**SET HAND HIST**  
import cv2

import numpy as np

import pickle

def build\_squares(img):

x, y, w, h = 420, 140, 10, 10

d = 10

imgCrop = None

for i in range(10):

for j in range(5):

if np.any(imgCrop == None):

imgCrop = img[y:y+h, x:x+w]

else:

imgCrop = np.vstack((imgCrop, img[y:y+h, x:x+w]))

cv2.rectangle(img, (x,y), (x+w, y+h), (0,255,0), 2)

x+=w+d

x = 420

y+=h+d

return imgCrop

def get\_hand\_hist():

cam = cv2.VideoCapture(1)

if cam.read()[0]==False:

cam = cv2.VideoCapture(0)

x, y, w, h = 300, 100, 300, 300

flagPressedC, flagPressedS = False, False

imgCrop = None

while True:

img = cam.read()[1]

img = cv2.flip(img, 1)

hsv = cv2.cvtColor(img, cv2.COLOR\_BGR2HSV)

keypress = cv2.waitKey(1)

if keypress == ord('c'):

hsvCrop = cv2.cvtColor(imgCrop, cv2.COLOR\_BGR2HSV)

flagPressedC = True

hist = cv2.calcHist([hsvCrop], [0, 1], None, [180, 256], [0, 180, 0, 256])

cv2.normalize(hist, hist, 0, 255, cv2.NORM\_MINMAX)

elif keypress == ord('s'):

flagPressedS = True

break

if flagPressedC:

dst = cv2.calcBackProject([hsv], [0, 1], hist, [0, 180, 0, 256], 1)

disc = cv2.getStructuringElement(cv2.MORPH\_ELLIPSE,(10,10))

cv2.filter2D(dst,-1,disc,dst)

blur = cv2.GaussianBlur(dst, (11,11), 0)

blur = cv2.medianBlur(blur, 15)

ret,thresh = cv2.threshold(blur,0,255,cv2.THRESH\_BINARY+cv2.THRESH\_OTSU)

thresh = cv2.merge((thresh,thresh,thresh))

res = cv2.bitwise\_and(img,thresh)

#cv2.imshow("res", res)

cv2.imshow("Thresh", thresh)

if not flagPressedS:

imgCrop = build\_squares(img)

#cv2.rectangle(img, (x,y), (x+w, y+h), (0,255,0), 2)

cv2.imshow("Set hand histogram", img)

cam.release()

cv2.destroyAllWindows()

with open("hist", "wb") as f:

pickle.dump(hist, f)

get\_hand\_hist()  
**RECOGNIZE GEST**

import cv2, pickle

import numpy as np

import tensorflow as tf

from cnn\_tf import cnn\_model\_fn

import os

import sqlite3

from keras.models import load\_model

os.environ['TF\_CPP\_MIN\_LOG\_LEVEL'] = '3'

tf.logging.set\_verbosity(tf.logging.ERROR)

classifier = tf.estimator.Estimator(model\_dir="tmp/cnn\_model2", model\_fn=cnn\_model\_fn)

prediction = None

model = load\_model('cnn\_model\_keras2.h5')

def get\_image\_size():

img = cv2.imread('gestures/0/100.jpg', 0)

return img.shape

image\_x, image\_y = get\_image\_size()

def tf\_process\_image(img):

img = cv2.resize(img, (image\_x, image\_y))

img = np.array(img, dtype=np.float32)

np\_array = np.array(img)

return np\_array

def tf\_predict(classifier, image):

'''

need help with prediction using tensorflow

'''

global prediction

processed\_array = tf\_process\_image(image)

pred\_input\_fn = tf.estimator.inputs.numpy\_input\_fn(x={"x":processed\_array}, shuffle=False)

pred = classifier.predict(input\_fn=pred\_input\_fn)

prediction = next(pred)

print(prediction)

def keras\_process\_image(img):

img = cv2.resize(img, (image\_x, image\_y))

img = np.array(img, dtype=np.float32)

img = np.reshape(img, (1, image\_x, image\_y, 1))

return img

def keras\_predict(model, image):

processed = keras\_process\_image(image)

pred\_probab = model.predict(processed)[0]

pred\_class = list(pred\_probab).index(max(pred\_probab))

return max(pred\_probab), pred\_class

def get\_pred\_text\_from\_db(pred\_class):

conn = sqlite3.connect("gesture\_db.db")

cmd = "SELECT g\_name FROM gesture WHERE g\_id="+str(pred\_class)

cursor = conn.execute(cmd)

for row in cursor:

return row[0]

def split\_sentence(text, num\_of\_words):

'''

Splits a text into group of num\_of\_words

'''

list\_words = text.split(" ")

length = len(list\_words)

splitted\_sentence = []

b\_index = 0

e\_index = num\_of\_words

while length > 0:

part = ""

for word in list\_words[b\_index:e\_index]:

part = part + " " + word

splitted\_sentence.append(part)

b\_index += num\_of\_words

e\_index += num\_of\_words

length -= num\_of\_words

return splitted\_sentence

def put\_splitted\_text\_in\_blackboard(blackboard, splitted\_text):

y = 200

for text in splitted\_text:

cv2.putText(blackboard, text, (4, y), cv2.FONT\_HERSHEY\_TRIPLEX, 2, (255, 255, 255))

y += 50

def get\_hand\_hist():

with open("hist", "rb") as f:

hist = pickle.load(f)

return hist

def recognize():

global prediction

cam = cv2.VideoCapture(1)

if cam.read()[0] == False:

cam = cv2.VideoCapture(0)

hist = get\_hand\_hist()

x, y, w, h = 300, 100, 300, 300

while True:

text = ""

img = cam.read()[1]

img = cv2.flip(img, 1)

imgCrop = img[y:y+h, x:x+w]

imgHSV = cv2.cvtColor(img, cv2.COLOR\_BGR2HSV)

dst = cv2.calcBackProject([imgHSV], [0, 1], hist, [0, 180, 0, 256], 1)

disc = cv2.getStructuringElement(cv2.MORPH\_ELLIPSE,(10,10))

cv2.filter2D(dst,-1,disc,dst)

blur = cv2.GaussianBlur(dst, (11,11), 0)

blur = cv2.medianBlur(blur, 15)

thresh = cv2.threshold(blur,0,255,cv2.THRESH\_BINARY+cv2.THRESH\_OTSU)[1]

thresh = cv2.merge((thresh,thresh,thresh))

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

thresh = thresh[y:y+h, x:x+w]

contours = cv2.findContours(thresh.copy(), cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_NONE)[1]

if len(contours) > 0:

contour = max(contours, key = cv2.contourArea)

#print(cv2.contourArea(contour))

if cv2.contourArea(contour) > 10000:

x1, y1, w1, h1 = cv2.boundingRect(contour)

save\_img = thresh[y1:y1+h1, x1:x1+w1]

if w1 > h1:

save\_img = cv2.copyMakeBorder(save\_img, int((w1-h1)/2) , int((w1-h1)/2) , 0, 0, cv2.BORDER\_CONSTANT, (0, 0, 0))

elif h1 > w1:

save\_img = cv2.copyMakeBorder(save\_img, 0, 0, int((h1-w1)/2) , int((h1-w1)/2) , cv2.BORDER\_CONSTANT, (0, 0, 0))

pred\_probab, pred\_class = keras\_predict(model, save\_img)

print(pred\_class, pred\_probab)

if pred\_probab\*100 > 80:

text = get\_pred\_text\_from\_db(pred\_class)

print(text)

blackboard = np.zeros((480, 640, 3), dtype=np.uint8)

splitted\_text = split\_sentence(text, 2)

put\_splitted\_text\_in\_blackboard(blackboard, splitted\_text)

#cv2.putText(blackboard, text, (30, 200), cv2.FONT\_HERSHEY\_TRIPLEX, 1.3, (255, 255, 255))

cv2.rectangle(img, (x,y), (x+w, y+h), (0,255,0), 2)

res = np.hstack((img, blackboard))

cv2.imshow("Recognizing gesture", res)

cv2.imshow("thresh", thresh)

if cv2.waitKey(1) == ord('q'):

break

keras\_predict(model, np.zeros((50, 50), dtype=np.uint8))

recognize()

**FINALWITHVOICE.PY**  
import cv2, pickle

import numpy as np

import tensorflow as tf

from cnn\_tf import cnn\_model\_fn

import os

import sqlite3, pyttsx3

from keras.models import load\_model

from threading import Thread

engine = pyttsx3.init()

engine.setProperty('rate', 150)

os.environ['TF\_CPP\_MIN\_LOG\_LEVEL'] = '3'

model = load\_model('cnn\_model\_keras2.h5')

def get\_hand\_hist():

with open("hist", "rb") as f:

hist = pickle.load(f)

return hist

def get\_image\_size():

img = cv2.imread('gestures/0/100.jpg', 0)

return img.shape

image\_x, image\_y = get\_image\_size()

def keras\_process\_image(img):

img = cv2.resize(img, (image\_x, image\_y))

img = np.array(img, dtype=np.float32)

img = np.reshape(img, (1, image\_x, image\_y, 1))

return img

def keras\_predict(model, image):

processed = keras\_process\_image(image)

pred\_probab = model.predict(processed)[0]

pred\_class = list(pred\_probab).index(max(pred\_probab))

return max(pred\_probab), pred\_class

def get\_pred\_text\_from\_db(pred\_class):

conn = sqlite3.connect("gesture\_db.db")

cmd = "SELECT g\_name FROM gesture WHERE g\_id="+str(pred\_class)

cursor = conn.execute(cmd)

for row in cursor:

return row[0]

def get\_pred\_from\_contour(contour, thresh):

x1, y1, w1, h1 = cv2.boundingRect(contour)

save\_img = thresh[y1:y1+h1, x1:x1+w1]

text = ""

if w1 > h1:

save\_img = cv2.copyMakeBorder(save\_img, int((w1-h1)/2) , int((w1-h1)/2) , 0, 0, cv2.BORDER\_CONSTANT, (0, 0, 0))

elif h1 > w1:

save\_img = cv2.copyMakeBorder(save\_img, 0, 0, int((h1-w1)/2) , int((h1-w1)/2) , cv2.BORDER\_CONSTANT, (0, 0, 0))

pred\_probab, pred\_class = keras\_predict(model, save\_img)

if pred\_probab\*100 > 70:

text = get\_pred\_text\_from\_db(pred\_class)

return text

hist = get\_hand\_hist()

x, y, w, h = 300, 100, 300, 300

is\_voice\_on = True

def get\_img\_contour\_thresh(img):

img = cv2.flip(img, 1)

imgHSV = cv2.cvtColor(img, cv2.COLOR\_BGR2HSV)

dst = cv2.calcBackProject([imgHSV], [0, 1], hist, [0, 180, 0, 256], 1)

disc = cv2.getStructuringElement(cv2.MORPH\_ELLIPSE,(10,10))

cv2.filter2D(dst,-1,disc,dst)

blur = cv2.GaussianBlur(dst, (11,11), 0)

blur = cv2.medianBlur(blur, 15)

thresh = cv2.threshold(blur,0,255,cv2.THRESH\_BINARY+cv2.THRESH\_OTSU)[1]

thresh = cv2.merge((thresh,thresh,thresh))

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

thresh = thresh[y:y+h, x:x+w]

contours = cv2.findContours(thresh.copy(), cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_NONE)[1]

return img, contours, thresh

def say\_text(text):

if not is\_voice\_on:

return

while engine.\_inLoop:

pass

engine.say(text)

engine.runAndWait()

def text\_mode(cam):

global is\_voice\_on

text = ""

word = ""

count\_same\_frame = 0

while True:

img = cam.read()[1]

img, contours, thresh = get\_img\_contour\_thresh(img)

old\_text = text

if len(contours) > 0:

contour = max(contours, key = cv2.contourArea)

if cv2.contourArea(contour) > 10000:

text = get\_pred\_from\_contour(contour, thresh)

if old\_text == text:

count\_same\_frame += 1

else:

count\_same\_frame = 0

if count\_same\_frame > 20:

if len(text) == 1:

Thread(target=say\_text, args=(text, )).start()

word = word + text

if word.startswith('I/Me '):

word = word.replace('I/Me ', 'I ')

elif word.endswith('I/Me '):

word = word.replace('I/Me ', 'me ')

count\_same\_frame = 0

elif cv2.contourArea(contour) < 1000:

if word != '':

#print('yolo')

#say\_text(text)

Thread(target=say\_text, args=(word, )).start()

text = ""

word = ""

else:

if word != '':

#print('yolo1')

#say\_text(text)

Thread(target=say\_text, args=(word, )).start()

text = ""

word = ""

blackboard = np.zeros((480, 640, 3), dtype=np.uint8)

cv2.putText(blackboard, "Text Mode", (180, 50), cv2.FONT\_HERSHEY\_TRIPLEX, 1.5, (255, 0,0))

cv2.putText(blackboard, "Predicted text- " + text, (30, 100), cv2.FONT\_HERSHEY\_TRIPLEX, 1, (255, 255, 0))

cv2.putText(blackboard, word, (30, 240), cv2.FONT\_HERSHEY\_TRIPLEX, 2, (255, 255, 255))

if is\_voice\_on:

cv2.putText(blackboard, "Voice on", (450, 440), cv2.FONT\_HERSHEY\_TRIPLEX, 1, (255, 127, 0))

else:

cv2.putText(blackboard, "Voice off", (450, 440), cv2.FONT\_HERSHEY\_TRIPLEX, 1, (255, 127, 0))

cv2.rectangle(img, (x,y), (x+w, y+h), (0,255,0), 2)

res = np.hstack((img, blackboard))

cv2.imshow("Recognizing gesture", res)

cv2.imshow("thresh", thresh)

keypress = cv2.waitKey(1)

if keypress == ord('q') or keypress == ord('c'):

break

if keypress == ord('v') and is\_voice\_on:

is\_voice\_on = False

elif keypress == ord('v') and not is\_voice\_on:

is\_voice\_on = True

if keypress == ord('c'):

return 2

else:

return 0

def recognize():

cam = cv2.VideoCapture(1)

if cam.read()[0]==False:

cam = cv2.VideoCapture(0)

text = ""

word = ""

count\_same\_frame = 0

keypress = 1

while True:

if keypress == 1:

keypress = text\_mode(cam)

#elif keypress == 2:

else:

break

keras\_predict(model, np.zeros((50, 50), dtype = np.uint8))

recognize()

**LOAD\_IMAGE**  
  
import cv2, os

import numpy as np

import random

from sklearn.utils import shuffle

import pickle

def pickle\_images\_labels():

gest\_folder = "gestures"

images\_labels = []

images = []

labels = []

for g\_id in os.listdir(gest\_folder):

for i in range(1200):

img = cv2.imread(gest\_folder+"/"+g\_id+"/"+str(i+1)+".jpg", 0)

if np.any(img == None):

continue

images\_labels.append((np.array(img, dtype=np.float32), int(g\_id)))

return images\_labels

def split\_images\_labels(images\_labels):

images = []

labels = []

for (image, label) in images\_labels:

images.append(image)

labels.append(label)

return images, labels

images\_labels = pickle\_images\_labels()

images\_labels = shuffle(shuffle(shuffle(images\_labels)))

images, labels = split\_images\_labels(images\_labels)

print("Length of images\_labels", len(images\_labels))

train\_images = images[:int(5/6\*len(images))]

print("Length of train\_images", len(train\_images))

with open("train\_images", "wb") as f:

pickle.dump(train\_images, f)

del train\_images

train\_labels = labels[:int(5/6\*len(labels))]

print("Length of train\_labels", len(train\_labels))

with open("train\_labels", "wb") as f:

pickle.dump(train\_labels, f)

del train\_labels

test\_images = images[int(5/6\*len(images)):]

print("Length of test\_images", len(test\_images))

with open("test\_images", "wb") as f:

pickle.dump(test\_images, f)

del test\_images

test\_labels = labels[int(5/6\*len(labels)):]

print("Length of test\_labels", len(test\_labels))

with open("test\_labels", "wb") as f:

pickle.dump(test\_labels, f)

del test\_labels

**DISPLAY GEST**

import cv2, os, random

import numpy as np

def get\_image\_size():

img = cv2.imread('gestures/0/100.jpg', 0)

return img.shape

gestures = os.listdir('gestures/')

gestures.sort(key = int)

begin\_index = 0

end\_index = 5

image\_x, image\_y = get\_image\_size()

if len(gestures)%5 != 0:

rows = int(len(gestures)/5)+1

else:

rows = int(len(gestures)/5)

full\_img = None

for i in range(rows):

col\_img = None

for j in range(begin\_index, end\_index):

img\_path = "gestures/%s/%d.jpg" % (j, random.randint(1, 1200))

img = cv2.imread(img\_path, 0)

if np.any(img == None):

img = np.zeros((image\_y, image\_x), dtype = np.uint8)

if np.any(col\_img == None):

col\_img = img

else:

col\_img = np.hstack((col\_img, img))

begin\_index += 5

end\_index += 5

if np.any(full\_img == None):

full\_img = col\_img

else:

full\_img = np.vstack((full\_img, col\_img))

cv2.imshow("gestures", full\_img)

cv2.waitKey(0)  
  
  
**CREATE GESTURE**  
  
  
import cv2

import numpy as np

import pickle, os, sqlite3

image\_x, image\_y = 50, 50

def get\_hand\_hist():

with open("hist", "rb") as f:

hist = pickle.load(f)

return hist

def init\_create\_folder\_database():

# create the folder and database if not exist

if not os.path.exists("gestures"):

os.mkdir("gestures")

if not os.path.exists("gesture\_db.db"):

conn = sqlite3.connect("gesture\_db.db")

create\_table\_cmd = "CREATE TABLE gesture ( g\_id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT UNIQUE, g\_name TEXT NOT NULL )"

conn.execute(create\_table\_cmd)

conn.commit()

def create\_folder(folder\_name):

if not os.path.exists(folder\_name):

os.mkdir(folder\_name)

def create\_empty\_images(folder\_name, n\_images):

create\_folder("gestures/"+folder\_name)

black = np.zeros(shape=(image\_x, image\_y, 1), dtype=np.uint8)

for i in range(n\_images):

cv2.imwrite("gestures/"+folder\_name+"/"+str(i+1)+".jpg", black)

def store\_in\_db(g\_id, g\_name):

conn = sqlite3.connect("gesture\_db.db")

cmd = "INSERT INTO gesture (g\_id, g\_name) VALUES (%s, \'%s\')" % (g\_id, g\_name)

try:

conn.execute(cmd)

except sqlite3.IntegrityError:

choice = input("g\_id already exists. Want to change the record? (y/n): ")

if choice.lower() == 'y':

cmd = "UPDATE gesture SET g\_name = \'%s\' WHERE g\_id = %s" % (g\_name, g\_id)

conn.execute(cmd)

else:

print("Doing nothing...")

return

conn.commit()

def store\_images(g\_id):

total\_pics = 1200

if g\_id == str(0):

create\_empty\_images("0", total\_pics)

return

hist = get\_hand\_hist()

cam = cv2.VideoCapture(1)

if cam.read()[0]==False:

cam = cv2.VideoCapture(0)

x, y, w, h = 300, 100, 300, 300

create\_folder("gestures/"+str(g\_id))

pic\_no = 0

flag\_start\_capturing = False

frames = 0

while True:

img = cam.read()[1]

img = cv2.flip(img, 1)

imgHSV = cv2.cvtColor(img, cv2.COLOR\_BGR2HSV)

dst = cv2.calcBackProject([imgHSV], [0, 1], hist, [0, 180, 0, 256], 1)

disc = cv2.getStructuringElement(cv2.MORPH\_ELLIPSE,(10,10))

cv2.filter2D(dst,-1,disc,dst)

blur = cv2.GaussianBlur(dst, (11,11), 0)

blur = cv2.medianBlur(blur, 15)

thresh = cv2.threshold(blur,0,255,cv2.THRESH\_BINARY+cv2.THRESH\_OTSU)[1]

thresh = cv2.merge((thresh,thresh,thresh))

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

thresh = thresh[y:y+h, x:x+w]

contours = cv2.findContours(thresh.copy(), cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_NONE)[1]

if len(contours) > 0:

contour = max(contours, key = cv2.contourArea)

if cv2.contourArea(contour) > 10000 and frames > 50:

x1, y1, w1, h1 = cv2.boundingRect(contour)

pic\_no += 1

save\_img = thresh[y1:y1+h1, x1:x1+w1]

if w1 > h1:

save\_img = cv2.copyMakeBorder(save\_img, int((w1-h1)/2) , int((w1-h1)/2) , 0, 0, cv2.BORDER\_CONSTANT, (0, 0, 0))

elif h1 > w1:

save\_img = cv2.copyMakeBorder(save\_img, 0, 0, int((h1-w1)/2) , int((h1-w1)/2) , cv2.BORDER\_CONSTANT, (0, 0, 0))

save\_img = cv2.resize(save\_img, (image\_x, image\_y))

cv2.putText(img, "Capturing...", (30, 60), cv2.FONT\_HERSHEY\_TRIPLEX, 2, (127, 255, 255))

cv2.imwrite("gestures/"+str(g\_id)+"/"+str(pic\_no)+".jpg", save\_img)

cv2.rectangle(img, (x,y), (x+w, y+h), (0,255,0), 2)

cv2.putText(img, str(pic\_no), (30, 400), cv2.FONT\_HERSHEY\_TRIPLEX, 1.5, (127, 127, 255))

cv2.imshow("Capturing gesture", img)

cv2.imshow("thresh", thresh)

keypress = cv2.waitKey(1)

if keypress == ord('c'):

if flag\_start\_capturing == False:

flag\_start\_capturing = True

else:

flag\_start\_capturing = False

frames = 0

if flag\_start\_capturing == True:

frames += 1

if pic\_no == total\_pics:

break

init\_create\_folder\_database()

g\_id = input("Enter gesture no.: ")

g\_name = input("Enter gesture name/text: ")

store\_in\_db(g\_id, g\_name)

store\_images(g\_id)

**CNN TF .PY**import tensorflow as tf

import numpy as np

import pickle, os, cv2

tf.logging.set\_verbosity(tf.logging.INFO)

os.environ['TF\_CPP\_MIN\_LOG\_LEVEL'] = '3'

def get\_image\_size():

img = cv2.imread('gestures/0/100.jpg', 0)

return img.shape

def get\_num\_of\_classes():

return len(os.listdir('gestures/'))

image\_x, image\_y = get\_image\_size()

def cnn\_model\_fn(features, labels, mode):

input\_layer = tf.reshape(features["x"], [-1, image\_x, image\_y, 1], name="input")

conv1 = tf.layers.conv2d(

inputs=input\_layer,

filters=32,

kernel\_size=[5, 5],

padding="same",

activation=tf.nn.relu,

name="conv1")

print("conv1",conv1.shape)

pool1 = tf.layers.max\_pooling2d(inputs=conv1, pool\_size=[2, 2], strides=2, name="pool1")

print("pool1",pool1.shape)

conv2 = tf.layers.conv2d(

inputs=pool1,

filters=64,

kernel\_size=[5, 5],

padding="same",

activation=tf.nn.relu,

name="conv2")

print("conv2",conv2.shape)

pool2 = tf.layers.max\_pooling2d(inputs=conv2, pool\_size=[5, 5], strides=5, name="pool2")

print("pool2",pool2.shape)

# Dense Layer

pool2\_flat = tf.reshape(pool2, [-1, 5\*5\*64], name="pool2\_flat")

print(pool2\_flat.shape)

dense = tf.layers.dense(inputs=pool2\_flat, units=1024, activation=tf.nn.relu, name="dense")

print(dense.shape)

dropout = tf.layers.dropout(inputs=dense, rate=0.4, training=mode == tf.estimator.ModeKeys.TRAIN, name="dropout")

# Logits Layer

num\_of\_classes = get\_num\_of\_classes()

logits = tf.layers.dense(inputs=dropout, units=num\_of\_classes, name="logits")

output\_class = tf.argmax(input=logits, axis=1, name="output\_class")

output\_probab = tf.nn.softmax(logits, name="softmax\_tensor")

predictions = {"classes": tf.argmax(input=logits, axis=1), "probabilities": tf.nn.softmax(logits, name="softmax\_tensor")}

#tf.Print(tf.nn.softmax(logits, name="softmax\_tensor"), [tf.nn.softmax(logits, name="softmax\_tensor")])

if mode == tf.estimator.ModeKeys.PREDICT:

return tf.estimator.EstimatorSpec(mode=mode, predictions=predictions)

# Calculate Loss (for both TRAIN and EVAL modes)

onehot\_labels = tf.one\_hot(indices=tf.cast(labels, tf.int32), depth=num\_of\_classes)

loss = tf.losses.softmax\_cross\_entropy(onehot\_labels=onehot\_labels, logits=logits)

# Configure the Training Op (for TRAIN mode)

if mode == tf.estimator.ModeKeys.TRAIN:

optimizer = tf.train.GradientDescentOptimizer(learning\_rate=1e-3)

train\_op = optimizer.minimize(loss=loss, global\_step=tf.train.get\_global\_step())

return tf.estimator.EstimatorSpec(mode=mode, loss=loss, train\_op=train\_op)

# Add evaluation metrics (for EVAL mode)

eval\_metric\_ops = {"accuracy": tf.metrics.accuracy(labels=labels, predictions=predictions["classes"])}

return tf.estimator.EstimatorSpec(mode=mode, loss=loss, eval\_metric\_ops=eval\_metric\_ops)

def main(argv):

with open("train\_images", "rb") as f:

train\_images = np.array(pickle.load(f))

with open("train\_labels", "rb") as f:

train\_labels = np.array(pickle.load(f), dtype=np.int32)

with open("test\_images", "rb") as f:

test\_images = np.array(pickle.load(f))

with open("test\_labels", "rb") as f:

test\_labels = np.array(pickle.load(f), dtype=np.int32)

#print(len(train\_images[1]), len(train\_labels))

classifier = tf.estimator.Estimator(model\_fn=cnn\_model\_fn, model\_dir="tmp/cnn\_model3")

tensors\_to\_log = {"probabilities": "softmax\_tensor"}

logging\_hook = tf.train.LoggingTensorHook(tensors=tensors\_to\_log, every\_n\_iter=50)

train\_input\_fn = tf.estimator.inputs.numpy\_input\_fn(x={"x": train\_images}, y=train\_labels, batch\_size=100, num\_epochs=None, shuffle=True)

classifier.train(input\_fn=train\_input\_fn, steps=1500, hooks=[logging\_hook])

# Evaluate the model and print results

eval\_input\_fn = tf.estimator.inputs.numpy\_input\_fn(

x={"x": test\_images},

y=test\_labels,

num\_epochs=1,

shuffle=False)

test\_results = classifier.evaluate(input\_fn=eval\_input\_fn)

print(test\_results)

if \_\_name\_\_ == "\_\_main\_\_":

tf.app.run()

**CNN KERAS.PY**

import numpy as np

import pickle

import cv2, os

from keras import optimizers

from keras.models import Sequential

from keras.layers import Dense

from keras.layers import Dropout

from keras.layers import Flatten

from keras.layers.convolutional import Conv2D

from keras.layers.convolutional import MaxPooling2D

from keras.utils import np\_utils

from keras.callbacks import ModelCheckpoint

from keras import backend as K

K.set\_image\_dim\_ordering('tf')

os.environ['TF\_CPP\_MIN\_LOG\_LEVEL'] = '3'

def get\_image\_size():

img = cv2.imread('gestures/0/100.jpg', 0)

return img.shape

def get\_num\_of\_classes():

return len(os.listdir('gestures/'))

image\_x, image\_y = get\_image\_size()

def cnn\_model():

num\_of\_classes = get\_num\_of\_classes()

model = Sequential()

model.add(Conv2D(32, (5,5), input\_shape=(image\_x, image\_y, 1), activation='sigmoid'))

model.add(MaxPooling2D(pool\_size=(2, 2), strides=(2, 2), padding='same'))

model.add(Conv2D(64, (5,5), activation='sigmoid'))

model.add(MaxPooling2D(pool\_size=(5, 5), strides=(5, 5), padding='same'))

model.add(Flatten())

model.add(Dense(1024, activation='relu'))

model.add(Dropout(0.6))

model.add(Dense(num\_of\_classes, activation='softmax'))

sgd = optimizers.SGD(lr=1e-2)

model.compile(loss='categorical\_crossentropy', optimizer=sgd, metrics=['accuracy'])

filepath="cnn\_model\_keras2.h5"

checkpoint1 = ModelCheckpoint(filepath, monitor='val\_acc', verbose=1, save\_best\_only=True, mode='max')

#checkpoint2 = ModelCheckpoint(filepath, monitor='val\_loss', verbose=1, save\_best\_only=True, mode='min')

callbacks\_list = [checkpoint1]

return model, callbacks\_list

def train():

with open("train\_images","r+b") as f:

train\_images = np.array(pickle.load(f))

with open("train\_labels", "r+b") as f:

train\_labels = np.array(pickle.load(f), dtype=np.int32)

with open("test\_images", "r+b") as f:

test\_images = np.array(pickle.load(f))

with open("test\_labels", "r+b") as f:

test\_labels = np.array(pickle.load(f), dtype=np.int32)

train\_images = np.reshape(train\_images, (train\_images.shape[0], image\_x, image\_y, 1))

test\_images = np.reshape(test\_images, (test\_images.shape[0], image\_x, image\_y, 1))

train\_labels = np\_utils.to\_categorical(train\_labels)

test\_labels = np\_utils.to\_categorical(test\_labels)

model, callbacks\_list = cnn\_model()

model.fit(train\_images, train\_labels, validation\_data=(test\_images, test\_labels), epochs=50, batch\_size=100, callbacks=callbacks\_list)

scores = model.evaluate(test\_images, test\_labels, verbose=0)

print("CNN Error: %.2f%%" % (100-scores[1]\*100))

#model.save('cnn\_model\_keras2.h5')

train()

K.clear\_session();