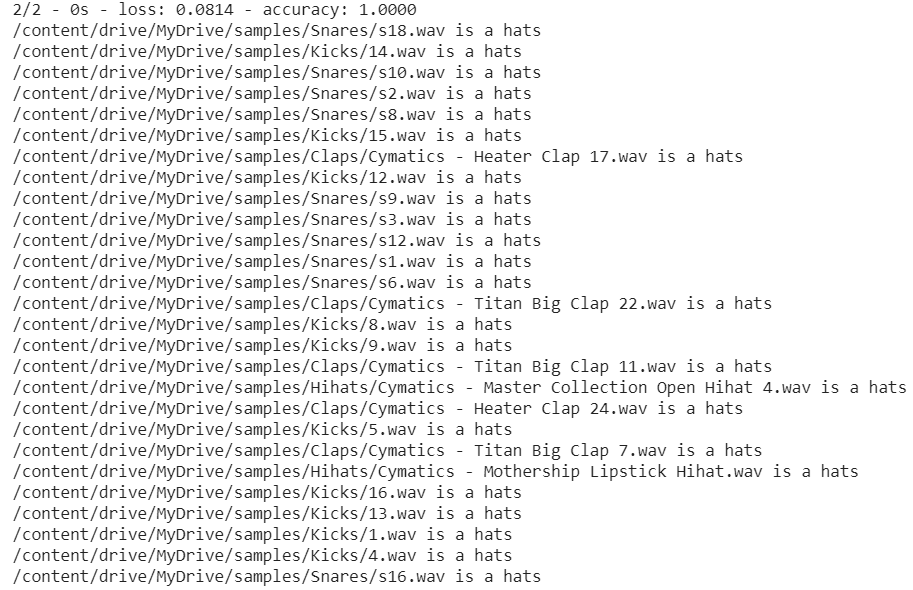
# print("\n")

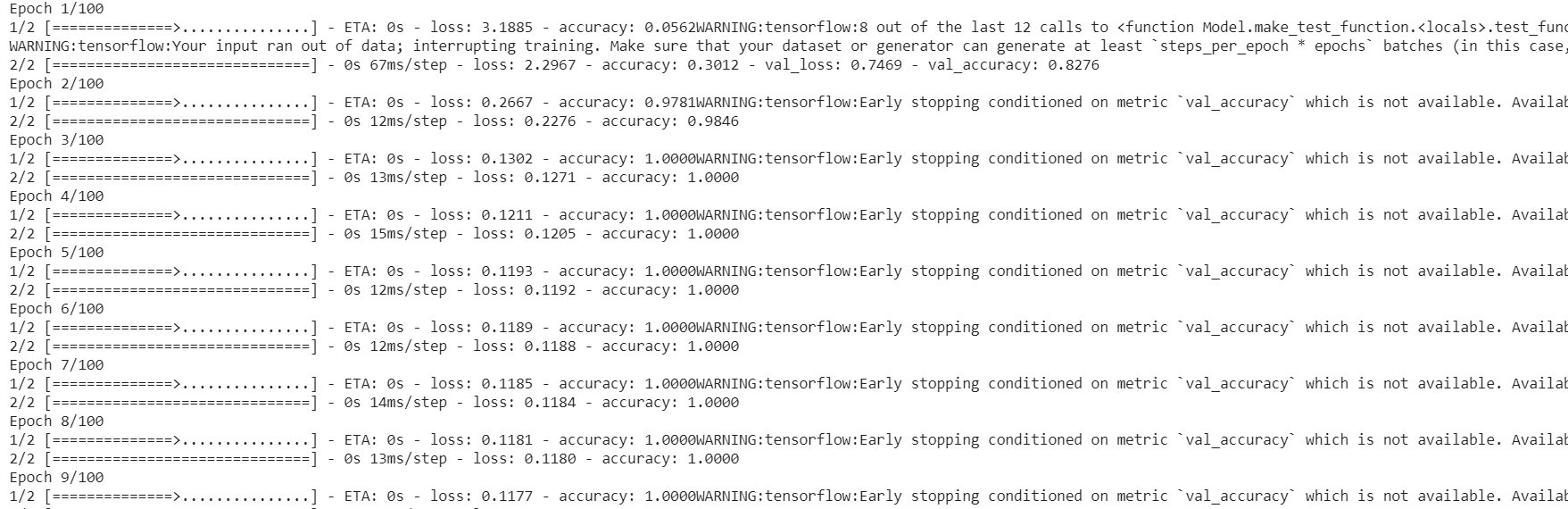
answer = np.argmax(result[0])

print(arr[i] + " is a " + class\_names[answer])

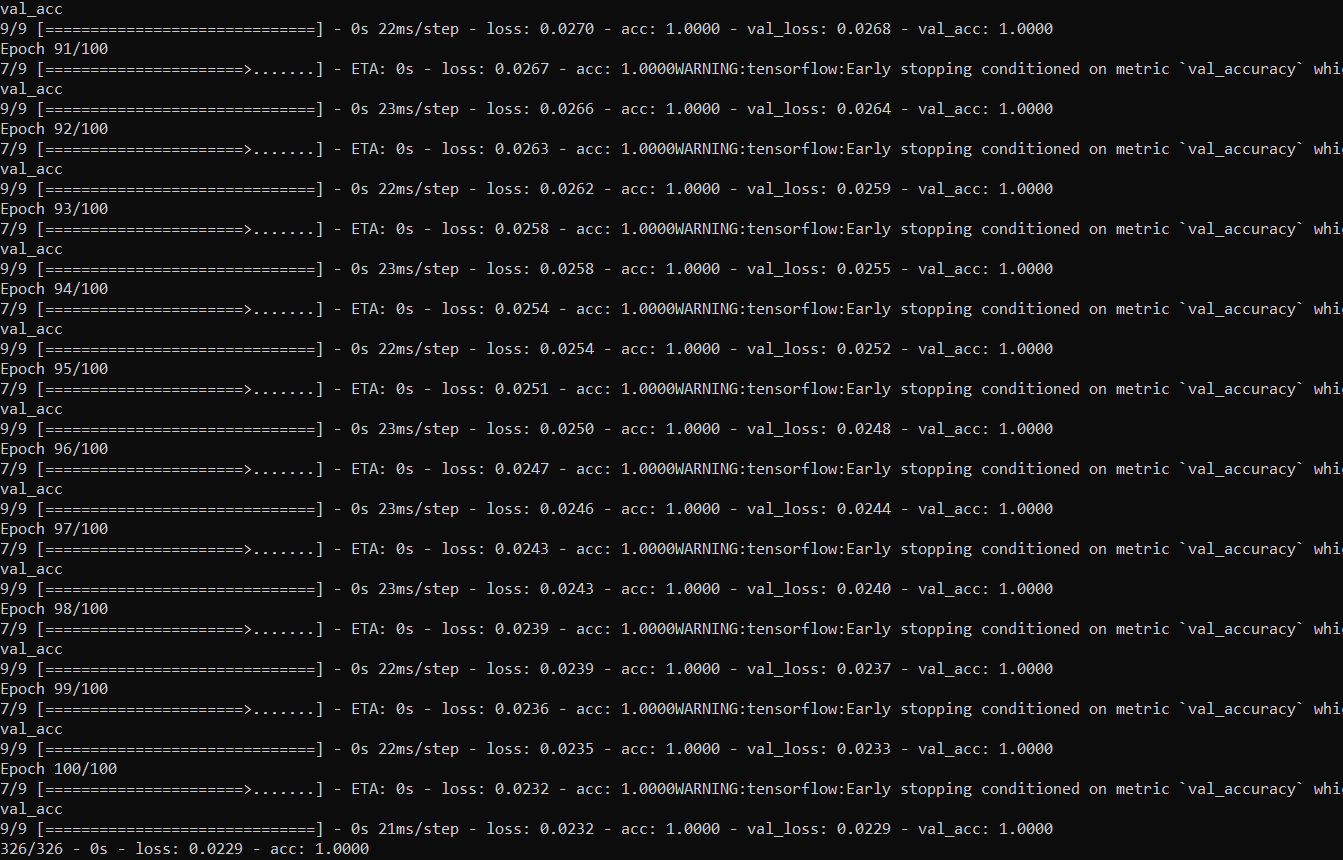
Screenshot-uri:

Preluarea datelor





Rezultat pentru 326 de sample-uri



Concluzie:

In urma efectuarii laboratorului am creat un model automat de invatare si sortarea asupra unui set de date. Un instrument foarte folositor in efectuarea sarcinii a fost libraria tensorflow.