**Implementation**

Project – Sign Language Recognition using Machine Learning

Name – Priyanka Rokhade

USN – 01FM19MCA014

Source code

**Realtimerecognition.py**

import cv2

import numpy as np

import tensorflow as tf

from tensorflow.keras.preprocessing.image import ImageDataGenerator

import os

import pyttsx3

#engine = pyttsx3.init()

# load saved model from PC

model = tf.keras.models.load\_model(r"new\_model\_20ep3.h5")

model.summary()

data\_dir = 'dataset'

#getting the labels form data directory

labels = sorted(os.listdir(r"data"))

labels[-1] = 'Nothing'

print(labels)

#initiating the video source, 0 for internal camera

cap = cv2.VideoCapture(0)

while(True):

\_ , frame = cap.read()

cv2.rectangle(frame, (100, 100), (300, 300), (0, 0, 255), 5)

#region of intrest

roi = frame[100:300, 100:300]

img = cv2.resize(roi, (50, 50))

cv2.imshow('roi', roi)

img = img/255

#make predication about the current frame

prediction = model.predict(img.reshape(1,50,50,3))

char\_index = np.argmax(prediction)

#print(char\_index,prediction[0,char\_index]\*100)

confidence = round(prediction[0,char\_index]\*100, 1)

predicted\_char = labels[char\_index]

# Initialize the engine

engine = pyttsx3.init()

engine.say(predicted\_char)

engine.runAndWait()

font = cv2.FONT\_HERSHEY\_TRIPLEX

fontScale = 1

color = (0,255,255)

thickness = 2

#writing the predicted char and its confidence percentage to the frame

msg = predicted\_char +', Conf: ' +str(confidence)+' %'

cv2.putText(frame, msg, (80, 80), font, fontScale, color, thickness)

cv2.imshow('frame',frame)

#close the camera when press 'q'

if cv2.waitKey(10) & 0xFF == ord('q'):

break

#release the camera and close all windows

cap.release()

cv2.destroyAllWindows()

**train.ipynb**

{

"cells": [

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"# importing needed libraries"

]

},

{

"cell\_type": "code",

"execution\_count": 1,

"metadata": {},

"outputs": [],

"source": [

"import pandas as pd\n",

"importnumpy as np\n",

"import matplotlib.pyplot as plt\n",

"importtensorflow as tf\n",

"importos\n",

"from tensorflow.keras.preprocessing.image import ImageDataGenerator\n",

"from sklearn.model\_selection import train\_test\_split\n",

"from tensorflow.keras.utils import to\_categorical\n",

"import cv2\n",

"importpydot"

]

},

{

"cell\_type": "markdown",

"metadata": {},

"source": [

"# to allow the code working on GPU instead CPU"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"gpu = tf.config.experimental.list\_physical\_devices('GPU')\n",

"tf.config.experimental.set\_memory\_growth(gpu[0], True)"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"#load\_dataset function to load the data and resize the images to 50x50\n",

"def load\_dataset(directory):\n",

" images = []\n",

" labels = []\n",

" foridx, label in enumerate(uniq\_labels):\n",

" for file in os.listdir(directory + '/'+label):\n",

" filepath = directory +'/'+ label + \"/\" + file\n",

" img = cv2.resize(cv2.imread(filepath),(50,50))\n",

" images.append(img)\n",

" labels.append(idx)\n",

" images = np.asarray(images)\n",

" labels = np.asarray(labels)\n",

" return images, labels"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"#display\_images function to show examples\n",

"def display\_images(x\_data,y\_data, title, display\_label = True):\n",

" x, y = x\_data,y\_data\n",

" fig, axes = plt.subplots(5, 8, figsize = (18, 5))\n",

" fig.subplots\_adjust(hspace = 0.5, wspace = 0.5)\n",

" fig.suptitle(title, fontsize = 18)\n",

" for i, ax in enumerate(axes.flat):\n",

" ax.imshow(cv2.cvtColor(x[i], cv2.COLOR\_BGR2RGB))\n",

" ifdisplay\_label:\n",

" ax.set\_xlabel(uniq\_labels[y[i]])\n",

" ax.set\_xticks([])\n",

" ax.set\_yticks([])\n",

" plt.show()"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"#loading\_dataset into X\_pre and Y\_pre\n",

"data\_dir = r'D:\\yasser\_sulaiman\\ANN project\\new model\\dataset\\train'\n",

"uniq\_labels = sorted(os.listdir(data\_dir))\n",

"X\_pre, Y\_pre = load\_dataset(data\_dir)\n",

"print(X\_pre.shape, Y\_pre.shape)"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"#spliting dataset into 80% train, 10% validation and 10% test data\n",

"X\_train, X\_test, Y\_train, Y\_test = train\_test\_split(X\_pre, Y\_pre, test\_size = 0.8)\n",

"X\_test, X\_eval, Y\_test, Y\_eval = train\_test\_split(X\_test, Y\_test, test\_size = 0.5)"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"%matplotlib notebook"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {

"scrolled": true

},

"outputs": [],

"source": [

"#print shapes and show examples for each set\n",

"print(\"Train images shape\",X\_train.shape, Y\_train.shape)\n",

"print(\"Test images shape\",X\_test.shape, Y\_test.shape)\n",

"print(\"Evaluate image shape\",X\_eval.shape, Y\_eval.shape)\n",

"print(\"Printing the labels\",uniq\_labels, len(uniq\_labels))\n",

"display\_images(X\_train,Y\_train,'Samples from Train Set')\n",

"display\_images(X\_test,Y\_test,'Samples from Test Set')\n",

"display\_images(X\_eval,Y\_eval,'Samples from Validation Set')"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"# converting Y\_tes and Y\_train to One hot vectors using to\_categorical\n",

"# example of one hot => '1' is represented as [0. 1. 0. . . . . 0.]\n",

"Y\_train = to\_categorical(Y\_train)\n",

"Y\_test = to\_categorical(Y\_test)\n",

"Y\_eval = to\_categorical(Y\_eval)\n",

"X\_train = X\_train / 255.\n",

"X\_test = X\_test/ 255.\n",

"X\_eval = X\_eval/ 255."

]

},

{

"cell\_type": "code",

"execution\_count": 2,

"metadata": {},

"outputs": [

{

"name": "stdout",

"output\_type": "stream",

"text": [

"Model: \"sequential\"\n",

"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n",

"Layer (type) Output Shape Param # \n",

"=================================================================\n",

"conv2d (Conv2D) (None, 48, 48, 16) 448 \n",

"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n",

"conv2d\_1 (Conv2D) (None, 46, 46, 16) 2320 \n",

"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n",

"conv2d\_2 (Conv2D) (None, 44, 44, 16) 2320 \n",

"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n",

"max\_pooling2d (MaxPooling2D) (None, 22, 22, 16) 0 \n",

"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n",

"conv2d\_3 (Conv2D) (None, 20, 20, 32) 4640 \n",

"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n",

"conv2d\_4 (Conv2D) (None, 18, 18, 32) 9248 \n",

"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n",

"conv2d\_5 (Conv2D) (None, 16, 16, 32) 9248 \n",

"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n",

"max\_pooling2d\_1 (MaxPooling2 (None, 8, 8, 32) 0 \n",

"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n",

"conv2d\_6 (Conv2D) (None, 6, 6, 64) 18496 \n",

"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n",

"conv2d\_7 (Conv2D) (None, 4, 4, 64) 36928 \n",

"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n",

"conv2d\_8 (Conv2D) (None, 2, 2, 64) 36928 \n",

"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n",

"flatten (Flatten) (None, 256) 0 \n",

"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n",

"dense (Dense) (None, 128) 32896 \n",

"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n",

"dense\_1 (Dense) (None, 25) 3225 \n",

"=================================================================\n",

"Total params: 156,697\n",

"Trainable params: 156,697\n",

"Non-trainable params: 0\n",

"\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n"

]

}

],

"source": [

"# building our model\n",

"model = tf.keras.Sequential([\n",

" tf.keras.layers.Conv2D(16, (3,3), activation ='relu', input\_shape=(50,50,3)),\n",

" tf.keras.layers.Conv2D(16, (3,3), activation ='relu'),\n",

" tf.keras.layers.Conv2D(16, (3,3), activation ='relu'),\n",

" tf.keras.layers.MaxPool2D((2,2)),\n",

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

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

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

" tf.keras.layers.MaxPool2D((2,2)),\n",

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

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

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

" tf.keras.layers.Flatten(),\n",

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

" tf.keras.layers.Dense(25, activation='softmax')\n",

"])\n",

"\n",

"model.summary()"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"#compiling the model\n",

"#default batch size 32\n",

"#default learning rate is 0.001\n",

"model.compile(optimizer = 'adam',\n",

" loss = 'categorical\_crossentropy',\n",

" metrics=['accuracy'],)"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"#start training(fitting) the data\n",

"history = model.fit(X\_train, Y\_train, epochs=20, verbose=1,\n",

" validation\_data=(X\_eval, Y\_eval))"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"#testing\n",

"model.evaluate(X\_test, Y\_test)"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"#save the model\n",

"model.save(r'D:\\yasser\_sulaiman\\ANN project\\new model\\new\_model\_40ep.h5')"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"train\_loss = history.history['loss']\n",

"train\_acc = history.history['accuracy']\n",

"val\_loss = history.history['val\_loss']\n",

"val\_accuracy = history.history['val\_accuracy']\n"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": []

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"#ploting training and validation loss vs. epochs\n",

"%matplotlib notebook\n",

"epochs = list(range(1,41))\n",

"plt.plot(epochs, train\_loss, label = \"training loss\")\n",

"plt.plot(epochs, val\_loss, label = \"validation loss\")\n",

"plt.legend()\n",

"plt.show()"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": [

"#ploting training and validation accuracy vs. epochs\n",

"%matplotlib notebook\n",

"plt.plot(epochs, train\_acc, label = \"training accuracy\")\n",

"plt.plot(epochs, val\_accuracy, label = \"validation accuracy\")\n",

"plt.legend()\n",

"plt.show()"

]

},

{

"cell\_type": "code",

"execution\_count": null,

"metadata": {},

"outputs": [],

"source": []

}

],

"metadata": {

"kernelspec": {

"display\_name": "Python 3",

"language": "python",

"name": "python3"

},

"language\_info": {

"codemirror\_mode": {

"name": "ipython",

"version": 3

},

"file\_extension": ".py",

"mimetype": "text/x-python",

"name": "python",

"nbconvert\_exporter": "python",

"pygments\_lexer": "ipython3",

"version": "3.7.9"

}

},

"nbformat": 4,

"nbformat\_minor": 5

}