Breaking Captcha

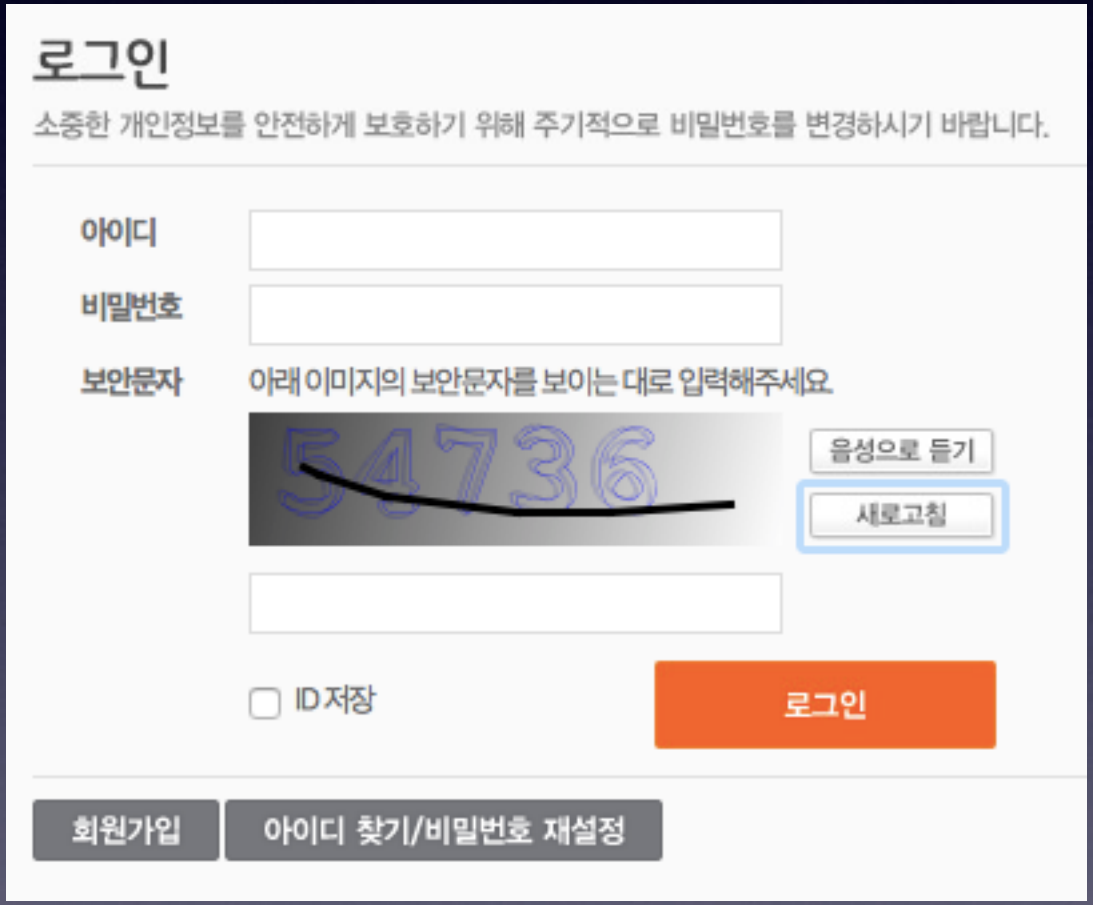
Machine Learing + Computer Vision

JongWon Kim

https://github.com/dikien/break-capctha

Motivation

- Annoying Pictures!



로그인

소중한 개인정보를 안전하게 보호하기 위해 주기적으로 비밀번호를 변경하시기 바랍니다.

아이디

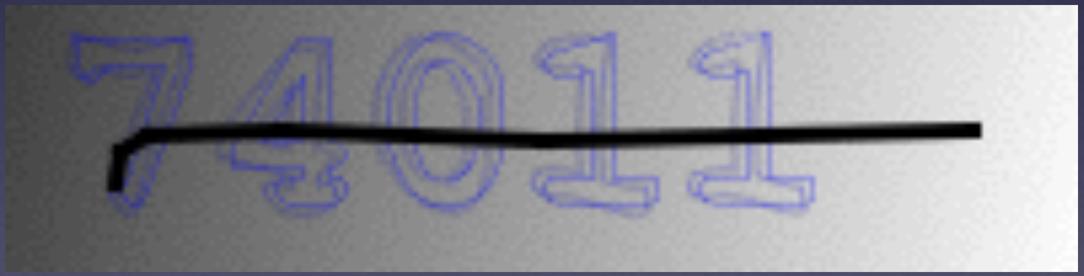
비밀번호

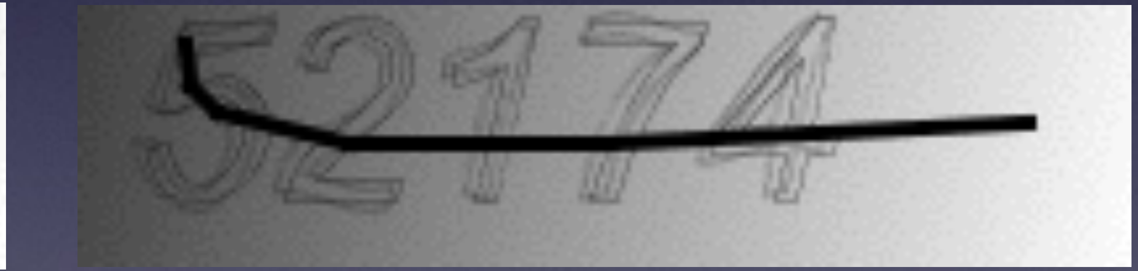
보안문자 아래 이미지의 보안문자를 보이는 대로 입력해주세요.

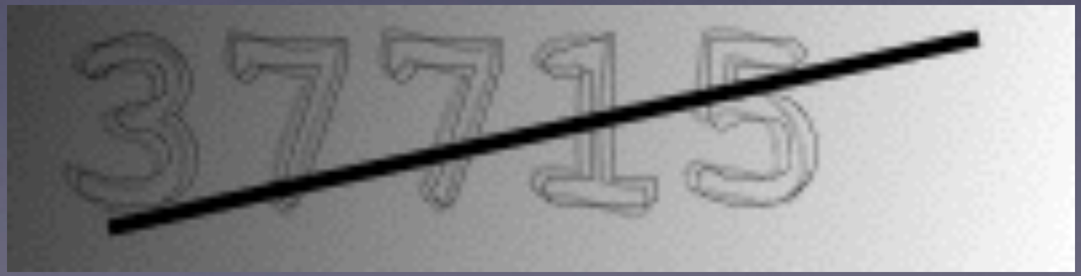
http://www.sktmembership.co.kr/web/html/login/LoginMain.jsp

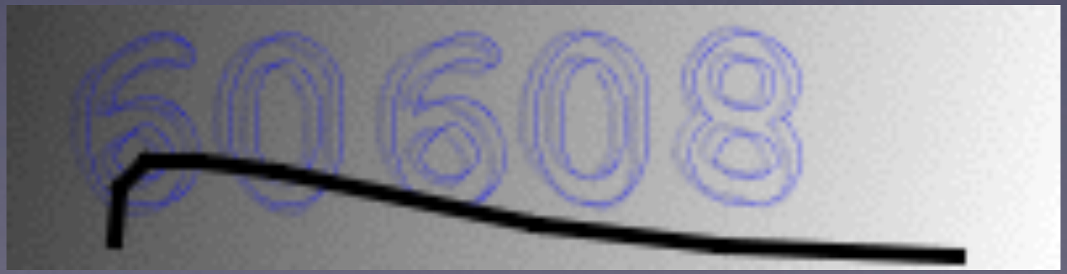
Sample Images

- 9 digits and No alphabet
- Black Line Noise
- 5-digit







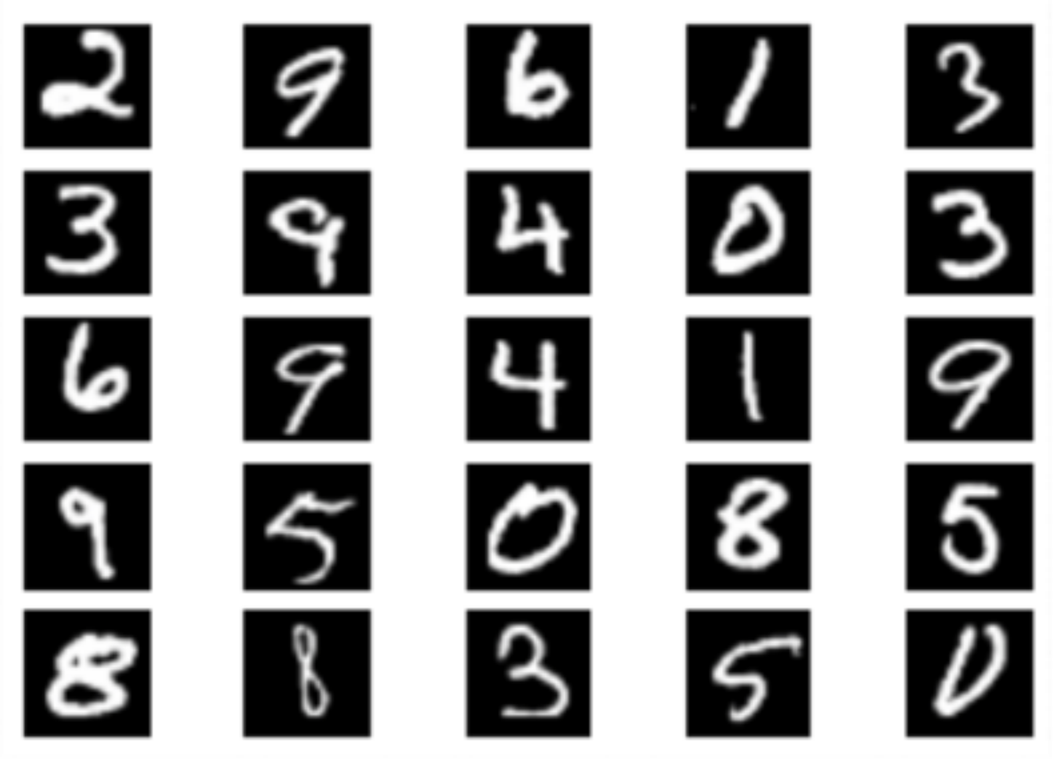


How?

1. **Removing Noises**
    - Black Line
2. **Seperating Digits**
    - Don’t put classifier if not classify into 5-digit
    - Don’t care gray image
3. **Extracting Features**
    - Histogram of Oriented Gradients
4. **Classifiers**
    - LinearSVC, SVC, BernoulliNB, RandomForestClassifier, KNeighborsClassifier, DecisionTreeClassifier, LogisticRegression, RBM + Logistic Regression

MNIST

- The MNIST database of handwritten digits has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image.



[Example of MNIST data]

Plan

- 8 kinds of Classifier X 2 kinds of Feature X 2 kinds of Data Splitting : 32 cases
  - LinearSVC, SVC, BernoulliNB, RandomForestClassifier, KNeighborsClassifier, DecisionTreeClassifier, LogisticRegression, RBM + Logistic Regression

	Train Data	Test Data	Feature
Plan 1	Captcha	Captcha	HOG
Plan 2	Captcha	Captcha	No Feature
Plan 3	MNIST	Captcha	HOG
Plan 4	MNIST	Captcha	No Feature

Background Knowledge

- Computer Vision
  - Computer vision is a field that includes methods for acquiring, processing, analyzing, and understanding images and, in general, high-dimensional data from the real world in order to produce numerical or symbolic information.
- Machine Learning
  - Machine learning is a subfield of computer science that evolved from the study of pattern recognition and computational learning theory in artificial intelligence. Machine learning explores the construction and study of algorithms that can learn from and make predictions on data.

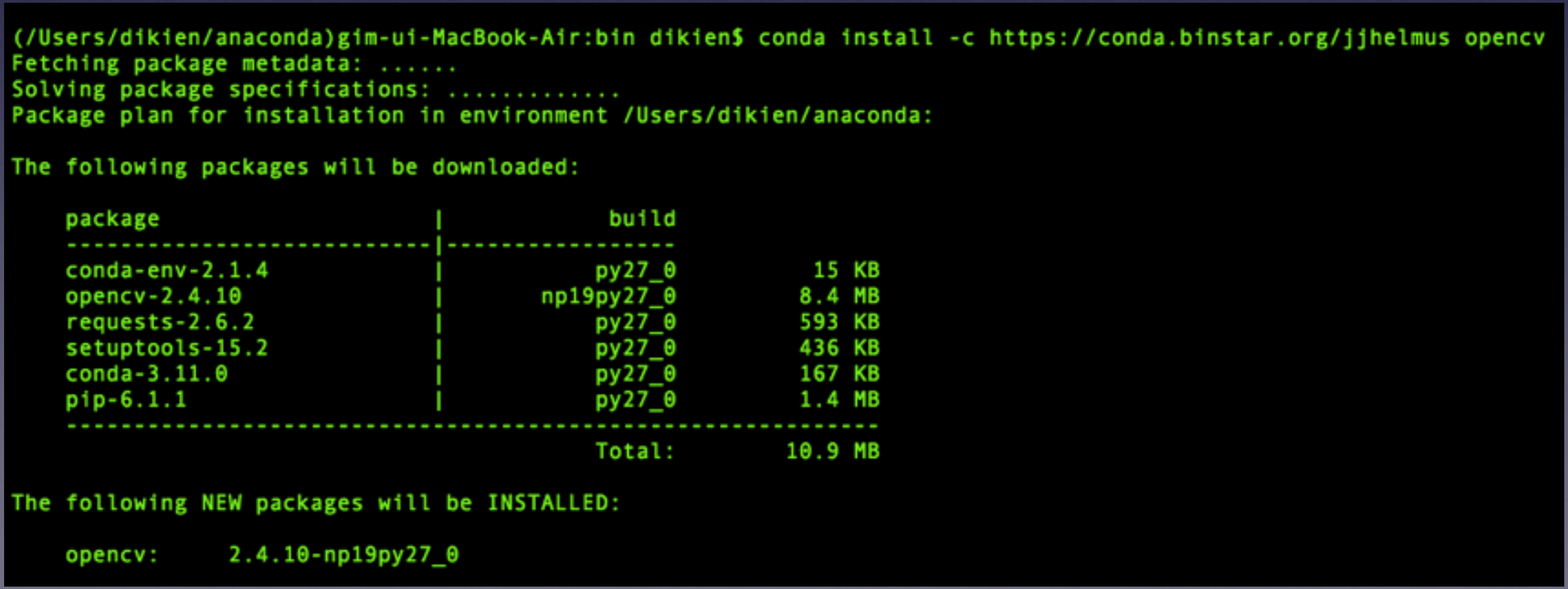
Development Environment

- Anaconda
  - python 2.7.9
  - ipython 3.0.0
  - Maching Learning : Scikit-Learn(v0.15.2)
  - Image Processing : Scikit-Image(v0.11.2), OpenCV(v2.4.10), PIL(v1.1.7)
  - Array processing : Numpy(v1.9.2)
  - Plotting : Matplotlib(v1.4.3)

Install OpenCV on OSX

- ENV Setting : source /Users/dikien/anaconda/bin/activate /Users/dikien/anaconda/

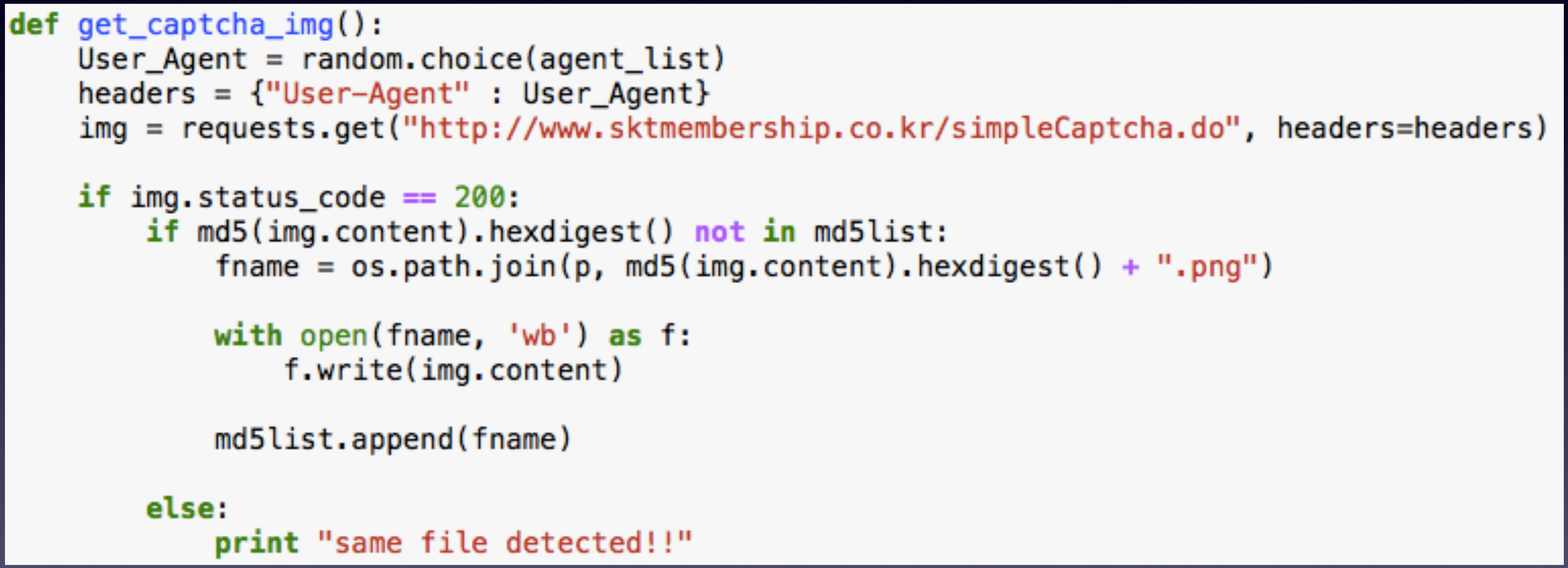
- Install OpenCV : conda install -c https://conda.binstar.org/jjhelmus opencv



```
(/Users/dikien/anaconda)gim-ui-MacBook-Air:bin dikien$ conda install -c https://conda.binstar.org/jjhelmus opencv
Fetching package metadata: ......
Solving package specifications: .............
Package plan for installation in environment /Users/dikien/anaconda:

The following packages will be downloaded:

    package                    |            build
    ---------------------------|-----------------
    conda-env-2.1.4            |            py27_0          15 KB
    opencv-2.4.10              |        np19py27_0         8.4 MB
    requests-2.6.2             |            py27_0         593 KB
    setuptools-15.2            |            py27_0         436 KB
    conda-3.11.0               |            py27_0         167 KB
    pip-6.1.1                  |            py27_0         1.4 MB
    ------------------------------------------------------------
                                           Total:        10.9 MB

The following NEW packages will be INSTALLED:

    opencv:     2.4.10-np19py27_0
```

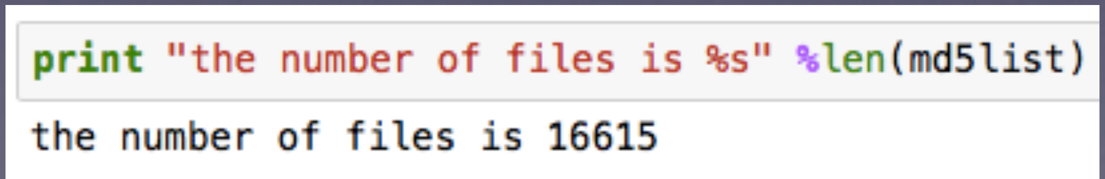
Step1. Get Captcha Image

- Save captcha images from servers



```
def get_captcha_img():
    User_Agent = random.choice(agent_list)
    headers = {"User-Agent" : User_Agent}
    img = requests.get("http://www.sktmembership.co.kr/simpleCaptcha.do", headers=headers)

    if img.status_code == 200:
        if md5(img.content).hexdigest() not in md5list:
            fname = os.path.join(p, md5(img.content).hexdigest() + ".png")

            with open(fname, 'wb') as f:
                f.write(img.content)

            md5list.append(fname)

        else:
            print "same file detected!!"
```

- The number of images is 16615

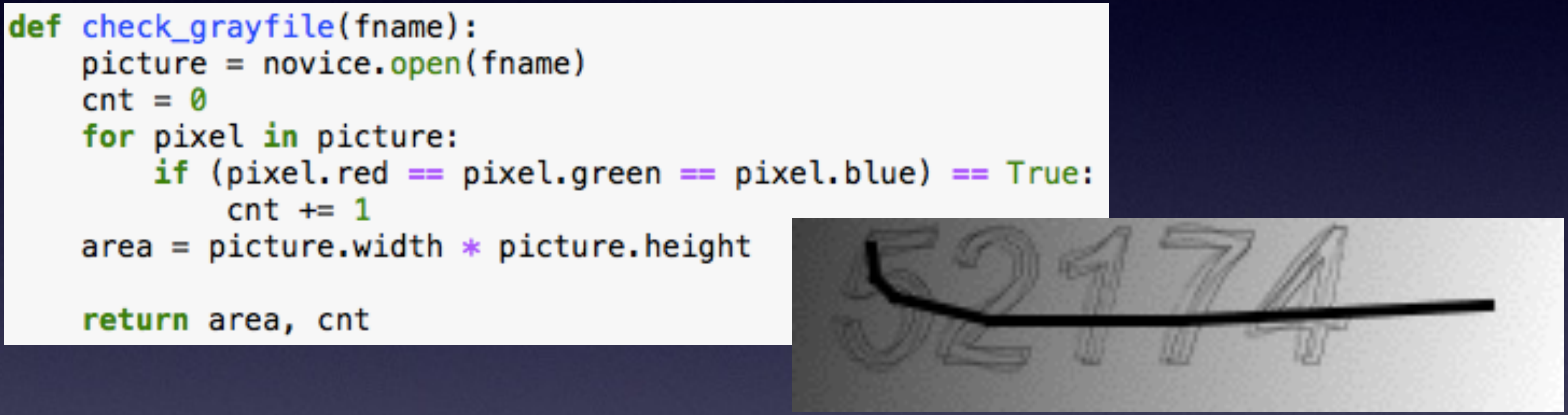


```
print "the number of files is %s" %len(md5list)
```

```
the number of files is 16615
```

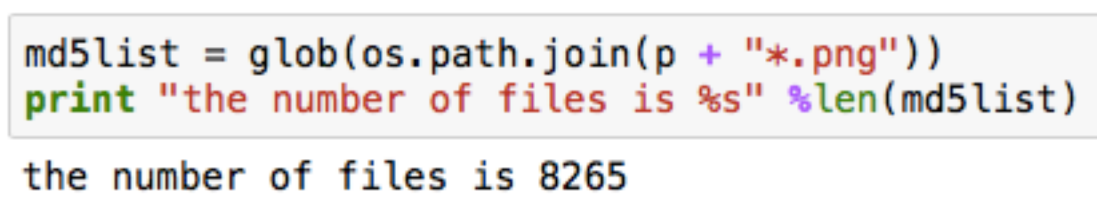
Step1. Get Captcha Image

- Remove images like below gray style



```python
def check_grayfile(fname):
    picture = novice.open(fname)
    cnt = 0
    for pixel in picture:
        if (pixel.red == pixel.green == pixel.blue) == True:
            cnt += 1
    area = picture.width * picture.height

    return area, cnt
```

- After removing, left 8265 images

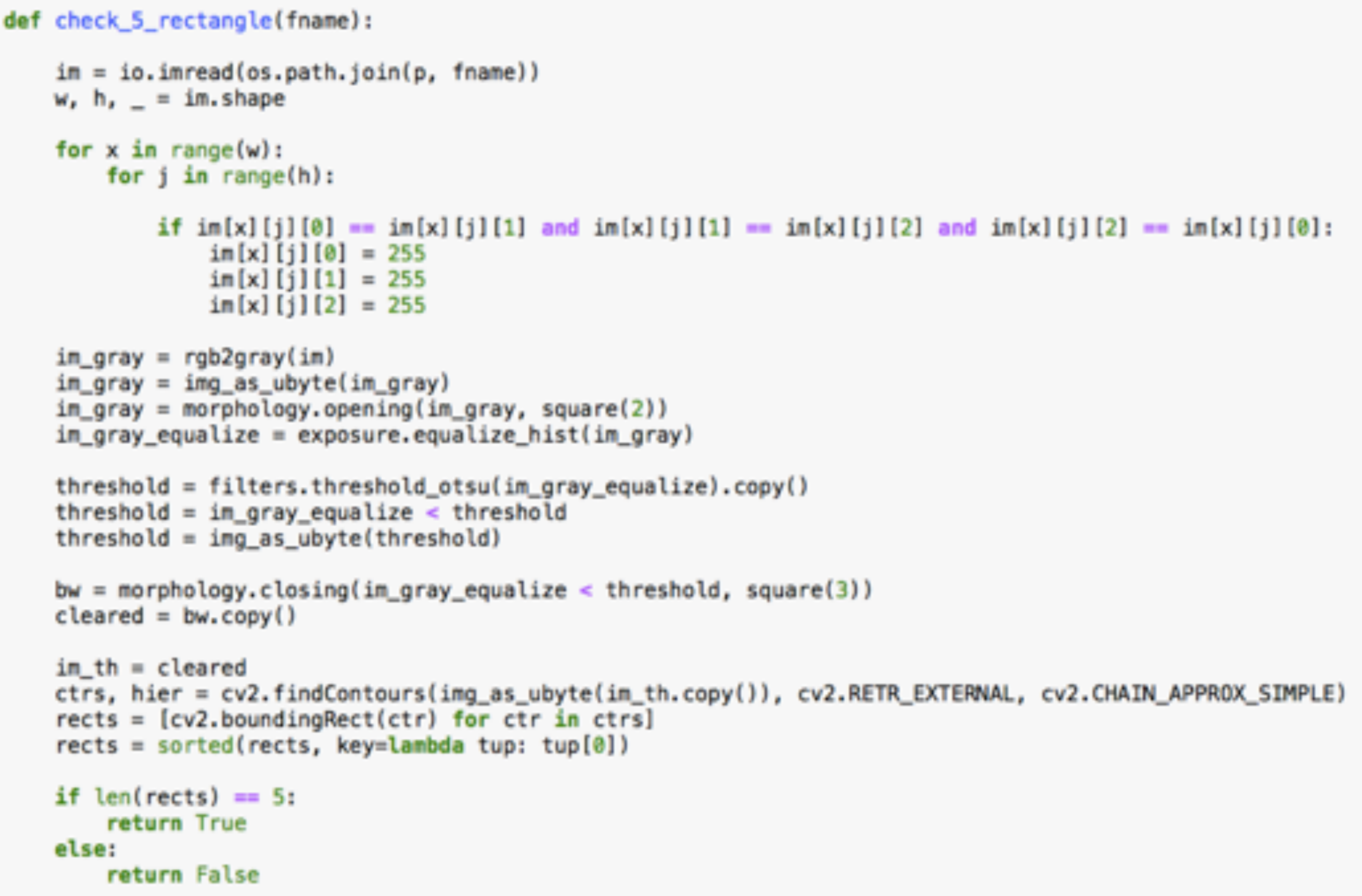


```python
md5list = glob(os.path.join(p + "*.png"))
print "the number of files is %s" %len(md5list)
```

```
the number of files is 8265
```

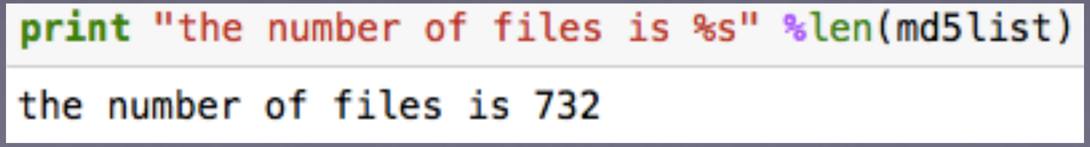
Step1. Get Captcha Image

- If we can’t recognize 5-digits, remove them



```python
def check_5_rectangle(fname):

    im = io.imread(os.path.join(p, fname))
    w, h, _ = im.shape

    for x in range(w):
        for j in range(h):

            if im[x][j][0] == im[x][j][1] and im[x][j][1] == im[x][j][2] and im[x][j][2] == im[x][j][0]:
                im[x][j][0] = 255
                im[x][j][1] = 255
                im[x][j][2] = 255

    im_gray = rgb2gray(im)
    im_gray = img_as_ubyte(im_gray)
    im_gray = morphology.opening(im_gray, square(2))
    im_gray_equalize = exposure.equalize_hist(im_gray)

    threshold = filters.threshold_otsu(im_gray_equalize).copy()
    threshold = im_gray_equalize < threshold
    threshold = img_as_ubyte(threshold)

    bw = morphology.closing(im_gray_equalize < threshold, square(3))
    cleared = bw.copy()

    im_th = cleared
    ctrs, hier = cv2.findContours(img_as_ubyte(im_th.copy()), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    rects = [cv2.boundingRect(ctr) for ctr in ctrs]
    rects = sorted(rects, key=lambda tup: tup[0])

    if len(rects) == 5:
        return True
    else:
        return False
```

- After removing, left 732 images

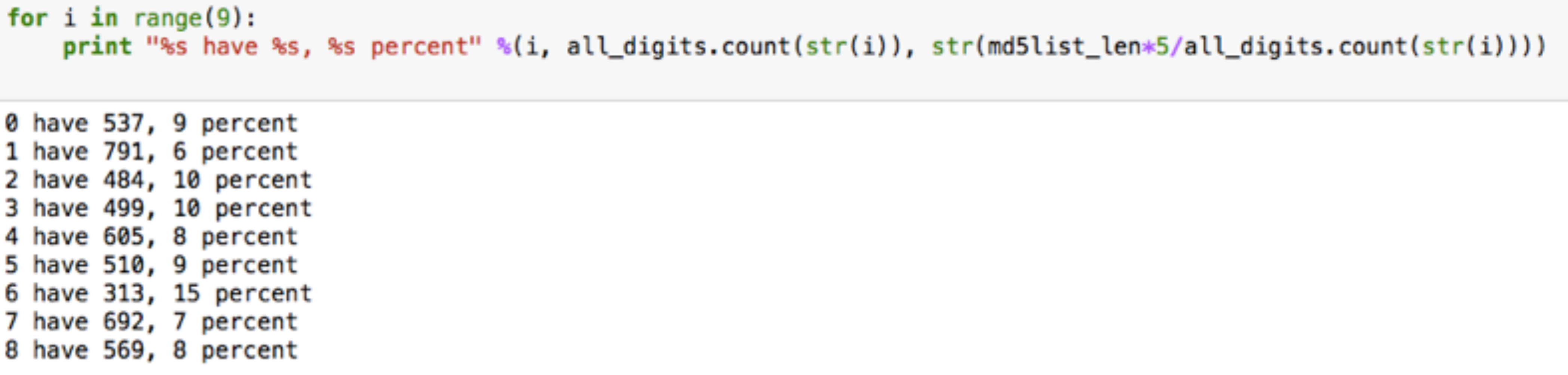


```python
print "the number of files is %s" %len(md5list)
```

```
the number of files is 732
```

Step1. Get Captcha Image

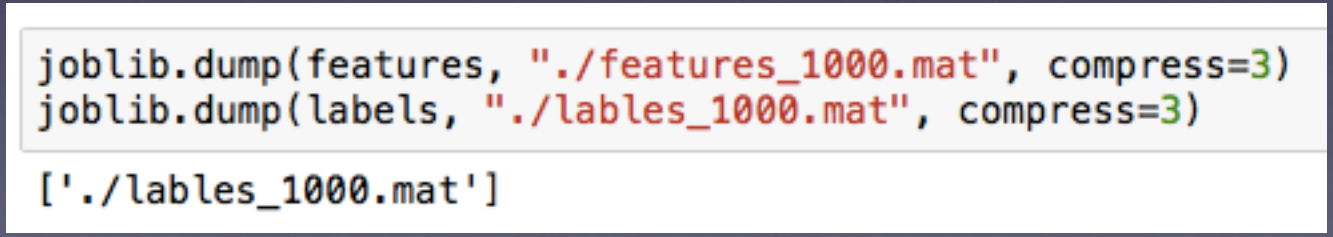
- Digits ratio from 732 images



```
for i in range(9):
    print "%s have %s, %s percent" %(i, all_digits.count(str(i)), str(md5list_len*5/all_digits.count(str(i))))
```

```
0 have 537, 9 percent
1 have 791, 6 percent
2 have 484, 10 percent
3 have 499, 10 percent
4 have 605, 8 percent
5 have 510, 9 percent
6 have 313, 15 percent
7 have 692, 7 percent
8 have 569, 8 percent
```

- Convert images to numpy type and save them

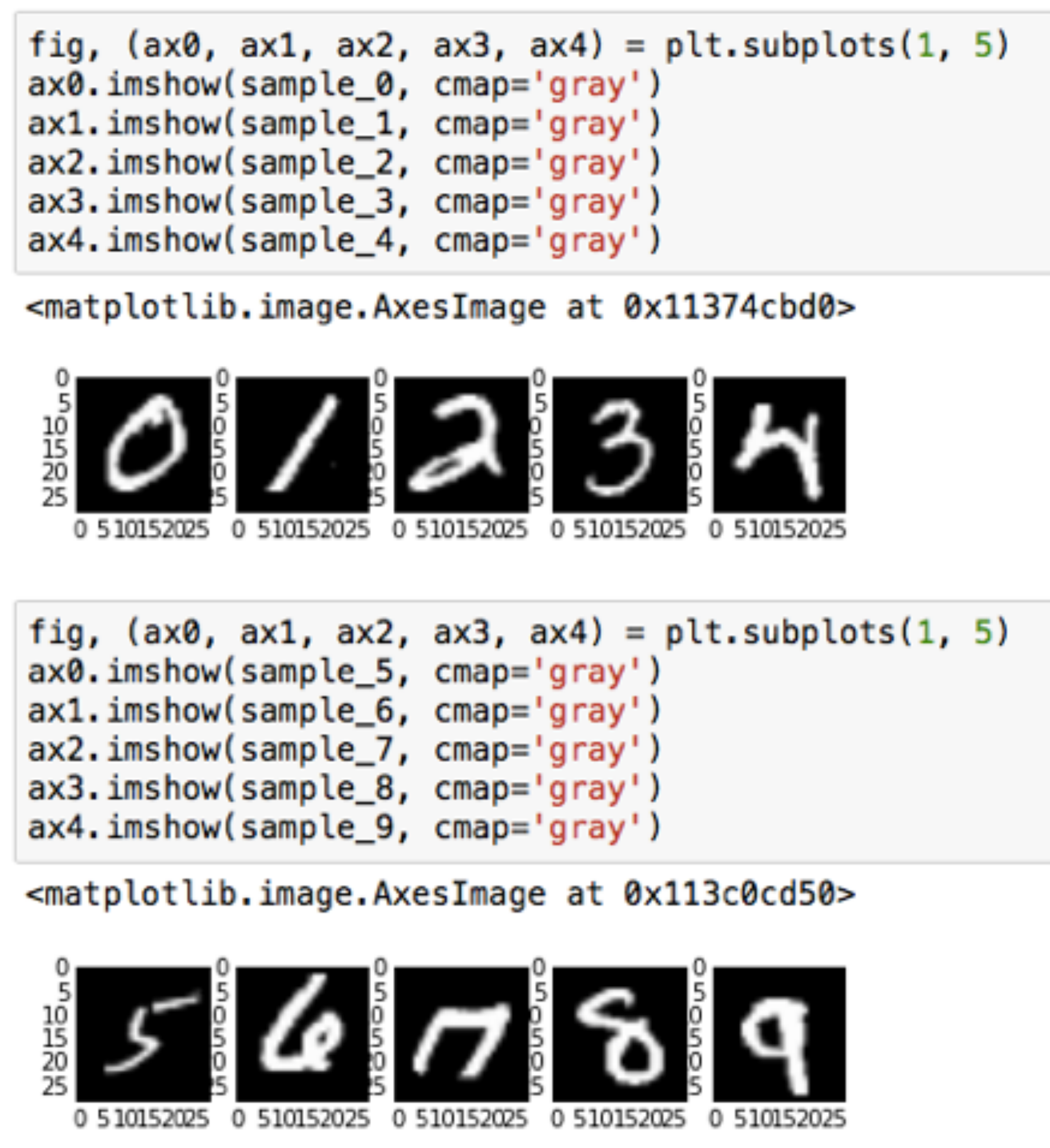


```
joblib.dump(features, "./features_1000.mat", compress=3)
joblib.dump(labels, "./lables_1000.mat", compress=3)
```

```
['./lables_1000.mat']
```

Step2. Data Exploratory

- MNIST Dataset



```
fig, (ax0, ax1, ax2, ax3, ax4) = plt.subplots(1, 5)
ax0.imshow(sample_0, cmap='gray')
ax1.imshow(sample_1, cmap='gray')
ax2.imshow(sample_2, cmap='gray')
ax3.imshow(sample_3, cmap='gray')
ax4.imshow(sample_4, cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x11374cbd0>
```

```
fig, (ax0, ax1, ax2, ax3, ax4) = plt.subplots(1, 5)
ax0.imshow(sample_5, cmap='gray')
ax1.imshow(sample_6, cmap='gray')
ax2.imshow(sample_7, cmap='gray')
ax3.imshow(sample_8, cmap='gray')
ax4.imshow(sample_9, cmap='gray')
```

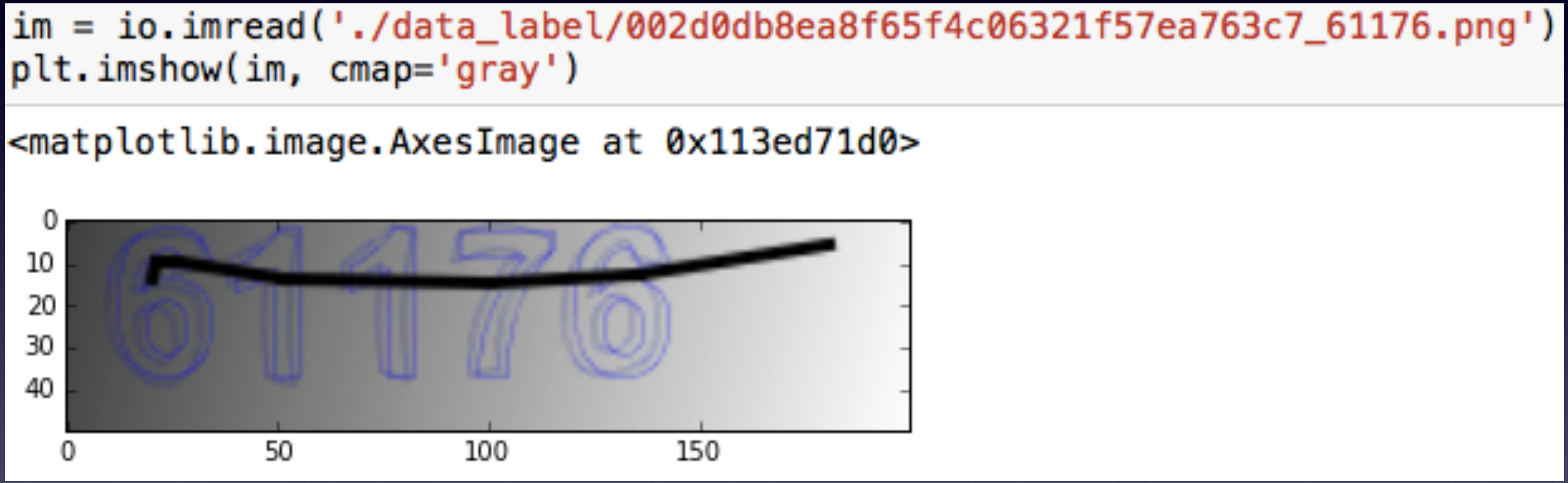
```
<matplotlib.image.AxesImage at 0x113c0cd50>
```

Step2. Data Exploratory

- Q1. If we train mnist dataset, can we predict the captcha image?
- Q2. If we train captcha dataset, can we predict the new captcha image?

Step2. Data Exploratory

- Captcha Image

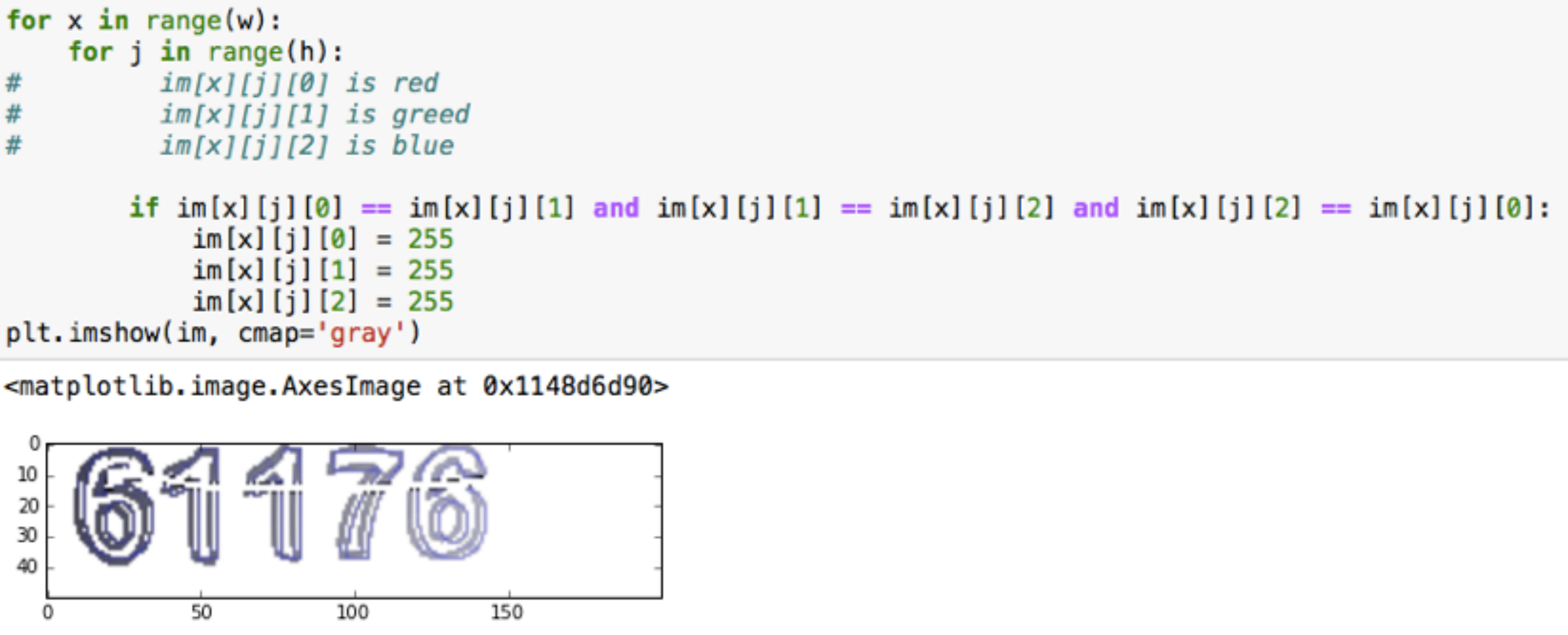


```
im = io.imread('./data_label/002d0db8ea8f65f4c06321f57ea763c7_61176.png')
plt.imshow(im, cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x113ed71d0>
```

Step2. Data Exploratory

- Remove Noise
  - Background and Black line

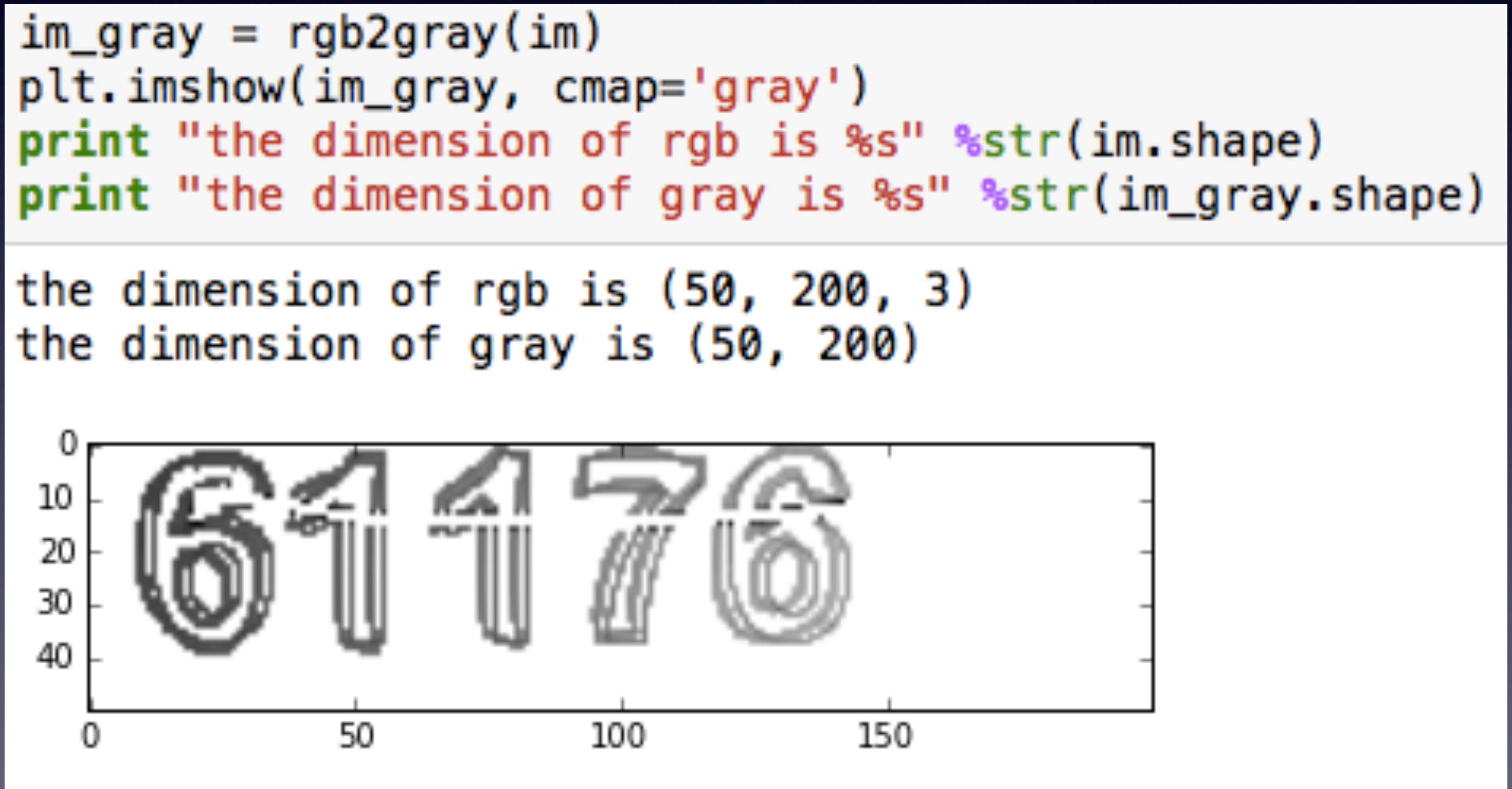


```
for x in range(w):
    for j in range(h):
#           im[x][j][0] is red
#           im[x][j][1] is greed
#           im[x][j][2] is blue

        if im[x][j][0] == im[x][j][1] and im[x][j][1] == im[x][j][2] and im[x][j][2] == im[x][j][0]:
            im[x][j][0] = 255
            im[x][j][1] = 255
            im[x][j][2] = 255
plt.imshow(im, cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x1148d6d90>
```

Step2. Data Exploratory

- Chagne RGB to Gray mode

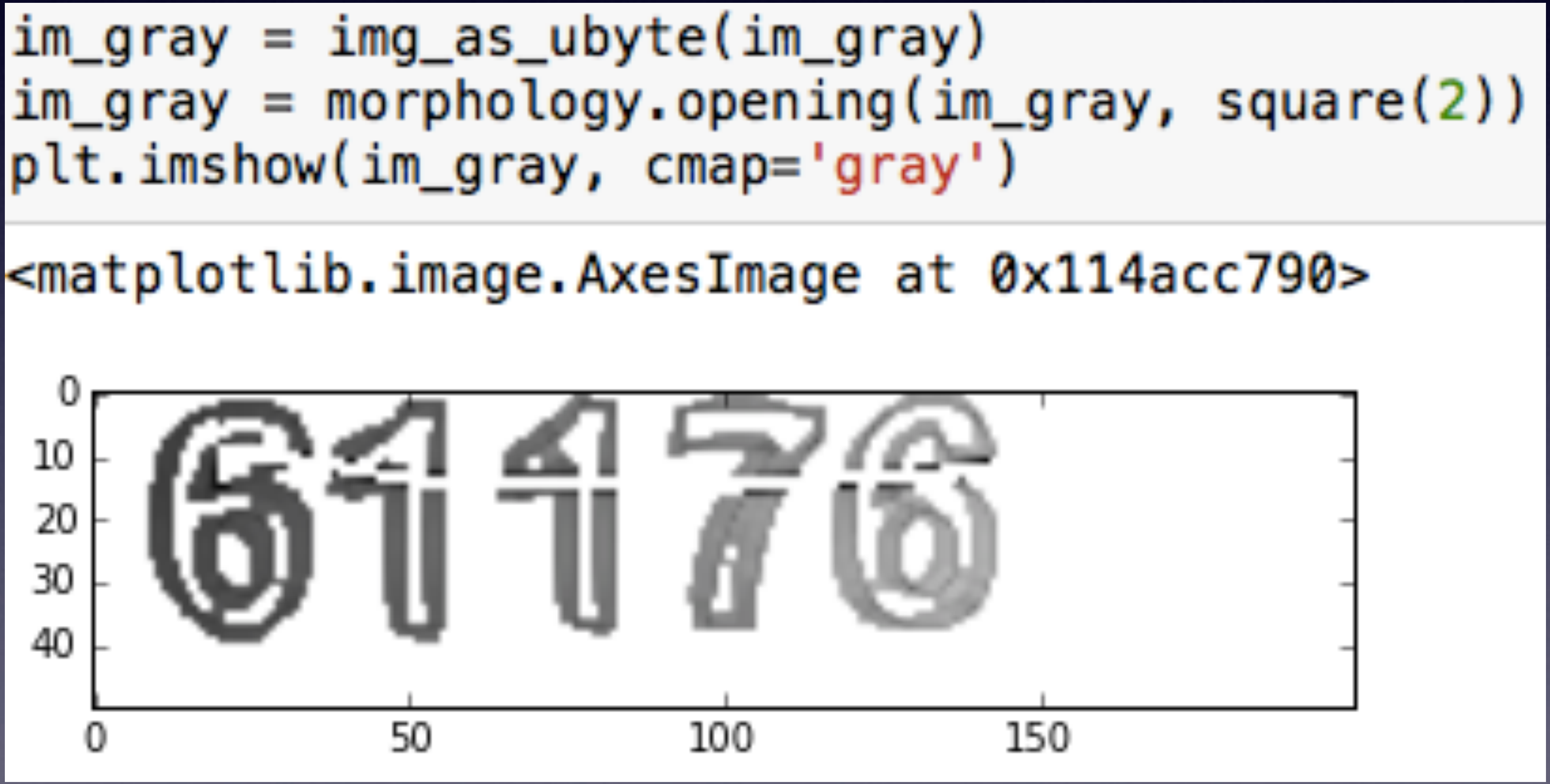


```
im_gray = rgb2gray(im)
plt.imshow(im_gray, cmap='gray')
print "the dimension of rgb is %s" %str(im.shape)
print "the dimension of gray is %s" %str(im_gray.shape)
```

```
the dimension of rgb is (50, 200, 3)
the dimension of gray is (50, 200)
```

Step2. Data Exploratory

- Apply Opening

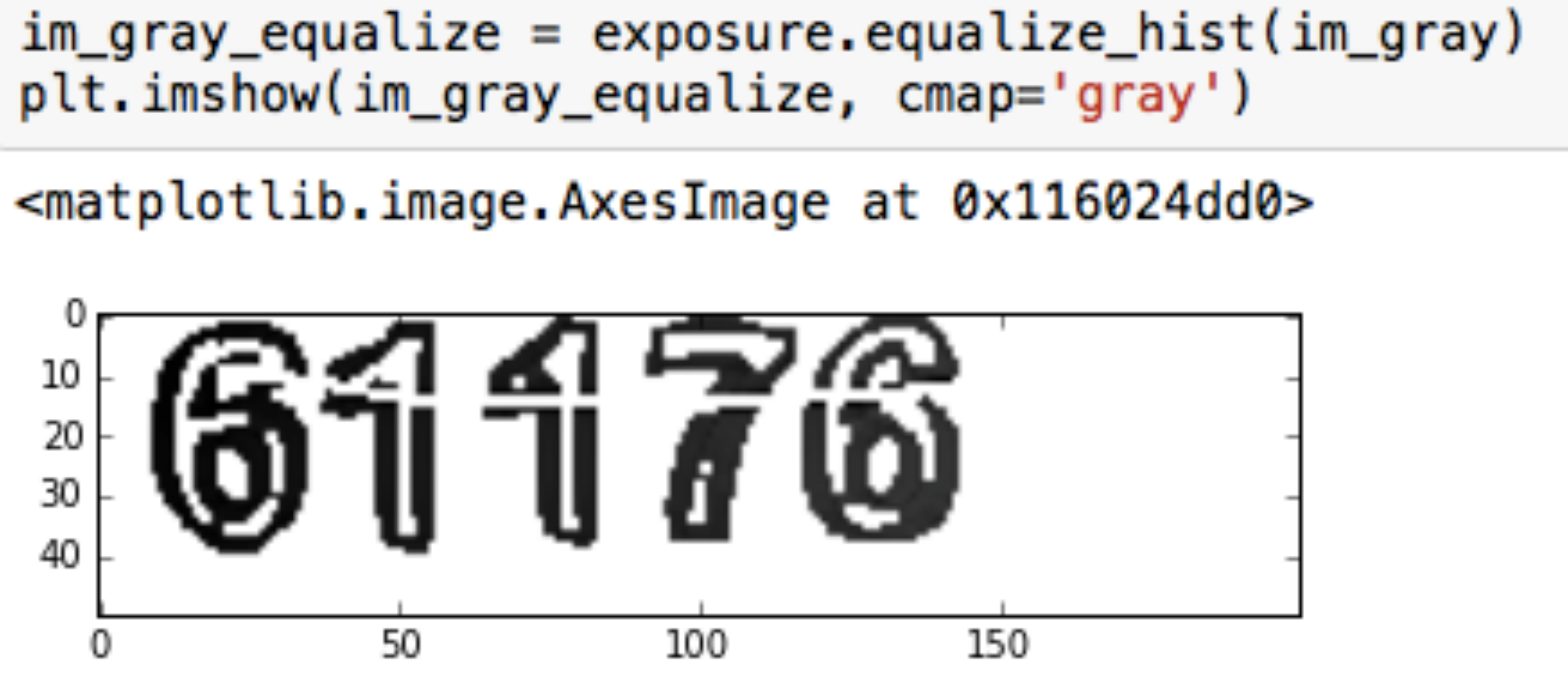


```
im_gray = img_as_ubyte(im_gray)
im_gray = morphology.opening(im_gray, square(2))
plt.imshow(im_gray, cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x114acc790>
```

Step2. Data Exploratory

- Apply Equalize Hist

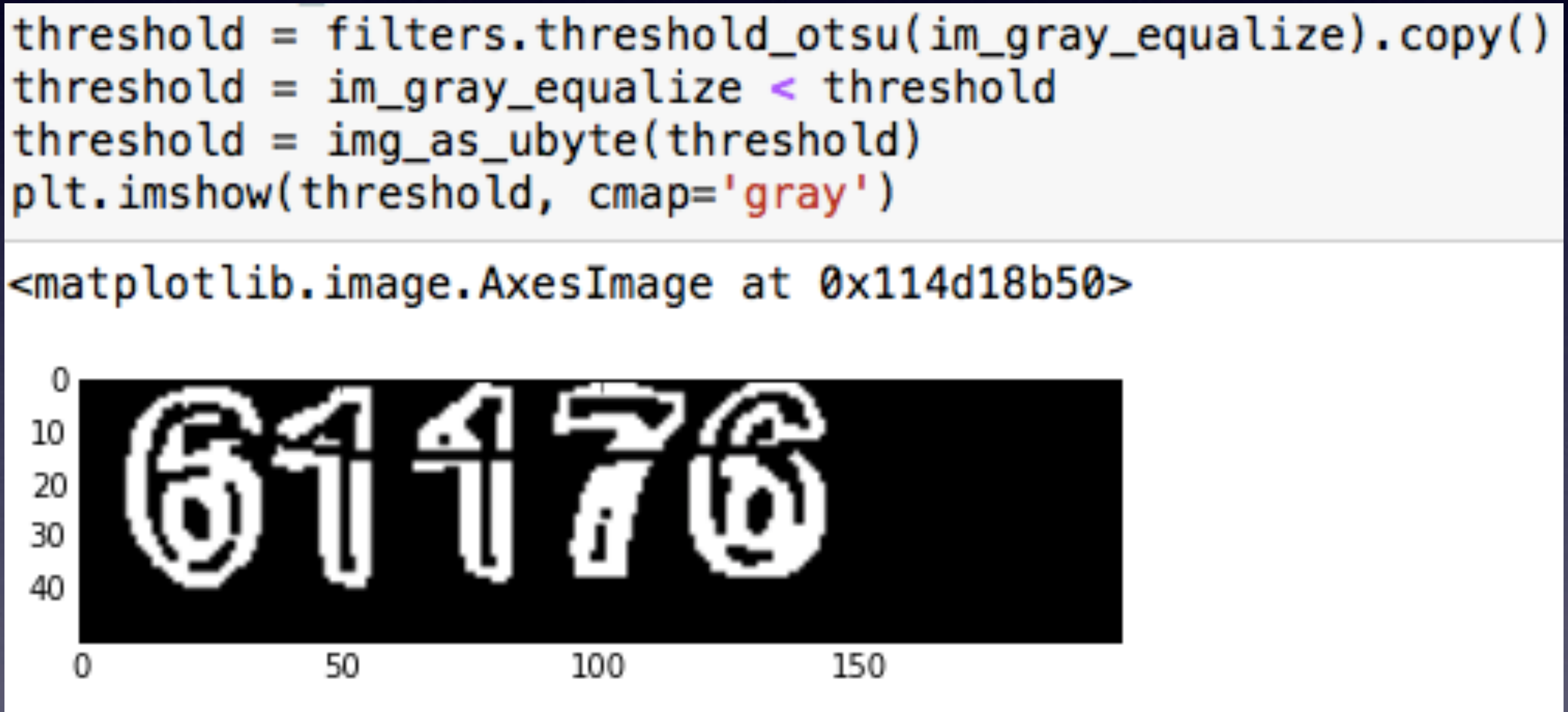


```
im_gray_equalize = exposure.equalize_hist(im_gray)
plt.imshow(im_gray_equalize, cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x116024dd0>
```

Step2. Data Exploratory

- Apply Threshold Otsu

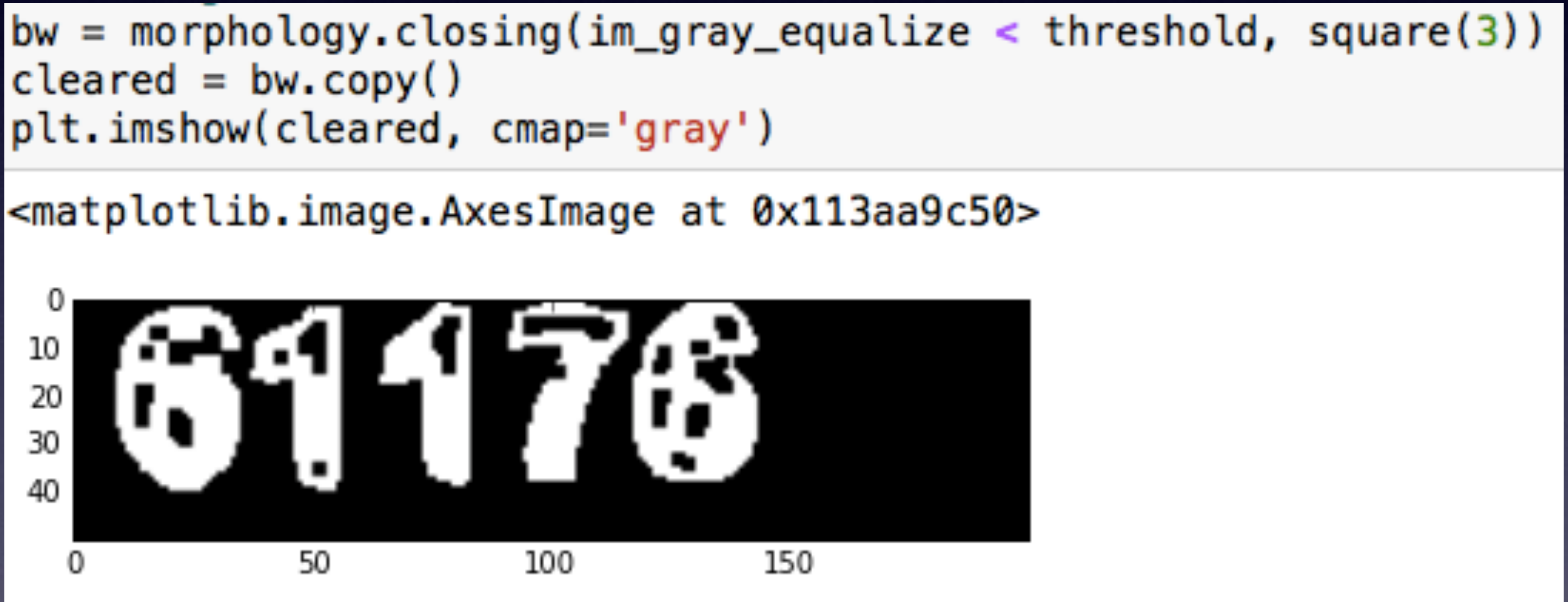


```
threshold = filters.threshold_otsu(im_gray_equalize).copy()
threshold = im_gray_equalize < threshold
threshold = img_as_ubyte(threshold)
plt.imshow(threshold, cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x114d18b50>
```

Step2. Data Exploratory

- Apply Closing

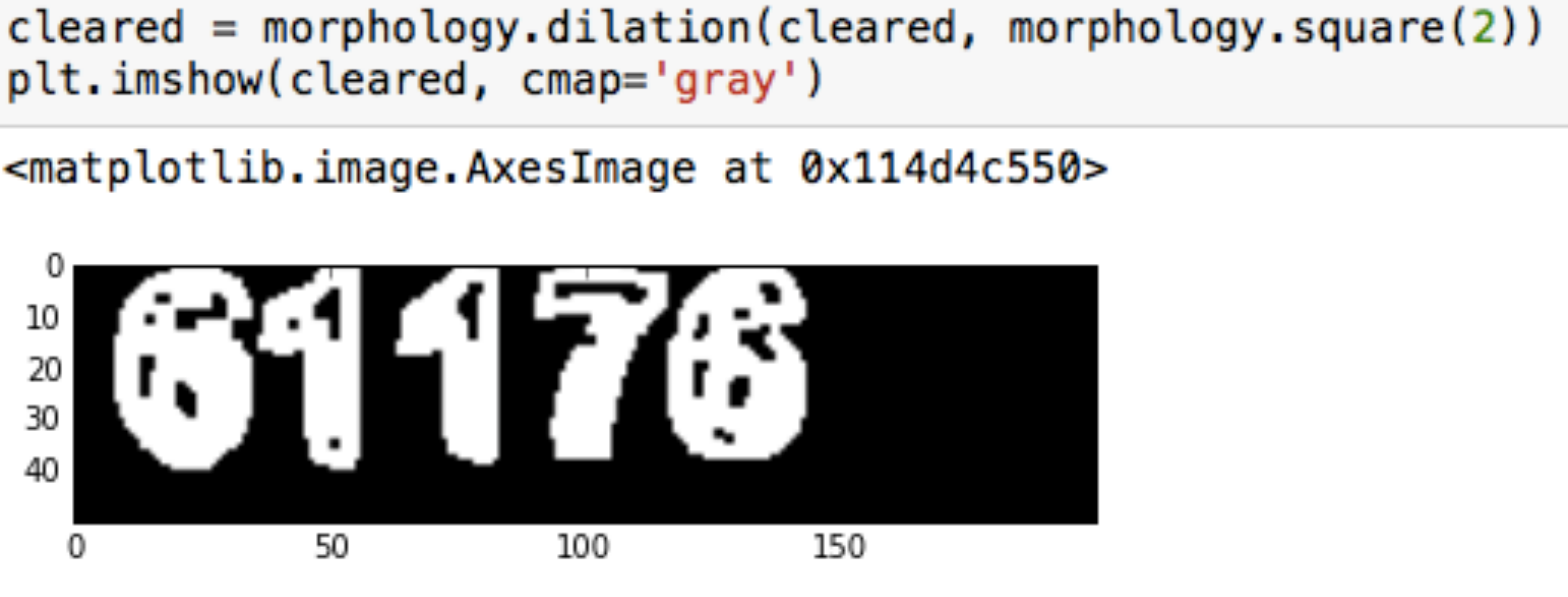


```
bw = morphology.closing(im_gray_equalize < threshold, square(3))
cleared = bw.copy()
plt.imshow(cleared, cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x113aa9c50>
```

Step2. Data Exploratory

- Apply Dilation

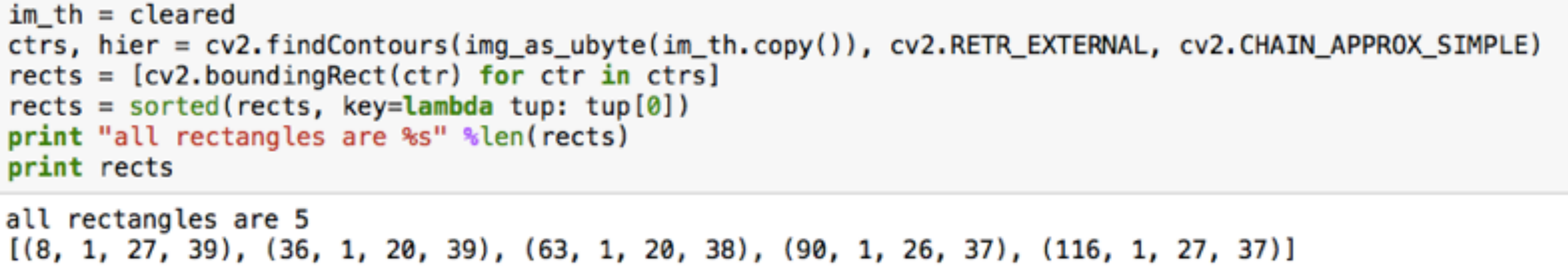


```
cleared = morphology.dilation(cleared, morphology.square(2))
plt.imshow(cleared, cmap='gray')
```

```
<matplotlib.image.AxesImage at 0x114d4c550>
```

Step2. Data Exploratory

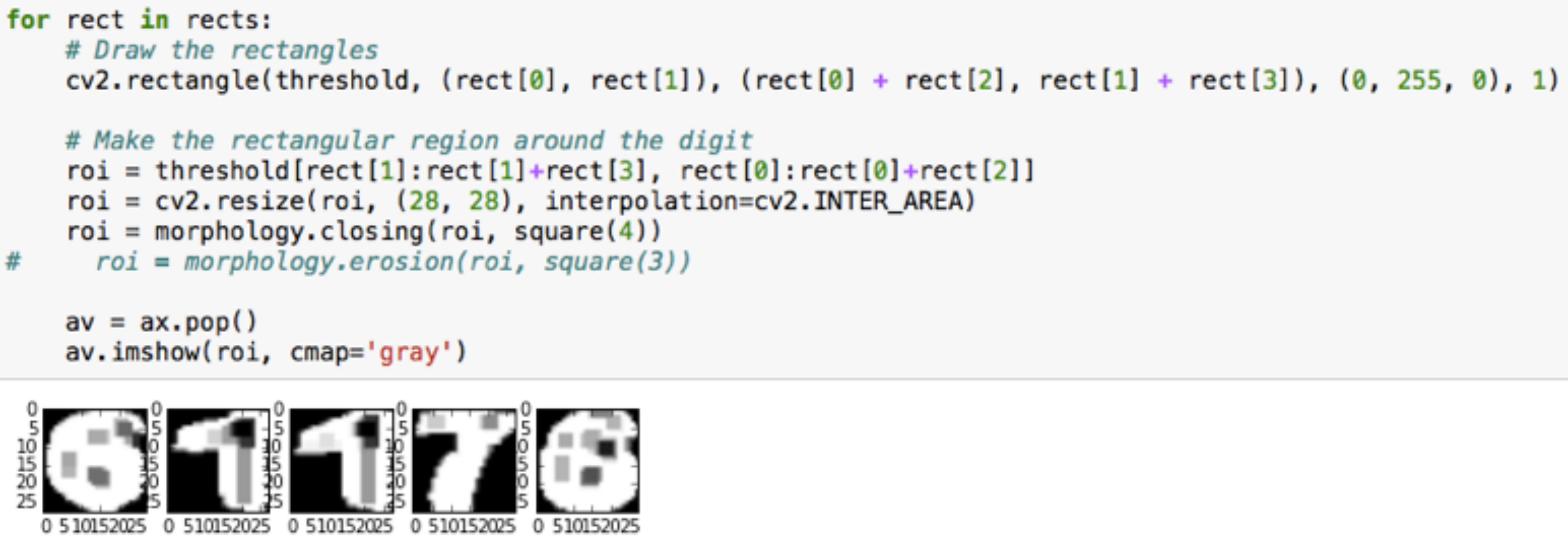
- Divide into 5-digits



```
im_th = cleared
ctrs, hier = cv2.findContours(img_as_ubyte(im_th.copy()), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
rects = [cv2.boundingRect(ctr) for ctr in ctrs]
rects = sorted(rects, key=lambda tup: tup[0])
print "all rectangles are %s" %len(rects)
print rects
```

```
all rectangles are 5
[(8, 1, 27, 39), (36, 1, 20, 39), (63, 1, 20, 38), (90, 1, 26, 37), (116, 1, 27, 37)]
```

- Print out each of digits



```
for rect in rects:
    # Draw the rectangles
    cv2.rectangle(threshold, (rect[0], rect[1]), (rect[0] + rect[2], rect[1] + rect[3]), (0, 255, 0), 1)

    # Make the rectangular region around the digit
    roi = threshold[rect[1]:rect[1]+rect[3], rect[0]:rect[0]+rect[2]]
    roi = cv2.resize(roi, (28, 28), interpolation=cv2.INTER_AREA)
    roi = morphology.closing(roi, square(4))
#     roi = morphology.erosion(roi, square(3))

    av = ax.pop()
    av.imshow(roi, cmap='gray')
```

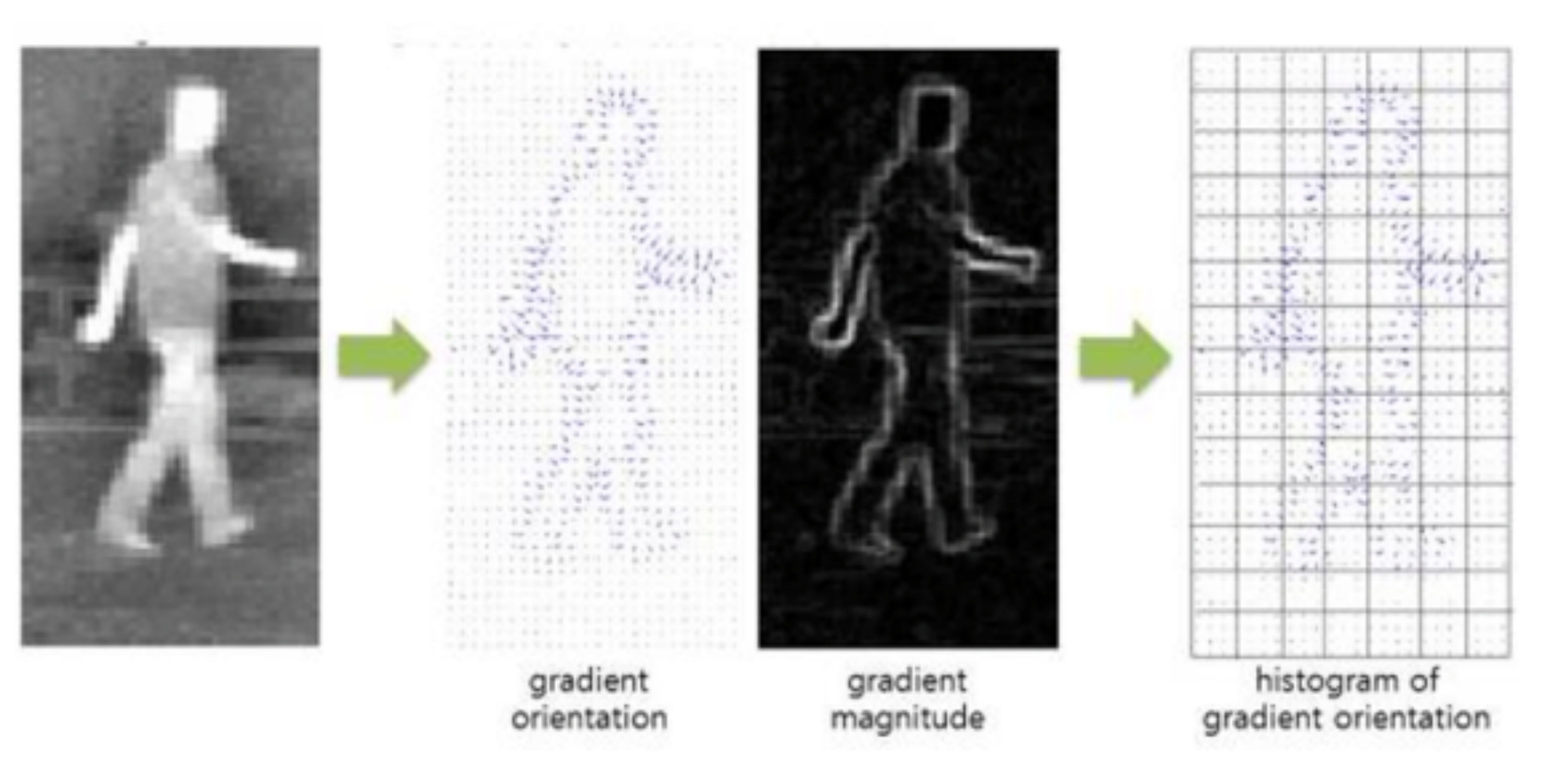
Step3. Clssifier GridSearch

	Train Data	Test Data	Feature
Plan 1	Captcha	Captcha	HOG
Plan 2	Captcha	Captcha	No Feature
Plan 3	MNIST	Captcha	HOG
Plan 4	MNIST	Captcha	No Feature

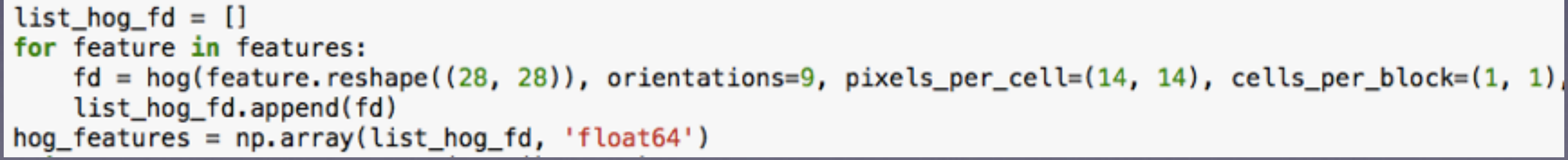
- Score 1 : Train(700 mnist images), Test(300 captcha images)
- Score 2 : Train(700 mnist images), Test(200 captcha images)

Step3. Clssifier GridSearch

- Histogram of oriented gradients



- Apply Histogram of oriented gradients and train



```
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),
    list_hog_fd.append(fd)
hog_features = np.array(list_hog_fd, 'float64')
```

Step3. Clssifier GridSearch

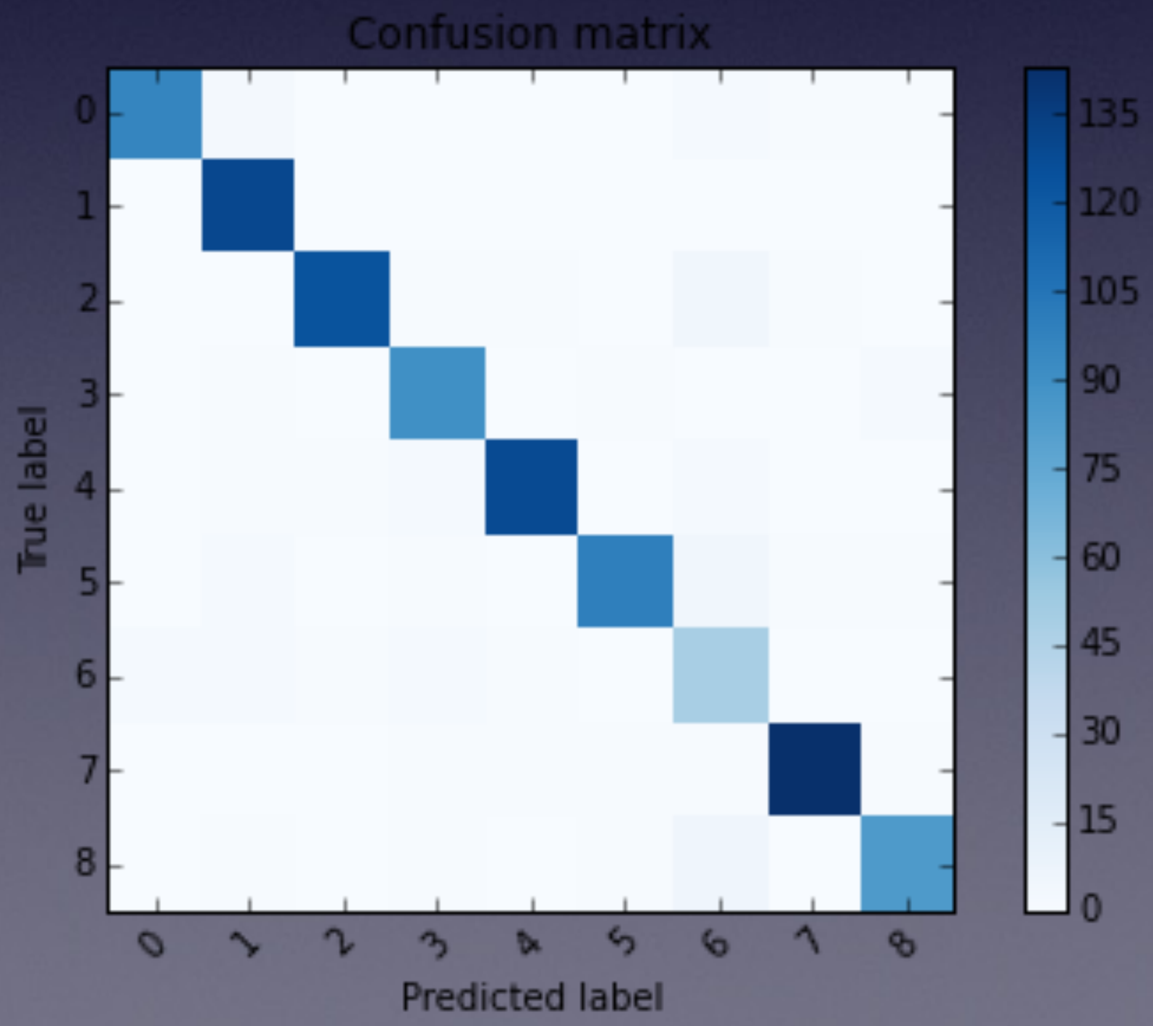
	Train Data	Test Data	Feature
Plan 1	Captcha	Captcha	HOG

Classifier	Score1	Score2
LinearSVC	0.9174	0.922
SVC	0.9294	0.943
BernoulliNB	0.713	0.731
RandomForestClassifier	0.9322	0.937
KNeighborsClassifier	0.9304	0.941
DecisionTreeClassifier	0.8904	0.897
LogisticRegression	0.920	0.926
RBM + Logistic Regression	0.840	0.853
Avg	0.8841	0.8937

Step3. Clssifier GridSearch

	Train Data	Test Data	Feature
Plan 1	Captcha	Captcha	HOG

- Confusion matrix of SVC(Best Classifier)



Step3. Clssifier GridSearch

	Train Data	Test Data	Feature
Plan 1	Captcha	Captcha	HOG

- Confusion matrix of SVC(Best Classifier)

	0	1	2	3	4	5	6	7	8
0	96	3	0	0	0	0	2	1	1
1	0	130	0	0	0	0	0	0	0
2	0	0	124	1	1	0	5	1	0
3	0	1	0	90	0	1	0	0	2
4	0	1	1	2	128	0	2	0	0
5	0	2	0	1	0	99	5	1	1
6	2	2	1	2	1	0	49	0	0
7	0	0	0	1	1	1	1	143	1
8	0	1	0	1	0	1	6	0	84

Step3. Clssifier GridSearch

	Train Data	Test Data	Feature
Plan 1	Captcha	Captcha	HOG
Plan 2	Captcha	Captcha	No Feature
Plan 3	MNIST	Captcha	HOG
Plan 4	MNIST	Captcha	No Feature

- Score 1 : Train(700 mnist images), Test(300 captcha images)
- Score 2 : Train(700 mnist images), Test(200 captcha images)

Step3. Clssifier GridSearch

	Train Data	Test Data	Feature
Plan 2	Captcha	Captcha	No Feature

Classifier	Score1	Score2
LinearSVC	0.9286	0.932
SVC	0.9294	0.947
BernoulliNB	0.9064	0.915
RandomForestClassifier	0.936	0.942
KNeighborsClassifier	0.9322	0.937
DecisionTreeClassifier	0.9156	0.911
LogisticRegression	0.934	0.939
RBM + Logistic Regression	0.931	0.937
Avg	0.9266	0.9325

Step3. Clssifier GridSearch

	Train Data	Test Data	Feature
Plan 1	Captcha	Captcha	HOG
Plan 2	Captcha	Captcha	No Feature
Plan 3	MNIST	Captcha	HOG
Plan 4	MNIST	Captcha	No Feature

- Score 1 : Train(700 mnist images), Test(200 captcha Images)
- Score 2 : Train(700 mnist images), Test(200 captcha Images)

Step3. Clssifier GridSearch

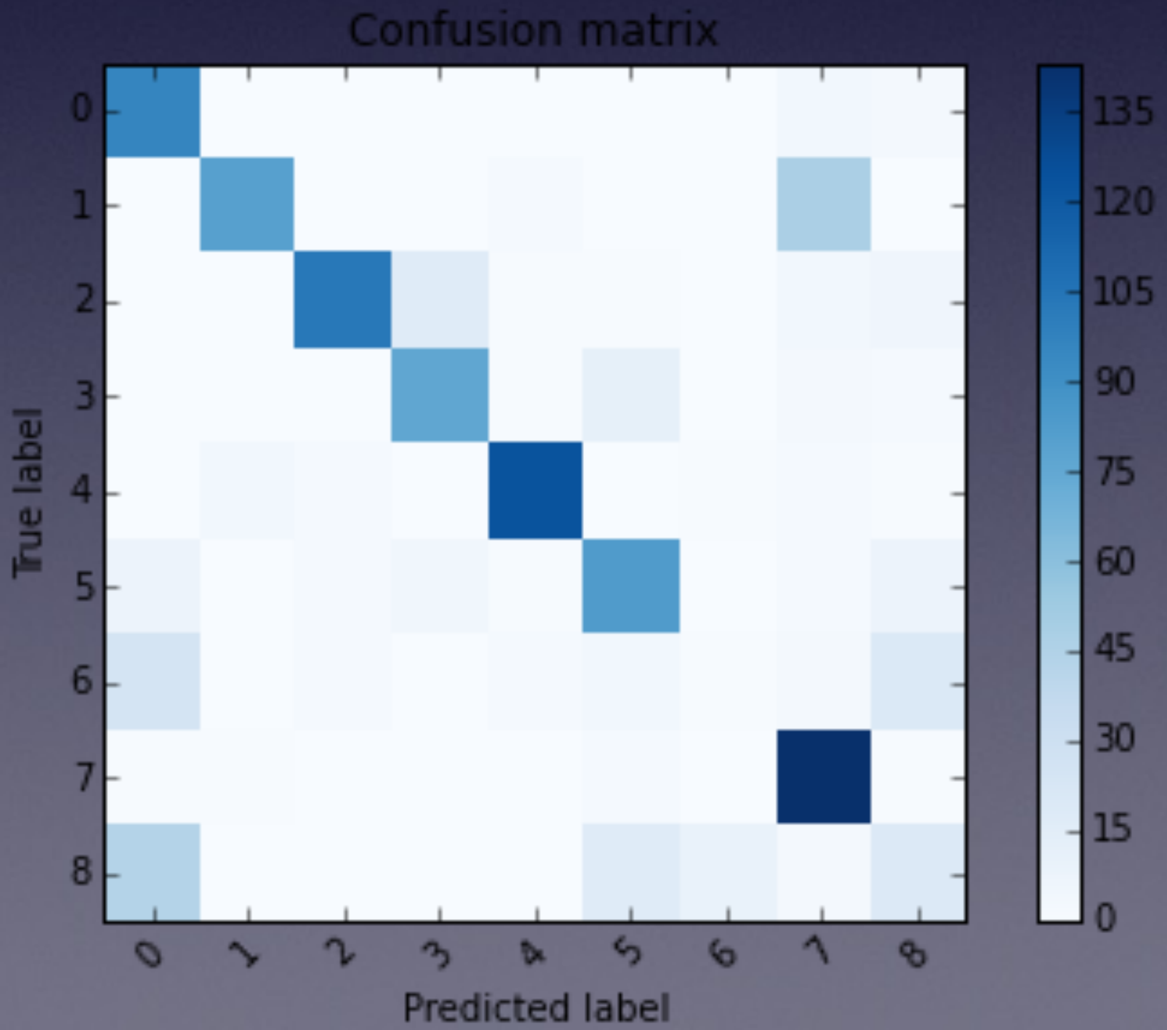
	Train Data	Test Data	Feature
Plan 3	MNIST	Captcha	HOG

Classifier	Score1	Score2
LinearSVC	0.8916	0.66
SVC	0.9525	0.721
BernoulliNB	0.6316	0.353
RandomForestClassifier	0.9356	0.637
KNeighborsClassifier	0.9316	0.725
DecisionTreeClassifier	0.8200	0.46
LogisticRegression	0.893	0.673
RBM + Logistic Regression	0.793	0.401
Avg	0.8612	0.5787

Step3. Clssifier GridSearch

	Train Data	Test Data	Feature
Plan 3	MNIST	Captcha	HOG

- Confusion matrix of KNeighbors Classifier(Best Classifier)



Step3. Clssifier GridSearch

	Train Data	Test Data	Feature
Plan 3	MNIST	Captcha	HOG

- Confusion matrix of KNeighbors Classifier(Best Classifier)

	0	1	2	3	4	5	6	7	8
0	96	0	0	0	0	0	0	4	3
1	0	80	0	0	2	0	0	48	0
2	0	0	103	17	1	1	0	4	6
3	0	0	0	76	1	12	0	3	2
4	0	4	2	0	124	0	1	2	1
5	8	0	2	5	1	83	0	2	8
6	25	0	2	0	2	4	1	3	20
7	1	1	0	0	0	2	0	143	1
8	44	0	0	0	0	17	10	3	19

Step3. Clssifier GridSearch

	Train Data	Test Data	Feature
Plan 1	Captcha	Captcha	HOG
Plan 2	Captcha	Captcha	No Feature
Plan 3	MNIST	Captcha	HOG
Plan 4	MNIST	Captcha	No Feature

- Score 1 : Train(700 mnist images), Test(200 captcha Images)
- Score 2 : Train(700 mnist images), Test(200 captcha Images)

Step3. Clssifier GridSearch

	Train Data	Test Data	Feature
Plan 4	MNIST	Captcha	No Feature

Classifier	Score1	Score2
LinearSVC	0.9136	0.027
SVC	0.9863	0.128
BernoulliNB	0.8571	0.129
RandomForestClassifier	0.9725	0.373
KNeighborsClassifier	0.973	0.373
DecisionTreeClassifier	0.8802	0.293
LogisticRegression	0.928	0.101
RBM + Logistic Regression	0.965	0.126
Avg	0.9344	0.1937

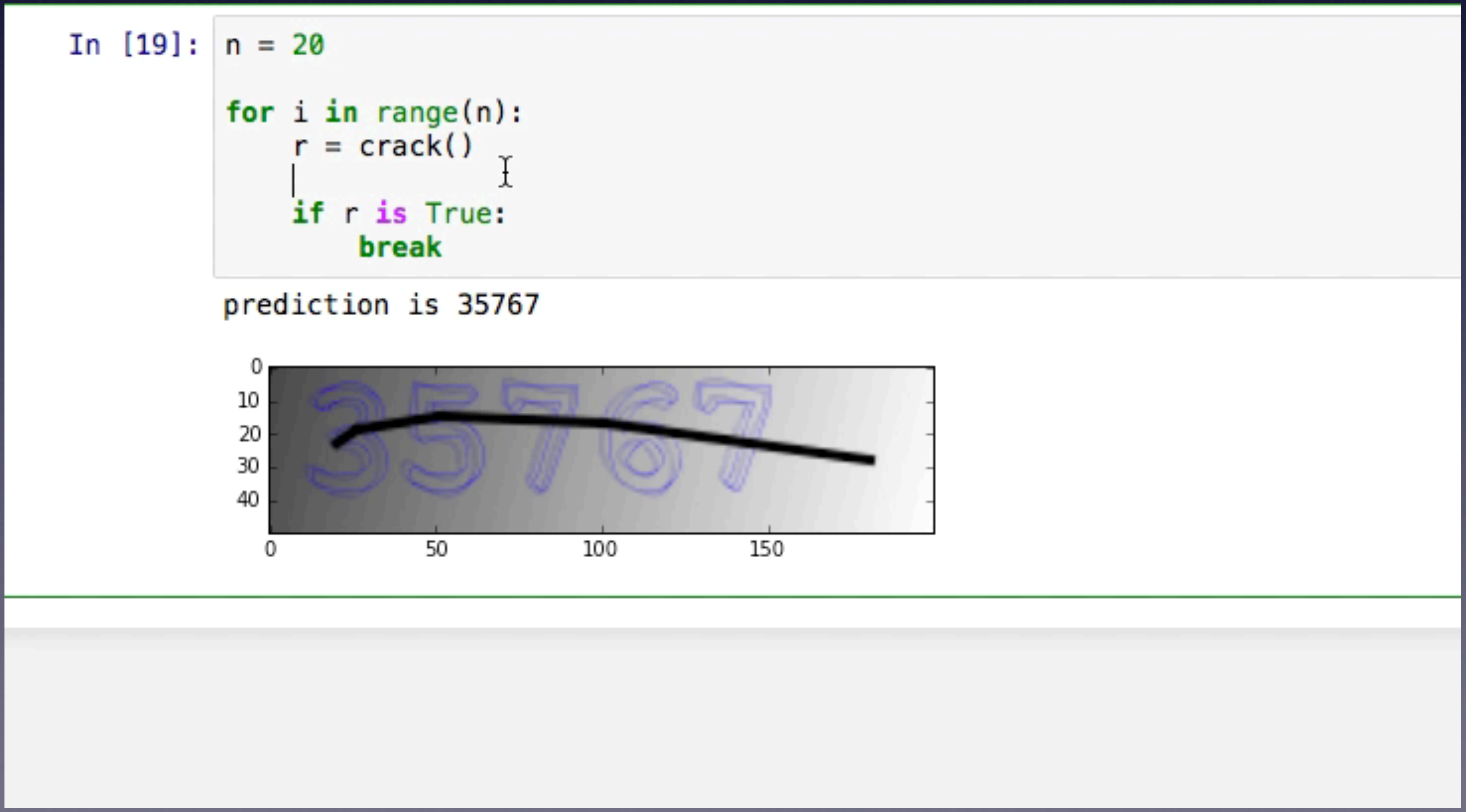
Result

	Train Data	Test Data	Feature
Plan 1	Captcha	Captcha	HOG
Plan 2	Captcha	Captcha	No Feature
Plan 3	MNIST	Captcha	HOG
Plan 4	MNIST	Captcha	No Feature

	Plan 1	Plan 2	Plan 3	Plan 4
LinearSVC	0.922	0.932	0.66	0.027
SVC	0.943	0.947	0.721	0.128
BernoulliNB	0.731	0.915	0.353	0.129
RandomForestClassifier	0.937	0.942	0.637	0.373
KNeighborsClassifier	0.941	0.937	0.725	0.373
DecisionTreeClassifier	0.897	0.911	0.46	0.293
LogisticRegression	0.926	0.939	0.673	0.101
RBM + Logistic Regression	0.853	0.937	0.401	0.126
Avg	0.8937	0.9325	0.5787	0.1937

Demo

- Predict Captcha Images with SVC
  - parameter : C(10.0), gamma(0.03125)



```
In [19]: n = 20

         for i in range(n):
             r = crack()

             if r is True:
                 break
```

```
prediction is 35767
```

Reference - opencv

1. Applying deep learning and a RBM to MNIST using Python : http://www.pyimagesearch.com/2014/06/23/applying-deep-learning-rbm-mnist-using-python/

2. opencv-practice : http://nbviewer.ipython.org/gist/mirosval/8752402

3. Thresholding : http://opencvpython.blogspot.kr/2013/05/thresholding.html

4. Skeletonization using OpenCV-Python : http://opencvpython.blogspot.kr/2012/05/skeletonization-using-opencv-python.html

5. OPENCV 얼굴검출 : http://blog.naver.com/kyaforeverl/220112044374

6. 얼굴 인식 기술을 소개합니다 : http://blog.naver.com/kkh8137/220231592219

7. Image Processing in OpenCV : http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_imgproc/py_table_of_contents_imgproc/py_table_of_contents_imgproc.html

8. Image - OpenCV BGR : Matplotlib RGB : http://www.bogotobogo.com/python/OpenCV_Python/python_opencv3_matplotlib_rgb_brg_image_load_display_save.php

9. High Performance Eye-Tracking : http://nbviewer.ipython.org/gist/mirosval/8752402

Reference - opencv

1. OpenCV Python: Digit Recognition : http://arnab.org/blog/so-i-suck-24-automating-card-games-using-opencv-and-python
2. http://projectproto.blogspot.kr/2014/07/opencv-python-digit-recognition.html
3. So I Suck At 24: Automating Card Games Using OpenCV and Python :
4. Converting PIL image to OpenCV image : http://opencv-users.1802565.n2.nabble.com/Converting-PIL-image-to-OpenCV-image-td7580733.html
5. Installing OpenCV with Anaconda on OS X : http://answers.opencv.org/question/28351/installing-opencv-with-anaconda-on-os-x/
6. TRAIN YOUR OWN OPENCV HAAR CLASSIFIER : http://coding-robin.de/2013/07/22/train-your-own-opencv-haar-classifier.html
7. OpenCV Haar/cascade training 튜토리얼 : http://darkpgmr.tistory.com/70

Reference - GitHub

1. Digit Image Recognition : https://github.com/ayng/digit-recognition
2. https://github.com/RobertGawron/snippets/tree/master/captcha
3. captcha-decoder : https://github.com/mekarpeles/captcha-decoder
4. Simple CAPTCHA solver in python : https://github.com/ptigas/simple-captcha-solver
5. captcha-ocr : https://github.com/bshillingford/captcha-ocr
6. Webcam-Face-Detect : https://github.com/shantnu/Webcam-Face-Detect
7. Captcha Intruder is an automatic pentesting tool to bypass captchas : https://github.com/epsylon/cintruder
8. Train your own OpenCV Haar classifier : https://github.com/mrnugget/opencv-haar-classifier-training
9. FaceDetect : https://github.com/shantnu/FaceDetect
10. Breaking a captcha : https://github.com/aflag/captcha-study

Reference - Articles

1. Handwriting Recognition Revisited: Kernel Support Vector Machines : http://www.codeproject.com/Articles/106583/Handwriting-Recognition-Revisited-Kernel-Support-V

2. Breaking a captcha : http://rafael.kontesti.me/blog/posts/Breaking_a_captcha/

3. DECODING WEIBO CAPTCHA IN PYTHON : http://slides.com/jingchaohu/decoding-weibo-captcha-in-python#/

4. Online RGB Color : http://www.colorspire.com/rgb-color-wheel/

5. Python - Remove(convert to white) non black pixels of an image [closed] : http://stackoverflow.com/questions/18282139/python-removeconvert-to-white-non-black-pixels-of-an-image

6. Digit Recognition using OpenCV, sklearn and Python : http://hanzratech.in/2015/02/24/handwritten-digit-recognition-using-opencv-sklearn-and-python.html

7. De-noising stripes of a captcha : http://stackoverflow.com/questions/24279918/de-noising-stripes-of-a-captcha

8. Object recognition in images with cuda-convnet : http://fastml.com/object-recognition-in-images-with-cuda-convnet/

9. Handwritten digits recognition based on neural networks : http://www.olivier-augereau.com/blog/?cat=18

10. How can i convert images from scikit-image to opencv2 and other libraries? : http://stackoverflow.com/questions/16156788/how-can-i-convert-images-from-scikit-image-to-opencv2-and-other-libraries

Reference - Articles

1. Handwritten digits recognition using OpenCV : http://perso.ens-lyon.fr/vincent.neiger/publications/report_digit_recognition.pdf

2. Python image processing libraries performance: OpenCV vs Scipy vs Scikit-Image : http://www.mastortosa.com/blog/python-image-processing-libraries-performance-opencv-scipy-scikit-image

3. 영상 데이터의 처리와 정보의 추출 : http://www.slideshare.net/neuralix/ss-35304929

4. The MNIST Database of Handwritten Digit Images for Machine Learning Research : http://research.microsoft.com/pubs/204699/MNIST-SPM2012.pdf

5. K Nearest Neighbors: Simplest Machine Learning : http://andrew.gibiansky.com/blog/machine-learning/k-nearest-neighbors-simplest-machine-learning

6. 컴퓨터 비전과 영상처리의 차이 : http://blog.naver.com/10200305

7. Histograms of Oriented Gradients (HOG) : https://www.youtube.com/watch?v=0Zib1YEE4LU

8. Neural Networks in the Wild: Handwriting Recognition : http://www.slideshare.net/guard0g/neural-networks-in-the-wild-handwriting-recognition

9. A gallery of interesting IPython Notebooks : https://github.com/ipython/ipython/wiki/A-gallery-of-interesting-IPython-Notebooks

10. MNIST who is the best in MNIST? : http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html

Reference - Articles

1. Trung Huynh’s tech blog : http://www.trungh.com/2013/04/digit-recognition-using-svm-in-python/
2. Segmentation: A SLIC Superpixel Tutorial using Python : http://www.pyimagesearch.com/2014/07/28/a-slic-superpixel-tutorial-using-python/
3. Deep Learning in Python with Pylearn2 and Amazon EC2 :
4. http://www.kurtsp.com/deep-learning-in-python-with-pylearn2-and-amazon-ec2.html
5. 우분투 12.04 - 삼바(samba) 서버 구축하기 : http://norus.tistory.com/5
6. Deep Learning through Examples - Kaggle #1 : http://www.slideshare.net/0xdata/deep-learning-through-examples-kaggle-1
7. Programming Assignment 2: Classification of MNIST Handwritten Digits and Regression of House Prices through Boston Dataset : http://saravanan-thirumuruganathan.github.io/cse5334Spring2015/assignments/PA2/PA2_MNIST_Classification_Boston_Regression.html
8. Deep Learning/Installing Pylearn2 on CUDA : http://labs.beatcraft.com/en/index.php?Deep%20Learning%2FInstalling%20Pylearn2%20on%20CUDA
9. Grid Searches for Hyper Parameters : https://github.com/amueller/pydata-strata-2015/blob/master/Chapter%204%20-%20Grid%20Searches%20for%20Hyper%20Parameters.ipynb
10. The MNIST Database of Handwritten Digit Images for Machine Learning Research :
11. http://research.microsoft.com/pubs/204699/MNIST-SPM2012.pdf

Reference - Articles

1. computer vision : http://www.aistudy.com/physiology/vision/computer_vision.htm

2. Confusion matrix : http://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html

3. How can we evaluate the predicted values using Scikit-Learn : http://stats.stackexchange.com/questions/95146/how-can-we-evaluate-the-predicted-values-using-scikit-learn

4. Data Workflows for Machine Learning - Seattle DAML : http://www.slideshare.net/pacoid/data-workflows-for-machine-learning

5. pylearn2-practice : https://github.com/zygmuntz/pylearn2-practice

Reference - skimage

1. Scikit-image tutorial at SciPy 2014 : http://tonysyu.github.io/scikit-image-tutorial-at-scipy-2014.html#.VUFzQM6UFjs

2. Image Processing with scikit-image : http://blog.yhathq.com/posts/image-processing-with-scikit-image.html

3. skimage-tutorials : http://nbviewer.ipython.org/github/scikit-image/skimage-tutorials/tree/master/

4. Adaptive Thresholding : http://scikit-image.org/docs/dev/auto_examples/plot_threshold_adaptive.html

5. CATEGORY ARCHIVES: SCIKIT-IMAGE : https://vcansimplify.wordpress.com/category/scikit-image/

6. scikit-image: image processing in Python : http://www.ncbi.nlm.nih.gov/pmc/articles/PMC4081273/

7. Label image regions : http://scikit-image.org/docs/dev/auto_examples/plot_label.html

Reference - PIL

1. PIL 이용한 이미지처리 - 3 : http://blog.daum.net/_blog/BlogTypeView.do?blogid=0Cf5V&articleno=13982241&categoryId=480155&regdt=20070920160619

2. PIL Handbook : http://pillow.readthedocs.org/en/latest/handbook/tutorial.html

3. ImageFilter : http://pillow-cn.readthedocs.org/zh_CN/latest/reference/ImageFilter.html