**AVEHICLE IDENTIFICATION BASED ON COMPUTER**

**VISION AND ACOUSTIC SIGNALS**

**CODE:**

**SOURCE CODE**

from keras.models import Sequential

from keras.layers import Convolution2D

from keras.layers import MaxPooling2D

from keras.layers import Flatten

from keras.layers import Dense

from keras.models import model\_from\_json

import matplotlib.pyplot as plt

import warnings

warnings.filterwarnings('ignore')

batch\_size = 32

from tensorflow.keras.preprocessing.image import ImageDataGenerator

# All images will be rescaled by 1./255

train\_datagen = ImageDataGenerator(rescale=1/255)

# Flow training images in batches of 128 using train\_datagen generator

train\_generator = train\_datagen.flow\_from\_directory(

'DataSet/train/', # This is the source directory for training images

target\_size=(200, 200), # All images will be resized to 200 x 200

batch\_size=batch\_size,

# Specify the classes explicitly

classes = ['Ambulance','Bicycle','Bus','Car','Motorcycle','Tank','Taxi','Truck'],# Since we use categorical\_crossentropy loss, we need categorical labels

class\_mode='categorical')

import tensorflow as tf

model = tf.keras.models.Sequential([

# Note the input shape is the desired size of the image 200x 200 with 3 bytes color

# The first convolution

tf.keras.layers.Conv2D(16, (3,3), activation='relu', input\_shape=(200, 200, 3)),

tf.keras.layers.MaxPooling2D(2, 2),

# The second convolution

tf.keras.layers.Conv2D(32, (3,3), activation='relu'),

tf.keras.layers.MaxPooling2D(2,2),

# The third convolution

tf.keras.layers.Conv2D(64, (3,3), activation='relu'),

tf.keras.layers.MaxPooling2D(2,2),

# The fourth convolution

tf.keras.layers.Conv2D(64, (3,3), activation='relu'),

tf.keras.layers.MaxPooling2D(2,2),

# The fifth convolution

tf.keras.layers.Conv2D(64, (3,3), activation='relu'),

tf.keras.layers.MaxPooling2D(2,2),

# Flatten the results to feed into a dense layer

tf.keras.layers.Flatten(),

35# 128 neuron in the fully-connected layer

tf.keras.layers.Dense(128, activation='relu'),

# 5 output neurons for 5 classes with the softmax activation

tf.keras.layers.Dense(8, activation='softmax')])

model.summary()

from tensorflow.keras.optimizers import RMSprop

early = tf.keras.callbacks.EarlyStopping(monitor='val\_loss',patience=5)

model.compile(loss='categorical\_crossentropy',

optimizer=RMSprop(lr=0.001),

metrics=['accuracy'])

total\_sample=train\_generator.n

n\_epochs = 10

history = model.fit\_generator(

train\_generator,

steps\_per\_epoch=int(total\_sample/batch\_size),

epochs=n\_epochs,

verbose=1)

model.save('model.h5')

acc = history.history['accuracy']

loss = history.history['loss']

epochs = range(1, len(acc) + 1)

# Train and validation accuracy

3637

plt.plot(epochs, acc, 'b', label='Training accurarcy')

plt.title('Training accurarcy')

plt.legend()

plt.figure()

# Train and validation loss

plt.plot(epochs, loss, 'b', label='Training loss')

plt.title('Training loss')

plt.legend()

plt.show()

import sys

import pathlib

working\_dir\_path = pathlib.Path().absolute()

if sys.platform.startswith('win32'):

TRAINING\_FILES\_PATH = str(working\_dir\_path) + '\\features\\'

SAVE\_DIR\_PATH = str(working\_dir\_path) + '\\joblib\_features\\'

MODEL\_DIR\_PATH = str(working\_dir\_path) + '\\model\\'

TESS\_ORIGINAL\_FOLDER\_PATH = str(working\_dir\_path) +

'\\sound\_set\_data\\'

EXAMPLES\_PATH = str(working\_dir\_path) + '\\examples\\'

else:

TRAINING\_FILES\_PATH = str(working\_dir\_path) + '/features/'

SAVE\_DIR\_PATH = str(working\_dir\_path) + '/joblib\_features/'

MODEL\_DIR\_PATH = str(working\_dir\_path) + '/model/'38

TESS\_ORIGINAL\_FOLDER\_PATH = str(working\_dir\_path) + '/sound\_set\_data/'

EXAMPLES\_PATH = str(working\_dir\_path) + '/examples/'

"""

This files creates the X and y features in joblib to be used by the predictive models.

"""

import os

import time

import joblib

import librosa

import numpy as np

from config import SAVE\_DIR\_PATH

from config import TRAINING\_FILES\_PATH

class CreateFeatures:

@staticmethod

def features\_creator(path, save\_dir) -> str:

"""

This function creates the dataset and saves both data and labels in two files, X.joblib

and y.joblib in the joblib\_features folder. With this method, you can persist your

features and train quickly new machine learning models instead of reloading the

featuresevery time with this pipeline.

"""lst = []

start\_time = time.time()

for subdir, dirs, files in os.walk(path):

for file in files:

print(file)

try:

# Load librosa array, obtain mfcss, store the file and the mcss information in a new

array

X, sample\_rate = librosa.load(os.path.join(subdir, file),

res\_type='kaiser\_fast')

mfccs = np.mean(librosa.feature.mfcc(y=X, sr=sample\_rate,

n\_mfcc=40).T, axis=0)

# The instruction below converts the labels (from 1 to 8) to a series from 0 to 7

# This is because our predictor needs to start from 0 otherwise it will try to predict

also 0.

file = str(file[0:3])

print(file)

if file == 'amb':

file = 0

else:

file = 1

print(file)

39arr = mfccs, file

print(arr)

lst.append(arr)

# If the file is not valid, skip it

except ValueError as err:

print(err)

continue

print("--- Data loaded. Loading time: %s seconds ---" % (time.time() - start\_time))

# Creating X and y: zip makes a list of all the first elements, and a list of all the

second elements.

X, y = zip(\*lst)

# Array conversion

X, y = np.asarray(X), np.asarray(y)

# Array shape check

print(X.shape, y.shape)

# Preparing features dump

X\_name, y\_name = 'X.joblib', 'y.joblib'

joblib.dump(X, os.path.join(save\_dir, X\_name))

joblib.dump(y, os.path.join(save\_dir, y\_name))

return "Completed"

if \_\_name\_\_ == '\_\_main\_\_':

print('Routine started')

40FEATURES = CreateFeatures.features\_creator(path=TRAINING\_FILES\_PATH,

save\_dir=SAVE\_DIR\_PATH)

print('Routine completed.')

"""

This file can be used to try a live prediction.

"""

import keras

import librosa

import numpy as np

from config import EXAMPLES\_PATH

from config import MODEL\_DIR\_PATH

class LivePredictions:

"""

Main class of the application.

"""

def \_\_init\_\_(self, file):

"""

Init method is used to initialize the main parameters.

"""

self.file = file

self.path = MODEL\_DIR\_PATH + 'sound\_Model.h5'

41self.loaded\_model = keras.models.load\_model(self.path)

def make\_predictions(self):

"""

Method to process the files and create your features.

"""

data, sampling\_rate = librosa.load(self.file)

mfccs = np.mean(librosa.feature.mfcc(y=data, sr=sampling\_rate, n\_mfcc=40).T,

axis=0)

x = np.expand\_dims(mfccs, axis=2)

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

predictions = self.loaded\_model.predict\_classes(x)

print( "Prediction is", " ", self.convert\_class\_to\_emotion(predictions))

@staticmethod

def convert\_class\_to\_emotion(pred):

"""

Method to convert the predictions (int) into human readable strings.

"""

label\_conversion = {'0': 'ambulance',

'1': 'traffic'

}

for key, value in label\_conversion.items():

if int(key) == pred:

42label = value

return label

if \_\_name\_\_ == '\_\_main\_\_':

live\_prediction = LivePredictions(file=EXAMPLES\_PATH + 'sound\_1.wav')

live\_prediction.loaded\_model.summary()

live\_prediction.make\_predictions()

live\_prediction = LivePredictions(file=EXAMPLES\_PATH + 'sound\_401.wav')

live\_prediction.make\_predictions()

"""

Neural network train file.

"""

import os

import joblib

import numpy as np

import matplotlib.pyplot as plt

from keras.layers import Dense

from keras.layers import Conv1D

from keras.layers import Flatten

from keras.layers import Dropout

from keras.layers import Activation

from keras.models import Sequential

from sklearn.metrics import confusion\_matrix

43from sklearn.metrics import classification\_report

from sklearn.model\_selection import train\_test\_split

from config import SAVE\_DIR\_PATH

from config import MODEL\_DIR\_PATH

class TrainModel:

@staticmethod

def train\_neural\_network(X, y) -> None:

"""

This function trains the neural network.

"""

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.33,

random\_state=42)

x\_traincnn = np.expand\_dims(X\_train, axis=2)

x\_testcnn = np.expand\_dims(X\_test, axis=2)

print(x\_traincnn.shape, x\_testcnn.shape)

model = Sequential()

model.add(Conv1D(64, 5, padding='same',

input\_shape=(40, 1)))

model.add(Activation('relu'))

model.add(Dropout(0.2))

model.add(Flatten())

model.add(Dense(2))

model.add(Activation('softmax'))

44print(model.summary)

model.compile(loss='sparse\_categorical\_crossentropy',

optimizer='rmsprop',

metrics=['accuracy'])

cnn\_history = model.fit(x\_traincnn, y\_train,

batch\_size=16, epochs=50,

validation\_data=(x\_testcnn, y\_test))

# Loss plotting

plt.plot(cnn\_history.history['loss'])

plt.plot(cnn\_history.history['val\_loss'])

plt.title('model loss')

plt.ylabel('loss')

plt.xlabel('epoch')

plt.legend(['train', 'test'], loc='upper left')

plt.savefig('loss.png')

plt.close()

# Accuracy plotting

plt.plot(cnn\_history.history['accuracy'])

plt.plot(cnn\_history.history['val\_accuracy'])

plt.title('model accuracy')

plt.ylabel('acc')

plt.xlabel('epoch')

45plt.legend(['train', 'test'], loc='upper left')

plt.savefig('accuracy.png')

predictions = model.predict\_classes(x\_testcnn)

new\_y\_test = y\_test.astype(int)

matrix = confusion\_matrix(new\_y\_test, predictions)

print(classification\_report(new\_y\_test, predictions))

print(matrix)

model\_name = 'sound\_Model.h5'

# Save model and weights

if not os.path.isdir(MODEL\_DIR\_PATH):

os.makedirs(MODEL\_DIR\_PATH)

model\_path = os.path.join(MODEL\_DIR\_PATH, model\_name)

model.save(model\_path)

print('Saved trained model at %s ' % model\_path)

if \_\_name\_\_ == '\_\_main\_\_':

print('Training started')

print(SAVE\_DIR\_PATH + 'X.joblib')

X = joblib.load(SAVE\_DIR\_PATH + 'X.joblib')

y = joblib.load(SAVE\_DIR\_PATH + 'y.joblib')

NEURAL\_NET = TrainModel.train\_neural\_network(X=X, y=y)

from flask import Flask, render\_template, flash, request, session, send\_file

from flask import render\_template, redirect, url\_for, request

46import warnings

import datetime

app = Flask(\_\_name\_\_)

app.config['DEBUG']

app.config['SECRET\_KEY'] = '7d441f27d441f27567d441f2b6176a'

@app.route("/")

def homepage():

return render\_template('index.html')

@app.route("/Test")

def Test():

return render\_template('Test.html')

@app.route("/ImageTest")

def ImageTest():

return render\_template('ImageTest.html')

@app.route("/imgetest", methods=['GET', 'POST'])

def imgetest():

if request.method == 'POST':

import tensorflow as tf

import numpy as np

from keras.preprocessing import image

file = request.files['fileupload']

file.save('static/upload/Test.jpg')

47fname = 'static/upload/Test.jpg'

import warnings

warnings.filterwarnings('ignore')

classifierLoad = tf.keras.models.load\_model('model.h5')

test\_image = image.load\_img('static/upload/Test.jpg', target\_size=(200, 200))

test\_image = np.expand\_dims(test\_image, axis=0)

result = classifierLoad.predict(test\_image)

print(result[0][0])

res = ''

result1 = ''

if result[0][0] == 1:

res = 'Ambulance'

result1 = 'Emergency'

return render\_template('Test.html', result=res, gry=fname)

elifresult[0][1] == 1:

res = 'Bicycle'

result1 = 'Normal'

elifresult[0][2] == 1:

res = 'Bus'

result1 = 'Normal'

elifresult[0][3] == 1:

res = 'Car'

48result1 = 'Normal'

elifresult[0][4] == 1:

res = 'Motorcycle'

result1 = 'Normal'

elifresult[0][5] == 1:

res = 'Tank'

result1 = 'Normal'

elifresult[0][6] == 1:

res = 'Taxi'

result1 = 'Normal'

elifresult[0][7] == 1:

res = 'Truck'

result1 = 'Normal'

return render\_template('ImageTest.html', result=res, result1=result1, gry=fname)

@app.route("/testsound", methods=['GET', 'POST'])

def testsound():

if request.method == 'POST':

file = request.files['fileupload']

file.save('static/Out/' + file.filename)

live\_prediction = LivePredictions(file='static/Out/' + file.filename)

live\_prediction.loaded\_model.summary()

live\_prediction.make\_predictions()

49res = session["out"]

gry = ''

if res == "ambulance":

res = 'Ambulance'

result1 = 'Emergency'

gry = 'static/emoji/green.jpg'

elif res == "traffic":

res = 'Traffic'

result1 = 'Normal'

gry = 'static/emoji/red.jpg'

return render\_template('Test.html', result=res, result1=result1, gry=gry)

import keras

import librosa

import numpy as np

from config import EXAMPLES\_PATH

from config import MODEL\_DIR\_PATH

class LivePredictions:

"""

Main class of the application.

"""

def \_\_init\_\_(self, file):

"""

50Init method is used to initialize the main parameters.

"""

self.file = file

self.path = MODEL\_DIR\_PATH + 'sound\_Model.h5'

self.loaded\_model = keras.models.load\_model(self.path)

def make\_predictions(self):

"""

Method to process the files and create your features.

"""

data, sampling\_rate = librosa.load(self.file)

mfccs = np.mean(librosa.feature.mfcc(y=data, sr=sampling\_rate, n\_mfcc=40).T,

axis=0)

x = np.expand\_dims(mfccs, axis=2)

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

predictions = self.loaded\_model.predict\_classes(x)

print("Prediction is", " ", self.convert\_class\_to\_emotion(predictions))

session["out"] = self.convert\_class\_to\_emotion(predictions)

@staticmethod

def convert\_class\_to\_emotion(pred):

"""

Method to convert the predictions (int) into human readable strings.

"""

label\_conversion = {'0': 'ambulance',

5152

'1': 'traffic',

}

for key, value in label\_conversion.items():

if int(key) == pred:

label = value

return label

def sendmsg(targetno, message):

import requests

requests.post(

"http://smsserver9.creativepoint.in/api.php?username=fantasy&password=596692&t

o=" + targetno + "&from=FSSMSS&message=Dear user yourmsg is " + message + "

Sent By FSMSG

FSSMSS&PEID=1501563800000030506&templateid=1507162882948811640")

if \_\_name\_\_ == '\_\_main\_\_':

app.run(debug=True, use\_reloader=True)