A Machine Learning Project

Report on

OWN EMOJI CREATOR WITH ML EMOJIFY

TABLE OF CONTENTS

1.0 INTRODUCTION:

Emojis are ideograms and smileys used in electronic messages and web pages. Emoji exist in various genres, including facial expressions, common objects, places and types of weather, and animals. This project aims to localize your hand gestures and interpret them to an emoji representation. This technology can be further rendered for translating sign language. Since we are working with a live stream of data, we want to make sure that the predictions are made at runtime. For doing this efficiently, we use several filters to downsize the image that in turn, increases the efficiency. For building a predictive model, we use deep learning technique to train and develop a model which could do the needful consuming the minimum amount of CPU resources.

2.0 ABSTRACT:

Deep learning is a method of machine learning that uses multiple layers to automate the process of feature extraction from inputs. It has been applied in various fields, including image recognition and text analysis. For our project, we proposed a hand gesture emojinator recognitor, which takes a real-time hand image as the input and gives the predicted emoji as the output. In the training process, we applied the Convolutional neural network and VGG16. After adjusting the hyper parameters, we get the training accuracy 1.00 and test accuracy 0.99.

3.0 DATASET

Our dataset is from the project on GitHub[1], which contains the training images and target emojis. In this project, we want to train our model to make it able to recognize the eleven emojis, and the sample of target (output) images and training images are shown in Figure 1 and Figure 2. For each of the emoji, there are 1,200 corresponding pixel style training images, which guarantee the balance of our training set. After the training process, we use a webcam to capture real-time hand gestures and process it to the pixel style images and make real-time predictions.

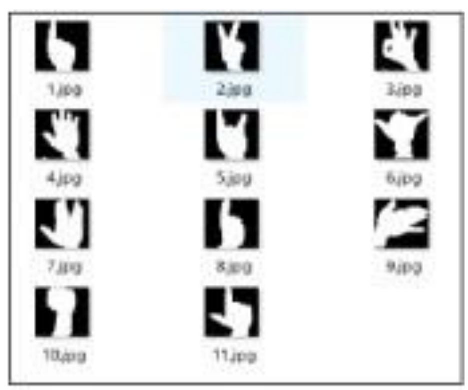


Figure 1. Sample of Target Image

Figure 2. Sample of Training Image

4.0 PRE-PROCESSING

We found that the model we trained could not perfectly identify the hand gesture during our intermediate test. The initial idea we had was that the dataset only contains the left-hand gestures, which led to failures when recognizing right-hand gestures. We used ImageDataGenerator horizontal flip to get the other copy of the dataset, except it is all right-hand gestures this time. The result showed significant improvement, but it remained some problems. We realized that it is not realistic that we can always put our hands in the perfect position in the detected area. Therefore, we used horizontal and vertical shift and set the rotation_range to 10 and recursively combine all the results from ImageDataGenerator as a single training set to the model. We later found that the model performs well on the hands that look identical to the ones in the training dataset but performs poorly when the hands differ too much with the training dataset, i.e., longer ring fingers.

5.0 MODEL

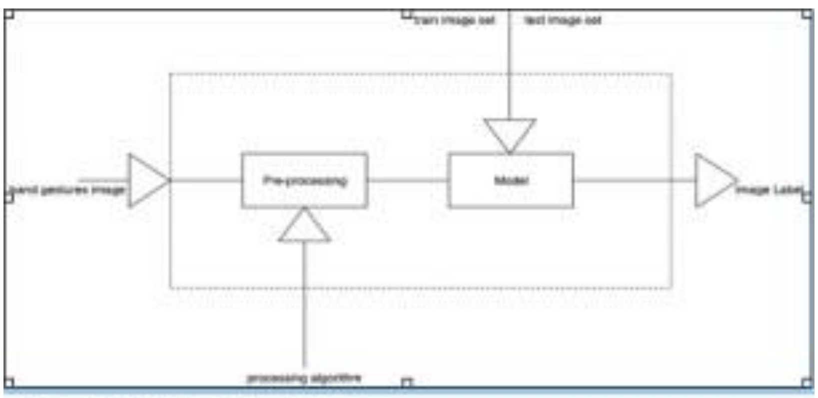


Figure 3: Workflow of the Model

Fig:work flow of model

6.0 Methodology / Approach

Procedure

1. First, you have to create a gesture database. For that, run CreateGest.py. Enter the gesture name and you will get 2 frames displayed. Look at the contour frame and adjust your hand to make sure that you capture the features of your hand. Press 'c' for capturing the images. It will take 1200 images of one gesture. Try moving your hand a little within the frame to make sure that your model doesn't overfit at the time of training.
2. Repeat this for all the features you want.
3. Run CreateCSV.py for converting the images to a CSV file 4.

If you want to train the model, run 'TrainEmojinator.py' 5.

Finally, run Emojinator.py for testing your model via webcam.

Functionalities
- Image processing filters to detect hand.
- CNN for training the model.

Network Used - Convolutional Neural Network

Technologies Used

Hardware Used
- Intel Powered PC (Intel i3).

Technologies Used -
- Intel Optimised Python.
- Intel Optimised TensorFlow.
- Keras
- OpenCV

7.0 CODE:

CreateGest.py.

```
import cv2

import numpy as np

import os

image_x, image_y = 50, 50

 cap = cv2.VideoCapture(0)

fbag = cv2.createBackgroundSubtractorMOG2()

def create_folder(folder_name):

 if not os.path.exists(folder_name):

os.mkdir(folder_name)

 def main(g_id):

 total_pics = 1200

 cap = cv2.VideoCapture(0)

 x, y, w, h = 300, 50, 350, 350

 create_folder("gestures/" + str(g_id))
```

```
pic_no = 0
 flag_start_capturing = False
 frames = 0
 while True:
 ret, frame = cap.read()
 frame = cv2.flip(frame, 1)
 hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
 mask2 = cv2.inRange(hsv, np.array([2, 50, 60]), np.array([25, 150, 255])) res
= cv2.bitwise_and(frame, frame, mask=mask2)
 gray = cv2.cvtColor(res, cv2.COLOR_BGR2GRAY)
 median = cv2.GaussianBlur(gray, (5, 5), 0)
 kernel_square = np.ones((5, 5), np.uint8)
 dilation = cv2.dilate(median, kernel_square, iterations=2)
opening=cv2.morphologyEx(dilation,cv2.MORPH_CLOSE,kernel_square) ret,
thresh = cv2.threshold(opening, 30, 255, cv2.THRESH_BINARY)  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(frame, "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(frame, (x, y), (x + w, y + h), (0, 255, 0), 2)
cv2.putText(frame, str(pic_no), (30, 400), cv2.FONT_HERSHEY_TRIPLEX,  1.5,
(127, 127, 255))
cv2.imshow("Capturing gesture", frame)
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
g_id = input("Enter gesture number: ")
main(g_id)
```

CreateCSV.py

```
from scipy.misc import imread
```

```
import numpy as np

import pandas as pd

import os

root = './gestures' # or './test' depending on for which the CSV is being created

 for directory, subdirectories, files in os.walk(root):

# go through each file in that directory

 for file in files:

 # read the image file and extract its pixels

 print(file)

 im = imread(os.path.join(directory,file))

 value = im.flatten()

 value = np.hstack((directory[11:],value))

 df = pd.DataFrame(value).T

 df = df.sample(frac=1) # shuffle the dataset

 with open('train_foo.csv', 'a') as dataset:

 df.to_csv(dataset, header=False, index=False)
```

3. TrainEmojinator.py

```
import numpy as np

from keras import layers

from keras.layers import Input, Dense, Activation, ZeroPadding2D, BatchNormalization,
Flatten, Conv2D

from keras.layers import AveragePooling2D, MaxPooling2D, Dropout, GlobalMaxPooling2D,
GlobalAveragePooling2D

from keras.utils import np_utils

from keras.models import Sequential

from keras.callbacks import ModelCheckpoint
import pandas as pd

 import keras.backend as K

 def keras_model(image_x, image_y):

 num_of_classes = 12

 model = Sequential()
```

```
model.add(Conv2D(32, (5, 5), input_shape=(image_x, image_y, 1), activation='relu'))
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'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
filepath = "emojinator.h5"
checkpoint1 = ModelCheckpoint(filepath, monitor='val_acc', verbose=1,
save_best_only=True, mode='max')
callbacks_list = [checkpoint1]
return model, callbacks_list
def main():
data = pd.read_csv("train_foo.csv")
dataset = np.array(data)
np.random.shuffle(dataset)
X = dataset
Y = dataset
X = X[:, 1:2501]
Y = Y[:, 0]
X_train = X[0:12000, :]
X_train = X_train / 255.
X_test = X[12000:13201, :]
X_test = X_test / 255.
Y = Y.reshape(Y.shape[0], 1)
Y_train = Y[0:12000, :]
Y_train = Y_train.T
Y_test = Y[12000:13201, :]
```

```
Y_test = Y_test.T
print ("number of training examples = " + str(X_train.shape[0]))
print("number of test examples = " + str(X_test.shape[0]))
print("X_train shape: " + str(X_train.shape))
print("Y_train shape: " + str(Y_train.shape))
print("X_test shape: " + str(X_test.shape))
print("Y_test shape: " + str(Y_test.shape))
image_x = 50
image_y = 50
train_y = np_utils.to_categorical(Y_train)
test_y = np_utils.to_categorical(Y_test)
train_y = train_y.reshape(train_y.shape[1], train_y.shape[2])
test_y = test_y.reshape(test_y.shape[1], test_y.shape[2])
X_train = X_train.reshape(X_train.shape[0], 50, 50, 1)
X_test = X_test.reshape(X_test.shape[0], 50, 50, 1)
print("X_train shape: " + str(X_train.shape))
print("X_test shape: " + str(X_test.shape))
model, callbacks_list = keras_model(image_x, image_y)
model.fit(X_train, train_y, validation_data=(X_test, test_y), epochs=10, batch_size=64,
callbacks=callbacks_list)
scores = model.evaluate(X_test, test_y, verbose=0)
print("CNN Error: %.2f%%" % (100 - scores[1] * 100))
model.save('emojinator.h5')
main()
```

4. Emojinator.py

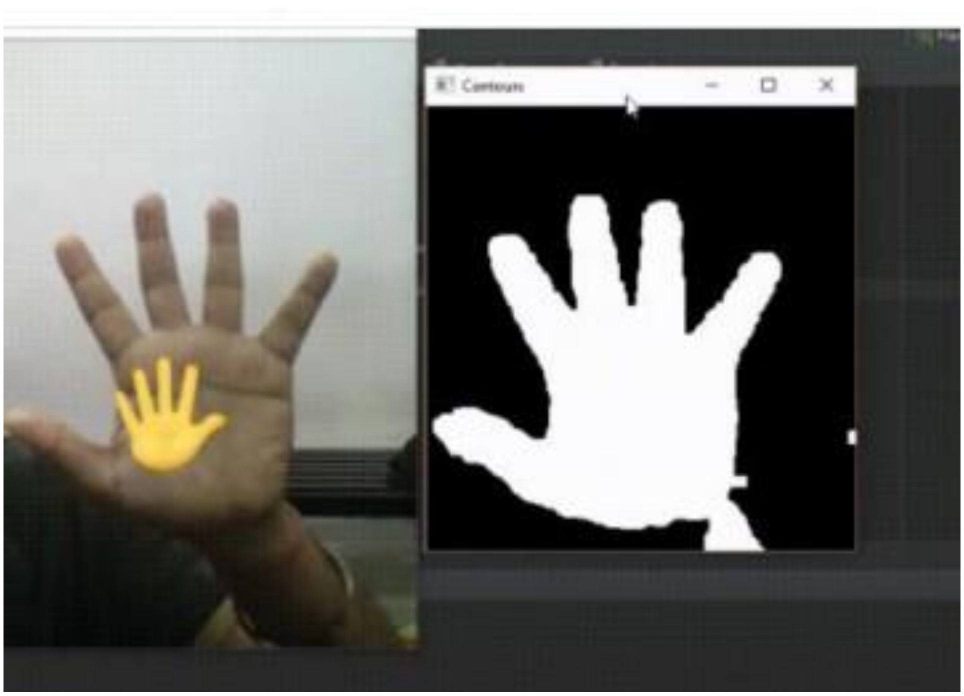
```
import cv2
from keras.models import load_model
import numpy as np
import os
model = load_model('emojinator.h5')
```

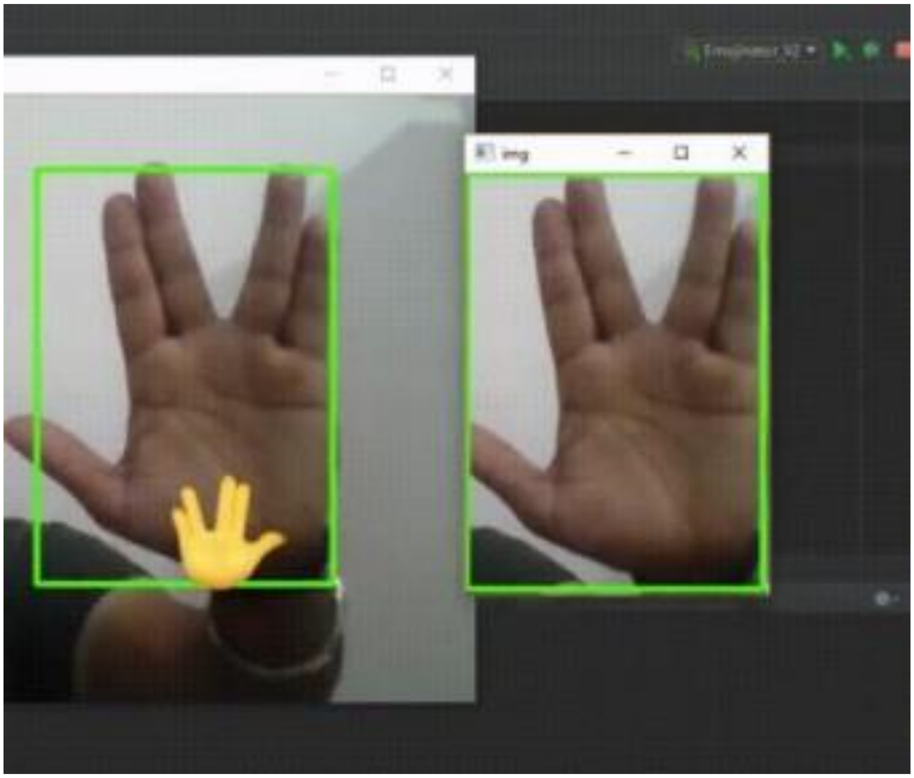
```
def main():
emojis = get_emojis()
cap = cv2.VideoCapture(0)
x, y, w, h = 300, 50, 350, 350
while (cap.isOpened()):
ret, img = cap.read()
img = cv2.flip(img, 1)
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
mask2 = cv2.inRange(hsv, np.array([2, 50, 60]), np.array([25, 150, 255]))  res
= cv2.bitwise_and(img, img, mask=mask2)
gray = cv2.cvtColor(res, cv2.COLOR_BGR2GRAY)
median = cv2.GaussianBlur(gray, (5, 5), 0)
kernel_square = np.ones((5, 5), np.uint8)
dilation = cv2.dilate(median, kernel_square, iterations=2)  opening =
cv2.morphologyEx(dilation, cv2.MORPH_CLOSE, kernel_square)  ret, thresh =
cv2.threshold(opening, 30, 255, cv2.THRESH_BINARY)
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) > 2500:
x, y, w1, h1 = cv2.boundingRect(contour)
newImage = thresh[y:y + h1, x:x + w1]
newImage = cv2.resize(newImage, (50, 50))  pred_probab,
pred_class = keras_predict(model, newImage)  print(pred_class,
pred_probab)
img = overlay(img, emojis[pred_class], 400, 250, 90, 90)  x, y, w,
h = 300, 50, 350, 350
```

```
cv2.imshow("Frame", img)
cv2.imshow("Contours", thresh)
k = cv2.waitKey(10)
if k == 27:
break
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 keras_process_image(img):
image_x = 50
image_y = 50
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 get_emojis():
emojis_folder = 'hand_emo/'
emojis = []
for emoji in range(len(os.listdir(emojis_folder))):
print(emoji)
emojis.append(cv2.imread(emojis_folder+str(emoji)+'.png', -1))
return emojis
def overlay(image, emoji, x,y,w,h):
emoji = cv2.resize(emoji, (w, h))
try:
image[y:y+h, x:x+w] = blend_transparent(image[y:y+h, x:x+w], emoji)
except:
pass
```

```
return image
def blend_transparent(face_img, overlay_t_img):
# Split out the transparency mask from the colour info
overlay_img = overlay_t_img[:,:,:3] # Grab the BRG planes
overlay_mask = overlay_t_img[:,:,3:] # And the alpha plane
background_mask = 255 - overlay_mask
overlay_mask = cv2.cvtColor(overlay_mask, cv2.COLOR_GRAY2BGR)
background_mask = cv2.cvtColor(background_mask, cv2.COLOR_GRAY2BGR)
face_part = (face_img * (1 / 255.0)) * (background_mask * (1 / 255.0))
overlay_part = (overlay_img * (1 / 255.0)) * (overlay_mask * (1 / 255.0)) return
np.uint8(cv2.addWeighted(face_part, 255.0, overlay_part, 255.0, 0.0))
keras_predict(model, np.zeros((50, 50, 1),
dtype=np.uint8)) main()
```

8.0 OUTPUTS:





9. Conclusion

For this project, we get pretty good test scores, that the accuracy rate is close to 1.0. Deep learning is pretty good at image recognition, and the result reflects that. Although the accuracy is already 1, there are a couple of things that we can do to improve our model.

First, we can test to find a better set of light sources and resolution of input images. For example, applying semantic segmentation pre-trained models with a label of lights will significantly reduce the unnecessary noise in our dataset. Second, we can use multiple snapshots from consecutive frames as input. From multiple label output, we can take the majority of the output labels as the final output label. Last but not least, we will want to include more datasets containing different sizes of hands to ensure our model can adapt to various hand shapes.

10. References:

[1] https://devmesh.intel.com/projects/emojinator-281f25

[2] akshaybahadur21. “akshaybahadur21/Emojinator.” GitHub, 3 Apr. 2019,

https://github.com/akshaybahadur21/Emojinator.