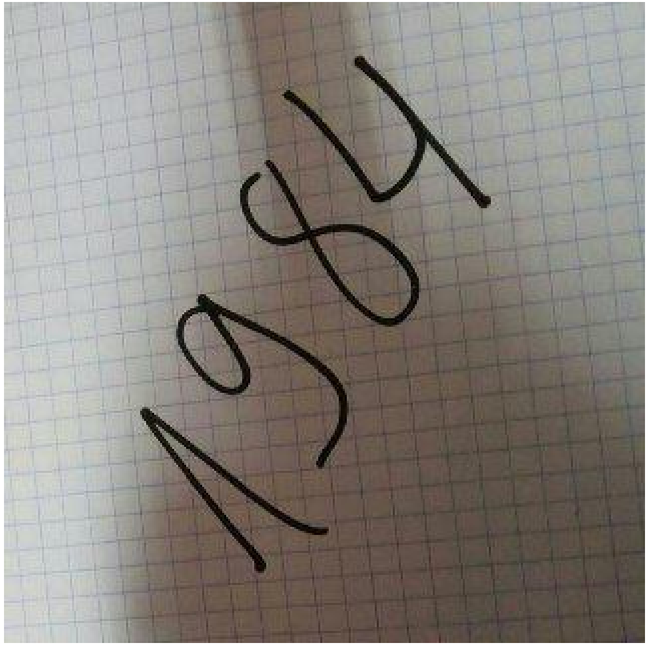
Digit Recognition

Lee and Ran

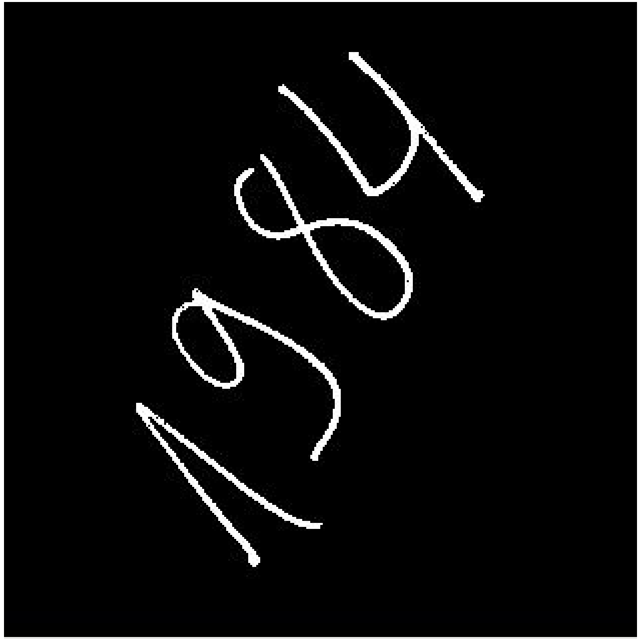
Future Learning, Course 1

Dec, 2017

Load Image in grayscale
Apply threshold for binary representation

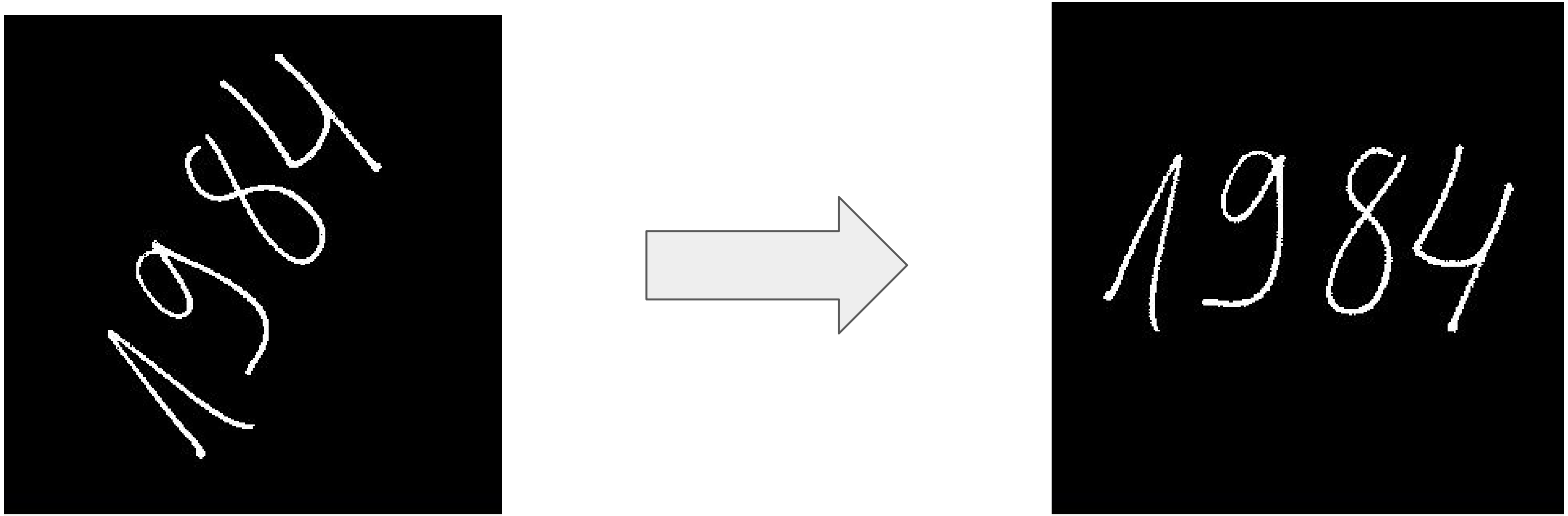


```
img = cv2.imread('big-Num1.jpg', 0)
```



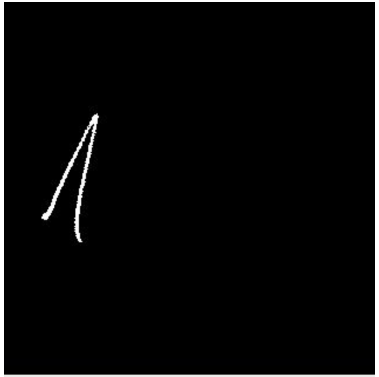
```
ret, thresh = cv2.threshold(img, 40, 255, cv2.THRESH_BINARY)
thresh = abs(thresh.astype(float) - 255)
thresh = thresh.astype('uint8')
cv2.imshow("thresh", thresh)
```

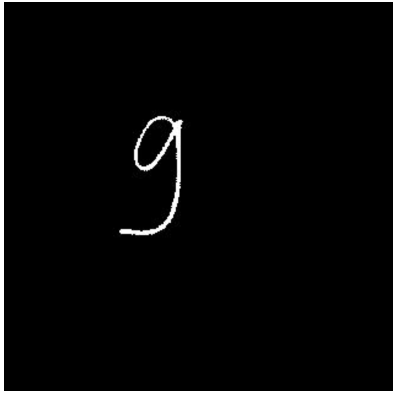
ROTATION

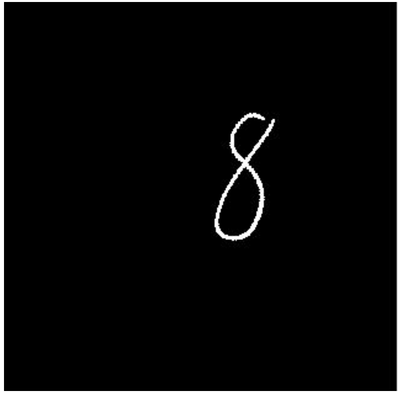


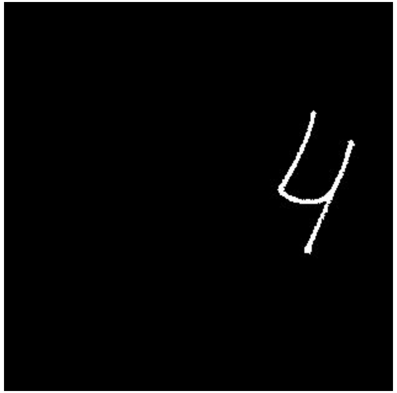
```
coords = np.column_stack(np.where(thresh > 0))
angle = cv2.minAreaRect(coords)[-1]
if angle < -45:
    angle = -angle
else:
    angle = - (90 + angle)
(h, w) = thresh.shape[:2]
center = (w // 2, h // 2)
M = cv2.getRotationMatrix2D(center, angle, 1.0)
rotated = cv2.warpAffine(thresh, M, (w, h),   flags=cv2.INTER_CUBIC, borderMode=cv2.BORDER_REPLICATE)
```

Segmentation



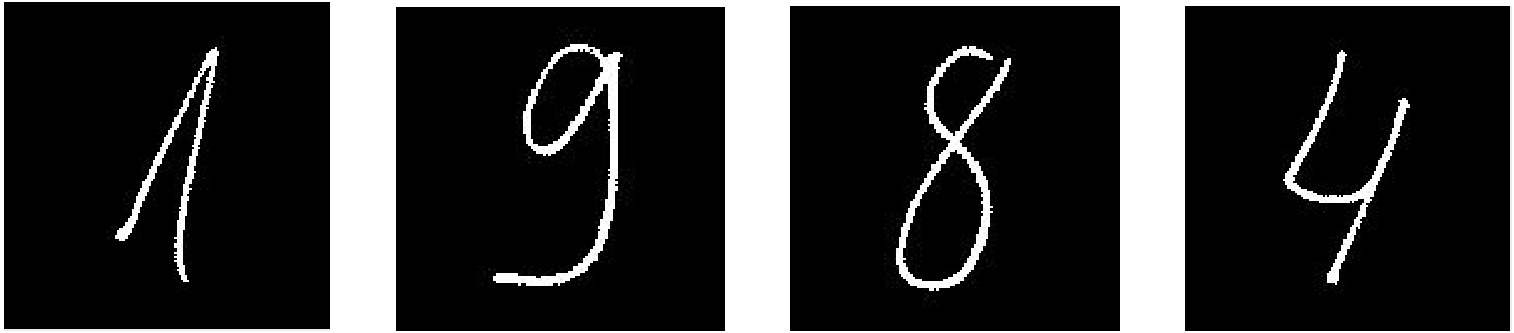






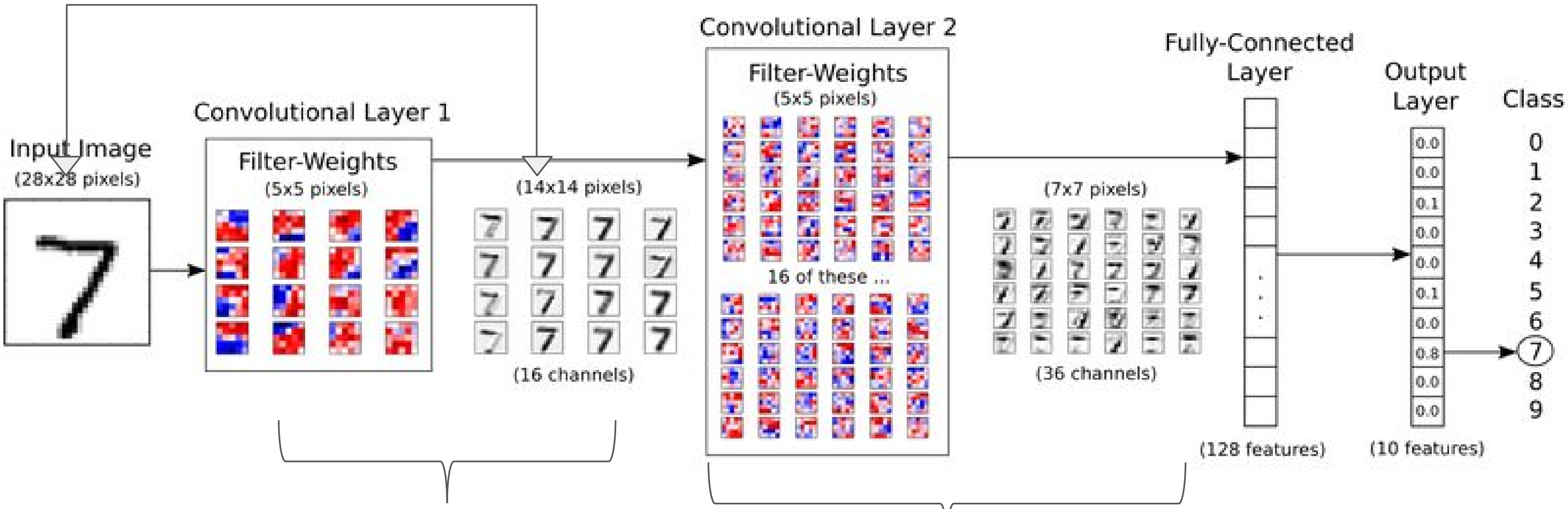
```
ret, markers = cv2.connectedComponents(rotated, connectivity=8)
marker = (markers == i).astype('uint8') * 255
_, contours, hierarchy = cv2.findContours(marker, 2, 2)
cnt = contours[0]
x, y, w, h = cv2.boundingRect(cnt)
```

Center, pad, resize to 28X28, threshold



```
square_edge = np.maximum(h, w)
square_edge = int(square_edge * 1.4)
top_bottom_padding = ((square_edge-h)/2, (square_edge-h)/2)
right_left_padding = ((square_edge-w)/2, (square_edge-w)/2)
only_digit = marker[y:y+h, x:x+w]
only_digit = np.pad(only_digit, [top_bottom_padding, right_left_padding], mode='constant')
resized_image = cv2.resize(only_digit, (28, 28), interpolation = cv2.INTER_AREA)
resized_image = (resized_image > 40).astype('uint8') * 255
```

Max pooling [2X2]->1



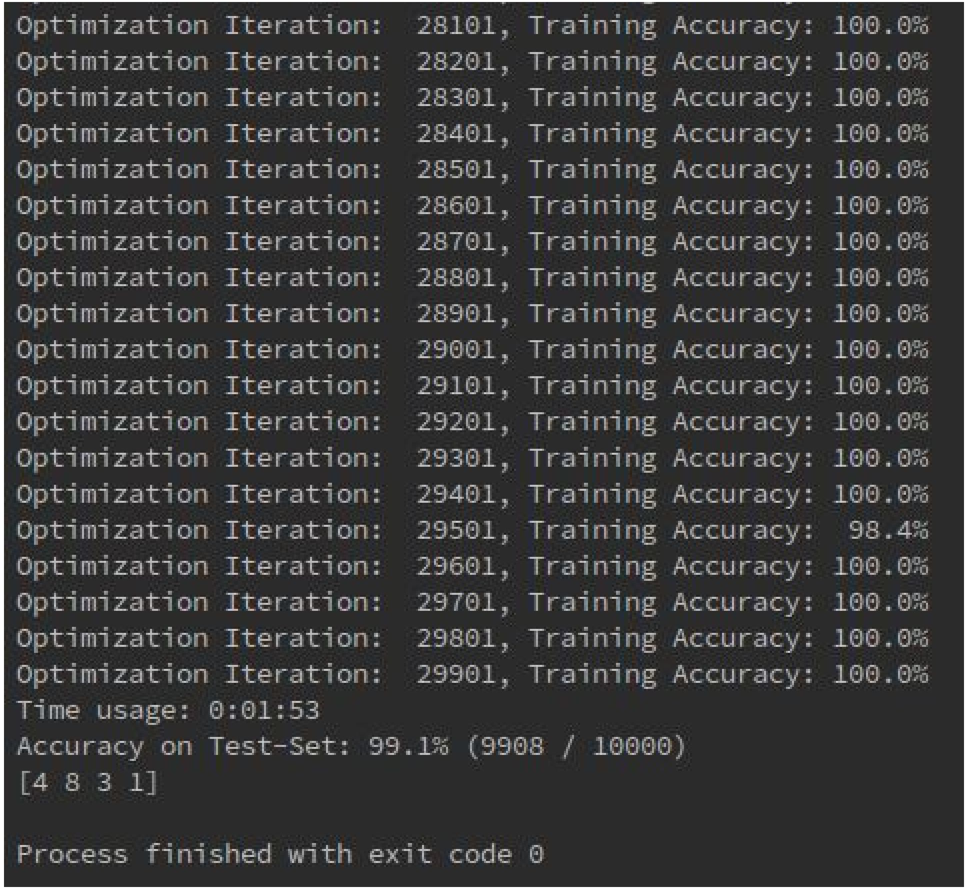
16 filters -> 16 images (channels)
Downsample: Each image 14X14 pixels

16 (images) x 36 (dedicated filters) = 576 filters
Downsample: Each image 7X7 pixels

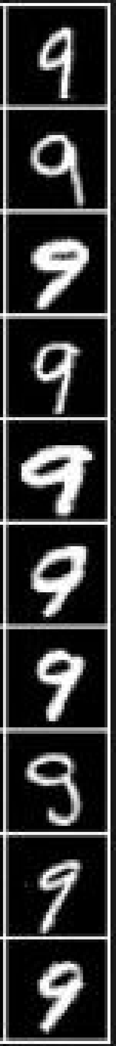
Convolutional Neural Network
Tensor Flow

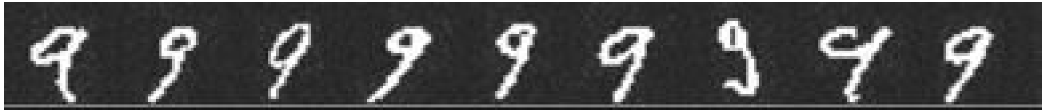
Based on:

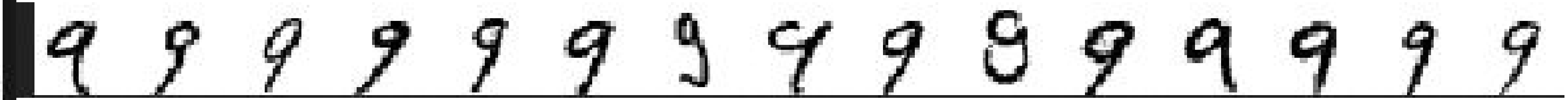
https://github.com/Hvass-Labs/TensorFlow-Tutorials/blob/master/02_Convolutional_Neural_Network.ipynb

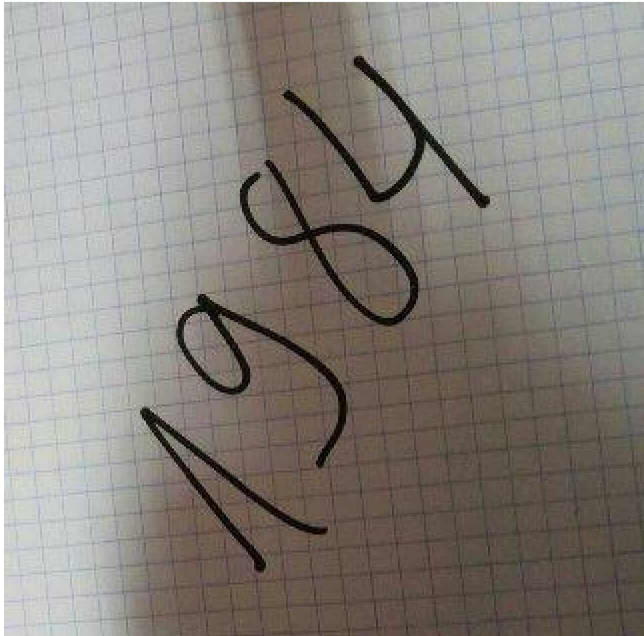


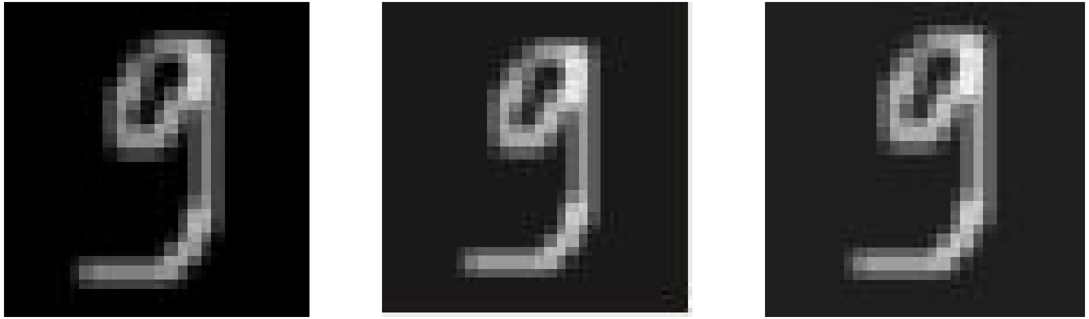
```
Optimization Iteration:  28101, Training Accuracy: 100.0%
Optimization Iteration:  28201, Training Accuracy: 100.0%
Optimization Iteration:  28301, Training Accuracy: 100.0%
Optimization Iteration:  28401, Training Accuracy: 100.0%
Optimization Iteration:  28501, Training Accuracy: 100.0%
Optimization Iteration:  28601, Training Accuracy: 100.0%
Optimization Iteration:  28701, Training Accuracy: 100.0%
Optimization Iteration:  28801, Training Accuracy: 100.0%
Optimization Iteration:  28901, Training Accuracy: 100.0%
Optimization Iteration:  29001, Training Accuracy: 100.0%
Optimization Iteration:  29101, Training Accuracy: 100.0%
Optimization Iteration:  29201, Training Accuracy: 100.0%
Optimization Iteration:  29301, Training Accuracy: 100.0%
Optimization Iteration:  29401, Training Accuracy: 100.0%
Optimization Iteration:  29501, Training Accuracy:  98.4%
Optimization Iteration:  29601, Training Accuracy: 100.0%
Optimization Iteration:  29701, Training Accuracy: 100.0%
Optimization Iteration:  29801, Training Accuracy: 100.0%
Optimization Iteration:  29901, Training Accuracy: 100.0%
Time usage: 0:01:53
Accuracy on Test-Set: 99.1% (9908 / 10000)
[4 8 3 1]

Process finished with exit code 0
```





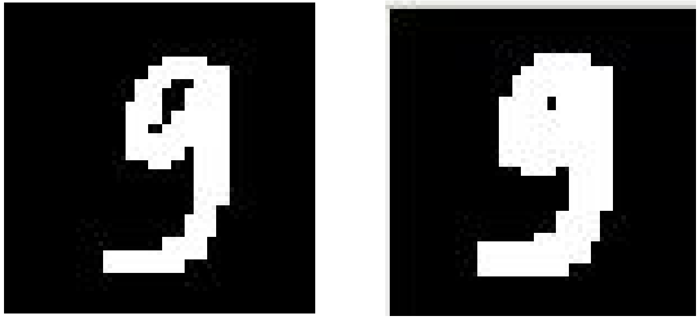



1984

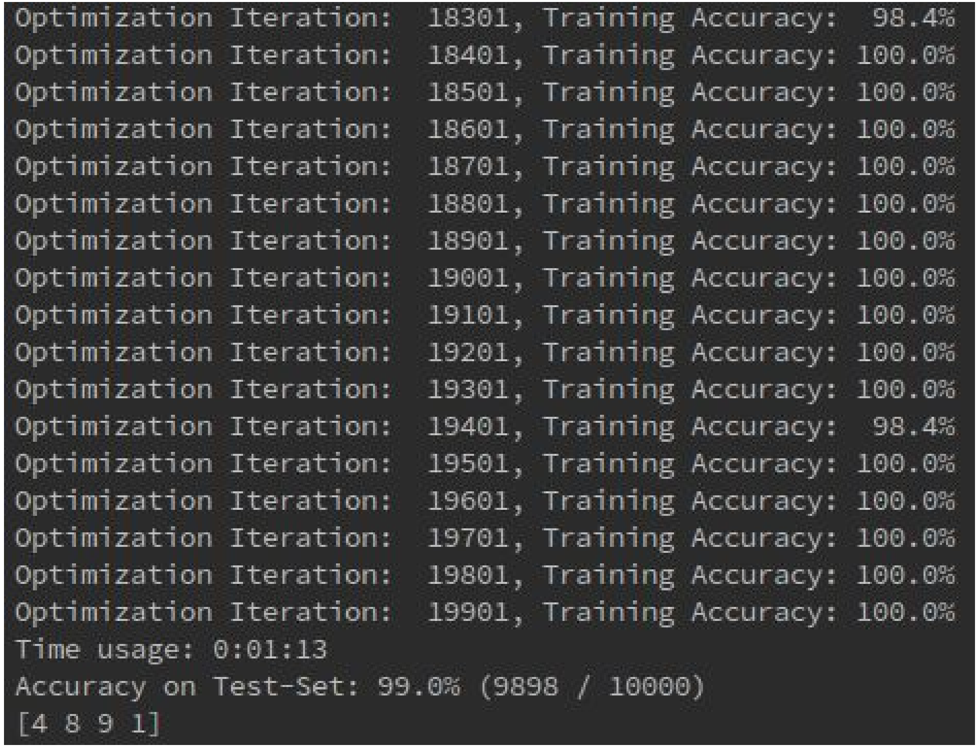


Gaussian Blurring at different levels

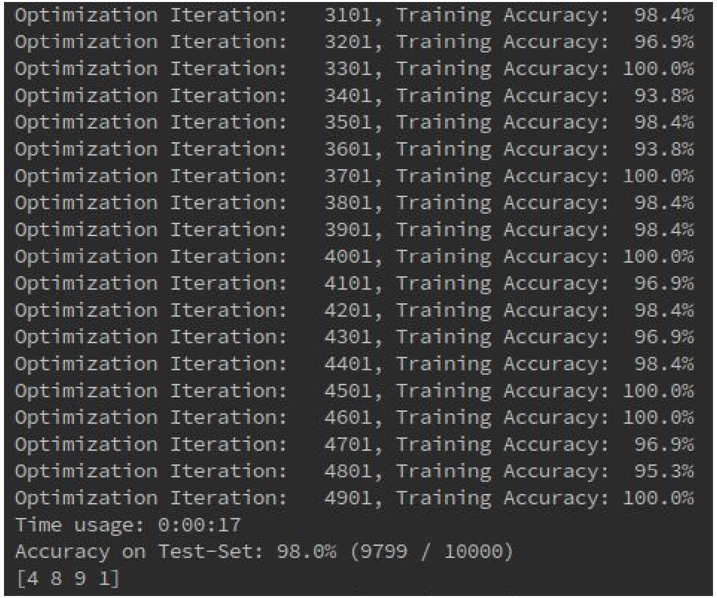
```
resized_image = cv2.GaussianBlur(resized_image, (3, 3), 0)
```



```
kernel = np.ones((2, 2), np.uint8)
resized_image = cv2.dilate(resized_image, kernel, iterations=1)
```



```
Optimization Iteration:  18301, Training Accuracy:  98.4%
Optimization Iteration:  18401, Training Accuracy: 100.0%
Optimization Iteration:  18501, Training Accuracy: 100.0%
Optimization Iteration:  18601, Training Accuracy: 100.0%
Optimization Iteration:  18701, Training Accuracy: 100.0%
Optimization Iteration:  18801, Training Accuracy: 100.0%
Optimization Iteration:  18901, Training Accuracy: 100.0%
Optimization Iteration:  19001, Training Accuracy: 100.0%
Optimization Iteration:  19101, Training Accuracy: 100.0%
Optimization Iteration:  19201, Training Accuracy: 100.0%
Optimization Iteration:  19301, Training Accuracy: 100.0%
Optimization Iteration:  19401, Training Accuracy:  98.4%
Optimization Iteration:  19501, Training Accuracy: 100.0%
Optimization Iteration:  19601, Training Accuracy: 100.0%
Optimization Iteration:  19701, Training Accuracy: 100.0%
Optimization Iteration:  19801, Training Accuracy: 100.0%
Optimization Iteration:  19901, Training Accuracy: 100.0%
Time usage: 0:01:13
Accuracy on Test-Set: 99.0% (9898 / 10000)
[4 8 9 1]
```



```
Optimization Iteration:   3101, Training Accuracy:  98.4%
Optimization Iteration:   3201, Training Accuracy:  96.9%
Optimization Iteration:   3301, Training Accuracy: 100.0%
Optimization Iteration:   3401, Training Accuracy:  93.8%
Optimization Iteration:   3501, Training Accuracy:  98.4%
Optimization Iteration:   3601, Training Accuracy:  93.8%
Optimization Iteration:   3701, Training Accuracy: 100.0%
Optimization Iteration:   3801, Training Accuracy:  98.4%
Optimization Iteration:   3901, Training Accuracy:  98.4%
Optimization Iteration:   4001, Training Accuracy: 100.0%
Optimization Iteration:   4101, Training Accuracy:  96.9%
Optimization Iteration:   4201, Training Accuracy:  98.4%
Optimization Iteration:   4301, Training Accuracy:  96.9%
Optimization Iteration:   4401, Training Accuracy:  98.4%
Optimization Iteration:   4501, Training Accuracy: 100.0%
Optimization Iteration:   4601, Training Accuracy: 100.0%
Optimization Iteration:   4701, Training Accuracy:  96.9%
Optimization Iteration:   4801, Training Accuracy:  95.3%
Optimization Iteration:   4901, Training Accuracy: 100.0%
Time usage: 0:00:17
Accuracy on Test-Set: 98.0% (9799 / 10000)
[4 8 9 1]
```

Extra Info