CSE4019

ACTION DETECTION-DUMB ASSISTOR

VIGNESH VAIDYNATHAN 15BCE0076
SAMUDRA PRATIM BORKAKOTI 15BCE0093
VOLETI RAVI 15BCE0082
TAJINDER SINGH SONDHI 15BCE0040

SLOT: E2+TE2

INTRODUCTION

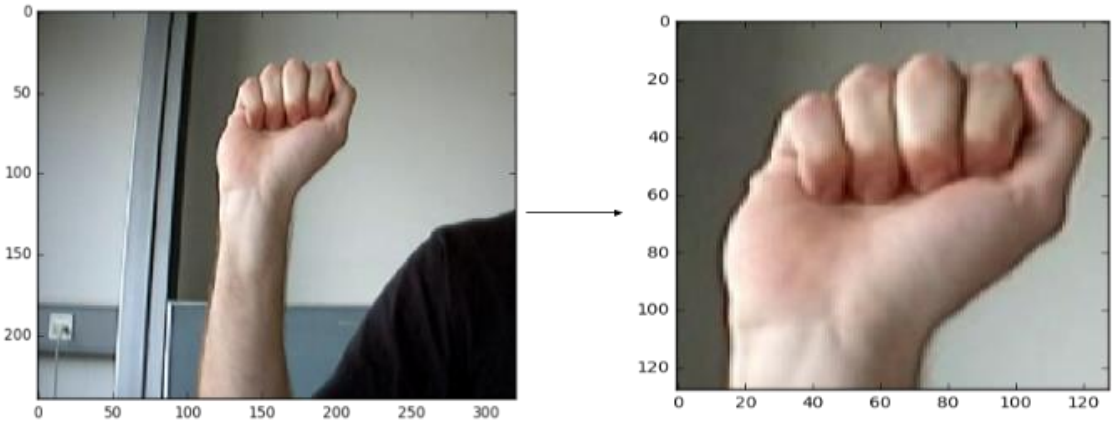
Gesture recognition is an open issue in the region of machine vision, a field of software engineering that empowers frameworks to copy human vision. Motion acknowledgment has numerous applications in enhancing human-PC communication, and one of them is in the field of Sign Language Translation, wherein a video grouping of typical hand signals is converted into common dialect. A scope of cutting edge strategies for the same have been created. Here, we'll take a gander at how to perform static-motion acknowledgment utilizing the scikit learn and scikit picture libraries.

We have used the following packages

- **Skikit-learn**
- **Skikit-image**
- **Python-Pip**
- **Pickle**
- **Pandas**
- **Json**
- **Random**
- **Keras**
- **Tensorflow**
- **Numpy**
- **Cv2**
- **Glob**
- **Os**
- **Time**
- **gzip**

BUILDING A STATIC-GESTURE RECOGNIZER

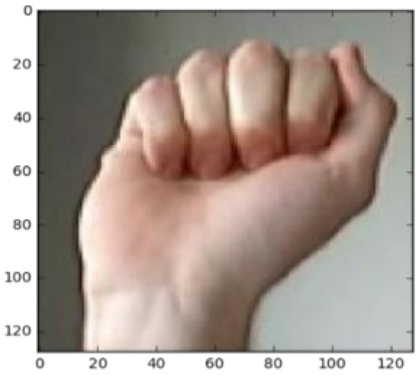
For this part, we utilize an informational collection involving crude pictures and a relating csv document with directions demonstrating the bouncing box for the deliver each picture. This informational index is composed user-wise and the catalog structure of the dataset is as per the following. The picture names demonstrate the letter set spoken to by the picture. The static-motion recognizer is basically a multi-class classifier that is prepared on info pictures speaking to the 24 static communication through signing signals (A-Y, barring J). Building a static-signal recognizer utilizing the crude pictures and the csv record is genuinely straightforward.



Raw image (320x240)

Cropped out section using given bounding box values (Rescaled to 128x128)

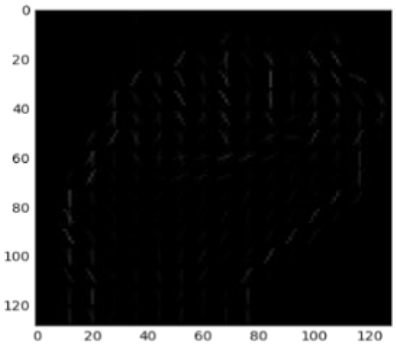
To utilize the multi-class classifiers from the scikit learn library, we'll have to first form the information set—that is, each picture must be changed over into an element vector (X) and each picture will have a mark relating to the gesture based communication letters in order that it indicates (Y). The key now is to utilize a suitable technique to vectorise the picture and concentrate important data to sustain to the classifier. Essentially utilizing the crude pixel qualities won't work on the off chance that we anticipate utilizing straightforward multi-class classifiers (rather than utilizing Convolution Networks). To vectorise our pictures, we utilize the Histogram of Oriented Gradients (HOG) approach, as it has been demonstrated to yield great outcomes on issues, for example, this one. Other component extractors that can be utilized incorporate Local Binary Patterns and Haar Filters.



Cropped out section using given bounding box values (Rescaled to 128x128)

```
[ 0.04204403  0.01927424
0.01888355 ...,  0.00379038
0.00314776  0.05292313]
(Length=15876)
```

HOG vector of the cropped out section



Visualization of the HOG image

CODE:

We utilize pandas in the get_data() capacity to stack the CSV record. Two capacities harvest() and convertToGrayToHog() are utilized to get the required hoard vector and attach it to the rundown of vectors that we're working, so as to prepare the multi-class classifier. The following stride is to encode the yield marks (the Y-qualities) to numerical qualities. We do this utilizing sklearn's mark encoder.

In our code, we have done this as takes after:

where, the label_encoder question is developed as takes after inside the signal recognizer class constructor: When this is done, the model can be prepared utilizing any Multi-class arrangement calculation of your decision from the scikit learn tool compartment. We have prepared our own utilizing Support Vector Classification, with a straight bit. Preparing a model utilizing sklearn does not include more than two lines of code. Here's the manner by which you do it: The hyper parameters (i.e., C=0.9 for this situation) can be tuned utilizing a Grid Search. For this situation, we don't have the foggiest idea about a mess about the information in that capacity (i.e., the hoard vectors). In this way, it'd be a smart thought to attempt and utilize calculations like xgboost (Extreme

Gradient Boosting) or Random Forest Classifiers and perceive how these calculations perform.

BUILDING THE LOCALIZER

This part requires a somewhat more exertion when contrasted with the first. Comprehensively, we'll utilize the accompanying strides in finishing this assignment. **Build a data set** containing pictures of hands and parts that are not-hand, utilizing the given informational collection and the jumping box values for each picture. **Train a binary classifier** to identify hand/not-hand pictures utilizing the above informational index. Utilize a **sliding window approach with different scales**, on the inquiry picture to segregate the area of intrigue.

BUILDING THE HAND/NOT HAND DATASET:

The informational collection could be assembled utilizing any system you like. One approach to do this, is to create irregular arranges and afterward check the proportion of range of crossing point to zone of union (i.e., the level of cover with the given jumping box) to decide whether it is a non-hand area. (Another approach could be to utilize a sliding window to decide the directions. Be that as it may, this is unpleasantly moderate and superfluous)

TRAINING A BINARY CLASSIFIER:

Once the informational index is prepared, preparing the classifier should be possible precisely as observed before in the first part. As a rule, for this situation, a system called Hard Negative Mining is utilized to diminish the quantity of false positive location and enhance the classifier. Maybe a couple iterations of hard negative mining utilizing a Random Forest Classifier, is sufficient to guarantee that your classifier achieves worthy characterization accuracies, which for this situation is anything over 80%.

DETECTING HANDS IN TEST IMAGES:

Presently, to really utilize the above classifier, we scale the test picture by different variables and after that utilization a sliding window approach on every one of them to pick the window which catches the district of intrigue splendidly. This is finished by choosing the locale comparing to the maximum of the certainty scores allocated by the twofold (hand/not-hand) classifier over all scales. The test pictures should be scaled in light of the fact that, we run a set estimated window (for our situation, it is 128x128) over all pictures to pick the locale of intrigue and it is conceivable that the district of intrigue does not fit superbly into this window measure.

PUTTING IT ALL TOGETHER

After both parts are finished, all that is left to do is to call them in progression to get the last yield when furnished with a test picture. That is, given a test picture, we initially get the different recognized locales crosswise over various sizes of the picture and pick the best one among them. This locale is then edited out, rescaled (to 128x128) and its comparing swine vector is bolstered to the multi-class classifier (i.e., the motion recognizer). The motion recognizer then predicts the signal signified by the turn in the picture.

KEY POINTS

Building the hand/not-hand dataset.

Changing over every one of the pictures i.e., trimmed areas with the motions and the hand, not-hand pictures, to its vectorised frame.

Building a twofold classifier for recognizing the segment with the hand and building a multi-class classifier for distinguishing the motion utilizing these informational indexes.

Utilizing the above classifiers in a steady progression to play out the required undertaking.

RESULTS

The program yields the following results after its execution. For input we have directly shown the path where we input the file to recognize_gesture()

INPUT 1:

```
image="/home/xelese/img_proj/hand_sign/custom/W.jpg"
```

OUTPUT 1:

```
xelese@xelese-Lenovo-Y50-70:~/img_proj/hand_sign$ sudo -s
[sudo] password for xelese:
root@xelese-Lenovo-Y50-70:~/img_proj/hand_sign# python new_sign.py
Using TensorFlow backend.
['user_3', 'user_4', 'user_5']
Train starts
Imageset, boundbox, hog_list,label_list Loaded!
Set training is done
Set training is done
Set training is done
Multiclass data loaded
handDetector trained
Sign Detector trained
The GestureRecognizer is saved to disk
LabelEncoder()
<__main__.GestureRecognizer object at 0x7fe722c55690>

The predicted gesture might be

W
root@xelese-Lenovo-Y50-70:~/img_proj/hand_sign#
```

INPUT 2:

```
image="/home/xelese/img_proj/hand_sign/custom/B.jpg"
```

OUTPUT 2:

```
root@xelese-Lenovo-Y50-70:~/img_proj/hand_sign# python new_sign.py
Using TensorFlow backend.
['user_3', 'user_4', 'user_5']
Train starts
Imageset, boundbox, hog_list,label_list Loaded!
Set training is done
Set training is done
Set training is done
Multiclass data loaded
handDetector trained
Sign Detector trained
The GestureRecognizer is saved to disk
LabelEncoder()
<__main__.GestureRecognizer object at 0x7f2589f1b690>

The predicted gesture might be

B
root@xelese-Lenovo-Y50-70:~/img_proj/hand_sign#
```

INPUT 3:

```
image="/home/xelese/img_proj/hand_sign/custom/H.jpg"
```

OUTPUT 3:

```
root@xelese-Lenovo-Y50-70:~/img_proj/hand_sign# python new_sign.py
Using TensorFlow backend.
['user_3', 'user_4', 'user_5']
Train starts
Imageset, boundbox, hog_list,label_list Loaded!
Set training is done
Set training is done
Set training is done
Multiclass data loaded
handDetector trained
Sign Detector trained
The GestureRecognizer is saved to disk
LabelEncoder()
<__main__.GestureRecognizer object at 0x7f082ca9b690>

The predicted gesture might be

H
```

INPUT 4:

```
image="/home/xelese/img_proj/hand_sign/custom/V.jpg"
```

OUTPUT 4:

```
root@xelese-Lenovo-Y50-70:~/img_proj/hand_sign# python new_sign.py
Using TensorFlow backend.
['user_3', 'user_4', 'user_5']
Train starts
Imageset, boundbox, hog_list,label_list Loaded!
Set training is done
Set training is done
Set training is done
Multiclass data loaded
handDetector trained
Sign Detector trained
The GestureRecognizer is saved to disk
LabelEncoder()
<__main__.GestureRecognizer object at 0x7ffaf1455690>

The predicted gesture might be

V
root@xelese-Lenovo-Y50-70:~/img_proj/hand_sign#
```

INPUT 5:

```
image="/home/xelese/img_proj/hand_sign/custom/C.jpg"
```

OUTPUT 5:

```
root@xelese-Lenovo-Y50-70:~/img_proj/hand_sign# python new_sign.py
Using TensorFlow backend.
['user_3', 'user_4', 'user_5']
Train starts
Imageset, boundbox, hog_list,label_list Loaded!
Set training is done
Set training is done
Set training is done
Multiclass data loaded
handDetector trained
Sign Detector trained
The GestureRecognizer is saved to disk
LabelEncoder()
<__main__.GestureRecognizer object at 0x7fba1badb690>

The predicted gesture might be

C
root@xelese-Lenovo-Y50-70:~/img_proj/hand_sign#
```

CONCLUSION:

We were successfully able to achieve the recognition of American sign language through the training of images using classifiers and the comparing our input images with the trained image thus gesture recognition was achieved with static images.

REFERENCES:

[1] https://medium.freecodecamp.com/weekend-projects-sign-language-and-static-gesture-recognition-using-scikit-learn-60813d600e79

[2] http://scikit-learn.org/

[3] http://pandas.pydata.org/

[4] https://stats.stackexchange.com/

[5] http://docs.opencv.org/2.4/doc/tutorials/tutorials.html

[6] http://docs.opencv.org/3.1.0/examples.html

[7] http://scikit-image.org/docs/dev/user_guide/tutorials.html

[8] http://cs231n.github.io/python-numpy-tutorial/

[9] https://elitedatascience.com/keras-tutorial-deep-learning-in-python