**HAND GESTURE RECOGNITION USING DEEP LEARNING**

**PROGRAM**

from keras.models import Sequential

from keras.layers import Conv2D

from keras.layers import MaxPooling2D

from keras.layers import Dropout

from keras.layers import Dense

from keras.layers import Flatten

from keras.preprocessing.image import ImageDataGenerator

from keras.callbacks import EarlyStopping, ModelCheckpoint

model = Sequential()

model.add(Conv2D(32, (3, 3), input\_shape = (256, 256, 1), activation = 'relu'))

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

model.add(Conv2D(64, (3, 3), activation = 'relu'))

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

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

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

model.add(Conv2D(256, (3, 3), activation = 'relu'))

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

model.add(Flatten())

model.add(Dense(units = 150, activation = 'relu'))

model.add(Dropout(0.25))

model.add(Dense(units = 6, activation = 'softmax'))

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

train\_datagen = ImageDataGenerator(rescale = 1./255,

                                   rotation\_range = 12.,

                                   width\_shift\_range = 0.2,

                                   height\_shift\_range = 0.2,

                                   zoom\_range=0.15,

                                   horizontal\_flip = True)

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

training\_set = train\_datagen.flow\_from\_directory('HandGestureDataset/train',

                                                 target\_size = (256, 256),

                                                 color\_mode = 'grayscale',

                                                 batch\_size = 8,

                                                 classes = ['NONE','ONE','TWO','THREE','FOUR','FIVE'],

                                                 class\_mode = 'categorical')

val\_set = val\_datagen.flow\_from\_directory('HandGestureDataset/test',

                                            target\_size = (256, 256),

                                            color\_mode='grayscale',

                                            batch\_size = 8,

                                            classes=['NONE', 'ONE', 'TWO', 'THREE', 'FOUR', 'FIVE'],

                                            class\_mode='categorical')

callback\_list = [

    EarlyStopping(monitor='val\_loss',patience=10),

    ModelCheckpoint(filepath="model.h5",monitor='val\_loss',save\_best\_only=True,verbose=1)]

model.fit\_generator(training\_set,

                         steps\_per\_epoch = 37,

                         epochs = 5,

                         validation\_data = val\_set,

                         validation\_steps = 7,

                         callbacks=callback\_list

                    )

model\_json = model.to\_json()

with open("model.json", "w") as json\_file:

    json\_file.write(model\_json)

model.save\_weights("model.h5")

print("Saved model to disk")

import cv2

import numpy as np

import math

cap = cv2.VideoCapture(0)

while(True):

    \_\_,img=cap.read()

    cv2.rectangle(img,(400,400),(50,50),(0,255,0),0)

    crop\_img = img[50:400, 50:400]

    grey = cv2.cvtColor(crop\_img, cv2.COLOR\_BGR2GRAY)

    value = (35, 35)

    blurred = cv2.GaussianBlur(grey, value, 0)

    \_, thresh1 = cv2.threshold(blurred, 127, 255,

                               cv2.THRESH\_BINARY\_INV+cv2.THRESH\_OTSU)

    cv2.imshow('Thresholded', thresh1)

    contours = cv2.findContours(thresh1.copy(), \

           cv2.RETR\_TREE, cv2.CHAIN\_APPROX\_NONE)[-2]

    cnt = max(contours, key = lambda x: cv2.contourArea(x))

    x,y,w,h = cv2.boundingRect(cnt)

    cv2.rectangle(crop\_img,(x,y),(x+w,y+h),(0,0,255),0)

    hull = cv2.convexHull(cnt)

    drawing = np.zeros(crop\_img.shape,np.uint8)

    cv2.drawContours(drawing,[cnt],0,(0,255,0),0)

    cv2.drawContours(drawing,[hull],0,(0,0,255),0)

    hull = cv2.convexHull(cnt,returnPoints = False)

    defects = cv2.convexityDefects(cnt,hull)

    count\_defects = 0

    cv2.drawContours(thresh1, contours, -1, (0,255,0), 3)

    for i in range(defects.shape[0]):

        s,e,f,d = defects[i,0]

        start = tuple(cnt[s][0])

        end = tuple(cnt[e][0])

        far = tuple(cnt[f][0])

        a = math.sqrt((end[0] - start[0])\*\*2 + (end[1] - start[1])\*\*2)

        b = math.sqrt((far[0] - start[0])\*\*2 + (far[1] - start[1])\*\*2)

        c = math.sqrt((end[0] - far[0])\*\*2 + (end[1] - far[1])\*\*2)

        angle = math.acos((b\*\*2 + c\*\*2 - a\*\*2)/(2\*b\*c)) \* 57

        if angle <= 90:

            count\_defects += 1

            cv2.circle(crop\_img,far,1,[0,0,255],-1)

        cv2.line(crop\_img,start,end,[0,255,0],2)

    if count\_defects == 1:

        cv2.putText(img,"GESTURE ONE", (50,50), cv2.FONT\_HERSHEY\_SIMPLEX, 2, 2)

    elif count\_defects == 2:

        str = "GESTURE TWO"

        cv2.putText(img, str, (5,50), cv2.FONT\_HERSHEY\_SIMPLEX, 1, 2)

    elif count\_defects == 3:

        cv2.putText(img,"GESTURE THREE", (50,50), cv2.FONT\_HERSHEY\_SIMPLEX, 2, 2)

    elif count\_defects == 4:

        cv2.putText(img,"GESTURE FOUR", (50,50), cv2.FONT\_HERSHEY\_SIMPLEX, 2, 2)

    else:

        cv2.putText(img,"Hello World!!!", (50,50),cv2.FONT\_HERSHEY\_SIMPLEX, 2, 2)

    cv2.imshow('Gesture', img)

    all\_img = np.hstack((drawing, crop\_img))

    cv2.imshow('Contours', all\_img)

    key = cv2.waitKey(1)

    if key == 27:

        break

cap.release()

cv2.destroyAllWindows()

**INPUT**

**A hand holding a peace sign

Description automatically generated**

**OUTPUT**

**A close-up of a number

Description automatically generated**

The given image is predicted as two