Digit Recognition

15CSE387,Open Lab

Team Members

Y.D.Sai Shreyas,CSE15133

B.Sudheesh Reddy,CSE15119

N.Sai Sree Nidhi,CSE15082

V.Bhuvaneswar Reddy,CSE15108

Abstract:

* In this project, we use images, from the MNIST dataset, as input. Uses OpenCV to pre-process the image and to extract the digits from the picture.
* Using K-Nearest Neighbours, digits are recognised in the image. After feature extraction from images for OCR -A HOG (Histogram of Gradients) is drawn.
* Basically, it tries to capture the shape of structures in the region by capturing information about gradients.
* Image gradient are simply intensity changes across pixels in an image. It works by dividing the image into small (usually 8x8 pixels) cells and blocks of 4x4 cells.
* Each cell has a fixed number of gradient orientation bins. Each pixel in the cell votes for a gradient orientation bin with a vote proportional to the gradient magnitude at that pixel or simply put, the "histogram" counts how many pixels have an edge with a specific orientation.

**Note:**

* User image should be a scanned (at least 300dpi) image.
* Image can be any format supported by OpenCV.

Description:

Detecting handwritten digits using HOG features and a multiclass Linear SVM is the objective.To do that,these steps should be followed:

1. A database of handwritten digits to train the classifier.
2. For each handwritten digit in the database, extract HOG features and train a Linear SVM.
3. Use the classifier trained in step 2 to predict digits.

### 1.MNIST database of handwritten digits:

The MNIST database is a set of 70000 samples of handwritten digits where each sample consists of a grayscale image of size 28×28. There are a total of 70,000 samples. We have calculated the number of samples for each digit using “collections.Counter” class.

The actual samples for each digit is like this below -

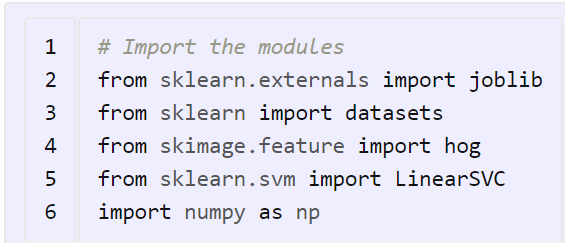
| **Digits** | **Number of samples** |
| --- | --- |
| 0 | 6903 |
| 1 | 7877 |
| 2 | 6990 |
| 3 | 7141 |
| 4 | 6824 |
| 5 | 6313 |
| 6 | 6876 |
| 7 | 7293 |
| 8 | 6825 |
| 9 | 6958 |
|  |  |

### 2.Training a Classifier:

To train the classifer,a certain modules & packages are used.

**Packages** : sklearn , skimage , sklearn.datasets ,sklearn.externals.joblib.

**Modules** : LinearSVC , numpy.



Sklearn.externals.joblib is used to save classifer in a file,so that the training & calculating HOG features for 70,000 samples need no to be done each time we run the code.

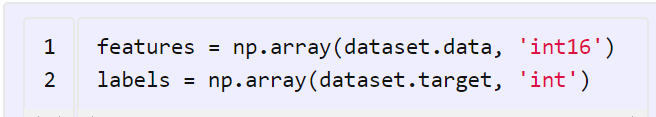
Sklearn.datasets package will be used to download the MNIST database for handwritten digits.

Skimage.feature.hog class is used to calculate the HOG features.

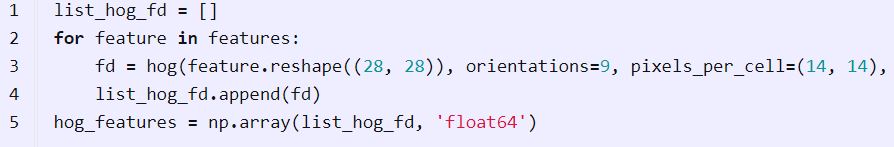
Sklearn.svm.LinearSVC is for the prediction of digits after the training of the classifier.

NumpyArrays are used to store HOG features and labels.

Sklearn.datasets.fetch\_mldata function is used to download the dataset which is used for training the classifier.



Once, the dataset is downloaded we will save the images of the digits in a numpy array ’features’ and the corresponding labels  i.e. the digit in another numpy array ‘labels’. Next, we calculate the HOG features for each image in the database and save them in another numpy array named hog\_feature.



To calculate the HOG features, we set the number of cells in each block equal to one and each individual cell is of size 14×14. Since our image is of size 28×28, we will have four blocks/cells of size 14×14 each. Also, we set the size of orientation vector equal to 9. So our HOG feature vector for each sample will be of size 4×9 = 36.’ list\_hog\_fd’ appends the HOG features for each sample in the database. We set the visualise parameter to false as we do not need the HOG features to be visible.



We create a Linear SVC object. Since there are 10 digits, we need a multi-class classifier. The Linear SVC that comes with sklearn can perform multi-class classification.So we are using clf as a object for running Linear SVC. We perform the training using the fit member function of the

Linear SVC to input the values.The fit function has 2 arguments ,one an array of the HOG features of the handwritten digit that was calculated earlier and a corresponding array of labels i.e.0,1,2,3,4,5..9. When the training finishes, we will save the classifier in a file named

“digits\_cls.pk1” by using “joblib.dump” function. ‘compress’ is used to compress the values taken.Higher the compress value decreases the read and write timing.so for best use ,we take values as 3.

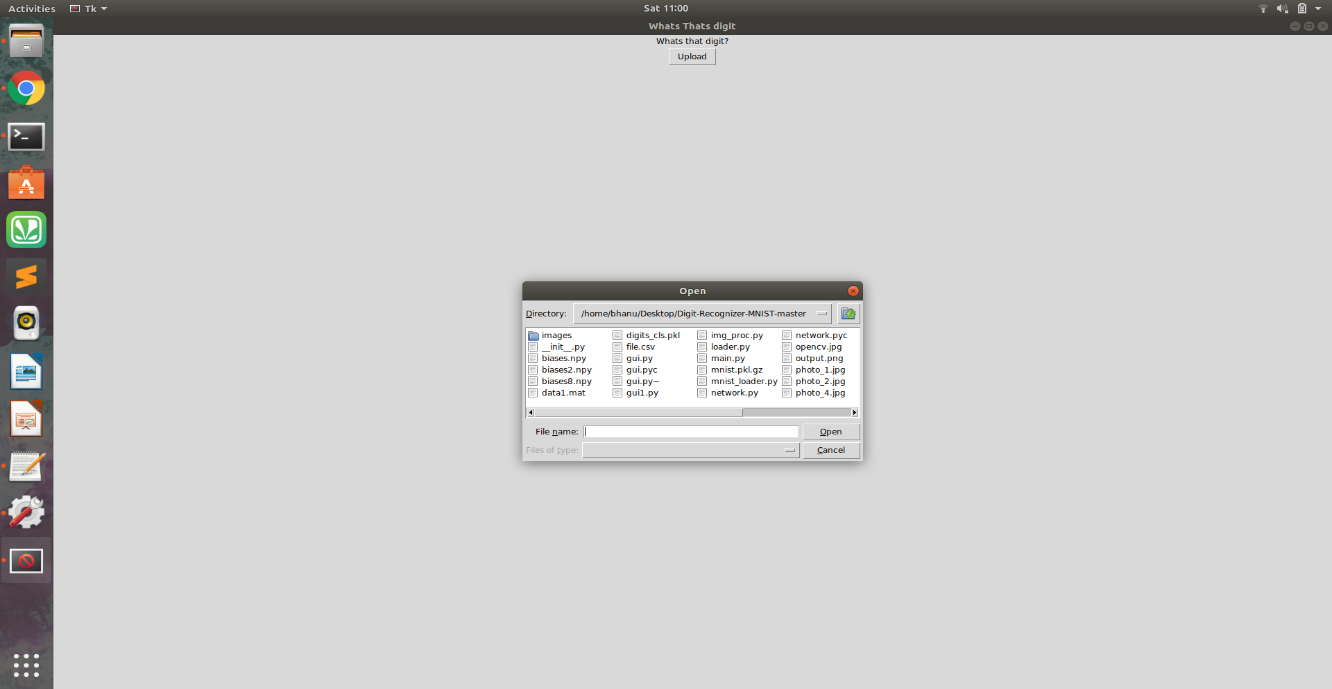
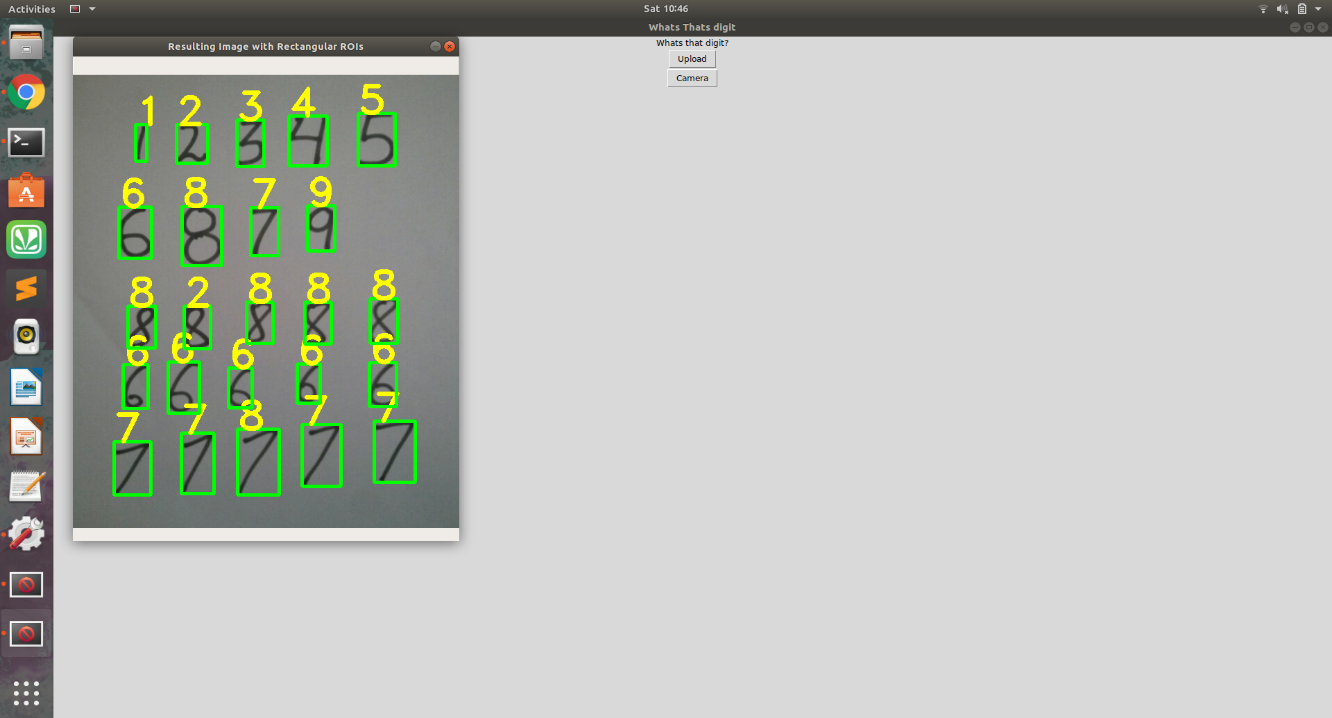


Now,we can test the classifier using all the above modules,classes,functions and packages.

3.Testing & Running the classifier:

**Packages used** : Tkinter,cv2

We load the classifier from the file “**digits\_cls.pkl “ and we load the image required by loading filepath from gui . The gui uses Tkinter package for displaying the required by running all the functions.After loading the image from the filepath , the test() is imported from user file where our main program exists and called in gui as test(filepath) by onclick() function for a button named ‘upload’.In test(filepath),** We convert the loaded image to a grayscale image using cv2.cvtColor function. We then apply a Gaussian filter to the grayscale image to remove noisy pixels.We convert the grayscale image into a binary image using a threshold value of 90. All the pixel locations with grayscale values greater than 90 are set to 0 in the binary image and all the pixel locations with grayscale values less than 90 are set to 255 in the binary image . We calculate the contours in the image and then we calculate the bounding box for each contour . For each bounding box, we generate a bounding square around each contour. We then resize each bounding square to a size of 28×28 and dilate it.We calculate the HOG features for each bounding square [HOG feature vector for each bounding square should be of the same size for which the classifier was trained, as to avoid error]. We predict the digit using our classifier and draw the bounding box and the predicted digit on the input image.

Sample Output:

Conclusion:

We recognized and predicted the handwritten digits from the user given input image and displayed the output image with boxes on the digits and printing the predicted values on the boxes in the output image.