**LPADIP 2018**

# 

# PROJECT ON :

**HANDWRITTEN DIGIT RECOGNITION**

GROUP 5:

* ANKUR VISHWAKARMA(GLA UNIVERSITY,MATHURA)
* SHIVANSH KESHARI(DIT UNIVERSITY)
* YASH MISHRA( REC,SONBHADRA)
* SHIKHAR(GALGOTIAS UNIVERSITY)
* DHIRGHRAJ KHUSHWAHA(JCER , JAIPUR )

**CONTENTS**

1. INTRODUCTION
2. WEBSITE DESIGNING
3. MACHINE LEARNING
4. MNIST DATASETS
5. CODE SCRIPTS
6. SCREENSHOTS
7. REFERENCES

# Dependencies

1. Numpy
2. pandas
3. cv2
4. sklearn
5. skimage
6. numpy
7. mathplot.lib
8. . Introduction
9. Handwritten digit recognition is an active topic in OCR
10. applications and pattern classication/learning research. In
11. OCR applications, digit recognition is dealt with in postal
12. mail sorting, bank check processing, form data entry, etc. For
13. these applications, the performance (accuracy and speed) of
14. digit recognition is crucial to the overall performance. While
15. in pattern classication and machine learning communities,
16. the problem of handwritten digit recognition is a good ex-
17. . Introduction
18. Handwritten digit recognition is an active topic in OCR
19. applications and pattern classication/learning research. In
20. OCR applications, digit recognition is dealt with in postal
21. mail sorting, bank check processing, form data entry, etc. For
22. these applications, the performance (accuracy and speed) of
23. digit recognition is crucial to the overall performance. While
24. in pattern classication and machine learning communities,
25. the problem of handwritten digit recognition is a good ex-

### Goals:

The goal in this competition is to take an image of a handwritten single digit, and determine what that digit is.  
For every ImageId in the test set, you should predict the correct label.

**DATASET DESCRIPTION**

The data files train.csv and test.csv contain gray-scale images of hand-drawn digits, from zero through nine.

Each image is 28 pixels in height and 28 pixels in width, for a total of 784 pixels in total. Each pixel has a single pixel-value associated with it, indicating the lightness or darkness of that pixel, with higher numbers meaning darker. This pixel-value is an integer between 0 and 255, inclusive.

The training data set, (train.csv), has 785 columns. The first column, called "label", is the digit that was drawn by the user. The rest of the columns contain the pixel-values of the associated image.

Each pixel column in the training set has a name like pixelx, where x is an integer between 0 and 783, inclusive. To locate this pixel on the image, suppose that we have decomposed x as x = i \* 28 + j, where i and j are integers between 0 and 27, inclusive. Then pixelx is located on row i and column j of a 28 x 28 matrix, (indexing by zero).

For example, pixel31 indicates the pixel that is in the fourth column from the left, and the second row from the top, as in the ascii-diagram below.

Visually, if we omit the "pixel" prefix, the pixels make up the image like this:

000 001 002 003 ... 026 027

028 029 030 031 ... 054 055

056 057 058 059 ... 082 083

| | | | ... | |

728 729 730 731 ... 754 755

756 757 758 759 ... 782 783

The test data set, (test.csv), is the same as the training set, except that it does not contain the "label" column.

Your submission file should be in the following format: For each of the 28000 images in the test set, output a single line containing the ImageId and the digit you predict. For example, if you predict that the first image is of a 3, the second image is of a 7, and the third image is of a 8, then your submission file would look like:

ImageId,Label  
1,3  
2,7  
3,8

(27997 more lines)

The evaluation metric for this contest is the categorization accuracy, or the proportion of test images that are correctly classified. For example, a categorization accuracy of 0.97 indicates that you have correctly classified all but 3% of the images.

**Training a Classifier**

Here, we will implement the following steps –

1. Calculate the HOG features for each sample in the database.
2. Train a multi-class linear SVM with the HOG features of each sample along with the corresponding label.
3. Save the classifier in a file

The first step is to import the required modules –

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | *# Import the modules*  from sklearn.externals import joblib  from sklearn import datasets  from skimage.feature import hog  from sklearn.svm import LinearSVC  import numpy as np |

We will use the sklearn.externals.joblib package to save the classifier in a file so that we can use the classifier again without performing training each time. Calculating HOG features for 70000 images is a costly operation, so we will save the classifier in a file and load it whenever we want to use it. As discussed above sklearn.datasets package will be used to download the MNIST database for handwritten digits. We will use skimage.feature.hog class to calculate the HOG features and sklearn.svm.LinearSVCclass to perform prediction after training the classifier. We will store our HOG features and labels in numpy arrays. The next step is to download the dataset using the sklearn.datasets.fetch\_mldata function. For the first time, it will take some time as 55.4 MB will be downloaded.

|  |  |
| --- | --- |
| 1 | dataset **=** datasets**.**fetch\_mldata("MNIST Original") |

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 as shown below –

|  |  |
| --- | --- |
| 1  2 | features **=** np**.**array(dataset**.**data, 'int16')  labels **=** np**.**array(dataset**.**target, 'int') |

Next, we calculate the HOG features for each image in the database and save them in another numpy array named hog\_feature.

|  |  |
| --- | --- |
| 1  2  3  4  5 | list\_hog\_fd **=** []  **for** feature **in** features:  fd **=** hog(feature**.**reshape((28, 28)), orientations**=**9, pixels\_per\_cell**=**(14, 14), cells\_per\_block**=**(1, 1), visualise**=**False)  list\_hog\_fd**.**append(fd)  hog\_features **=** np**.**array(list\_hog\_fd, 'float64') |

In **line 17** we initialize an empty list list\_hog\_fd, where we append the HOG features for each sample in the database. So, in the for loop in **line 18**, we calculate the HOG features and append them to the list list\_hog\_fd. Finally, we create an numpy array hog\_features containing the HOG features which will be used to train the classifier. This step will take some time, so be patient while this piece of code finishes.

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. We are not interesting in visualizing the HOG feature image, so we will set the visualise parameter to false.

The next step is to create a Linear SVM object. Since there are 10 digits, we need a multi-class classifier. The Linear SVM that comes with sklearn can perform multi-class classification.

|  |  |
| --- | --- |
| 1 | clf **=** LinearSVC() |

### Testing the Classifier

**Now, we will write another python script to test the classifier. The code for the second script is pretty easy and here is the code for the same** –

xtest=data[21000:,1:]

actual\_label=data[21000:,0]

n=int(input("INDEX: "))

d=xtest[n]

d.shape=(28,28)

pt.imshow(255-d,cmap='gray')

pt.savefig('books\_read.jpg')

print(clf.predict([xtest[n]]))

pt.show()

p=clf.predict(xtest)

count=0

for i in range(0,21000):

if(p[i]==actual\_label[i]):

count+=1

else:

count+=0

print("accuracy: ",(count/21000)\*100)

**SOURCE CODE:-**

Algorithm Applied : Decision Tree

import numpy as np

import matplotlib.pyplot as pt

import pandas as pd

from sklearn.tree import DecisionTreeClassifier

data=pd.read\_csv("C:/Users/Welcome/Desktop/per/dataset/train.csv").as\_matrix()

data

clf=DecisionTreeClassifier()

#training dataset

xtrain=data[0:21000,1:]

train\_label=data[0:21000,0]

clf.fit(xtrain,train\_label)

s=clf.score(xtrain,train\_label)

print(s)

#testing dataset

xtest=data[21000:,1:]

actual\_label=data[21000:,0]

n=int(input("INDEX: "))

d=xtest[n]

d.shape=(28,28)

pt.imshow(255-d,cmap='gray')

pt.savefig('books\_read.jpg')

print(clf.predict([xtest[n]]))

pt.show()

p=clf.predict(xtest)

count=0

for i in range(0,21000):

if(p[i]==actual\_label[i]):

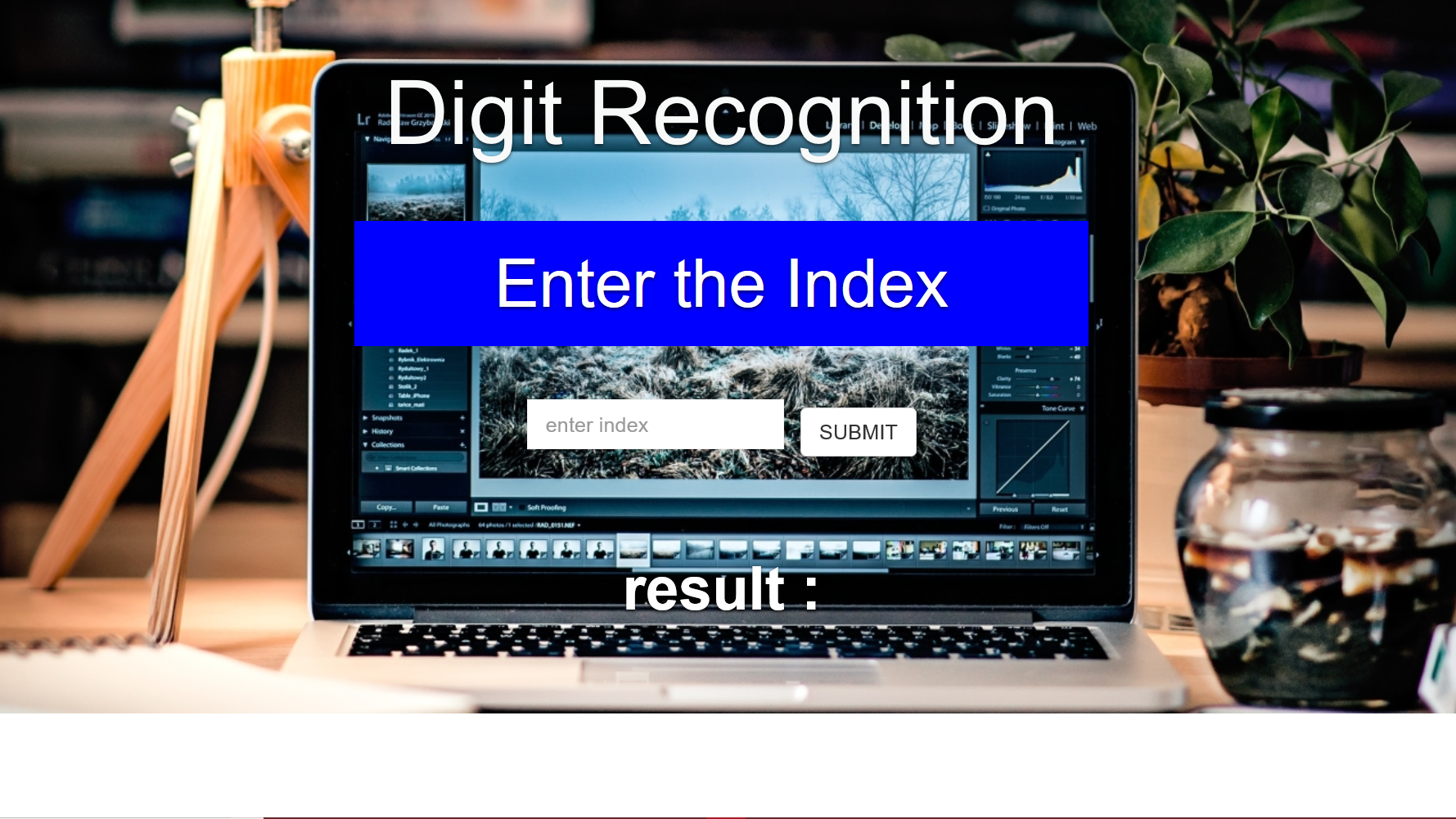
count+=1

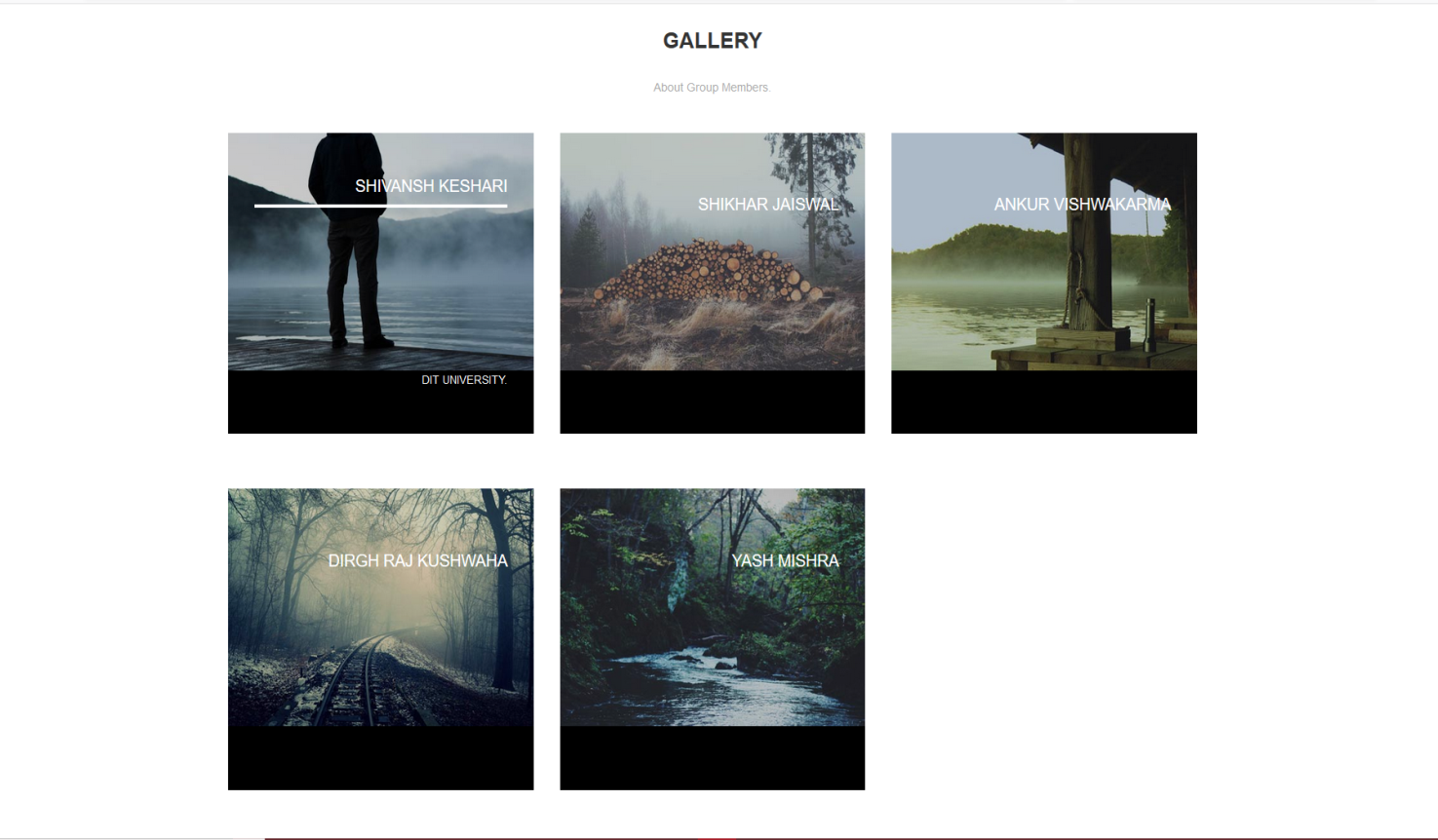
else:

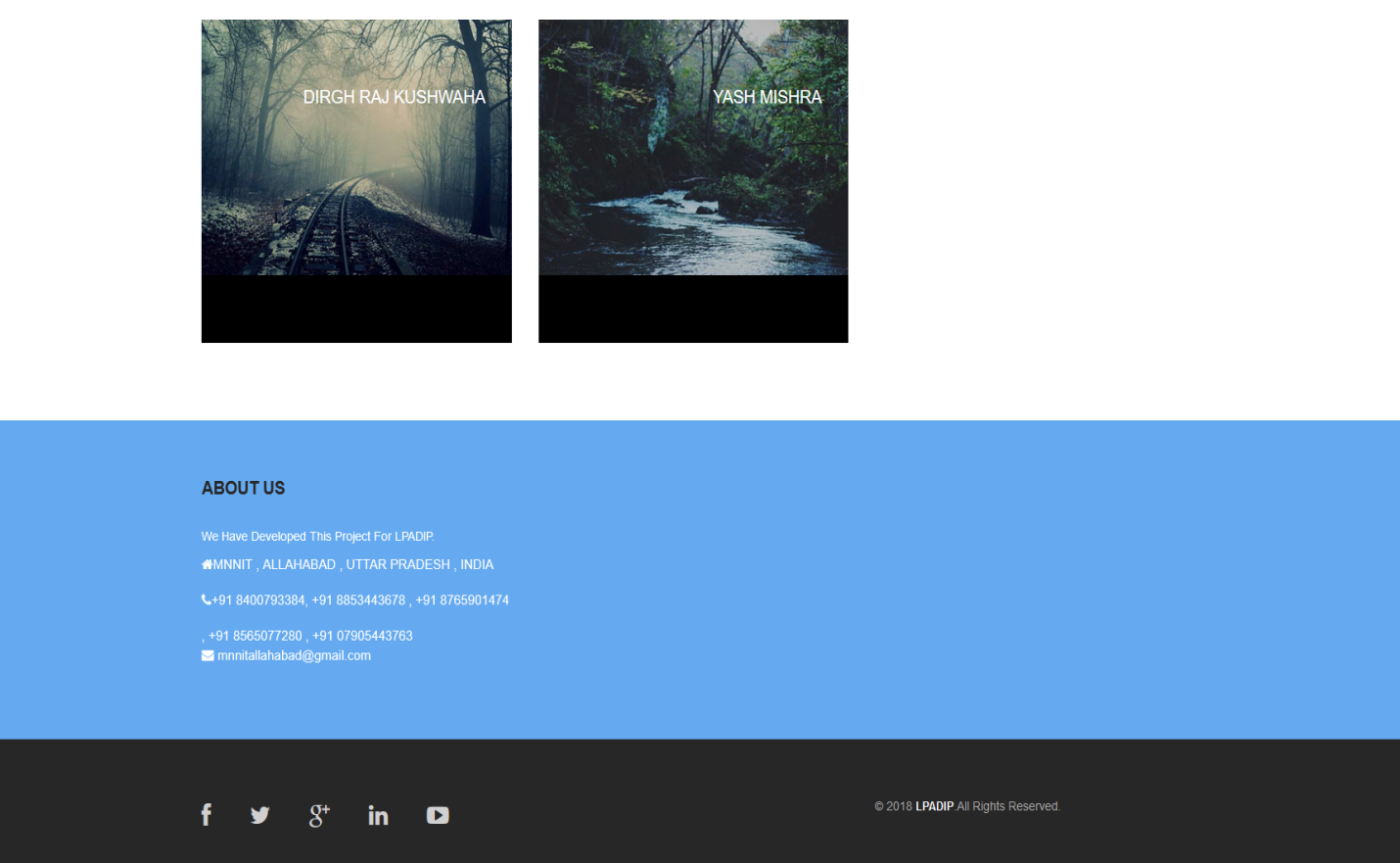
count+=0

print("accuracy: ",(count/21000)\*100)

***WEB APPLICATION USING DJANGO***







**REFERENCES**

* MNIST DATA SET
* KAGGLE
* IEEE.ORG
* MACHINE LEARNING IN SIX STEPS(BOOK)