CNN

Data Set Creation Code

import os

import cv2

directory= 'SignImage48x48/'

print(os.getcwd())

if not os.path.exists(directory):

os.mkdir(directory)

if not os.path.exists(f'{directory}/blank'):

os.mkdir(f'{directory}/blank')

for i in range(65,91):

letter = chr(i)

if not os.path.exists(f'{directory}/{letter}'):

os.mkdir(f'{directory}/{letter}')

cap=cv2.VideoCapture(0)

while True:

\_,frame=cap.read()

count = {

'a': len(os.listdir(directory+"/A")),

'b': len(os.listdir(directory+"/B")),

'c': len(os.listdir(directory+"/C")),

'd': len(os.listdir(directory+"/D")),

'e': len(os.listdir(directory+"/E")),

'f': len(os.listdir(directory+"/F")),

'g': len(os.listdir(directory+"/G")),

'h': len(os.listdir(directory+"/H")),

'i': len(os.listdir(directory+"/I")),

'j': len(os.listdir(directory+"/J")),

'k': len(os.listdir(directory+"/K")),

'l': len(os.listdir(directory+"/L")),

'm': len(os.listdir(directory+"/M")),

'n': len(os.listdir(directory+"/N")),

'o': len(os.listdir(directory+"/O")),

'p': len(os.listdir(directory+"/P")),

'q': len(os.listdir(directory+"/Q")),

'r': len(os.listdir(directory+"/R")),

's': len(os.listdir(directory+"/S")),

't': len(os.listdir(directory+"/T")),

'u': len(os.listdir(directory+"/U")),

'v': len(os.listdir(directory+"/V")),

'w': len(os.listdir(directory+"/W")),

'x': len(os.listdir(directory+"/X")),

'y': len(os.listdir(directory+"/Y")),

'z': len(os.listdir(directory+"/Z")),

'blank': len(os.listdir(directory+"/blank"))

}

row = frame.shape[1]

col = frame.shape[0]

cv2.rectangle(frame,(0,40),(300,300),(255,255,255),2)

cv2.imshow("data",frame)

frame=frame[40:300,0:300]

cv2.imshow("ROI",frame)

frame = cv2.cvtColor(frame,cv2.COLOR\_BGR2GRAY)

frame = cv2.resize(frame,(48,48))

interrupt = cv2.waitKey(10)

if interrupt & 0xFF == ord('a'):

cv2.imwrite(os.path.join(directory+'A/'+str(count['a']))+'.jpg',frame)

if interrupt & 0xFF == ord('b'):

cv2.imwrite(os.path.join(directory+'B/'+str(count['b']))+'.jpg',frame)

if interrupt & 0xFF == ord('c'):

cv2.imwrite(os.path.join(directory+'C/'+str(count['c']))+'.jpg',frame)

if interrupt & 0xFF == ord('d'):

cv2.imwrite(os.path.join(directory+'D/'+str(count['d']))+'.jpg',frame)

if interrupt & 0xFF == ord('e'):

cv2.imwrite(os.path.join(directory+'E/'+str(count['e']))+'.jpg',frame)

if interrupt & 0xFF == ord('f'):

cv2.imwrite(os.path.join(directory+'F/'+str(count['f']))+'.jpg',frame)

if interrupt & 0xFF == ord('g'):

cv2.imwrite(os.path.join(directory+'G/'+str(count['g']))+'.jpg',frame)

if interrupt & 0xFF == ord('h'):

cv2.imwrite(os.path.join(directory+'H/'+str(count['h']))+'.jpg',frame)

if interrupt & 0xFF == ord('i'):

cv2.imwrite(os.path.join(directory+'I/'+str(count['i']))+'.jpg',frame)

if interrupt & 0xFF == ord('j'):

cv2.imwrite(os.path.join(directory+'J/'+str(count['j']))+'.jpg',frame)

if interrupt & 0xFF == ord('k'):

cv2.imwrite(os.path.join(directory+'K/'+str(count['k']))+'.jpg',frame)

if interrupt & 0xFF == ord('l'):

cv2.imwrite(os.path.join(directory+'L/'+str(count['l']))+'.jpg',frame)

if interrupt & 0xFF == ord('m'):

cv2.imwrite(os.path.join(directory+'M/'+str(count['m']))+'.jpg',frame)

if interrupt & 0xFF == ord('n'):

cv2.imwrite(os.path.join(directory+'N/'+str(count['n']))+'.jpg',frame)

if interrupt & 0xFF == ord('o'):

cv2.imwrite(os.path.join(directory+'O/'+str(count['o']))+'.jpg',frame)

if interrupt & 0xFF == ord('p'):

cv2.imwrite(os.path.join(directory+'P/'+str(count['p']))+'.jpg',frame)

if interrupt & 0xFF == ord('q'):

cv2.imwrite(os.path.join(directory+'Q/'+str(count['q']))+'.jpg',frame)

if interrupt & 0xFF == ord('r'):

cv2.imwrite(os.path.join(directory+'R/'+str(count['r']))+'.jpg',frame)

if interrupt & 0xFF == ord('s'):

cv2.imwrite(os.path.join(directory+'S/'+str(count['s']))+'.jpg',frame)

if interrupt & 0xFF == ord('t'):

cv2.imwrite(os.path.join(directory+'T/'+str(count['t']))+'.jpg',frame)

if interrupt & 0xFF == ord('u'):

cv2.imwrite(os.path.join(directory+'U/'+str(count['u']))+'.jpg',frame)

if interrupt & 0xFF == ord('v'):

cv2.imwrite(os.path.join(directory+'V/'+str(count['v']))+'.jpg',frame)

if interrupt & 0xFF == ord('w'):

cv2.imwrite(os.path.join(directory+'W/'+str(count['w']))+'.jpg',frame)

if interrupt & 0xFF == ord('x'):

cv2.imwrite(os.path.join(directory+'X/'+str(count['x']))+'.jpg',frame)

if interrupt & 0xFF == ord('y'):

cv2.imwrite(os.path.join(directory+'Y/'+str(count['y']))+'.jpg',frame)

if interrupt & 0xFF == ord('z'):

cv2.imwrite(os.path.join(directory+'Z/'+str(count['z']))+'.jpg',frame)

if interrupt & 0xFF == ord('.'):

cv2.imwrite(os.path.join(directory+'blank/' + str(count['blank']))+ '.jpg',frame)

CNN-MODEL CREATION& TRAINING -GOOGLE COLAB

from keras.utils import to\_categorical

from keras.models import Sequential

from keras.layers import Dense, Conv2D, Dropout, Flatten, MaxPooling2D

from keras.preprocessing.image import ImageDataGenerator

from keras.callbacks import TensorBoard

import os

train\_datagen=ImageDataGenerator(

    rescale=1./255,

)

val\_datagen=ImageDataGenerator(rescale=1./255)

batch\_size=128

DATASET UPLOADED IN GOOGLE DRIVE AND TAKEN FROM THERE

train\_datagen=ImageDataGenerator(

    rescale=1./255,

)

val\_datagen=ImageDataGenerator(rescale=1./255)

batch\_size=128

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

    '/content/drive/MyDrive/SignLanguage/Splitdataset48x48/train',

    target\_size=(48,48),

    batch\_size=batch\_size,

    class\_mode='categorical',

    color\_mode='grayscale'

)

validation\_generator=val\_datagen.flow\_from\_directory(

    '/content/drive/MyDrive/SignLanguage/Splitdataset48x48/val',

    target\_size=(48,48),

    batch\_size=128,

    class\_mode='categorical',

    color\_mode='grayscale'

class\_names=list(train\_generator.class\_indices.keys())

MODEL  
model=Sequential()

#convulational layer

model.add(Conv2D(64,kernel\_size=(3,3),activation='relu',input\_shape=(48,48,1)))

model.add(MaxPooling2D(pool\_size=(2,2)))

model.add(Dropout(0.2))

model.add(Conv2D(128,kernel\_size=(3,3),activation='relu'))

model.add(MaxPooling2D(pool\_size=(2,2)))

model.add(Dropout(0.2))

model.add(Conv2D(512,kernel\_size=(3,3),activation='relu'))

model.add(MaxPooling2D(pool\_size=(2,2)))

model.add(Flatten())

#connected layer

model.add(Dense(512,activation='relu'))

model.add(Dropout(0.2))

model.add(Dense(256,activation='relu'))

model.add(Dropout(0.2))

model.add(Dense(64,activation='relu'))

model.add(Dropout(0.2))

model.add(Dense(27,activation='softmax'))

model.compile(optimizer='adam',loss='categorical\_crossentropy',metrics='accuracy')

!rm=rf Logs

logdir=os.path.join("Logs")

TensorBoard\_callback=TensorBoard(log\_dir=logdir)

model.fit(

    train\_generator,

    steps\_per\_epoch=train\_generator.samples // batch\_size,

    epochs=100,

    validation\_data=validation\_generator,

    validation\_steps=validation\_generator.samples // batch\_size,

    callbacks=[TensorBoard\_callback]

)

predictions=model.predict(validation\_generator)

MODEL SAVED AS JSON FILE

model\_json = model.to\_json()

with open("/content/drive/MyDrive/signlanguagedetectionmodel48x48.json", 'a') as json\_file:

    json\_file.write(model\_json)  # Append to existing content

model.save("/content/drive/MyDrive/signlanguagedetectionmodel48x48.h5")

REAL-TIME PREDICTION

from keras.models import model\_from\_json

import cv2

import numpy as np

json\_file = open("signlanguagedetectionmodel48x48main.json", "r")

model\_json = json\_file.read()

json\_file.close()

model = model\_from\_json(model\_json)

model.load\_weights("signlanguagedetectionmodel48x48main.h5")

def extract\_features(image):

    feature = np.array(image)

    feature = feature.reshape(1,48,48,1)

    return feature/255.0

cap = cv2.VideoCapture(0)

label = ['A','B','C','D','E','F','G','H','I','J','K','L','M','N','O','P','Q','R','S','T','U','V','W','X','Y','Z','blank']

while True:

    \_,frame = cap.read()

    cv2.rectangle(frame,(0,40),(300,300),(0, 165, 255),1)

    cropframe=frame[40:300,0:300]

    cropframe=cv2.cvtColor(cropframe,cv2.COLOR\_BGR2GRAY)

    cropframe = cv2.resize(cropframe,(48,48))

    cropframe = extract\_features(cropframe)

    pred = model.predict(cropframe)

    prediction\_label = label[pred.argmax()]

    cv2.rectangle(frame, (0,0), (300, 40), (0, 165, 255), -1)

    if prediction\_label == 'blank':

        cv2.putText(frame, " ", (10, 30),cv2.FONT\_HERSHEY\_SIMPLEX,1, (255, 255, 255),2,cv2.LINE\_AA)

    else:

        accu = "{:.2f}".format(np.max(pred)\*100)

        cv2.putText(frame, f'{prediction\_label}  {accu}%', (10, 30),cv2.FONT\_HERSHEY\_SIMPLEX,1, (255, 255, 255),2,cv2.LINE\_AA)

    cv2.imshow("output",frame)

    cv2.waitKey(27)

cap.release()

cv2.destroyAllWindows()