Breaking Captcha

Machine Learing + Computer Vision

JongWon Kim

dikien2012@gmail.com
https://github.com/dikien/break-capctha

Table

- Motivation
- Packet and Image Analysis
- Get Images
- Data Exploratory
- Machine Learning
- Deep Learning
- Future Work

Motivation

- Annoying Pictures!

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

Packet Analysis

- **Burpsuite** > Paros

Home > 뉴스

뉴스
토픽

KT 해킹에 사용된 프로그램 '파로스'...어떤 프로그램인가

인쇄 트위터 페이스북 미투데이 네이트커넥트 요즘 제보하기

등록 : 2014-03-06 17:21 , 데일리시큐 길민권기자 , mkgil@dailysecu.com

해커 "웹해킹 할 때 사용되는 툴로 해킹대회에서도 주로 사용돼"

인천지방경찰청(청장 이상원) 광역수사대는 해킹프로그램을 자체 제작해 KT 고객센터 홈페이지를 해킹해 1,200만명의 고객 정보를 빼내어 텔레마케팅 업체에 판매하는 방법으로 개인정보를 유출하고 115억원의 부당이득을 챙긴 사이버 범죄자와 텔레마케팅 대표 등 3명을 검거했다고 발표했다. 특히 이번에 범죄자가 사용한 해킹 프로그램이 바로 웹 프록시 프로그램인 파로스(Paros)라고 밝혔다.

경찰 측은 "악성해커는 인터넷상에 배포되어 있는 웹사이트에 대한 취약성 분석 등 강력한 해킹도구 프로그램인, 파로스 프로그램을 이용해 신종 해킹프로그램을 개발한 것이다. 이후 KT 홈페이지에 로그인 후 이용대금 조회란에 고유숫자 9개를 무작위로 자동 입력시키는 프로그램을 이용해 다른 고객들의 고유번호를 찾아내 고객정보를 해킹하는 방법을 사용했다"고 설명했다.

Packet Analysis

- http://www.sktmembership.co.kr/simpleCaptcha.do

Request

Raw | Params | Headers | Hex

```
GET /simpleCaptcha.do HTTP/1.1
Accept: text/html, application/xhtml+xml, */*
Accept-Language: ko-KR
User-Agent: Mozilla/5.0 (Windows NT 6.1; Trident/7.0;
rv:11.0) like Gecko
Accept-Encoding: gzip, deflate
Host: www.sktmembership.co.kr
DNT: 1
Proxy-Connection: Keep-Alive
Pragma: no-cache
Cookie: XTVID=Z1406032327288545;
 ga=GA1.3.1697950790.1401805656; _gat=1; OBBBOPEN=1;
JSESSIONID=G32GVjRhQhh3YtxnvWY0TNJGRDt5fXQ0tpcvxtDvGMBQh87Tvv
hk!-1017567127!-1399328284
```

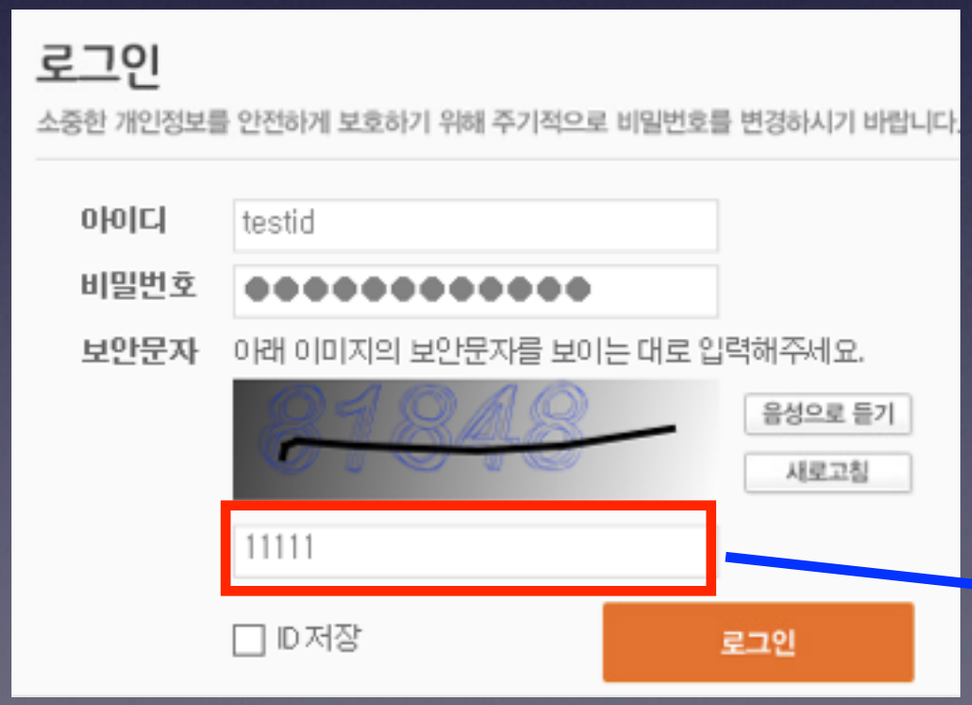
Response

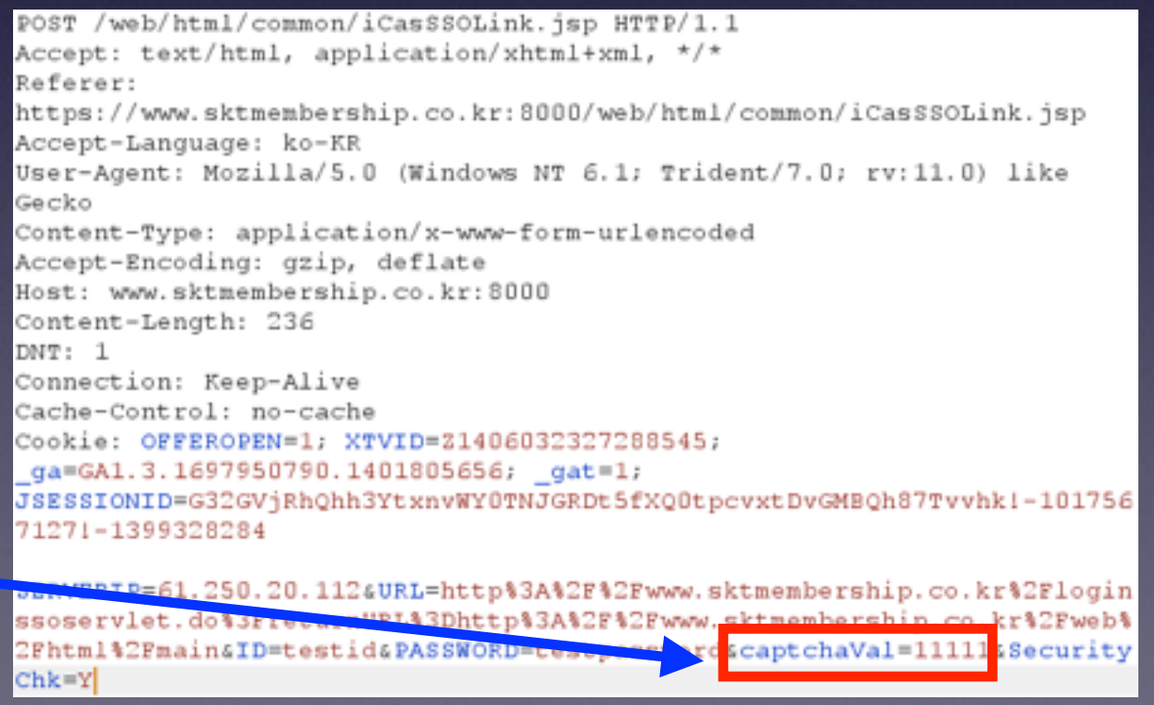
Raw | Headers | Hex | Render

```
HTTP/1.1 200 OK
Date: Mon, 25 May 2015 12:15:26 GMT
Content-type: image/png
Cache-Control: private,no-cache,no-store
Content-Length: 8906

□NG
□
□□□IHDR□□□□□□2□□□□□□□□□"□DATx□□T□□□N□iNsz;□□□i□m□□□□□□□b/
□@□□□□□□c68□□□:n(□{□(□9□□[EHXv□%=~□□,□□□□□□□□□>□
□□8□□□(□*□□□□□□□□□□□gdd□4:□□□□□□□□□□□□□G~~~□
□□□□□□nII
□+---+++□EEE□□□jjjkk□□□□□□ACCCcc□(l□KKK□D"□,nkkkoo□$□□J□□
□r□□*U*□□□V□□N□□□□□□□)}&□□□b□□w8□N□□□n□□□□□□□□□□□:::□□□□□
```

Packet Analysis

- http://www.sktmembership.co.kr/web/html/login/LoginMain.jsp
- ID : testid, Password : testpassword, Captcha: 11111





Sample Images

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

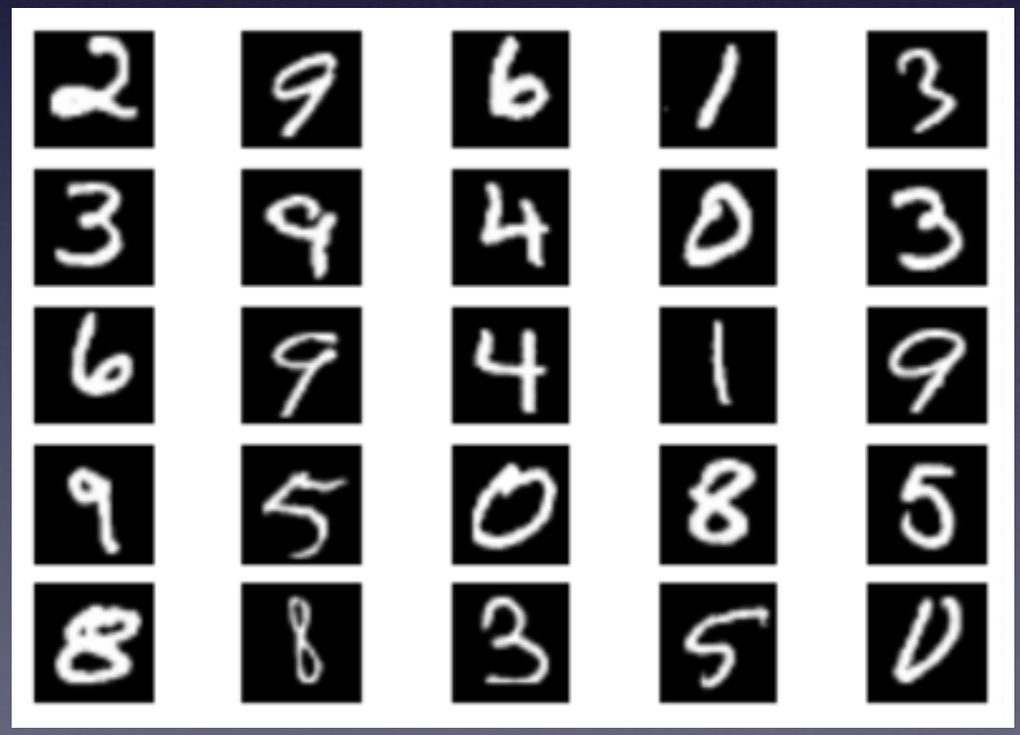



How?

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
   - LinearSVM, SVM, BernoulliNB, RandomForest, KNeighbors, DecisionTree, LogisticRegression

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

- 7 kinds of Classifier X 2 kinds of Feature X 2 kinds of Data Splitting : 28 cases
  - LinearSVM, SVM, BernoulliNB, RandomForest, KNeighbors, DecisionTree, LogisticRegression

	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

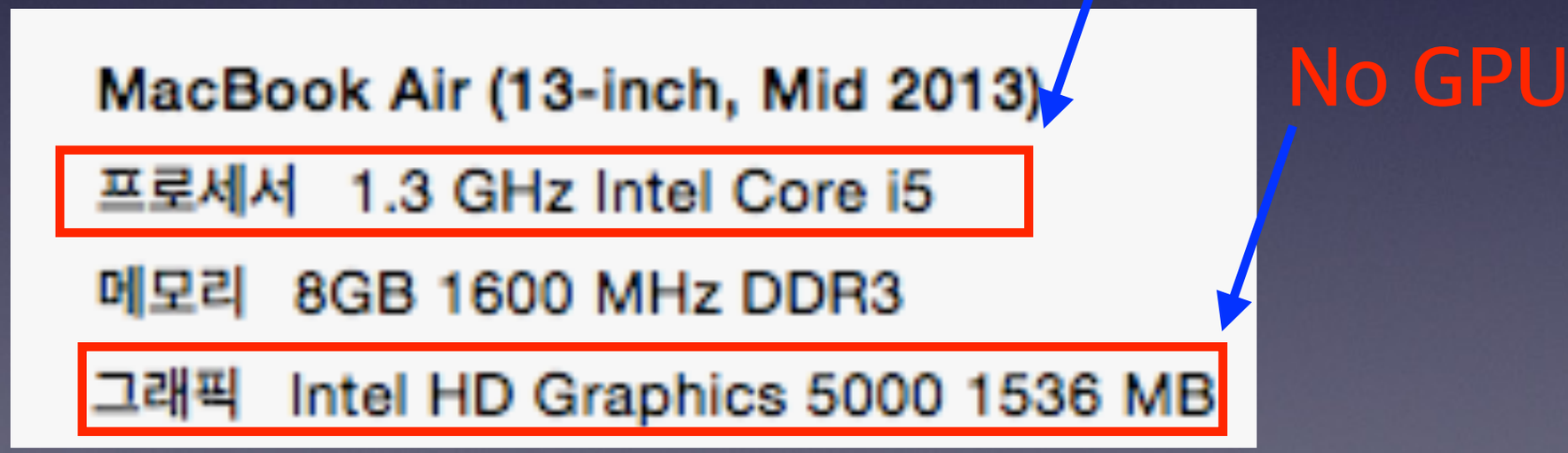
- Computer Vision
  - Computer vision is a field that includes methods for acquiring, processing, analyzing, and understanding images and, in general, high-dimensional data from the real world in order to produce numerical or symbolic information.
- Machine Learning
  - Machine learning is a subfield 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)
  - Array processing : Numpy(v1.9.2)
  - Plotting : Matplotlib(v1.4.3)

Hardware Requirment

- I have a terrible hardware to run machine learning.
- It doen’t matter because we deal with small dataset.
- Just be patient!



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

```
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

```
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

```
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

- Q1. If we train classifier with captcha dataset, can we predict the new captcha image?
  - Proof of Concept : Plan 1, 2
- Q2. If we train classifier with mnist dataset, can we predict the captcha image?
  - Proof of Concept : Plan 3, 4

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>
```

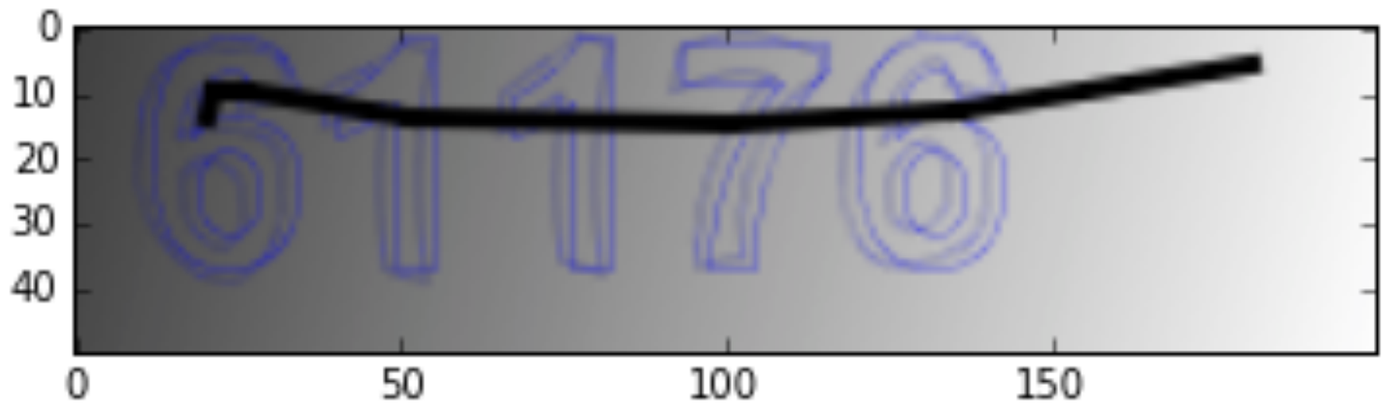
```
fig, (ax0, ax1, ax2, ax3, ax4) = plt.subplots(1, 5)
ax0.imshow(sample_5, cmap='gray')
ax1.imshow(sample_6, cmap='gray')
ax2.imshow(sample_7, cmap='gray')
ax3.imshow(sample_8, cmap='gray')
ax4.imshow(sample_9, cmap='gray')
```

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

Open Image

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

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

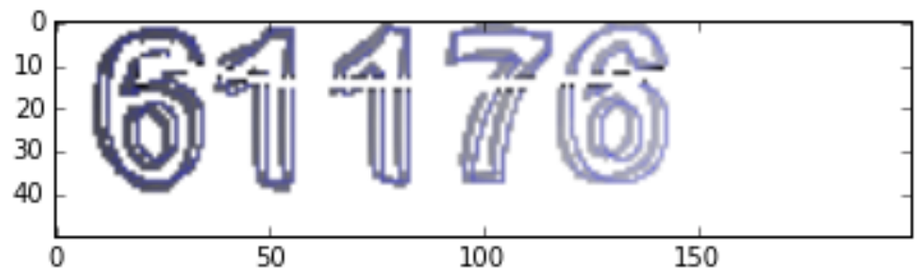


Remove Noise

- 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')
```

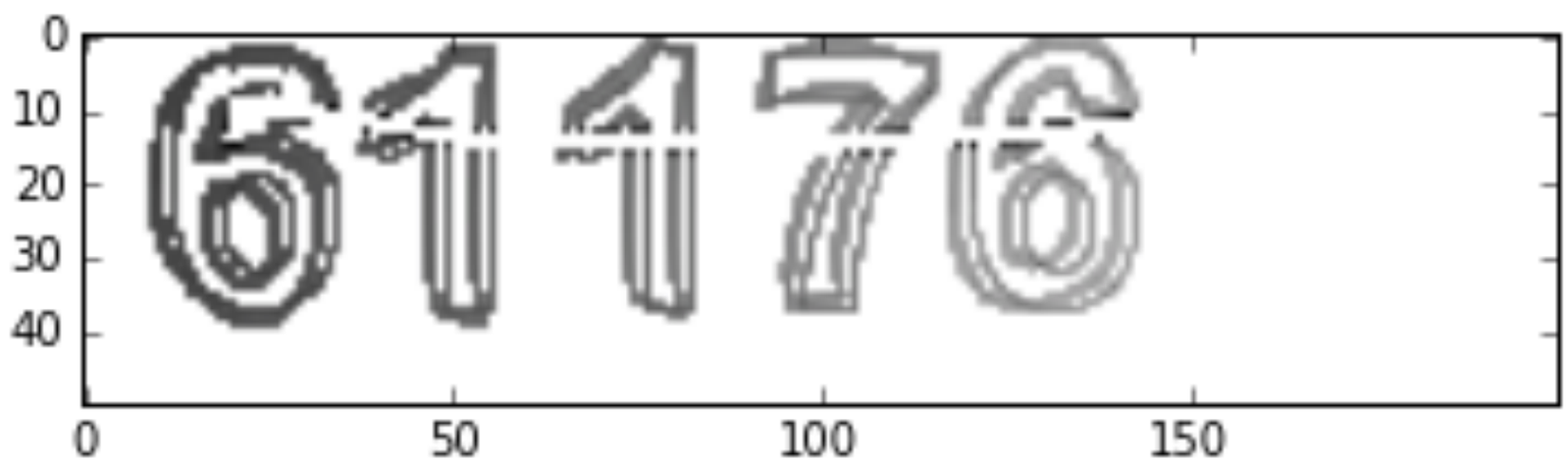
```
<matplotlib.image.AxesImage at 0x1148d6d90>
```



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)
```

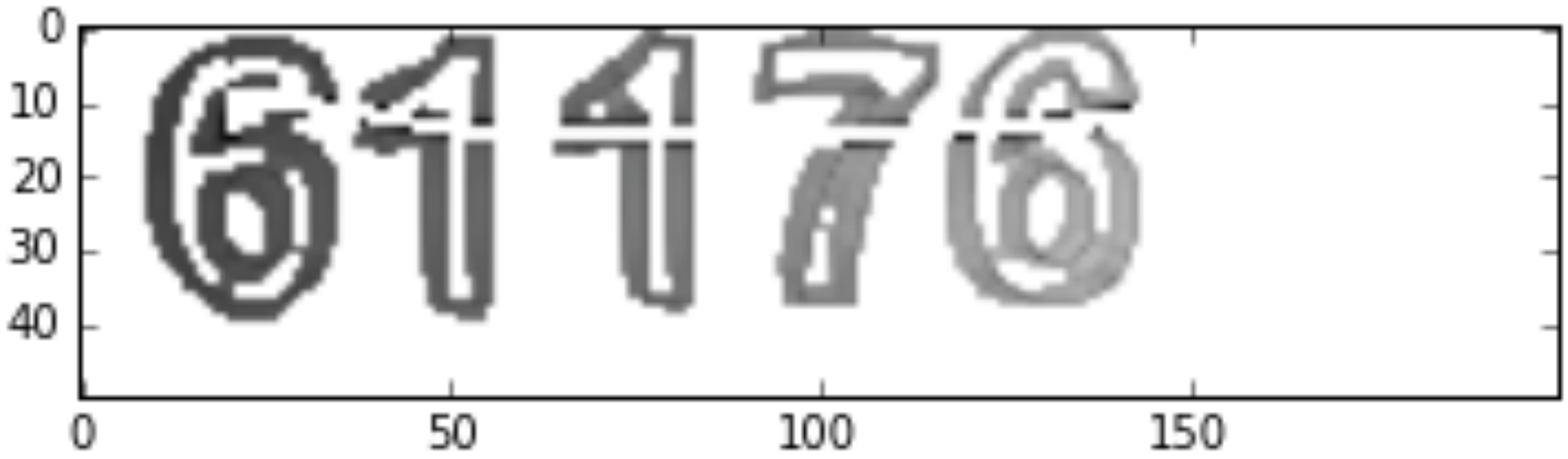
```
the dimension of rgb is (50, 200, 3)
the dimension of gray is (50, 200)
```



Opening

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

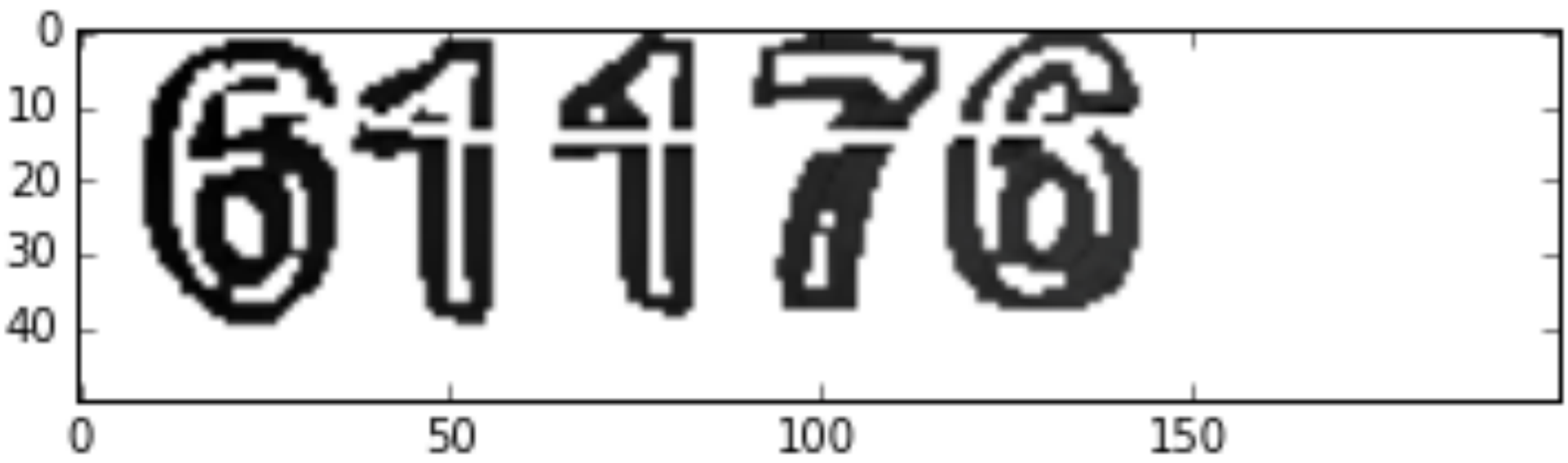
```
<matplotlib.image.AxesImage at 0x114acc790>
```



Equalize Hist

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

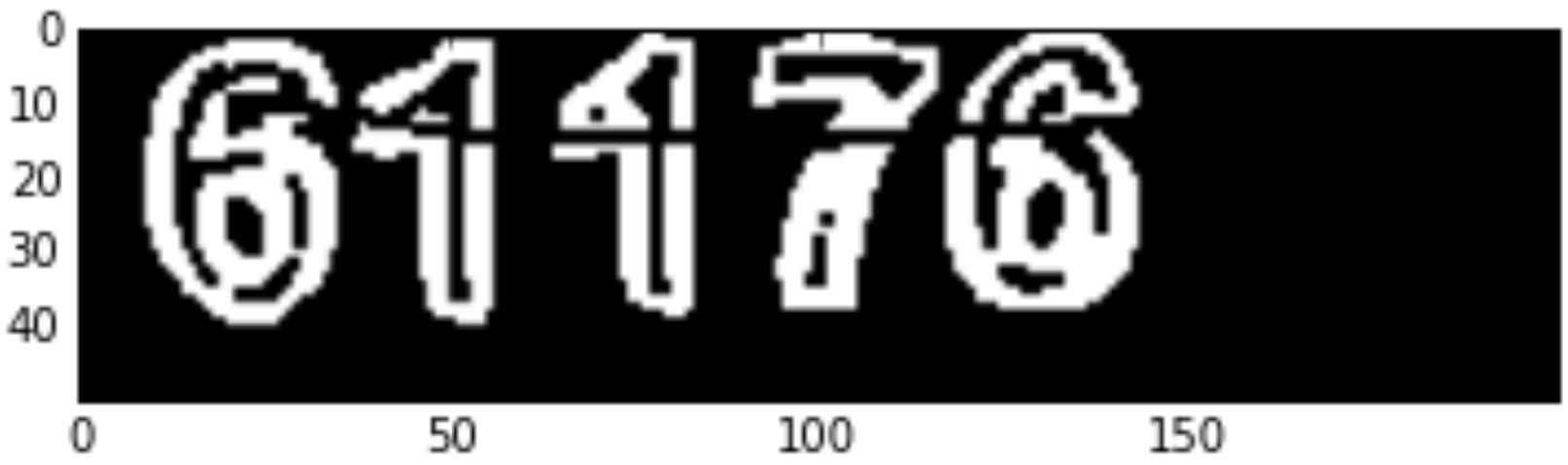
```
<matplotlib.image.AxesImage at 0x116024dd0>
```



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>
```



Closing

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

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

Dilation

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

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

5-Digits

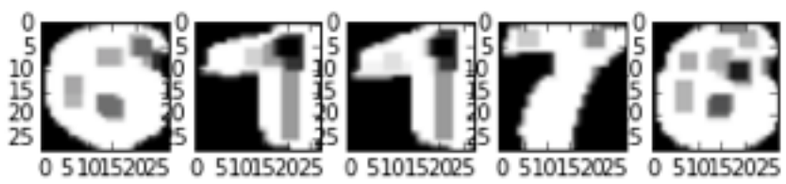
- Divide into 5-digits

```
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')
```



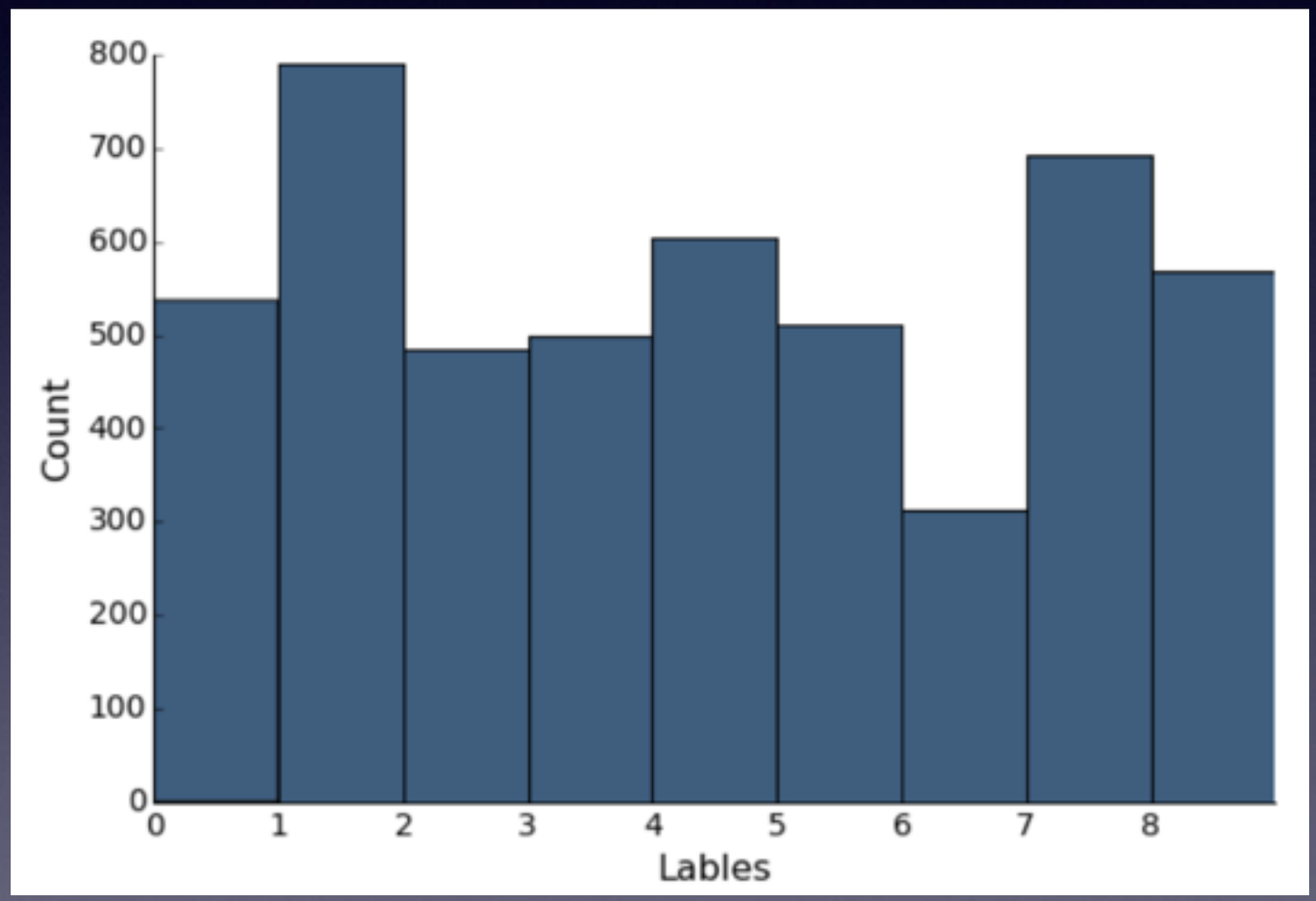
Step3. Clssifier GridSearch

	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(800 captcha images), Test(200 captcha images)
- Score 2 : Train(1000 captcha images), Test(200 captcha images)

Image Histogram

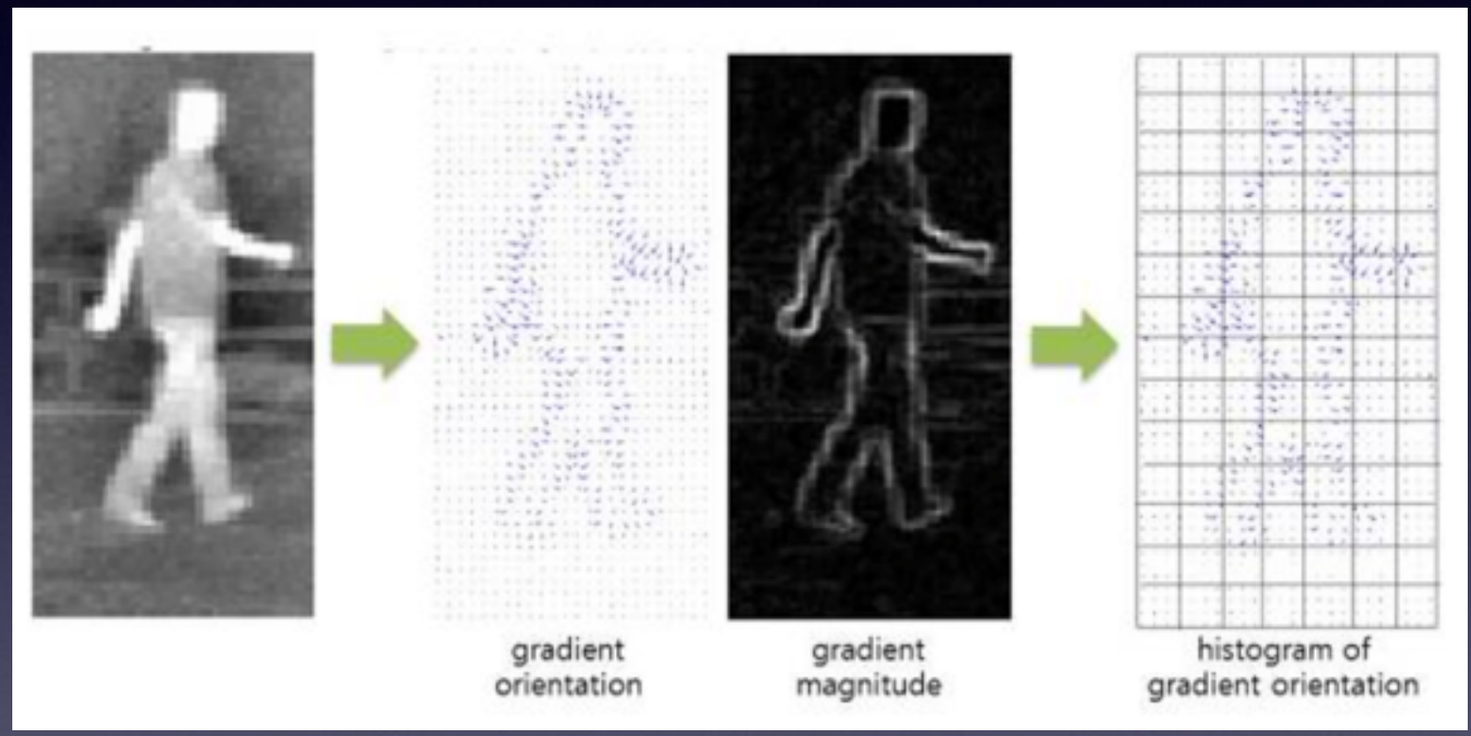
- 1000 Captcha Images Histogram



	0	1	2	3	4	5	6	7	8
1000	537	791	484	499	605	510	313	692	569

Feature

- 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')
```

Plan 1

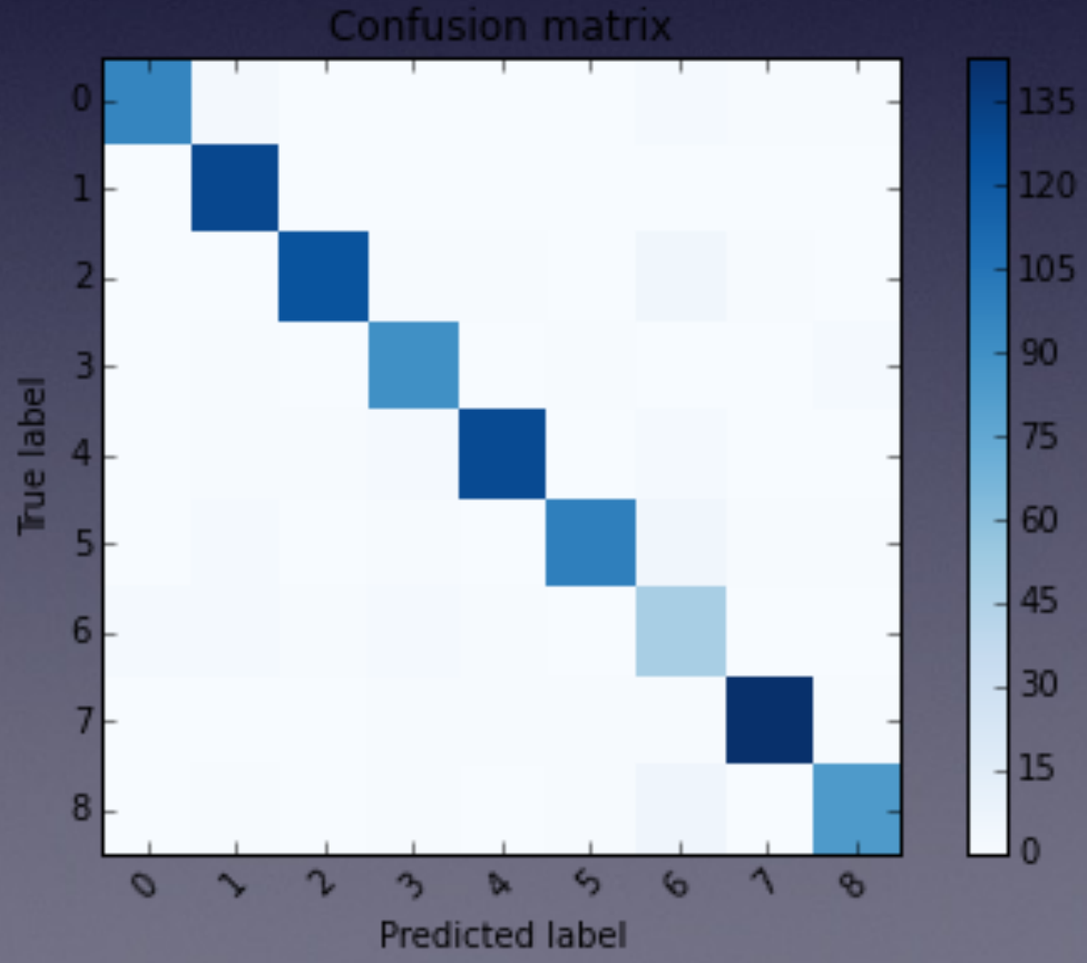
	Train Data	Test Data	Feature
Plan 1	Captcha	Captcha	HOG

Classifier	Score1	Score2
LinearSVM	0.9174	0.922
SVM	0.9294	0.943
BernoulliNB	0.713	0.731
RandomForest	0.9322	0.937
KNeighbor	0.9304	0.941
DecisionTree	0.8904	0.897
LogisticRegression	0.920	0.926
RBM + Logistic Regression	0.840	0.853
Avg	0.8841	0.8937

Plan 1

	Train Data	Test Data	Feature
Plan 1	Captcha	Captcha	HOG

- Confusion matrix of SVC(Best Classifier)



Plan 1

	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

Plan 2

	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(800 captcha images), Test(200 captcha images)
- Score 2 : Train(1000 captcha images), Test(200 captcha images)

Plan 2

	Train Data	Test Data	Feature
Plan 2	Captcha	Captcha	No Feature

Classifier	Score1	Score2
LinearSVM	0.9286	0.932
SVM	0.9294	0.947
BernoulliNB	0.9064	0.915
RandomForest	0.936	0.942
KNeighbors	0.9322	0.937
DecisionTree	0.9156	0.911
LogisticRegression	0.934	0.939
RBM + Logistic Regression	0.931	0.937
Avg	0.9266	0.9325

Plan 3

	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(MNIST images), Test(MNIST images)
- Score 2 : Train(MNIST images), Test(200 captcha Images)

Plan 3

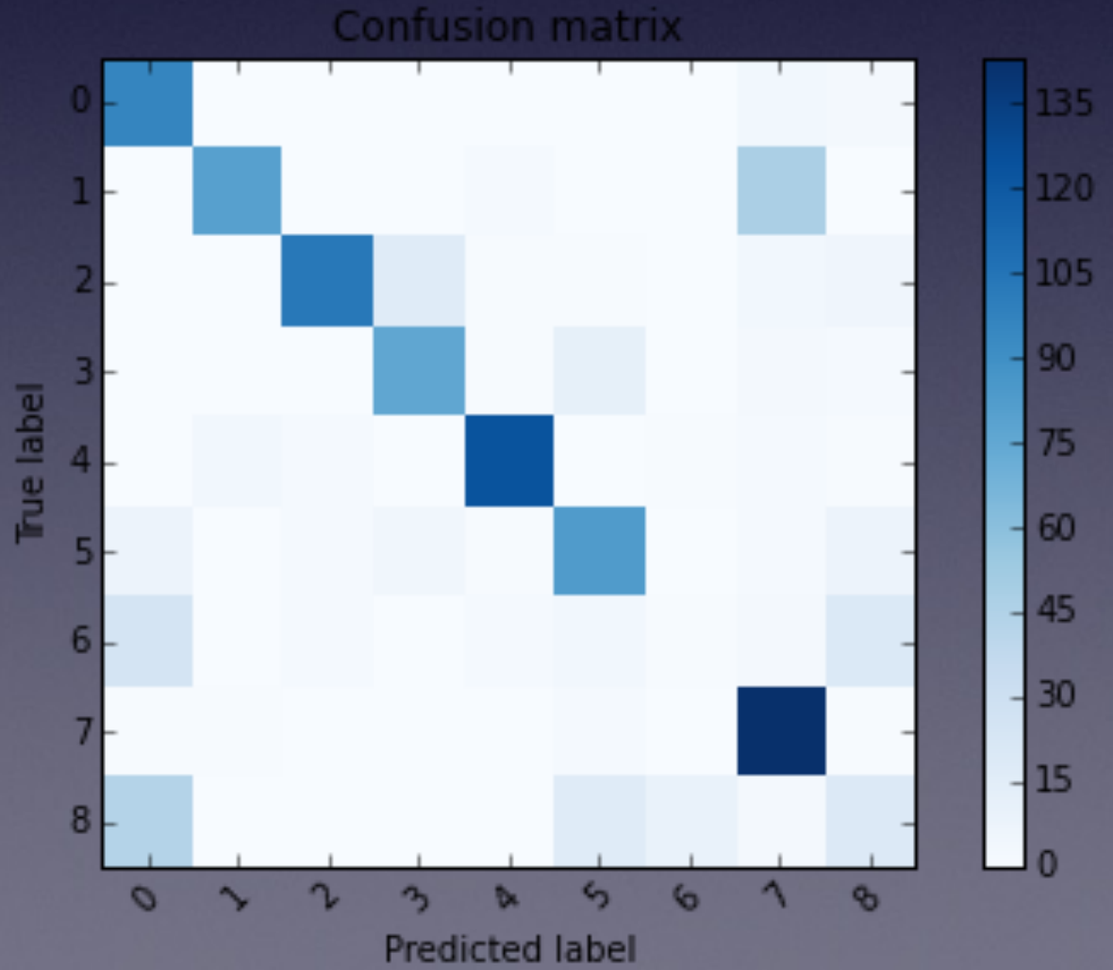
	Train Data	Test Data	Feature
Plan 3	MNIST	Captcha	HOG

Classifier	Score1	Score2
LinearSVM	0.8916	0.66
SVM	0.9525	0.721
BernoulliNB	0.6316	0.353
RandomForest	0.9356	0.637
KNeighbors	0.9316	0.725
DecisionTree	0.8200	0.46
LogisticRegression	0.893	0.673
RBM + Logistic Regression	0.793	0.401
Avg	0.8612	0.5787

Plan 3

	Train Data	Test Data	Feature
Plan 3	MNIST	Captcha	HOG

- Confusion matrix of KNeighbors Classifier(Best Classifier)



Better Idea?

	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

Plan 4

	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

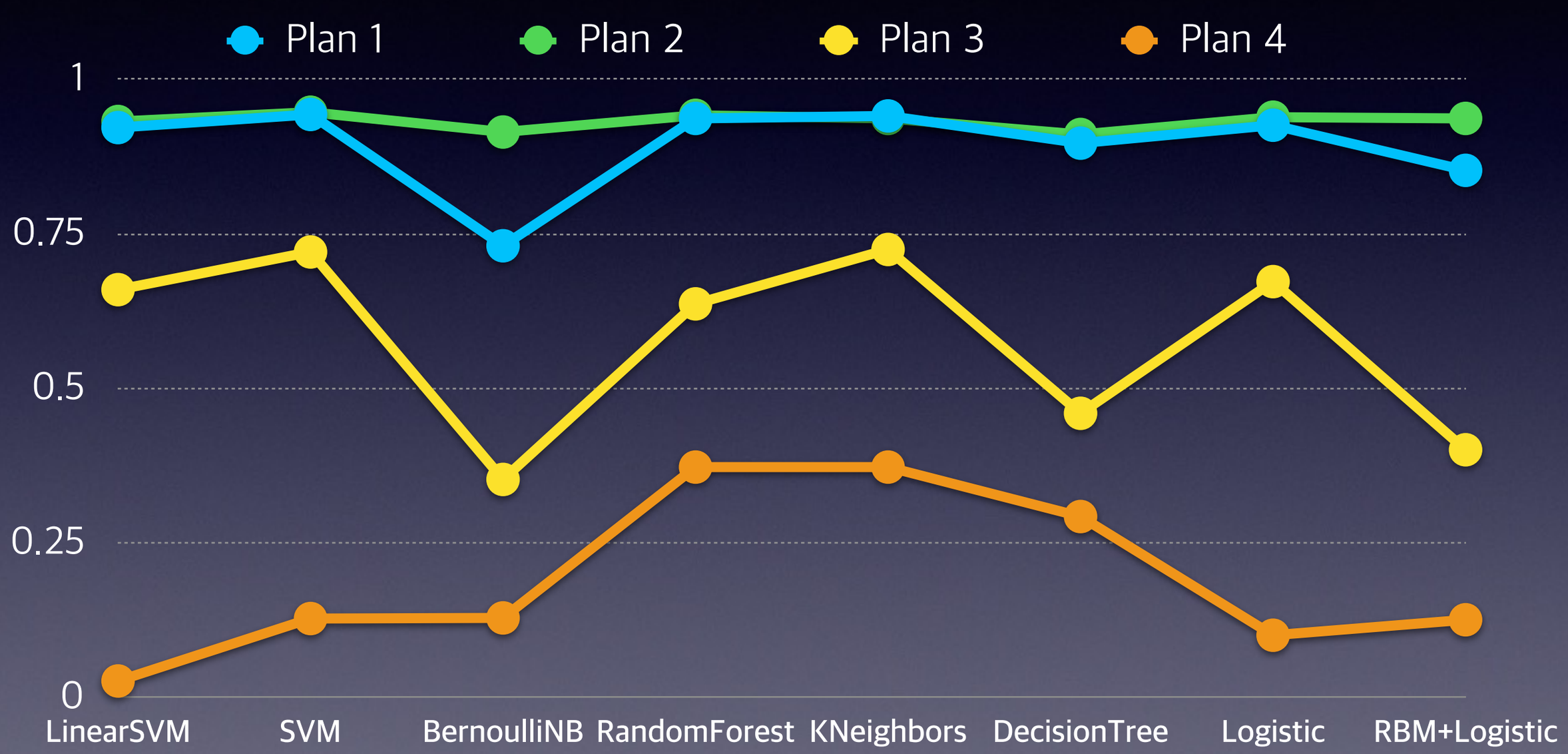
- Score 1 : Train(mnist images), Test(mnist images)
- Score 2 : Train(mnist images), Test(200 captcha Images)

Plan 4

	Train Data	Test Data	Feature
Plan 4	MNIST	Captcha	No Feature

Classifier	Score1	Score2
LinearSVM	0.9136	0.027
SVM	0.9863	0.128
BernoulliNB	0.8571	0.129
RandomForest	0.9725	0.373
KNeighbors	0.973	0.373
DecisionTree	0.8802	0.293
LogisticRegression	0.928	0.101
RBM + Logistic Regression	0.965	0.126
Avg	0.9344	0.1937

Which is the Best?



Support Vector Machine

	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

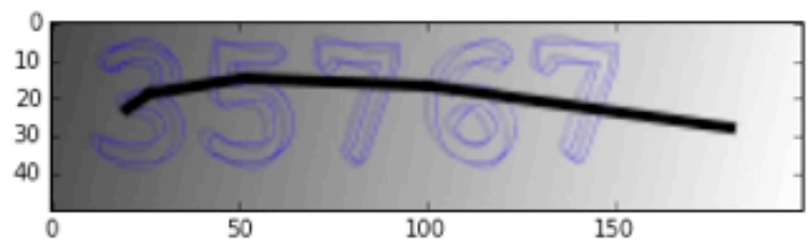
	Plan 1	Plan 2	Plan 3	Plan 4
LinearSVM	0.922	0.932	0.66	0.027
SVM	0.943	0.947	0.721	0.128
BernoulliNB	0.731	0.915	0.353	0.129
RandomForest	0.937	0.942	0.637	0.373
KNeighbors	0.941	0.937	0.725	0.373
DecisionTree	0.897	0.911	0.46	0.293
LogisticRegression	0.926	0.939	0.673	0.101
RBM + Logistic Regression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

Demo

- Predict Captcha Images with SVM
  - 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
```



https://raw.githubusercontent.com/dikien/break-captcha/master/demo/demo.mov

Step4. Deep Learning

- Q1. Can Deep Learning give a better result?
- Q2. What happen if we apply Deep Learning feature to Machine Learning?

Plan

	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

- No more Feature Engineering

How?

- What is Convolutional Neural and Deep Belief Network?
- What is difference between them?
- How to tuning parameters?
  - learning rate, kernel, hidden layers, activation functions, epochs, dropout, units
- How to visualize convolutional layers?

ToolBox

- Keras
- Scikit-Neuralnetwork(v0.1)
- Lasagne
- Nolearn(v0.5)
- Pylearn2
- Theano(v0.7.0rc2)

Install Pylearn2 on OSX(1/2)

- Install cuda 6.5
- Install Theano : conda install -c https://conda.binstar.org/jeprescottroy theano
- Install PyYaml Maually
- Env Configurations

```
gg-android:~ sds$ echo '[global]
> floatX = float32
> device = gpu0
>
> [nvcc]
> fastmath = True' > .theanorc
```

Install Pylearn2 on OSX(2/2)

- Env Configurations
  - export PATH="/Users/dikien/anaconda/bin:$PATH"
  - export PYLEARN2_DATA_PATH=/Users/dikien/Downloads/data
  - export LD_LIBRARY_PATH=/usr/local/cuda/lib:
  - export PYLEARN2_VIEWER_COMMAND="open -a Preview"
  - export PATH=/usr/local/cuda/bin:/Users/sds/anaconda/bin:/usr/local/bin:/usr/bin:/bin:/usr/sbin:/sbin
- Pylearn2 Install

```
(venv)gg-android:pylearn2 sds$ python setup.py install
Using gpu device 0: GeForce GT 755M
setup.py:18: UserWarning: Cython was not found and hence pylearn2
m (e.g. pylearn2.train_extensions.window_flip) will not be availa
  warnings.warn("Cython was not found and hence pylearn2.utils._w
/Users/sds/venv/lib/python2.7/site-packages/setuptools/dist.py:28
  normalized_version,
running install
```

Install Scikit-Neuralnetwork on OSX

- pip install scikit-neuralnetwork

```
(/Users/dikien/anaconda/envs/deeplearning)172:deep_learning_study dikien$ pip install scikit-neuralnetwork
Collecting scikit-neuralnetwork
  Downloading scikit-neuralnetwork-0.1.tar.gz (701kB)
    100% |████████████████████████████████| 704kB 235kB/s
Requirement already satisfied (use --upgrade to upgrade): scikit-learn in /Users/dikien/anaconda/envs/deepl
euralnetwork)
Requirement already satisfied (use --upgrade to upgrade): theano in /Users/dikien/anaconda/envs/deeplearnin
etwork)
Requirement already satisfied (use --upgrade to upgrade): pyyaml in /Users/dikien/anaconda/envs/deeplearnin
etwork)
Requirement already satisfied (use --upgrade to upgrade): numpy>=1.6.2 in /Users/dikien/anaconda/envs/deepl
scikit-neuralnetwork)
Requirement already satisfied (use --upgrade to upgrade): scipy>=0.11 in /Users/dikien/anaconda/envs/deeple
cikit-neuralnetwork)
Installing collected packages: scikit-neuralnetwork
  Running setup.py install for scikit-neuralnetwork
Successfully installed scikit-neuralnetwork-0.1
```

- Check 1 : nosetests -v sknn.tests

```
test_FitFloat64 (sknn.tests.test_types.TestScipySparseMatrix) ... ok
test_Predict32 (sknn.tests.test_types.TestScipySparseMatrix) ... ok
test_Predict64 (sknn.tests.test_types.TestScipySparseMatrix) ... ok

----------------------------------------------------------------------
Ran 127 tests in 244.524s

OK
```

Install Scikit-Neuralnetwork on OSX

- Check 2
  - python examples/bench_mnist.py sknn dbn

```
(/Users/dikien/anaconda)172:scikit-neuralnetwork dikien$ python examples/bench_mnist.py sknn db
gnumpy: failed to import cudamat. Using npmat instead. No GPU will be used.
[DBN] fitting X.shape=(46900, 784)
[DBN] layers [784, 300, 10]
[DBN] Fine-tune...
100%
Epoch 1:
  loss 0.290286798442
  err  0.0841871584699
  (0:00:07)
100%
Epoch 2:
  loss 0.174733093346
  err  0.0482197745902
  (0:00:08)
100%
```

Install Scikit-Neuralnetwork on OSX

- Result

```
sknn.mlp
        Accuracy: 98.00% ±0.00
        Times:    44.90s ±0.00
        Report:
             precision    recall  f1-score   support

          0       0.98      0.99      0.99      2312
          1       0.99      0.99      0.99      2589
          2       0.98      0.98      0.98      2241
          3       0.97      0.98      0.98      2430
          4       0.98      0.98      0.98      2253
          5       0.97      0.98      0.98      2051
          6       0.98      0.98      0.98      2246
          7       0.99      0.97      0.98      2374
          8       0.98      0.97      0.97      2230
          9       0.97      0.98      0.97      2374

avg / total       0.98      0.98      0.98     23100
```

```
nolearn.dbn
        Accuracy: 97.79% ±0.00
        Times:    130.29s ±0.00
        Report:
             precision    recall  f1-score   support

          0       0.98      0.99      0.98      2312
          1       0.98      0.99      0.99      2589
          2       0.98      0.98      0.98      2241
          3       0.97      0.98      0.97      2430
          4       0.98      0.97      0.98      2253
          5       0.98      0.96      0.97      2051
          6       0.98      0.98      0.98      2246
          7       0.98      0.98      0.98      2374
          8       0.96      0.97      0.97      2230
          9       0.98      0.97      0.97      2374

avg / total       0.98      0.98      0.98     23100

Initializing neural network with 2 layers, 784 inputs and 10 outputs.
  - Dense: Rectifier  Units:  300
  - Dense: Softmax    Units:  10
```

Install Keras on OSX

- git clone git@github.com:fchollet/keras.git
- Install Manually

```
(/Users/dikien/anaconda)172:keras dikien$ python setup.py install
running install
running bdist_egg
running egg_info
creating Keras.egg-info
writing requirements to Keras.egg-info/requires.txt
writing Keras.egg-info/PKG-INFO
writing top-level names to Keras.egg-info/top_level.txt
writing dependency_links to Keras.egg-info/dependency_links.txt
```

Install Lasagne on OSX

- git clone git@github.com:Lasagne/Lasagne.git
- Install Manually

```
(/Users/dikien/anaconda)gim-ui-MacBook-Air:Lasagne dikien$ python setup.py install
/Users/dikien/anaconda/lib/python2.7/site-packages/setuptools-16.0-py2.7.egg/setup
1.dev0'
running install
running bdist_egg
running egg_info
creating Lasagne.egg-info
writing requirements to Lasagne.egg-info/requires.txt
writing Lasagne.egg-info/PKG-INFO
writing top-level names to Lasagne.egg-info/top_level.txt
writing dependency_links to Lasagne.egg-info/dependency_links.txt
```

Install Lasagne on OSX

- Check 1 : python examples/mnist.py

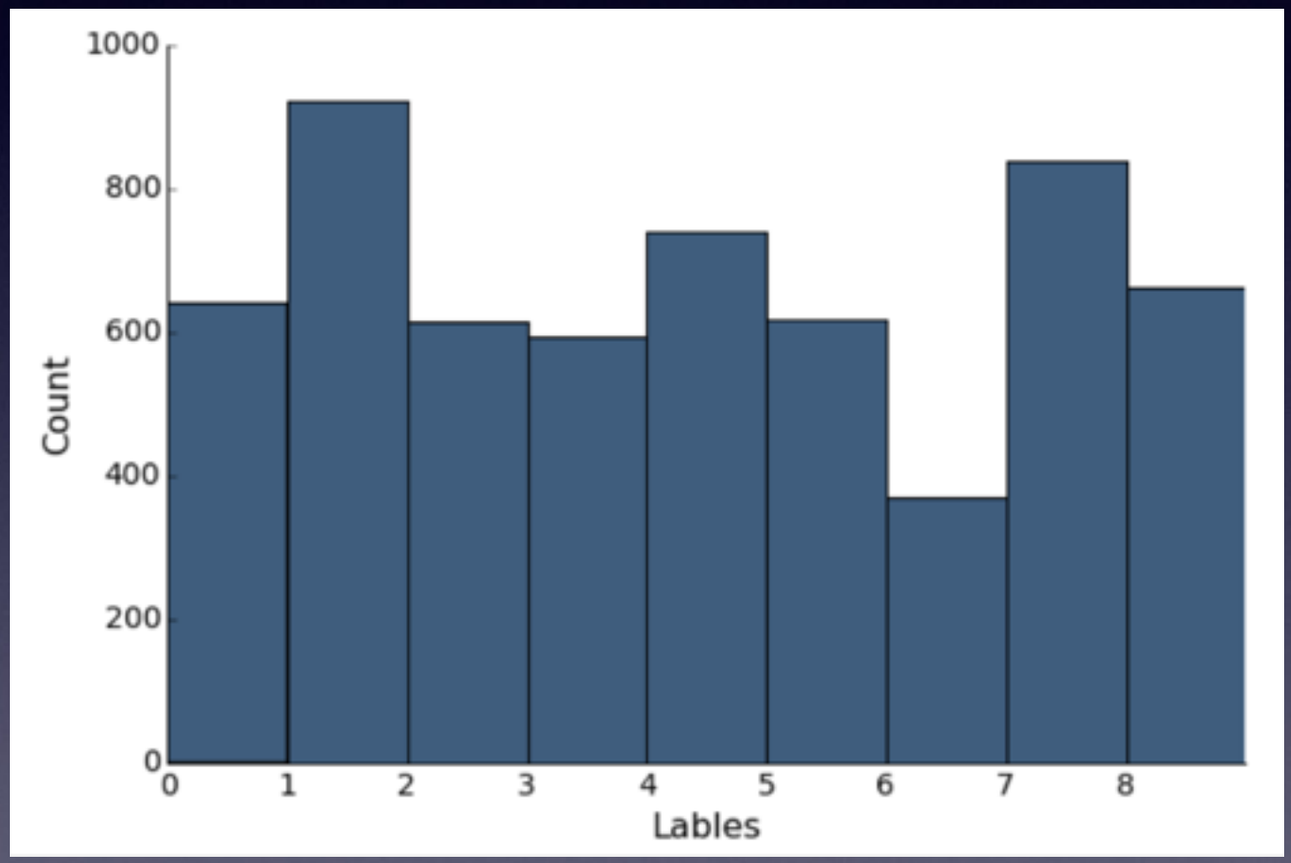
```
Epoch 500 of 500 took 6.755s
  training loss:                  0.007751
  validation loss:                0.062589
  validation accuracy:            98.56 %%
```

- Check 2 : python examples/mnist_conv.py

```
Epoch 200 of 200 took 173.266s
  training loss:                  0.005285
  validation loss:                0.033518
  validation accuracy:            99.20 %%
```

Image Histogram

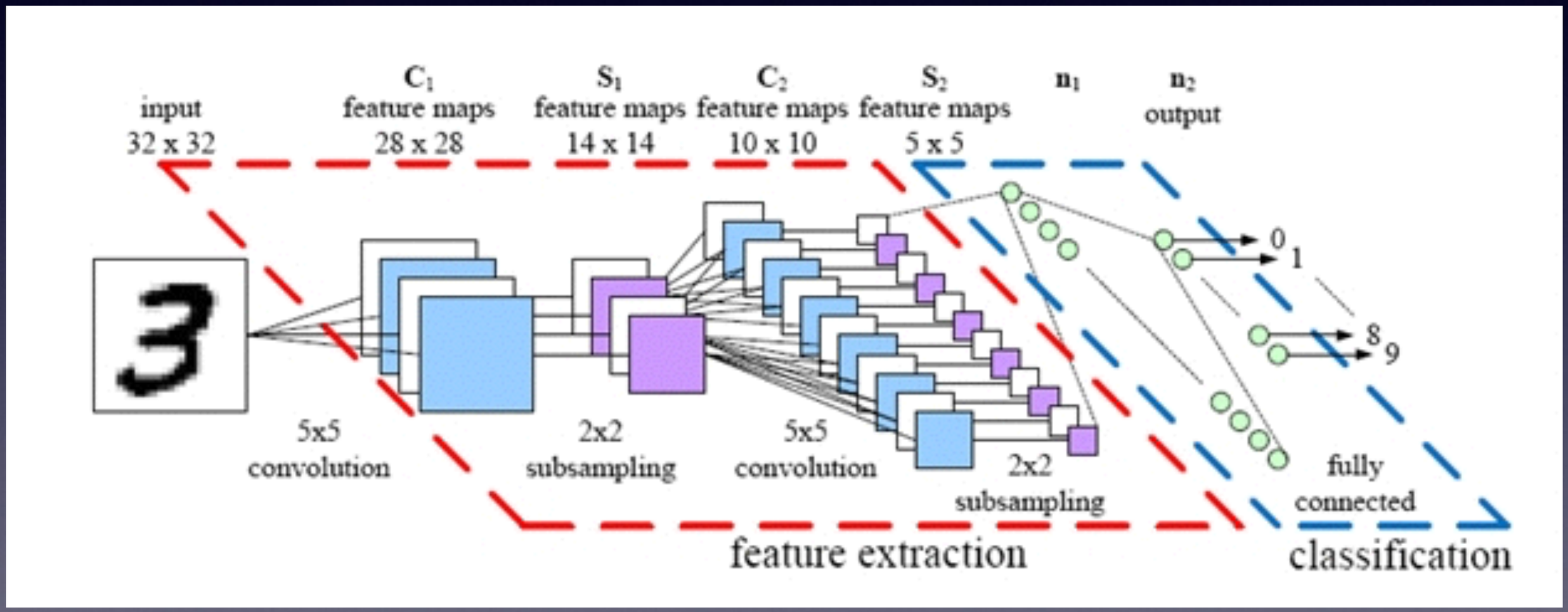
- (+200)1000 Captcha Images Histogram



	0	1	2	3	4	5	6	7	8
1000	537	791	484	499	605	510	313	692	569
1200	640	921	616	593	739	619	370	840	662

Convolutional Neural Network

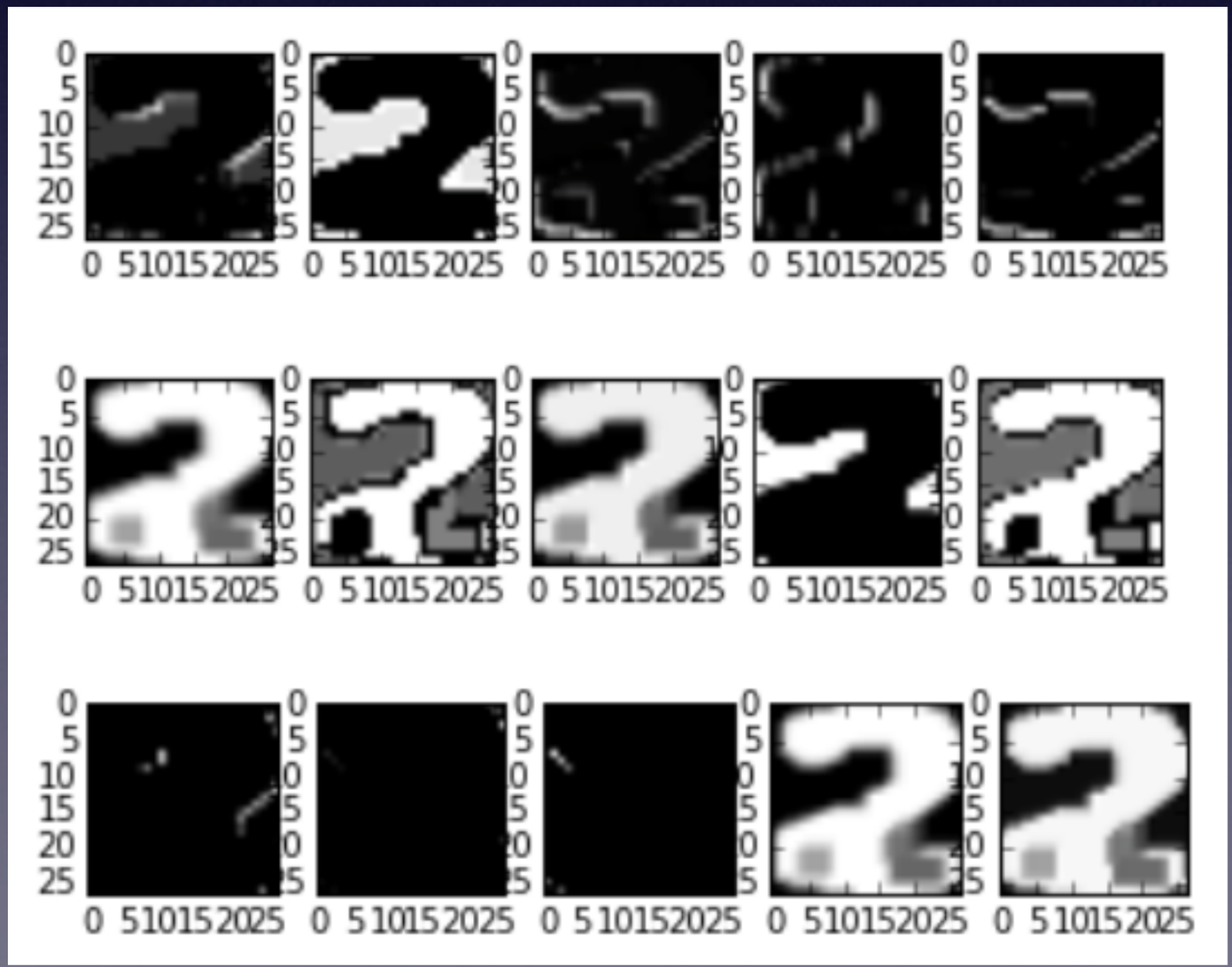
- Everyone says CNN is the 甲 to classify images



http://parse.ele.tue.nl/cluster/2/CNNArchitecture.jpg

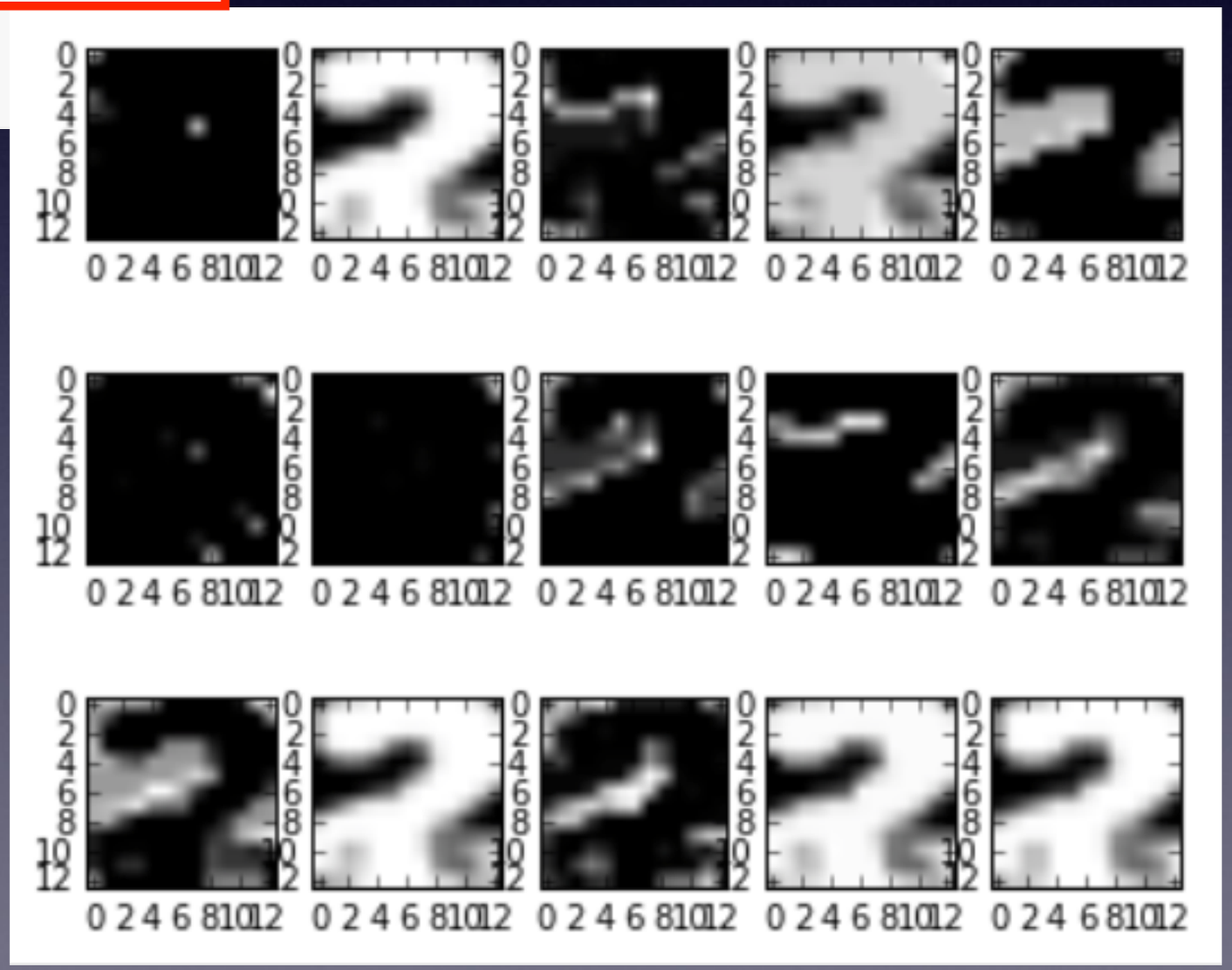
CNN-1

```
model1 = Sequential()
model1.add(Convolution2D(32, 1, 2, 2, weights=model.layers[0].get_weights()))
model1.add(Activation('relu'))
model1.compile(loss='categorical_crossentropy', optimizer=sgd)
```



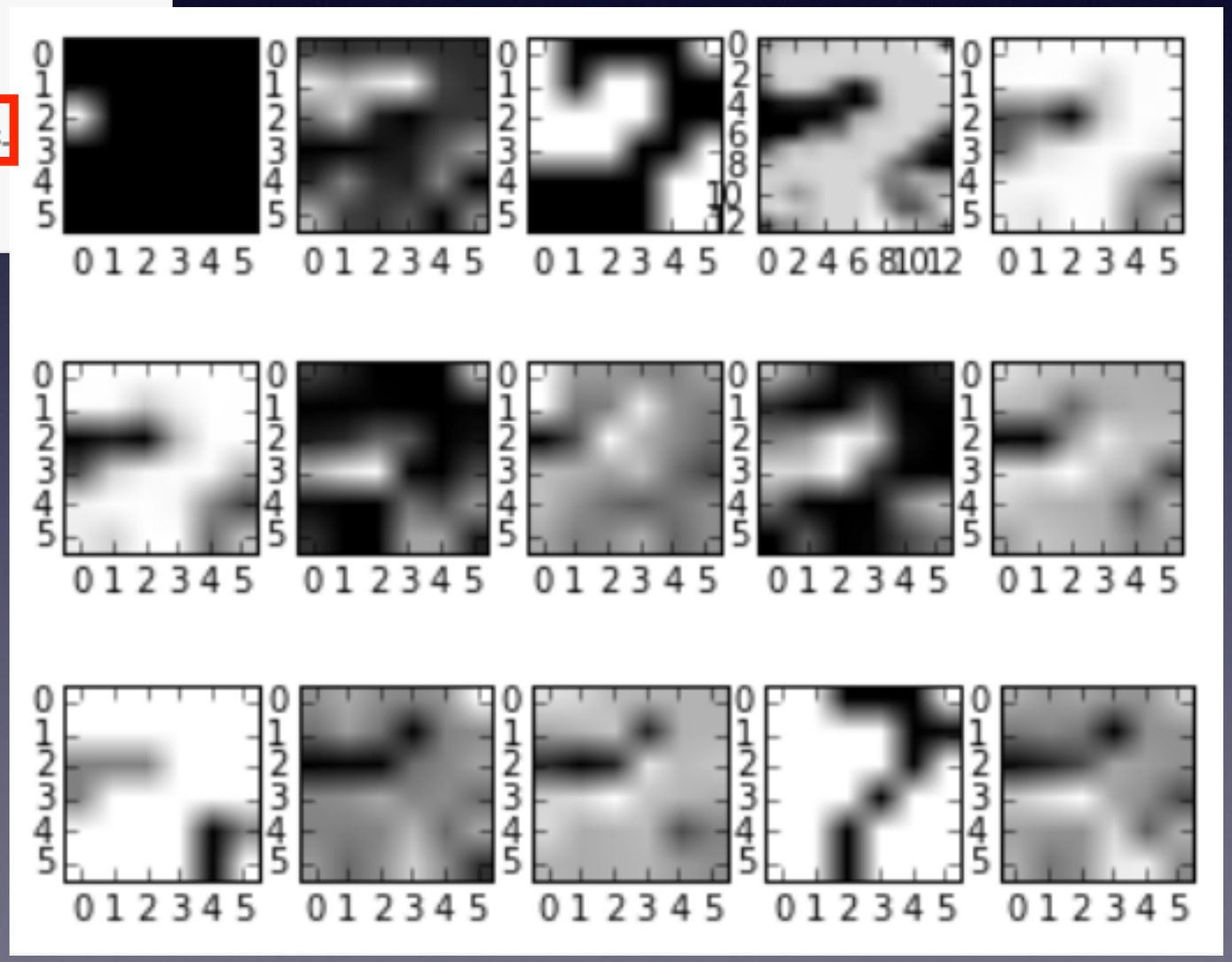
CNN-2

```
# Sequential wrapper model
model2 = Sequential()

# first convolutional layer
model2.add(Convolution2D(32, 1, 2, 2, weights=model.layers[0].get_weights()))
model2.add(Activation('relu'))

# second convolutional layer
model2.add(Convolution2D(48, 32, 2, 2, weights=model.layers[2].get_weights()))
model2.add(Activation('relu'))
model2.add(MaxPooling2D(poolsize=(2,2)))

model2.compile(loss='categorical_crossentropy', optimizer=sgd)
```



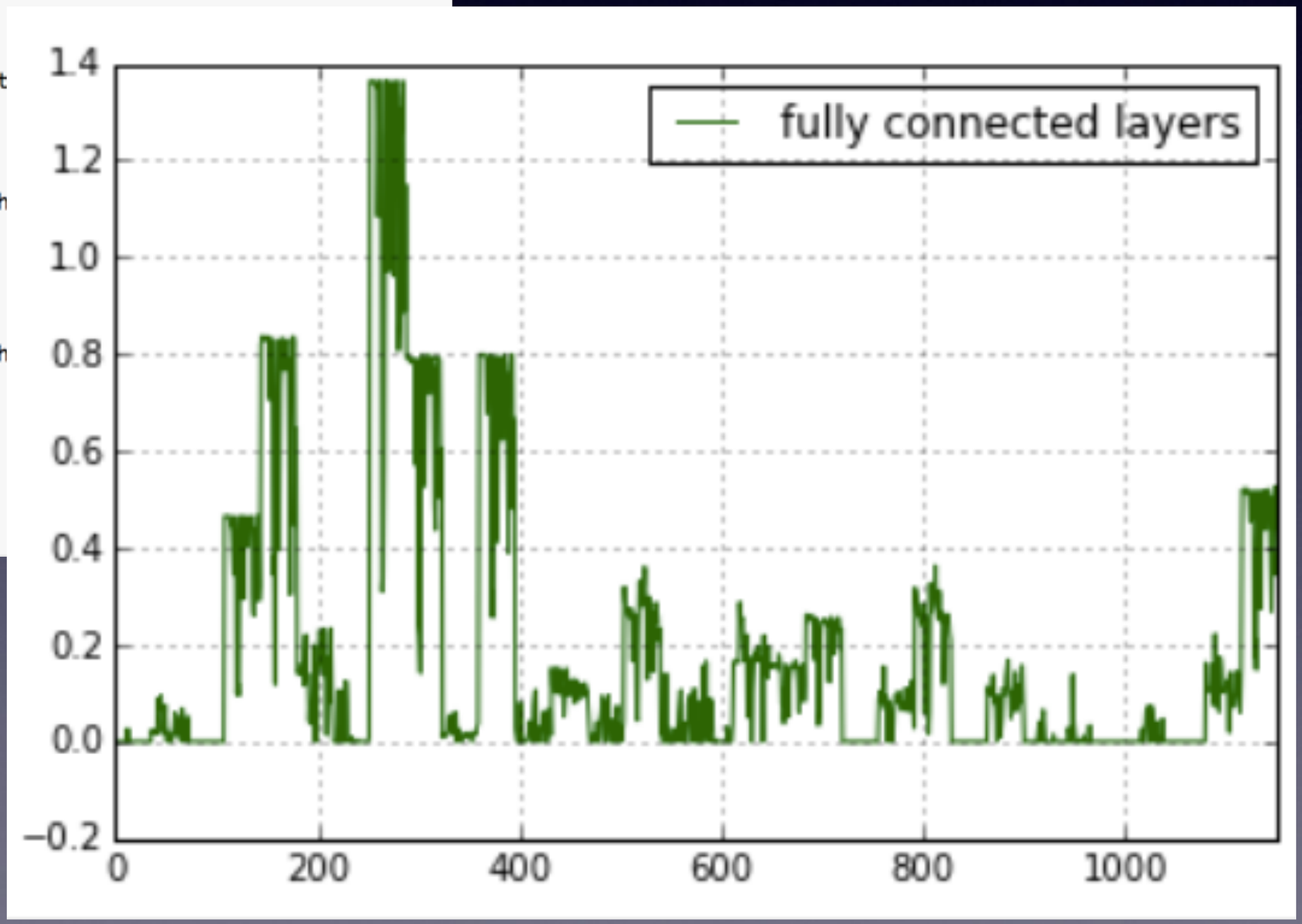
CNN-3

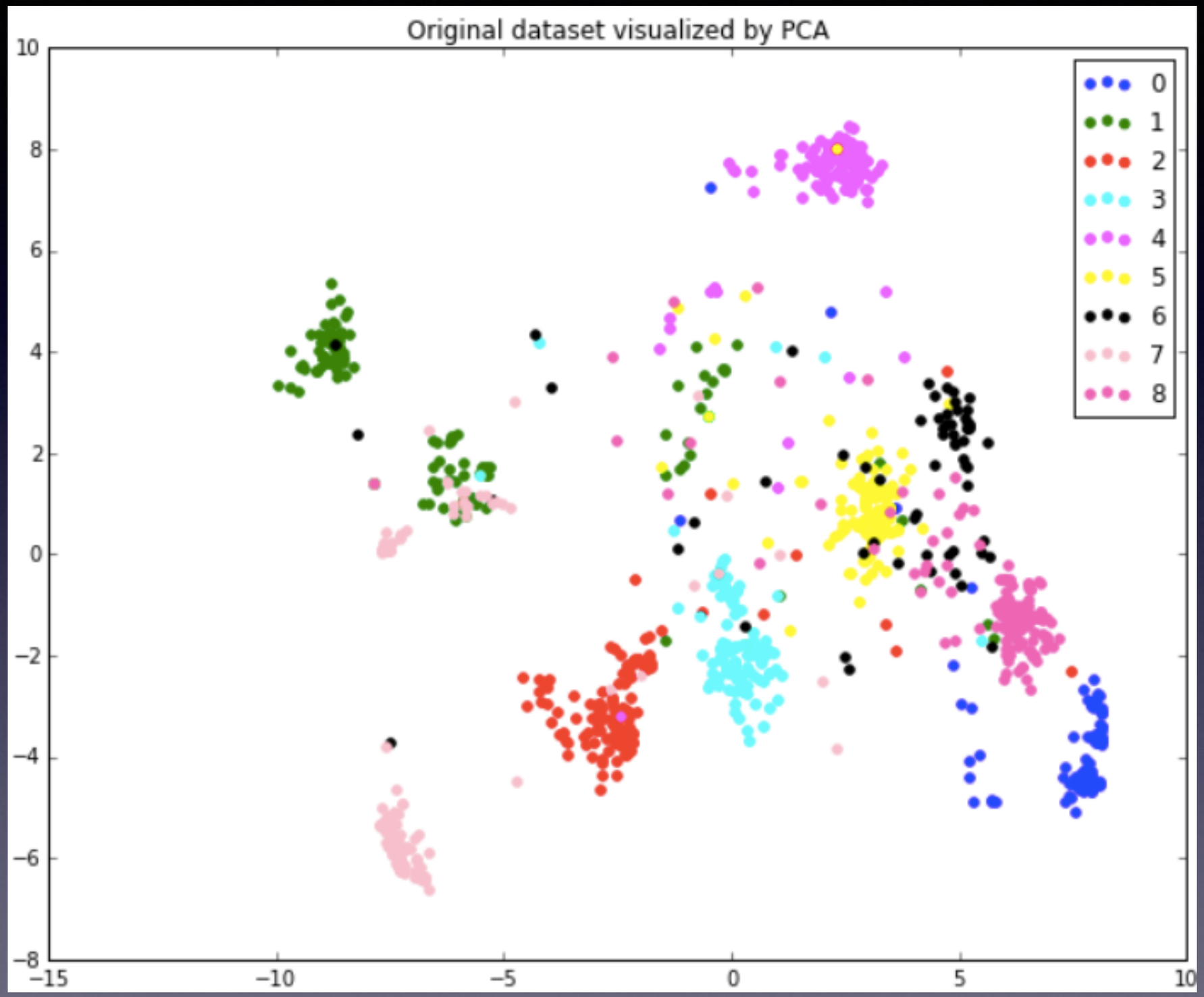
```
# Sequential wrapper model
model3 = Sequential()

# first convolutional layer
model3.add(Convolution2D(32, 1, 2, 2, weights=model.layers[0].get_weights()))
model3.add(Activation('relu'))

# second convolutional layer
model3.add(Convolution2D(48, 32, 2, 2, weights=model.layers[2].get_weights()))
model3.add(Activation('relu'))
model3.add(MaxPooling2D(poolsize=(2,2)))

# third convolutional layer
model3.add(Convolution2D(32, 48, 2, 2, weights=model.layers[5].get_
model3.add(Activation('relu'))
model3.add(MaxPooling2D(poolsize=(2,2)))
model3.compile(loss='categorical_crossentropy', optimizer=sgd)
```

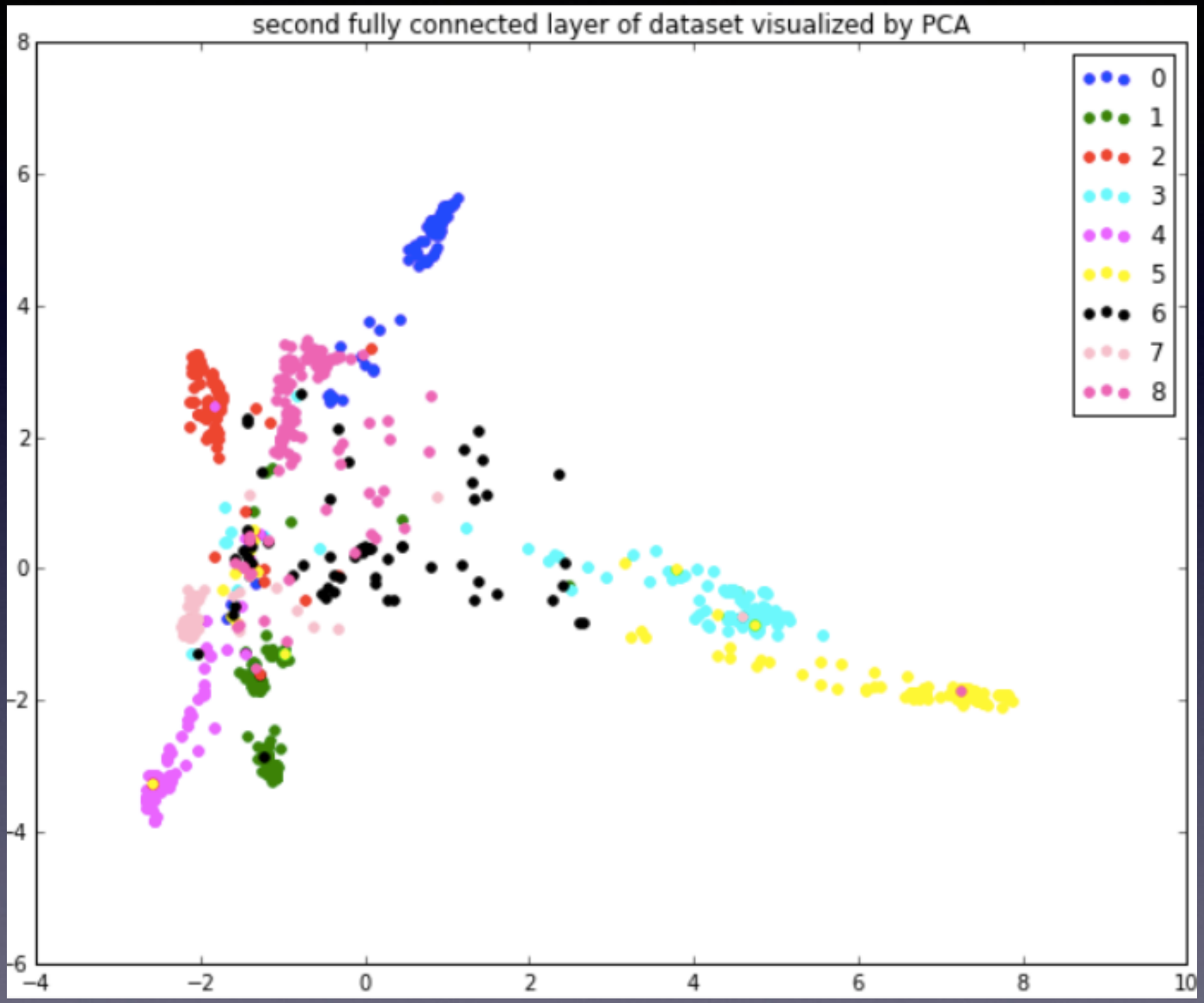


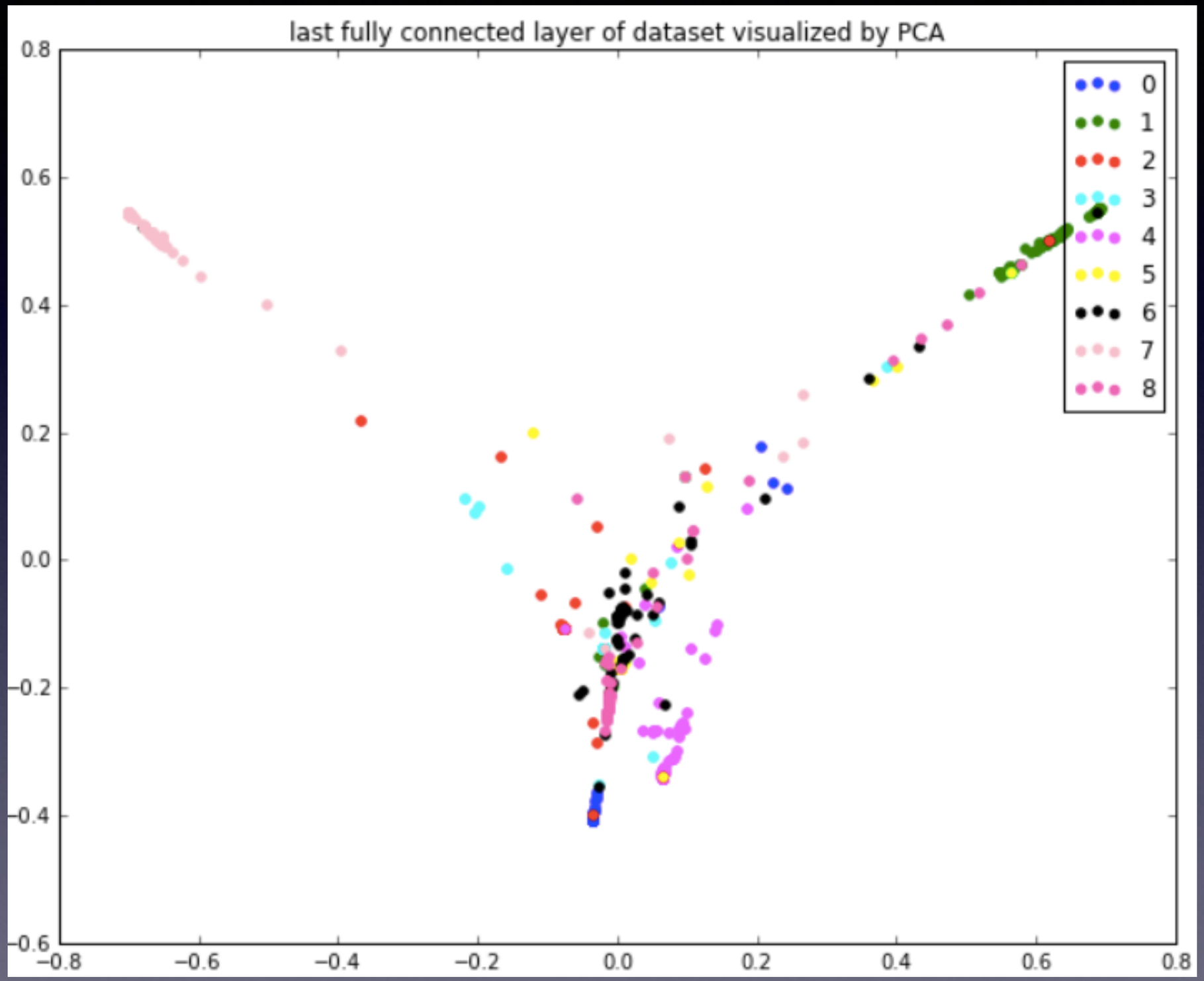
Fully Connected Layers

```
# Creating the model which consists of 3 conv layers followed by
# 2 fully conntected layers

# Sequential wrapper model
model4 = Sequential()

# first convolutional layer
model4.add(Convolution2D(32, 1, 2, 2, weight
model4.add(Activation('relu'))

# second convolutional layer
model4.add(Convolution2D(48, 32, 2, 2, weigh
model4.add(Activation('relu'))
model4.add(MaxPooling2D(poolsize=(2,2)))

# third convolutional layer
model4.add(Convolution2D(32, 48, 2, 2, weigh
model4.add(Activation('relu'))
model4.add(MaxPooling2D(poolsize=(2,2)))

# convert convolutional filters to flatt so
# fully connected layers
model4.add(Flatten())
```

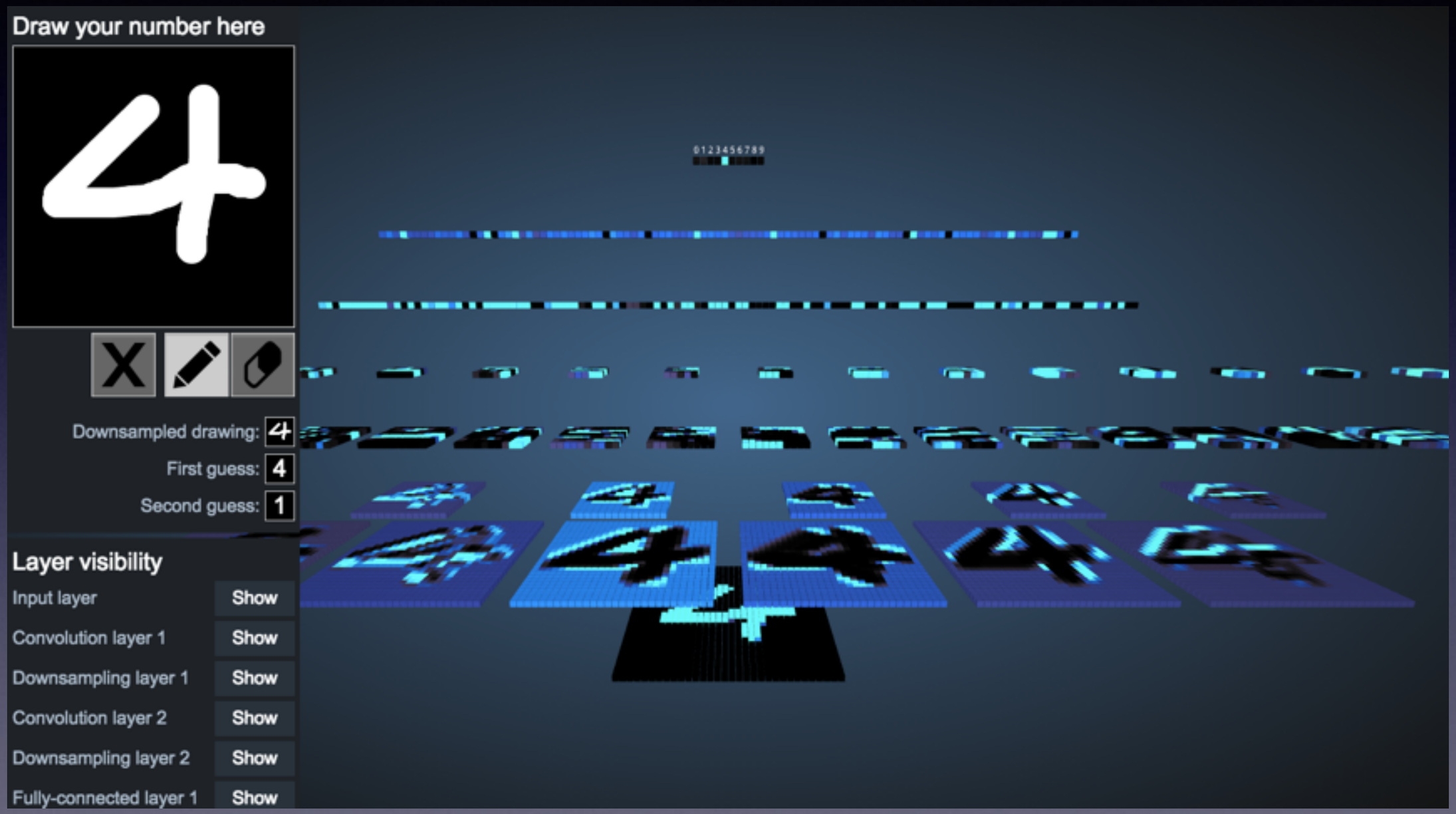



Original dataset visualized by PCA
10
8
6
4
2
0
−2
−4
−6
−8
−15
−10
−5
0
5
10
0
1
2
3
4
5
6
7
8


second fully connected layer of dataset visualized by PCA
0
1
2
3
4
5
6
7
8
8
6
4
2
0
−2
−4
−6
−4
−2
0
2
4
6
8
10


last fully connected layer of dataset visualized by PCA
0
1
2
3
4
5
6
7
8
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6
−0.8
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8

3D convolutional network visualization



http://scs.ryerson.ca/~aharley/vis/

Build Model

- Two Convolution Layers

```
l_in = lasagne.layers.InputLayer(
    shape=(batch_size, 1, input_width, input_height),
    )
```

```
l_conv1 = lasagne.layers.Conv2DLayer(
    l_in,
    num_filters=32,
    filter_size=(5, 5),
    nonlinearity=lasagne.nonlinearities.rectify,
    W=lasagne.init.GlorotUniform(),
    )
l_pool1 = lasagne.layers.MaxPool2DLayer(l_conv1, pool_size=(2, 2))
```

```
l_conv2 = lasagne.layers.Conv2DLayer(
    l_pool1,
    num_filters=32,
    filter_size=(5, 5),
    nonlinearity=lasagne.nonlinearities.rectify,
    W=lasagne.init.GlorotUniform(),
    )
l_pool2 = lasagne.layers.MaxPool2DLayer(l_conv2, pool_size=(2, 2))
```

Build Model

- Two Hidden Layers
- Output Layer

```
l_hidden1 = lasagne.layers.DenseLayer(
    l_pool2,
    num_units=256,
    nonlinearity=lasagne.nonlinearities.rectify,
    W=lasagne.init.GlorotUniform(),
    )

l_hidden1_dropout = lasagne.layers.DropoutLayer(l_hidden1, p=0.5)
```

```
l_hidden2 = lasagne.layers.DenseLayer(
    l_hidden1_dropout,
    num_units=256,
    nonlinearity=lasagne.nonlinearities.rectify,
    )
l_hidden2_dropout = lasagne.layers.DropoutLayer(l_hidden2, p=0.5)
```

```
l_out = lasagne.layers.DenseLayer(
    l_hidden1_dropout,
    num_units=output_dim,
    nonlinearity=lasagne.nonlinearities.softmax,
    W=lasagne.init.GlorotUniform(),
    )
```

```
Epoch 500 of 500 took 12.887s
  training loss:                0.075859
  validation loss:              0.896478
  validation accuracy:          91.00 %%
```

Tuning

- Train Data : 4000
- Each of Test and Validation Data : 1000

	Kernel	Epochs	Learning Rate	Accuracy
1	(5, 5)	500	0.01	91.0%
2	(3, 3)	50	0.01	91.0%
3	(3, 3)	50	0.009	91.0%
4	(3, 3)	100	0.001	89.3%
5	(4, 4)	50	0.009	91.5%
6	(4, 4)	200	0.009	91.5%

Tuning

- Train Data : 3000
- Each of Test and Validation Data : 1500

	Kernel	Epochs	Learning Rate	Accuracy
1	(5, 5)	500	0.01	92.9%
2	(3, 3)	50	0.01	92.0%
3	(3, 3)	50	0.009	91.6%
4	(3, 3)	100	0.001	91.0%
5	(4, 4)	50	0.009	91.8%
6	(4, 4)	200	0.009	92.2%

Tuning

- Train Data : 2000
- Each of Test and Validation Data : 2000

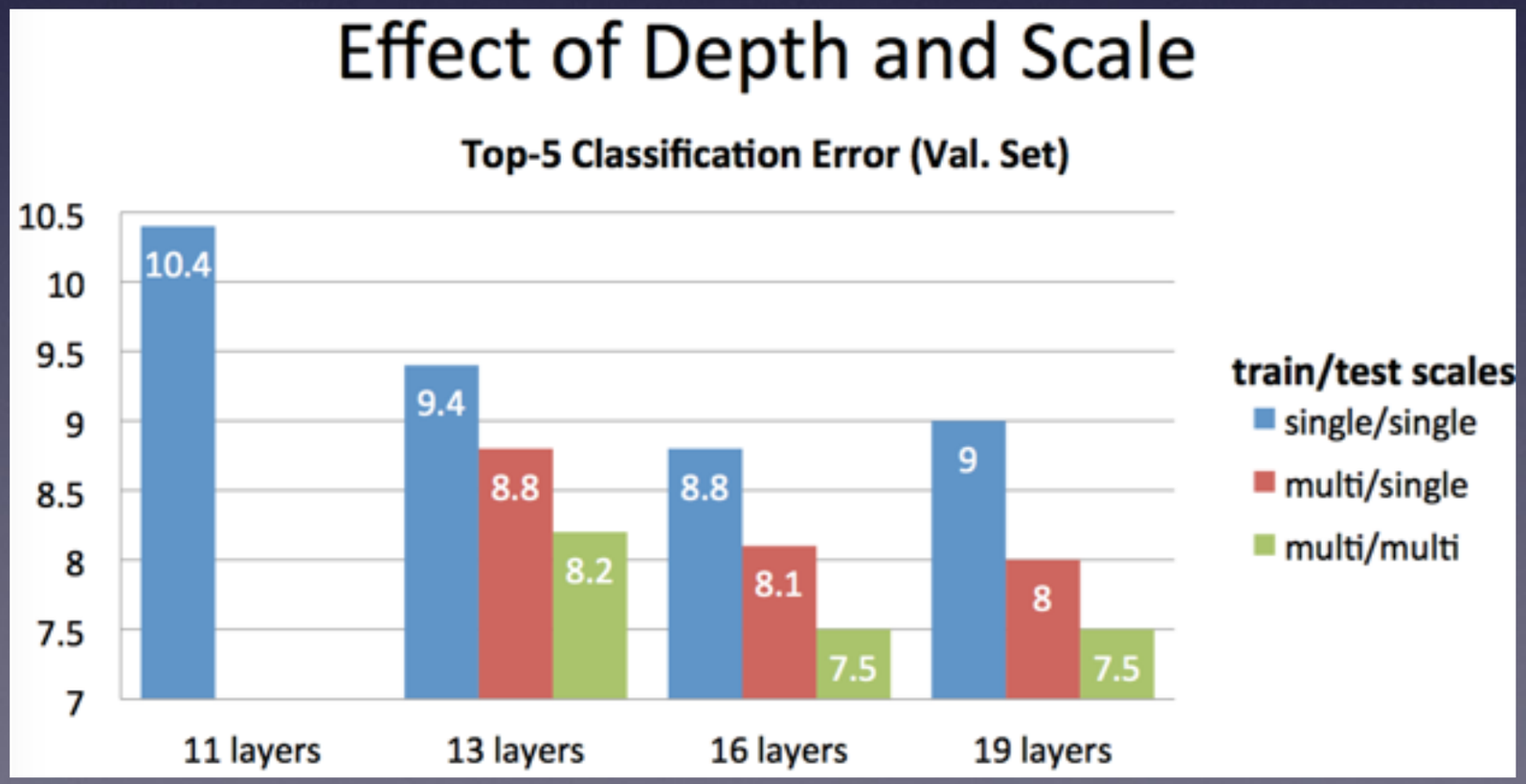
	Kernel	Epochs	Learning Rate	Accuracy
1	(5, 5)	500	0.01	91.6%
2	(3, 3)	50	0.01	90.8%
3	(3, 3)	50	0.009	90.4%
4	(3, 3)	100	0.001	89.7%
5	(4, 4)	50	0.009	91.1%
6	(4, 4)	200	0.009	91.6%

UNIVERSITY OF OXFORD

Google DeepMind

Very Deep ConvNets for Large-Scale Image Recognition

Karen Simonyan, Andrew Zisserman
University of Oxford



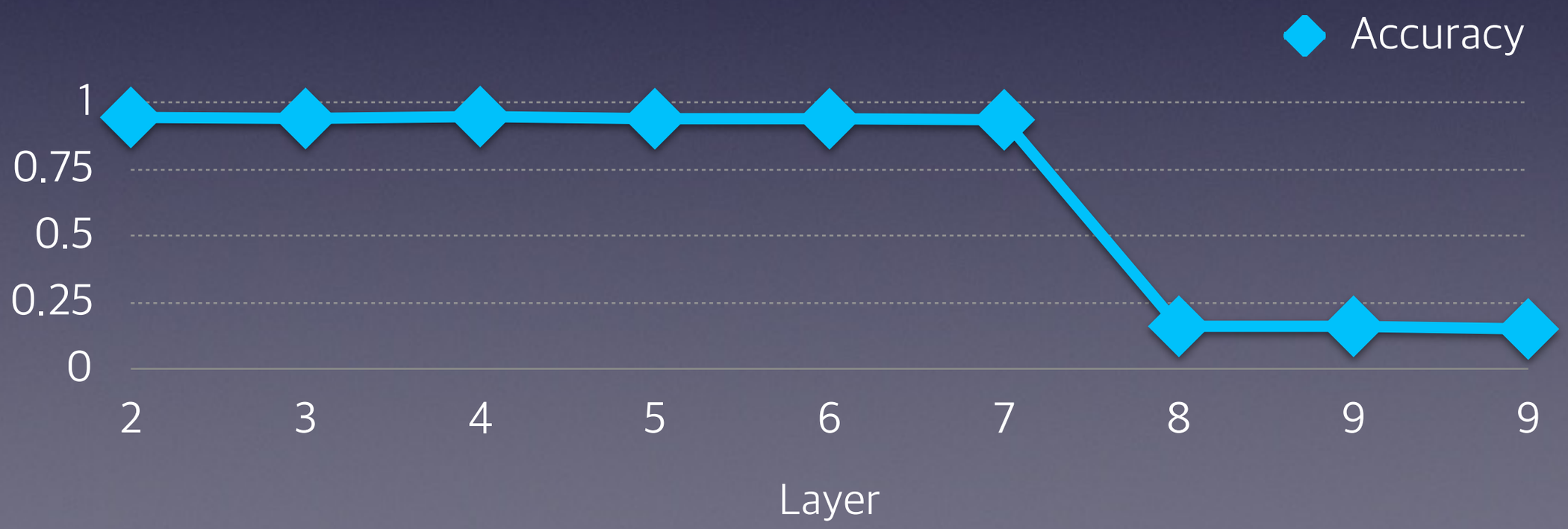
Add Convolutional Layers

```
# Convolution
nn = Classifier(
    layers=[
        Convolution("Rectifier", channels=9, kernel_shape=(3,3), border_mode='full'),
        Convolution("Rectifier", channels=9, kernel_shape=(3,3), border_mode='full'),
        Convolution("Rectifier", channels=9, kernel_shape=(3,3), border_mode='full'),
        Convolution("Rectifier", channels=9, kernel_shape=(3,3), border_mode='full'),
        Convolution("Rectifier", channels=9, kernel_shape=(3,3), border_mode='full'),
        Convolution("Rectifier", channels=9, kernel_shape=(3,3), border_mode='full'),
        Convolution("Rectifier", channels=9, kernel_shape=(3,3), border_mode='full'),
        Convolution("Rectifier", channels=9, kernel_shape=(3,3), border_mode='full'),
        Convolution("Rectifier", channels=9, kernel_shape=(3,3), border_mode='full'),
        Layer("Softmax")],
    learning_rate=0.01,
    n_iter=10,
    verbose=True)
nn.fit(trainX, trainY)
```

Add Convolutional Layers

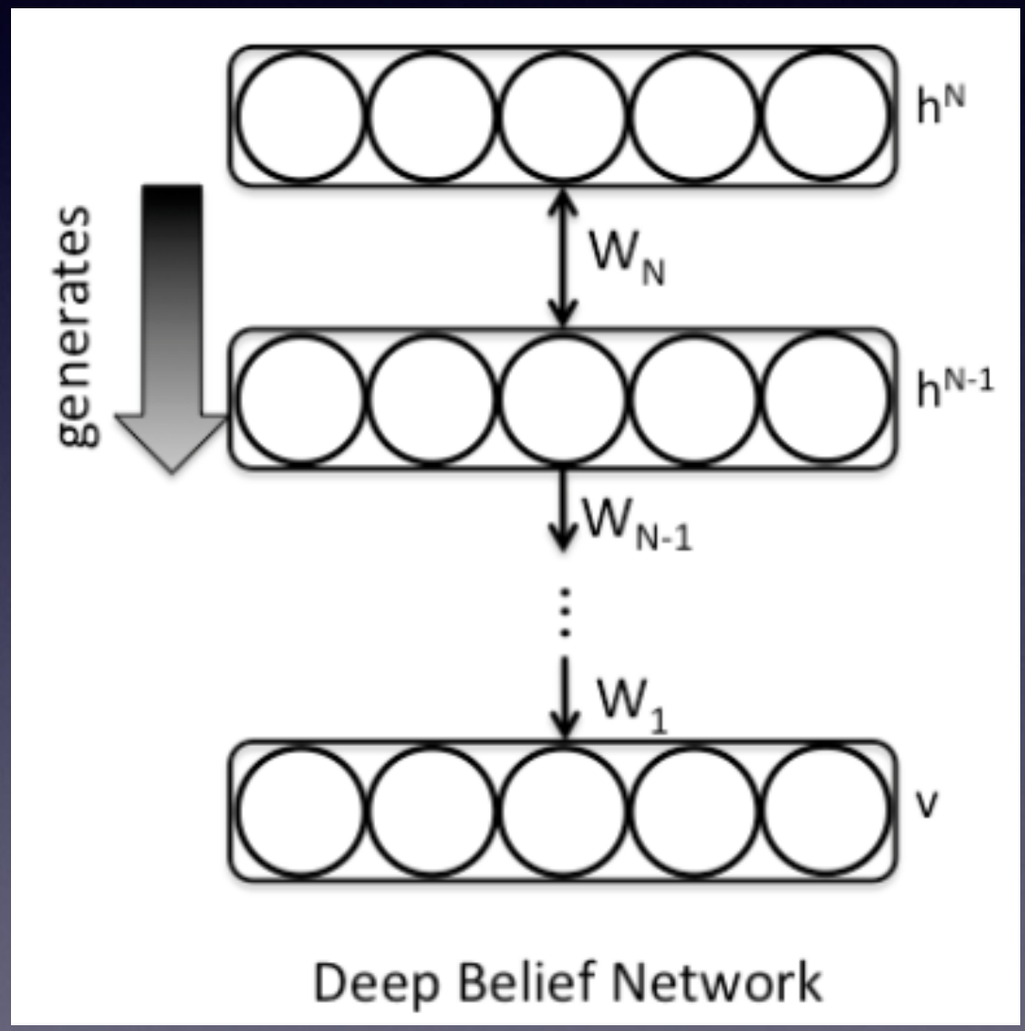
- **Convolution Layers is not black magic!**
  - kernel : (3,3), Epochs : 10, Learning Rate : 0.01

Layer	2	3	4	5	6	7	8	9	10
Accuracy	94.5%	94.1%	94.8%	94.0%	94.0%	93.6%	16.1%	16.1%	15.1%



Deep Belief Network

- An input layer : 784 inputs
- A hidden layer : 500 nodes
- Output layer : 9 nodes (0-8)
- learning rates : 0.009
- epochs : 10



Deep Belief Network

- Code

```
dbn = DBN(
    [trainX.shape[1], 500, 9],
    learn_rates = 0.009,
    learn_rate_decays = 0.9,
    epochs = 10,
    verbose = 1)
dbn.fit(trainX, trainY)
```

- Accuracy Score

```
accuracy score : 0.95
classification report :
             precision    recall  f1-score   support

          0       0.96      0.93      0.94        70
          1       0.93      0.96      0.94        93
          2       0.86      0.93      0.89        55
          3       0.98      0.95      0.96        58
          4       0.98      0.98      0.98        64
          5       0.98      0.94      0.96        62
          6       0.90      0.84      0.87        45
          7       1.00      0.99      0.99       100
          8       0.91      0.98      0.95        53

avg / total       0.95      0.95      0.95       600
```

Confusion matrix of DBN

Train Data	Test Data	Feature
Captcha	Captcha	No Feature

	0	1	2	3	4	5	6	7	8
0	65	0	3	0	0	0	1	0	1
1	2	89	1	0	0	0	0	0	1
2	0	1	51	1	0	0	1	0	1
3	0	1	2	55	0	0	0	0	0
4	0	0	0	0	63	0	0	0	1
5	0	2	1	0	0	58	1	0	0
6	1	3	1	0	0	1	38	0	1
7	0	0	0	0	0	0	1	99	0
8	0	0	0	0	1	0	0	0	52

Deep Belief Network

- Grid Search

```
nn = Classifier(
    layers=[
        Layer("Rectifier", units=300), # 첫번째 히든레이어
        Layer("Softmax")], # 아웃풋 레이어
    verbose=2)

gs = GridSearchCV(nn, param_grid={
    'learning_rate': [0.009],
    'n_iter' : [10],
    'hidden0__units': [100, 200, 300],
    'hidden0__weight_decay' : [0.9, 0.09, 0.009, 0.0009, 0.00009],
    'hidden0__type': ["Rectifier", "Tanh"] # 첫번째 히든레이어
    })
gs.fit(trainX, trainY)
```

```
avg / total       0.93      0.93      0.93       600

Classifier(batch_size=1, debug=False, dropout_rate=None, f_stable=0.001,
    hidden0=<sknn.nn.Layer `Rectifier`: name=u'hidden0', units=200, weight_decay=9e-05>
    layers=[<sknn.nn.Layer `Rectifier`: name=u'hidden0', units=200, weight_decay=9e-05>
    learning_momentum=0.9, learning_rate=0.009, learning_rule=u'sgd',
    loss_type=u'mse', n_iter=10, n_stable=50,
    output=<sknn.nn.Layer `Softmax`: name=u'output', units=9>,
    random_state=None, regularize=None, valid_set=None, valid_size=0.0,
```

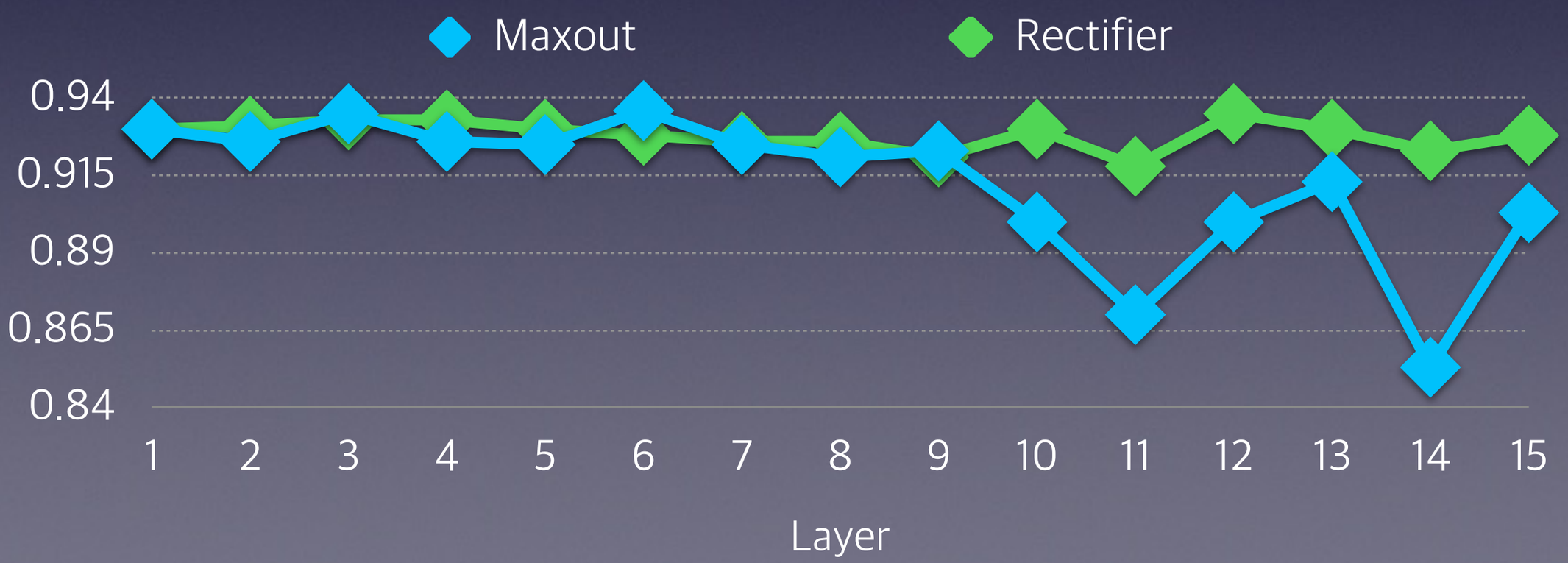
```
nn = Classifier(
    layers=[
        Layer("Rectifier", units=300), # 첫번째 히든레이어
        Layer("Softmax")], # 아웃풋 레이어
    verbose=2)

gs = GridSearchCV(nn, param_grid={
    'learning_rate': [0.09],
    'n_iter' : [10],
    'hidden0__units': [100, 200, 300],
    'hidden0__weight_decay' : [0.9, 0.09, 0.009, 0.0009, 0.00009],
    'hidden0__type': ["Rectifier", "Tanh"] # 첫번째 히든레이어
    })
gs.fit(trainX, trainY)
```

```
avg / total       0.01      0.10      0.02       600

Classifier(batch_size=1, debug=False, dropout_rate=None, f_stable=0.001,
    hidden0=<sknn.nn.Layer `Tanh`: name=u'hidden0', units=200, weight_decay=0.0009>
    layers=[<sknn.nn.Layer `Tanh`: name=u'hidden0', units=200, weight_decay=0.0009>
    learning_momentum=0.9, learning_rate=0.09, learning_rule=u'sgd',
    loss_type=u'mse', n_iter=10, n_stable=50,
    output=<sknn.nn.Layer `Softmax`: name=u'output', units=9>,
    random_state=None, regularize=None, valid_set=None, valid_size=0.0,
```

Adding Layers is not Black Magic

- Unit : 200, Epochs : 10
- Learning Rate : 0.009 ~ 0.000009, Weight Decay : 0.00009

Layer	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Maxout	93.0%	92.6%	93.5%	92.6%	92.5%	93.6%	92.5%	92.1%	92.3%	90.0%	87.0%	90.0%	91.3%	85.3%	90.3%
Rectifier	93.0%	93.1%	93.3%	93.3%	93.0%	92.8%	92.6%	92.6%	92.1%	93.0%	91.8%	93.5%	93.0%	92.3%	92.8%



Adding Layers is not Black Magic

	Layers	Unit	Epochs	Learning Rate	Weight Decay	Hidden Layers	Accuracy
1	Rectifier	200	10	0.009	0.00009	1	93.0%
2	Rectifier	200	10	0.009	0.00009	2	93.1%
3	Rectifier	200	10	0.009	0.00009	3	93.3%
4	Rectifier	200	10	0.009	0.00009	4	93.3%
5	Rectifier	200	10	0.009	0.00009	5	93.0%
6	Rectifier	200	10	0.009	0.00009	6	92.8%
7	Rectifier	200	10	0.009	0.00009	7	92.6%
8	Rectifier	200	10	0.009	0.00009	8	92.6%
9	Rectifier	200	10	0.009	0.00009	9	92.1%
10	Rectifier	200	10	0.009	0.00009	10	93.0%
11	Rectifier	200	10	0.009	0.00009	11	91.8%
12	Rectifier	200	10	0.009	0.00009	12	93.5%
13	Rectifier	200	10	0.009	0.00009	13	93.0%
14	Rectifier	200	10	0.009	0.00009	14	92.3%
15	Rectifier	200	10	0.009	0.00009	15	92.8%

Adding Layers is not Black Magic

	Layers	Unit	Epochs	Learning Rate	Weight Decay	Hidden Layers	Accuracy
1	Maxout	200	10	0.009	0.00009	1	93.0%
2	Maxout	200	10	0.009	0.00009	2	92.6%
3	Maxout	200	10	0.0009	0.00009	3	93.5%
4	Maxout	200	10	0.0009	0.00009	4	92.6%
5	Maxout	200	10	0.0009	0.00009	5	92.5%
6	Maxout	200	10	0.0009	0.00009	6	93.6%
7	Maxout	200	10	0.0009	0.00009	7	92.5%
8	Maxout	200	10	0.00009	0.00009	8	92.1%
9	Maxout	200	10	0.00009	0.00009	9	92.3%
10	Maxout	200	10	0.00009	0.00009	10	90.0%
11	Maxout	200	10	0.00009	0.00009	11	87.0%
12	Maxout	200	10	0.00009	0.00009	12	90.0%
13	Maxout	200	10	0.00009	0.00009	13	91.3%
14	Maxout	200	10	0.000009	0.00009	14	85.3%
15	Maxout	200	10	0.000009	0.00009	15	90.3%

GET
CURIOUS

NOT
FURIOUS

Deadly Simple

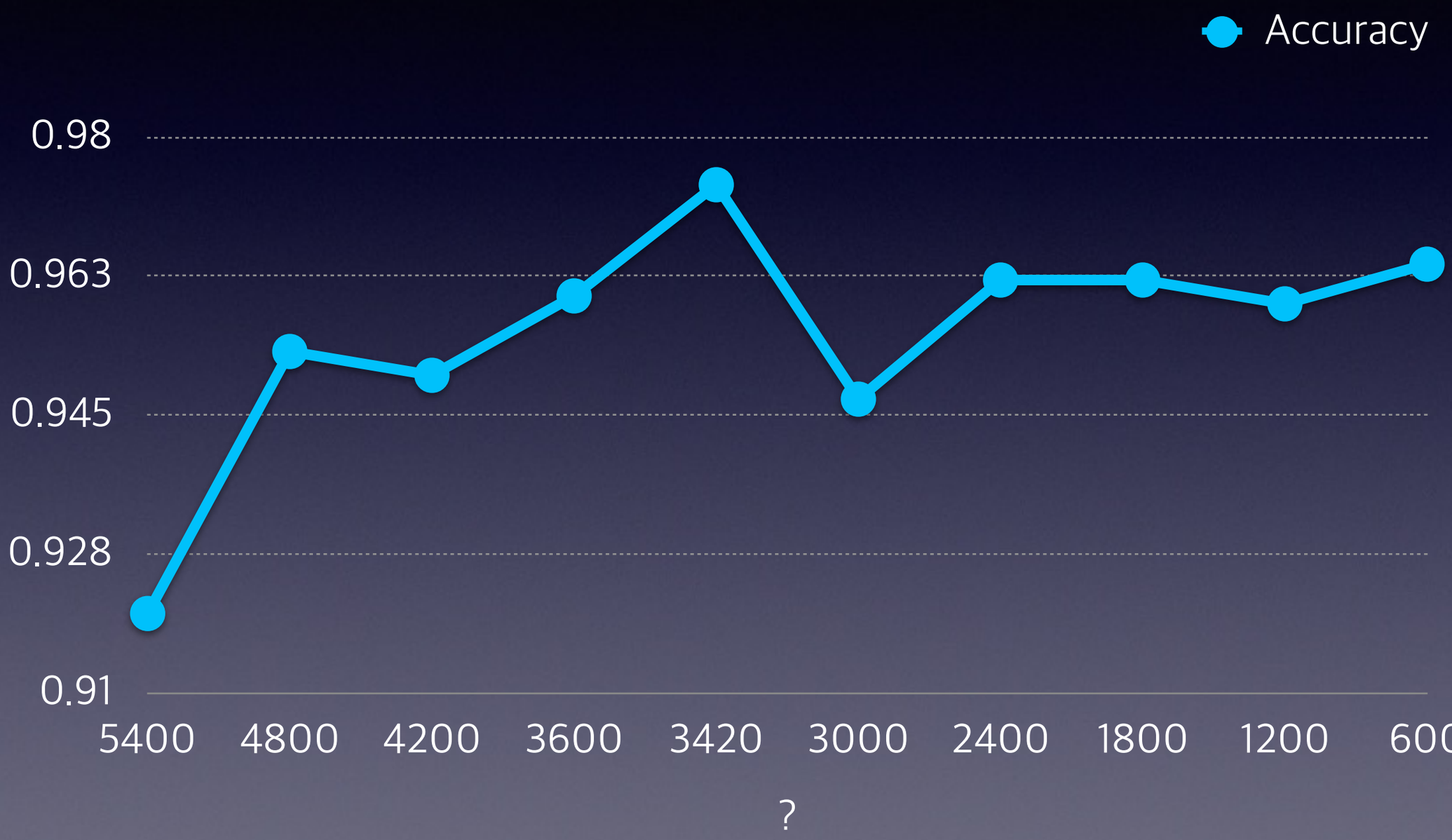
- Make a Model

```
In [4]: model = Sequential()

In [5]: model.add(Dense(input_dim=784, output_dim=200, init="uniform"))
        model.add(Activation("relu"))
        model.add(Dense(input_dim=200, output_dim=9, init="uniform"))
        model.add(Activation("softmax"))

In [6]: model.compile(loss='categorical_crossentropy', optimizer='sgd')
```

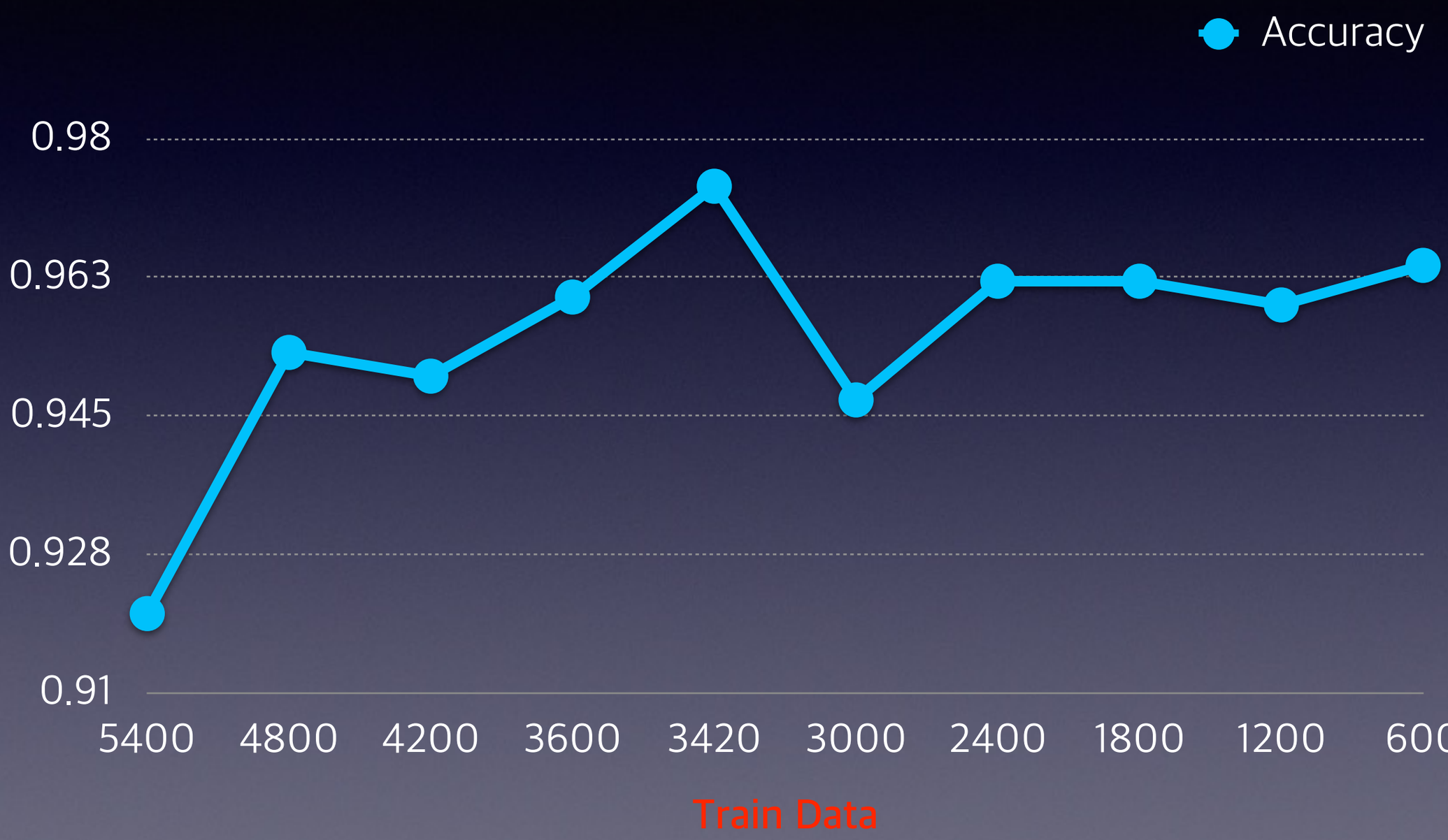
- Train!

```
mm = model.fit(trainX, trainY,
               batch_size=batch_size,
               nb_epoch=100,
               show_accuracy=True,
               verbose=0,
               validation_data=(testX, testY))
score = model.evaluate(valX, valY, show_accuracy=True, verbose=0, batch_size=32)
```

What makes it 97.4%?



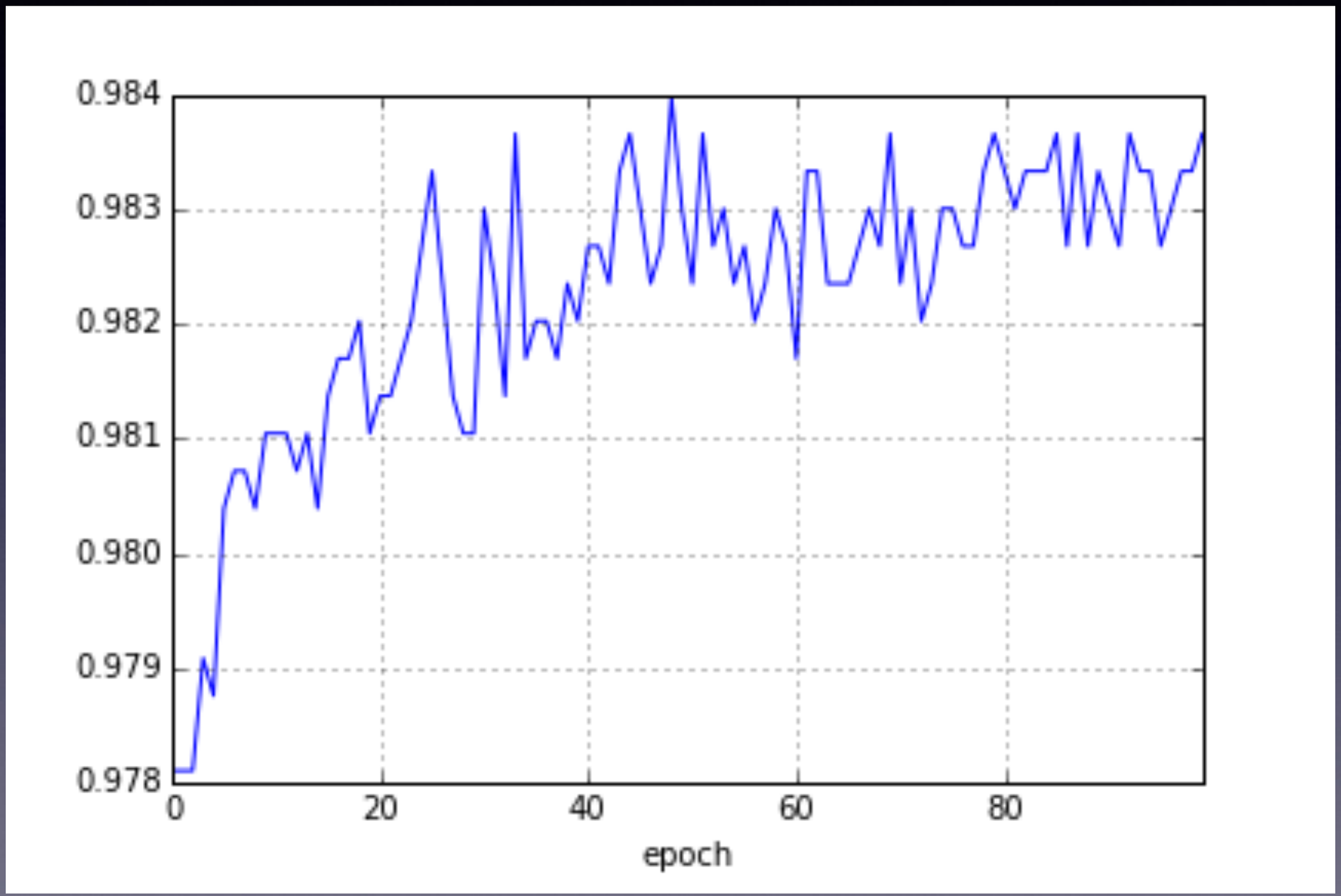
Rectifier Single Layer



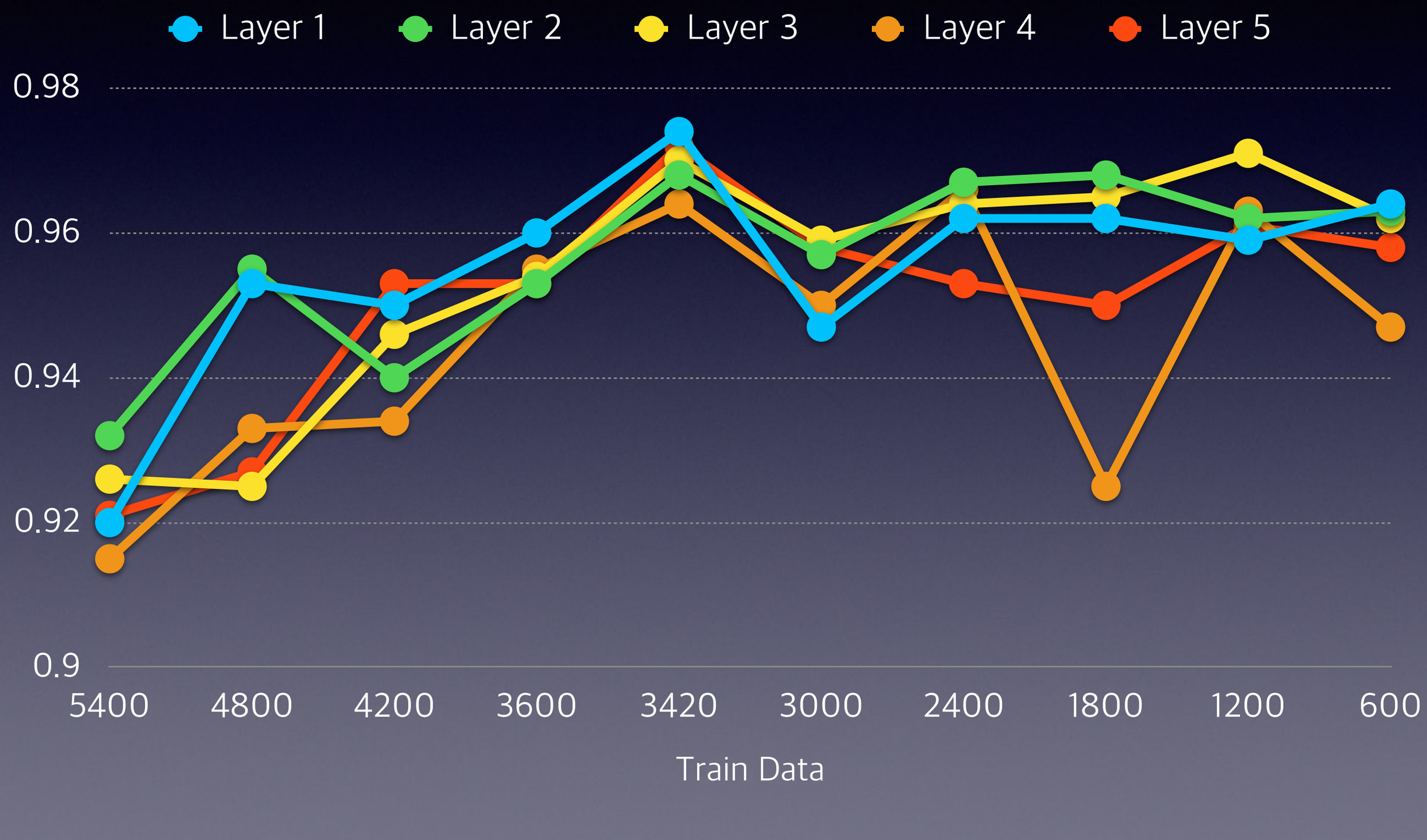
Rectifier Single Layer

	Unit	Epochs	Train	Test	Validation	Accuracy
1	200	100	5400	300	300	92.0%
2	200	100	4800	600	600	95.3%
3	200	100	4200	900	900	95.0%
4	200	100	3600	1200	1200	96.0%
5	200	100	3000	1500	1500	94.7%
6	200	100	2400	1800	1800	96.2%
7	200	100	1800	2100	2100	96.2%
8	200	100	1200	2400	2400	95.9%
9	200	100	600	2700	2700	96.4%
10	200	100	3420	1290	1290	97.4%

Train 3420



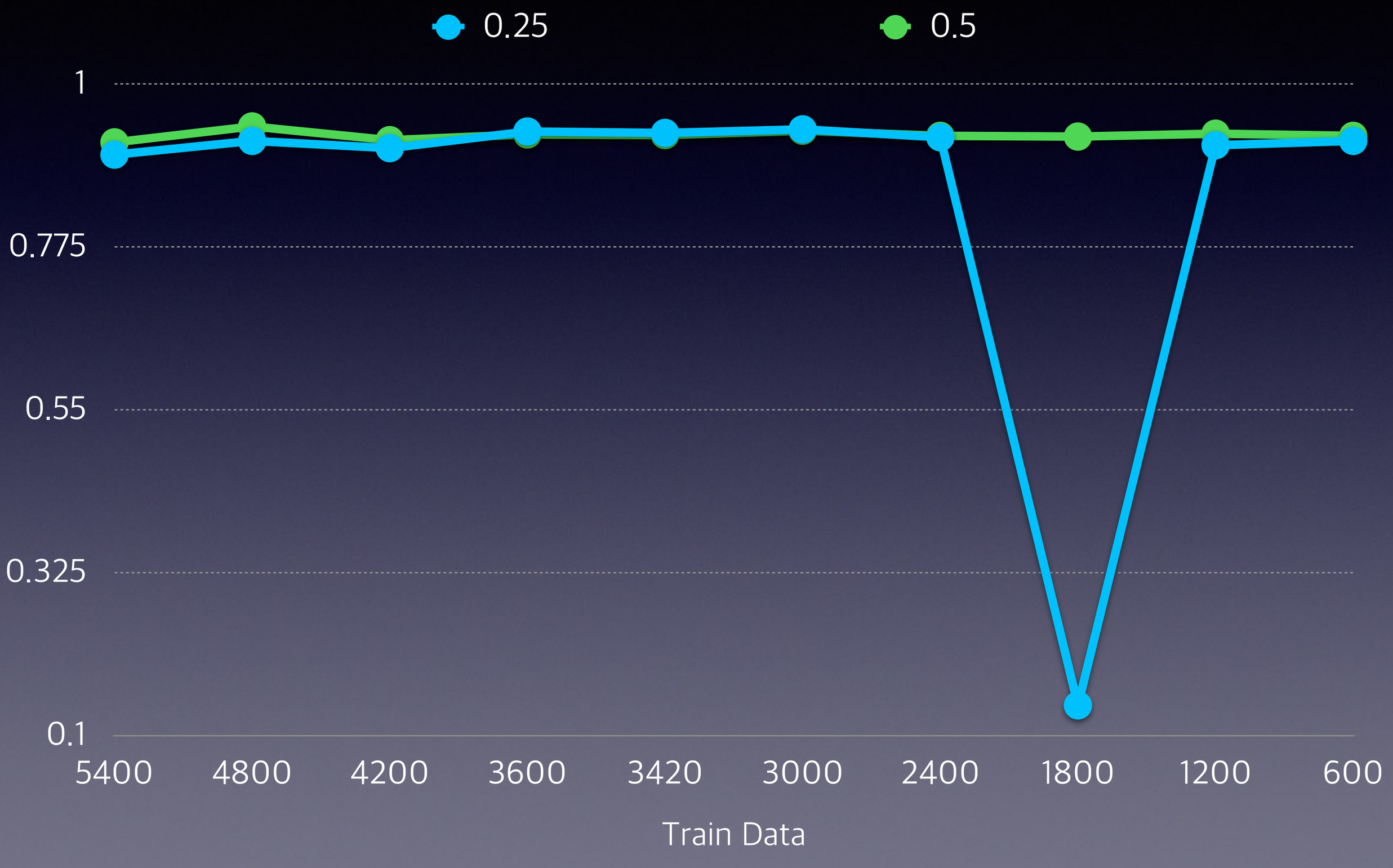
Rectifier Multiple Layers



Rectifier Multiple Layers

	Unit	Epochs	Train	Test	Val	**Accuracy (1-Layer)**	Accuracy (2-Layer)	Accuracy (3-Layer)	Accuracy (4-Layer)	Accuracy (5-Layer)
1	200	100	5400	300	300	**92.0%**	93.2%	92.6%	91.5%	92.1%
2	200	100	4800	600	600	**95.3%**	95.5%	92.5%	93.3%	92.7%
3	200	100	4200	900	900	**95.0%**	94.0%	94.6%	93.4%	95.3%
4	200	100	3600	1200	1200	**96.0%**	95.3%	95.4%	95.5%	95.3%
5	200	100	3000	1500	1500	**94.7%**	95.7%	95.9%	95.0%	95.8%
6	200	100	2400	1800	1800	**96.2%**	96.7%	96.4%	96.6%	95.3%
7	200	100	1800	2100	2100	**96.2%**	96.8%	96.5%	92.5%	95.0%
8	200	100	1200	2400	2400	**95.9%**	96.2%	97.1%	96.3%	96.1%
9	200	100	600	2700	2700	**96.4%**	96.3%	96.2%	94.7%	95.8%
10	200	100	3420	1290	1290	97.4%	96.8%	97.0%	96.4%	97.2%

Dropout


0.25
0.5
1
0.775
0.55
0.325
0.1
5400
4800
4200
3600
3420
3000
2400
1800
1200
600
Train Data

Dropout 0.25

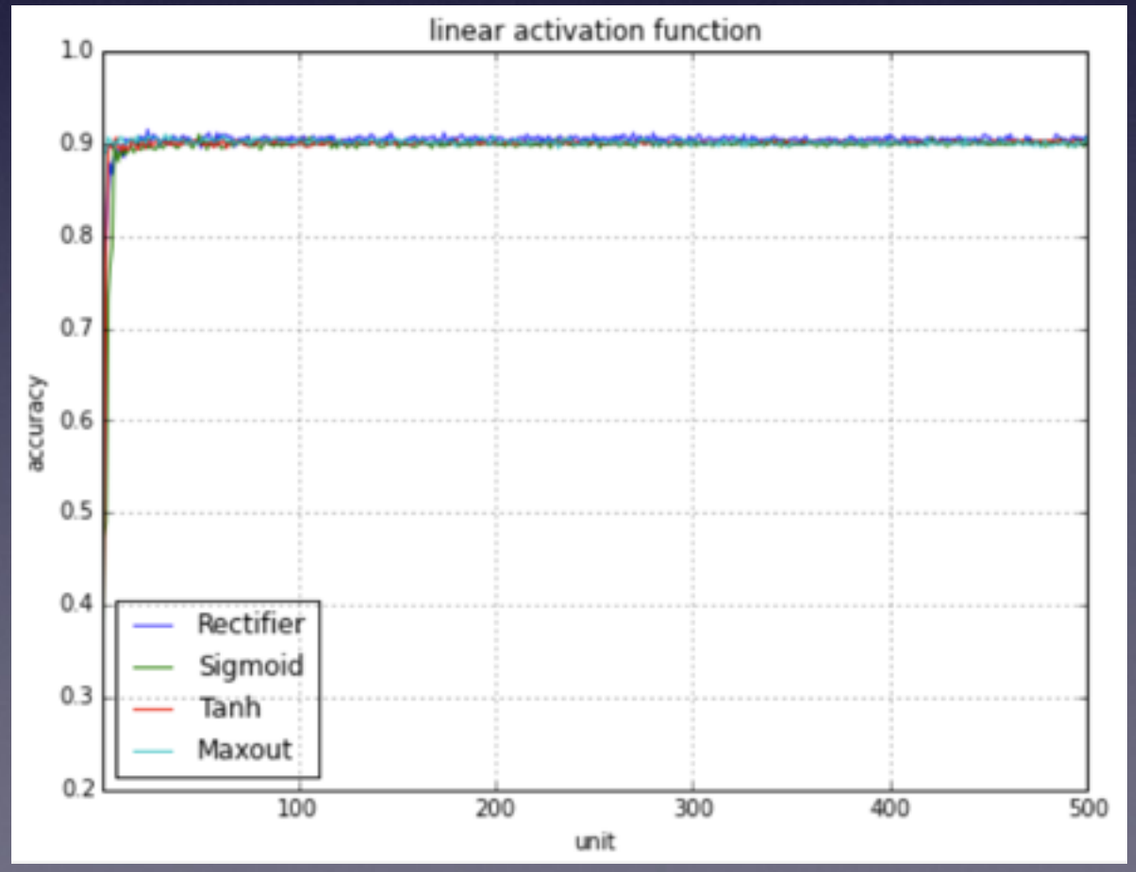
	Layer	Unit	Epochs	Train	Test	Validation	Accuracy
1	ReLU-1	128	100	5400	300	300	90.3%
2	ReLU-1	128	100	4800	600	600	92.2%
3	ReLU-1	128	100	4200	900	900	91.2%
4	ReLU-1	128	100	3600	1200	1200	93.5%
5	ReLU-1	128	100	3000	1500	1500	93.8%
6	ReLU-1	128	100	2400	1800	1800	92.7%
7	ReLU-1	128	100	1800	2100	2100	14.2%
8	ReLU-1	128	100	1200	2400	2400	91.6%
9	ReLU-1	128	100	600	2700	2700	92.2%
10	ReLU-1	128	100	3420	1290	1290	93.3%

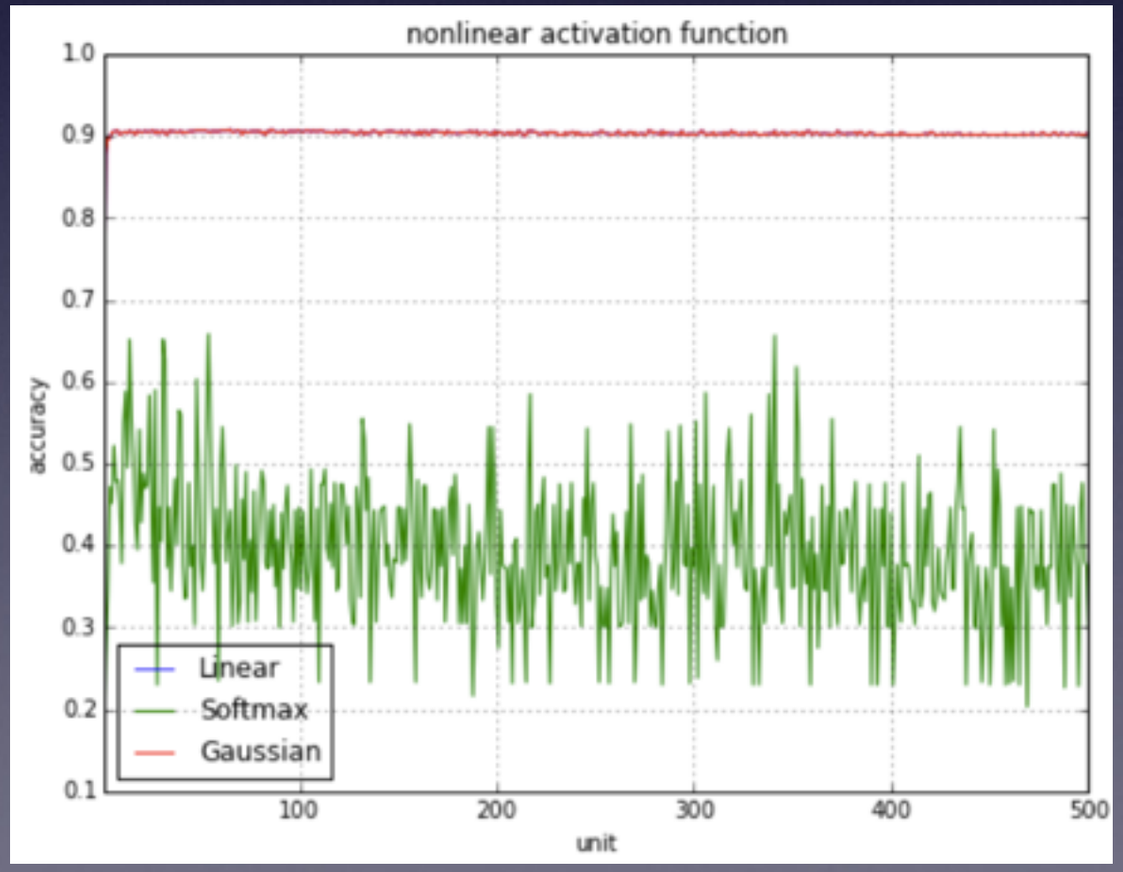
Dropout 0.5

	Layer	Unit	Epochs	Train	Test	Validation	Accuracy
1	ReLU-1	128	100	5400	300	300	92.0%
2	ReLU-1	128	100	4800	600	600	94.2%
3	ReLU-1	128	100	4200	900	900	92.3%
4	ReLU-1	128	100	3600	1200	1200	93.1%
5	ReLU-1	128	100	3000	1500	1500	93.6%
6	ReLU-1	128	100	2400	1800	1800	92.9%
7	ReLU-1	128	100	1800	2100	2100	92.8%
8	ReLU-1	128	100	1200	2400	2400	93.2%
9	ReLU-1	128	100	600	2700	2700	92.9%
10	ReLU-1	128	100	3420	1290	1290	93.0%

How Many Units need?
NonLinear vs Linear?

	Rectifier	Sigmoid	Tanh	Maxout	Linear	Softmax	Gaussian
mean	0.90	0.90	0.90	0.90	0.90	0.39	0.90
std	0.02	0.04	0.04	0.01	0.01	0.09	0.01
min	0.45	0.23	0.23	0.60	0.62	0.14	0.62
max	0.92	0.91	0.91	0.91	0.91	0.66	0.91





Deep Learning with Support Vector Machines

```
clf = SVC(cache_size=1000, kernel="rbf", C=10.0, gamma=0.03125)

clf.fit(trainX, trainY)

y_pred = clf.predict(testX)
score_accuracy = accuracy_score(y_pred, testY, normalize=True)
print "escape time : ", round(time()-t0, 3), "s"
print "accuracy is %s" % score_accuracy
```

```
escape time :  8.029 s
accuracy is 0.9275
```

```
pca = RandomizedPCA(n_components=144)

trainX = pca.fit_transform(trainX, trainY)
testX = pca.fit_transform(testX, testY)

t0 = time()
clf = SVC(cache_size=1000, kernel="rbf", C=10.0, gamma=0.03125)

clf.fit(trainX, trainY)

y_pred = clf.predict(testX)
score_accuracy = accuracy_score(y_pred, testY, normalize=True)
print "escape time : ", round(time()-t0, 3), "s"
print "accuracy is %s" % score_accuracy
```

```
escape time :  1.468 s
accuracy is 0.574166666667
```

```
# dimension reduction by CNN : 144
t0 = time()
clf = SVC(cache_size=1000, kernel="rbf", C=10.0, gamma=0.03125)

labels_train = np_utils.categorical_probas_to_classes(trainY)
labels_test = np_utils.categorical_probas_to_classes(testY)

clf.fit(trainX_deep, labels_train)

y_pred = clf.predict(testX_deep)
score_accuracy = accuracy_score(y_pred, labels_test, normalize=True)
print "escape time : ", round(time()-t0, 3), "s"
print "accuracy is %s" % score_accuracy
```

```
escape time :  0.615 s
accuracy is 0.9375
```

overfitting

1. Regularization : L1, L2, Dropout
2. The initialization of Weights and Biases
3. Hyper-Parameter : Learning Rate, Learning Rule, Epoch, Hidden Layer, Unit, Batch
4. Activation Function : Relu, Maxout, Sigmoid, Tanh, Softmax
5. Gradient Descent : SGD , Hessan Technique, Momentum, Nag, Adagrad, Adadelta, Rmsprop
6. Data Quantity : Train, Test, Validation
7. Cost Function : Sum-of-Squares, Cross-Entropy, L-BFGS

Summary

1. After preprocessing, MNIST can predict captcha image
2. Why don’t you get rid of “6” that break classifier?
3. Deep Learning is not necessarily the answer
4. CNN is not necessarily the answer
5. Many layers is not alwalys good choice
6. Do you know your data well?

Next

1. Detect Security Card Numbers
2. Skynet
3. Drone looking for wild ginseng
4. Drone seeing smell
5. Find out needle in sand beach
6. Classify the morphologies of distant galaxies in our Universe from Kaggle
7. AI reply bot
8. Predict the people who you will talk with tomorrow and conversation topics
9. Score human faces and detect whether or not plastic surgery