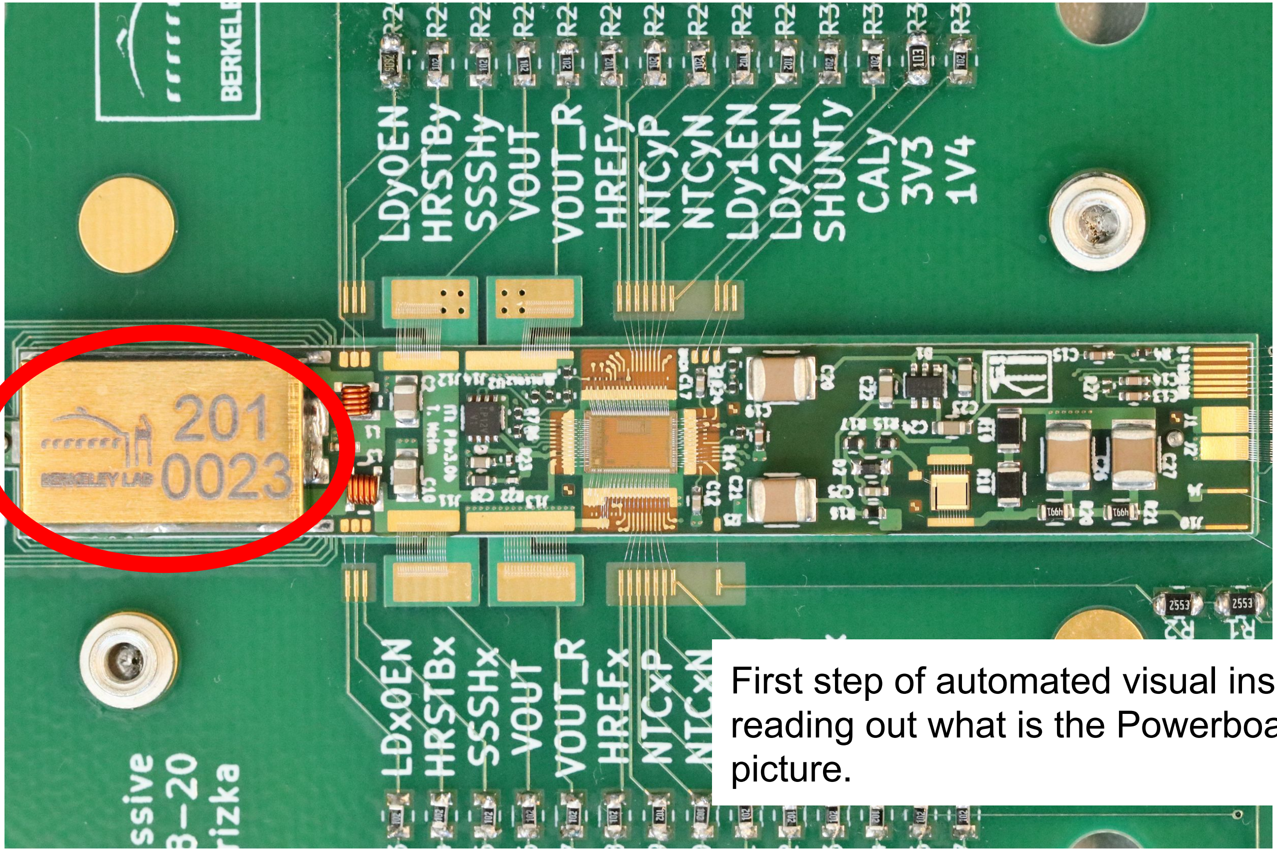
POWERBOARD IDENTIFICATION USING COMPUTER VISION

Ameya Kunder
24th July, 2020

Introduction



Read out the serial number

First step of automated visual inspection is reading out what is the Powerboard in the picture.

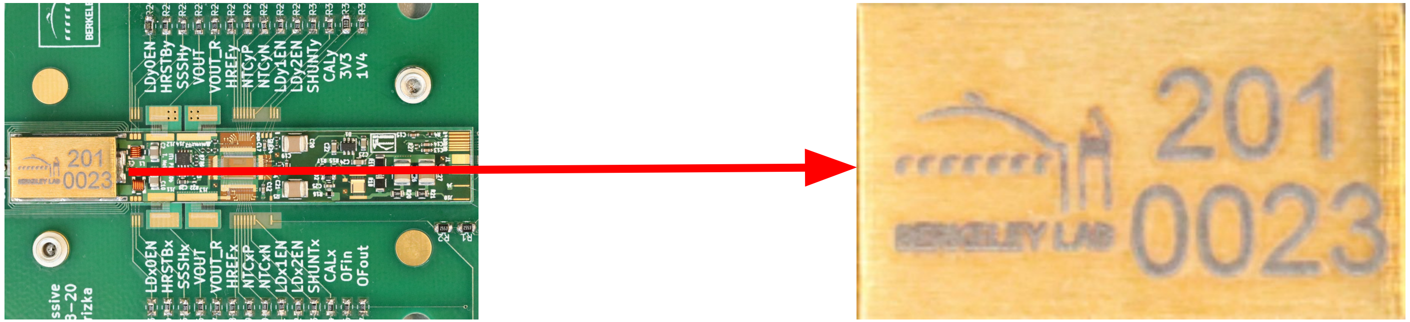
DIGITIZING IDENTIFICATION NUMBER

1. Cropping the etched serial number and logo from the image of a Powerboard and saving in a directory.
2. Applying pre processing methods (e.g.- thresholding) to get a black and white image of the serial number and logo.
3. Using machine learning to recognize the digits of the serial number (main focus of the project is to automate this step as this is the hardest to achieve)

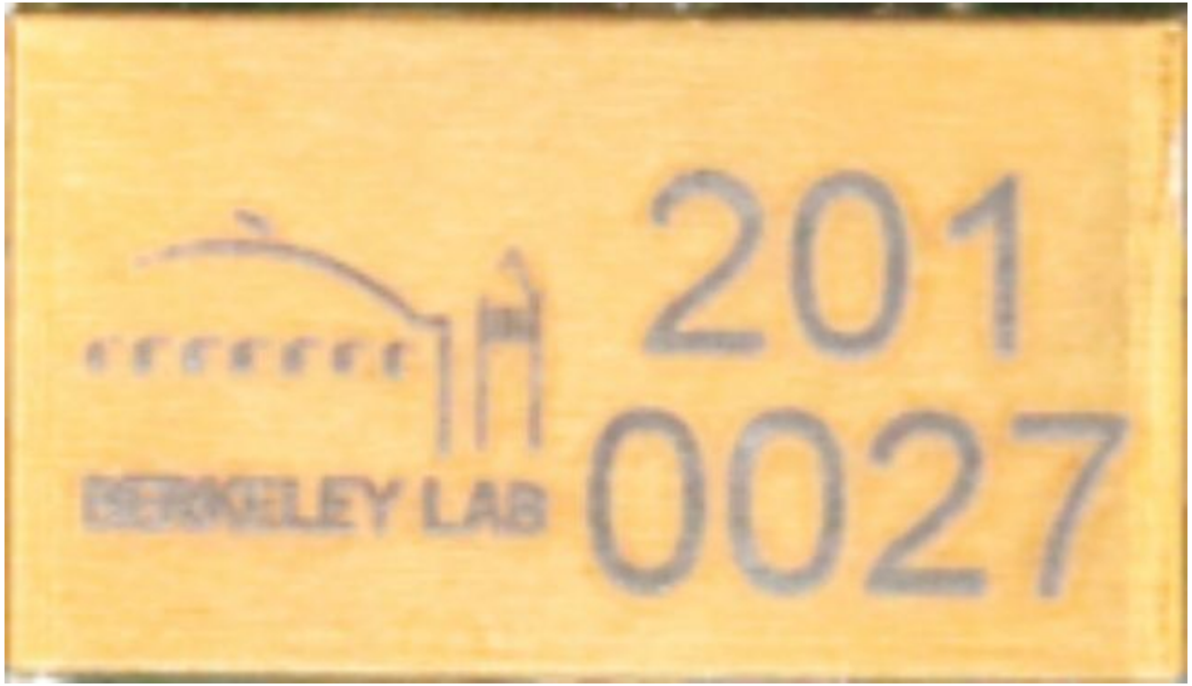
Link to online repository:
https://github.com/a-kunder/powerboard_qc

Code for cropping

1. Reads all the Powerboard images and displays them one after another letting the user crop the desired region.
2. Saves the cropped images in a separate folder which can be used later for post processing.

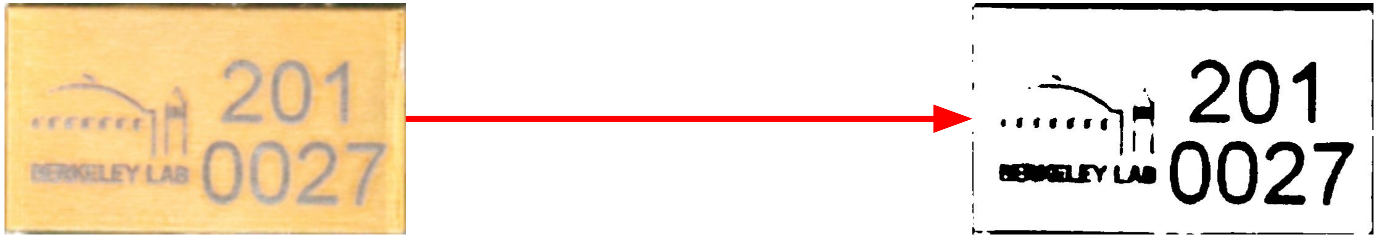


Cropped Image



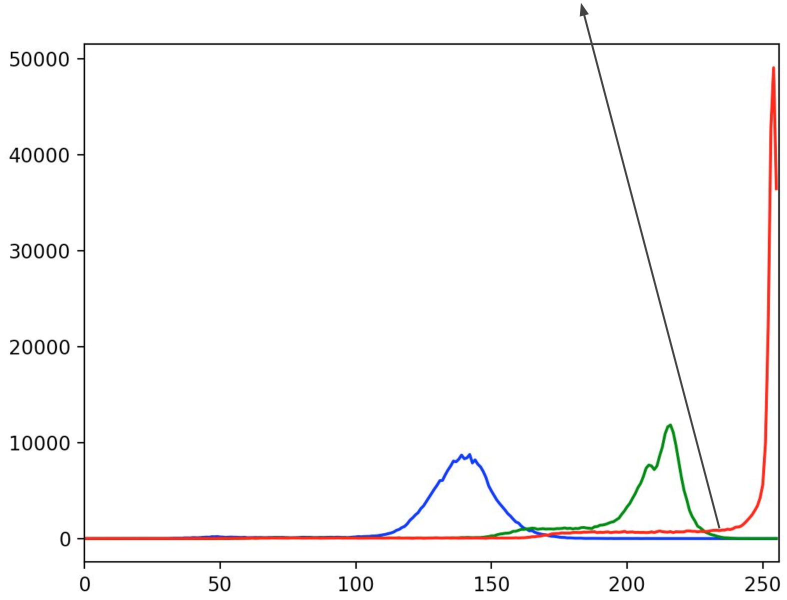
Code for processing

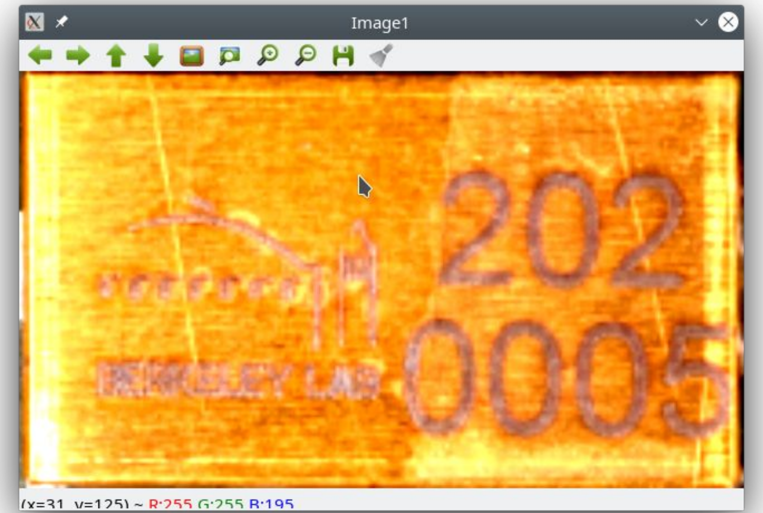
1. Displays the cropped image and lets the user input the channel (blue/green/red) for which the user wants to apply thresholding.
2. Lets the user input a pixel value for the channel that they chose.
3. From what I was able to observe, regular thresholding works for red (every pixel ***above*** the entered value is turned black) and inverse thresholding works for blue and green (every pixel ***below*** the entered value is turned black)



RGB histogram and the tool used for thresholding

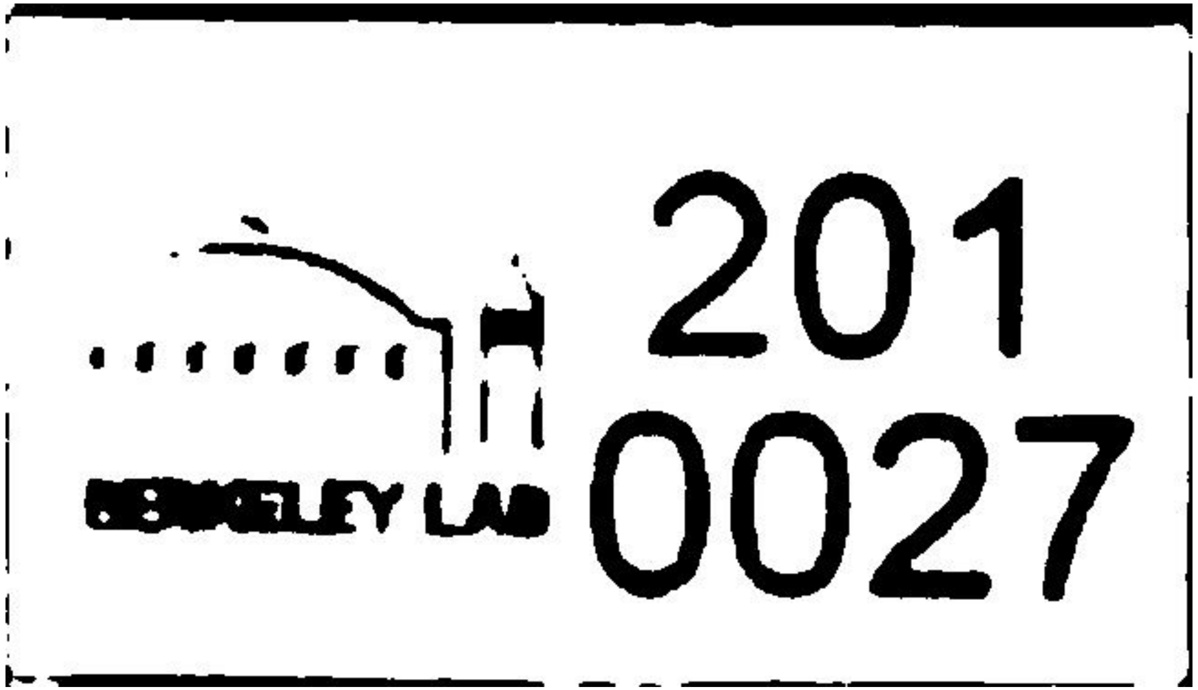
RED value: 230 (threshold)





(x=31, y=125) ~ R:255 G:255 B:195

Pre-processed image

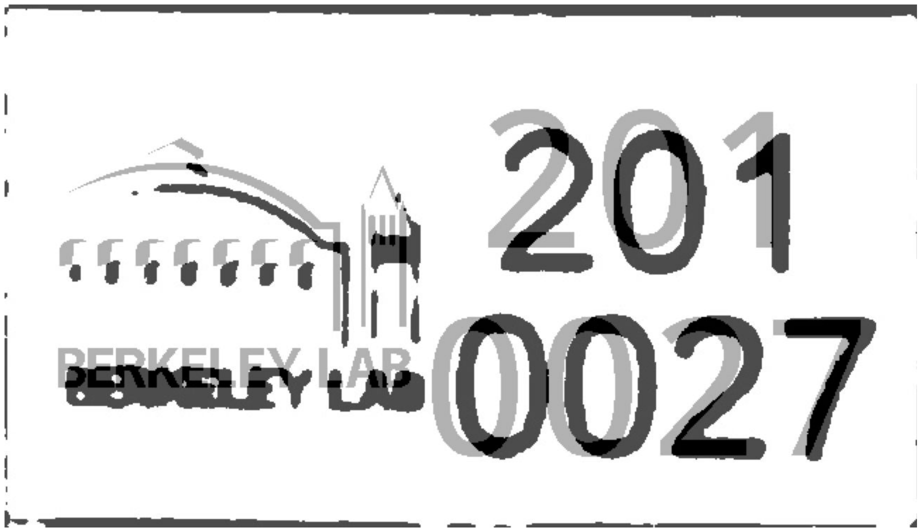


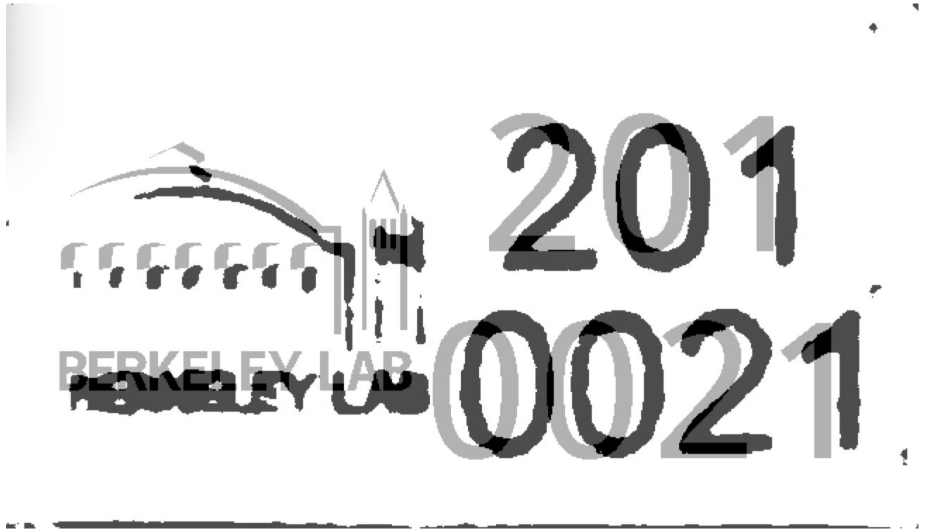
Recognizing the digits

This process can be split into two parts:

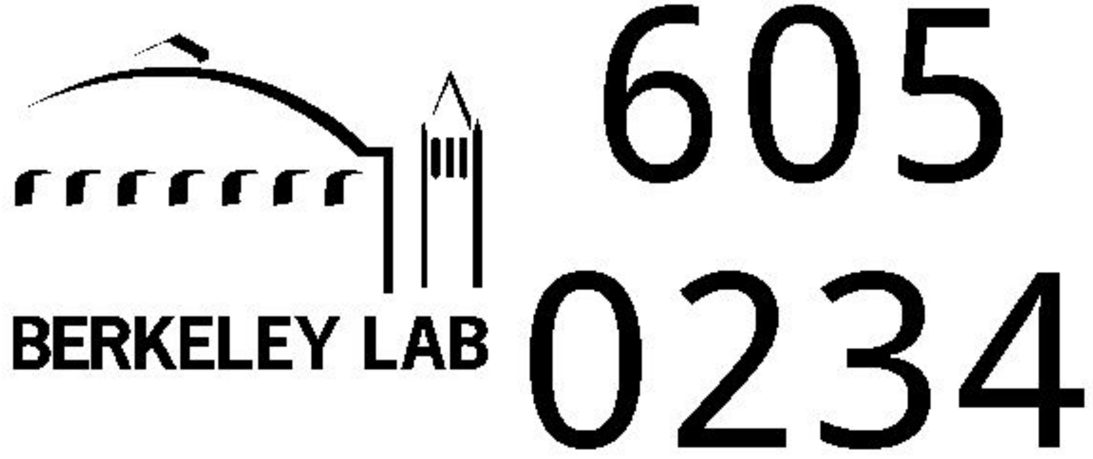
1. Generating images for the training dataset
2. Letting the neural network learn from the images in the training dataset and apply it on the post processed images.

Overlaying processed image from the powerboard and the generated image





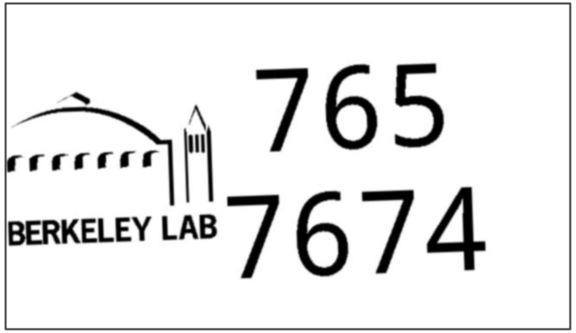
Randomly generated serial numbers

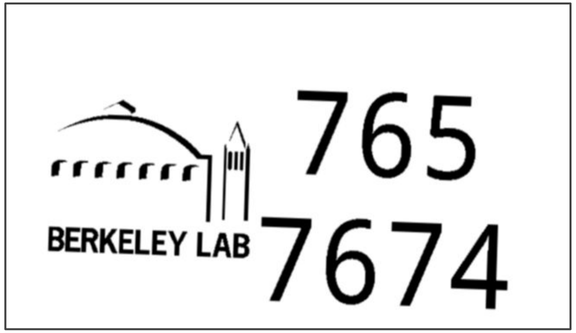


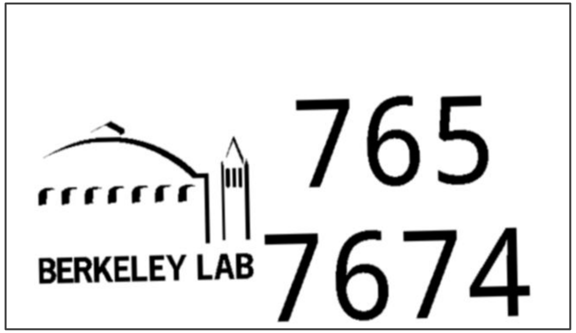
Code for generating labels

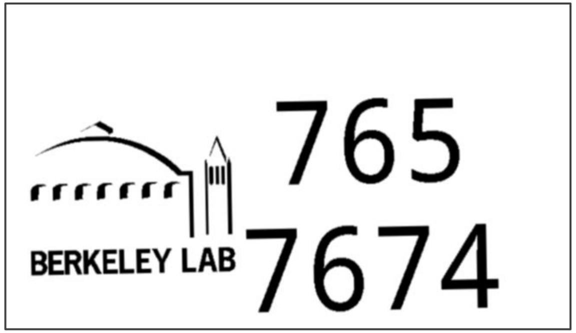
1. Generates random 7 digit serial number with the lab logo
2. Randomly applies Gaussian blur to some images
3. Uses image augmentation to create minor changes to the images such as horizontal/vertical shift, zoom and rotation.
4. Changing fonts to provide more variations in the training dataset.

Augmented images





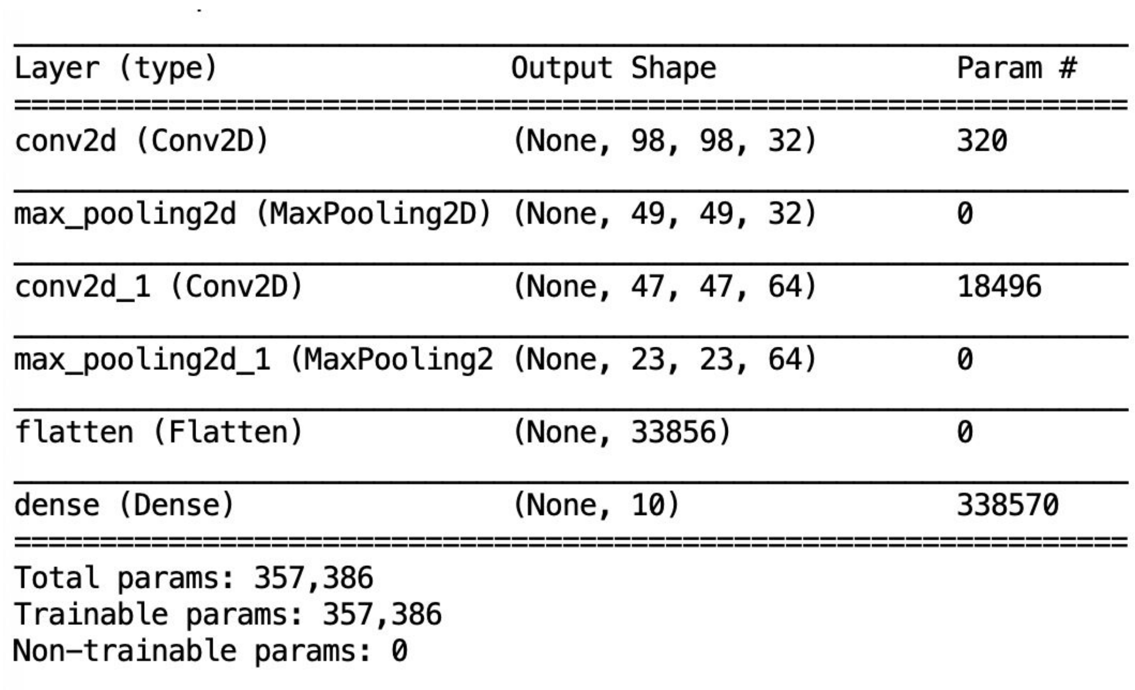




Training and testing

1. We train the model (simple CNN) using the the originally generated images (without augmentation) and the augmented images.
2. We also train the model using different neural networks, varying both in the type of the layers and the number of them.

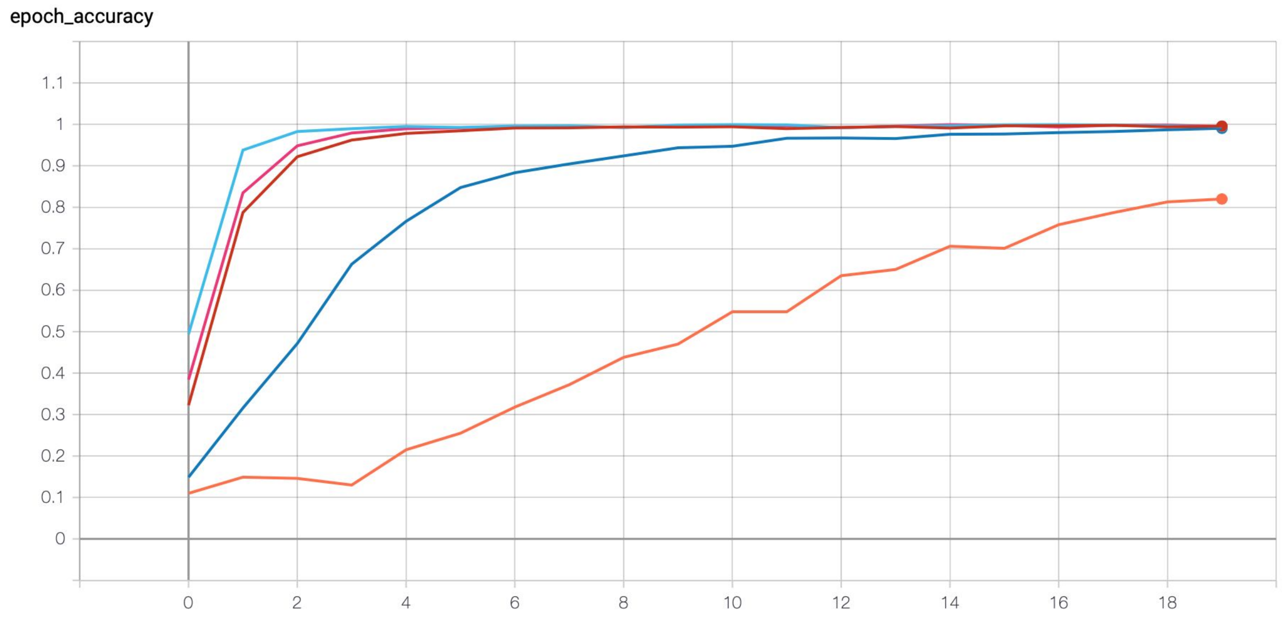
Model Summary



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 98, 98, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 49, 49, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 47, 47, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 23, 23, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 33856)             0
_________________________________________________________________
dense (Dense)                (None, 10)                338570
=================================================================
Total params: 357,386
Trainable params: 357,386
Non-trainable params: 0
```

Two layer CNN

Accuracy vs Epoch (for *epochs* = 20)



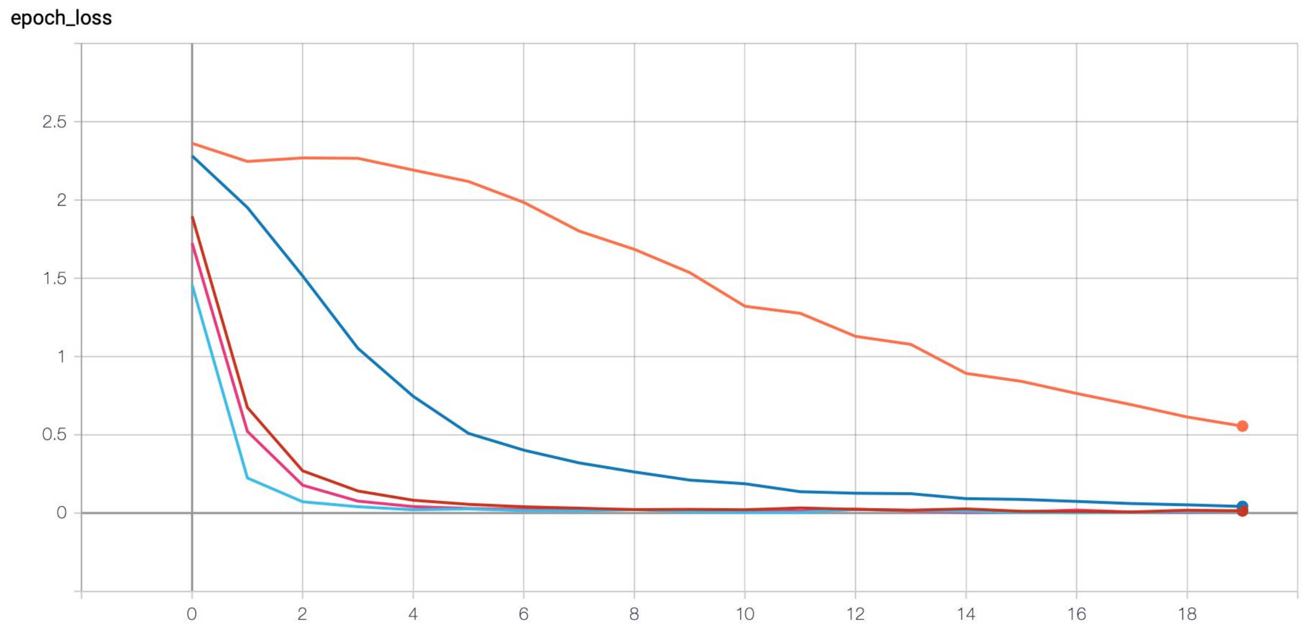
Accuracy for the training dataset (not the testing dataset)

Training data size :

- 1,000 O
- 5,000 O
- 10,000 O
- 15,000 O
- 20,000 O

For final results

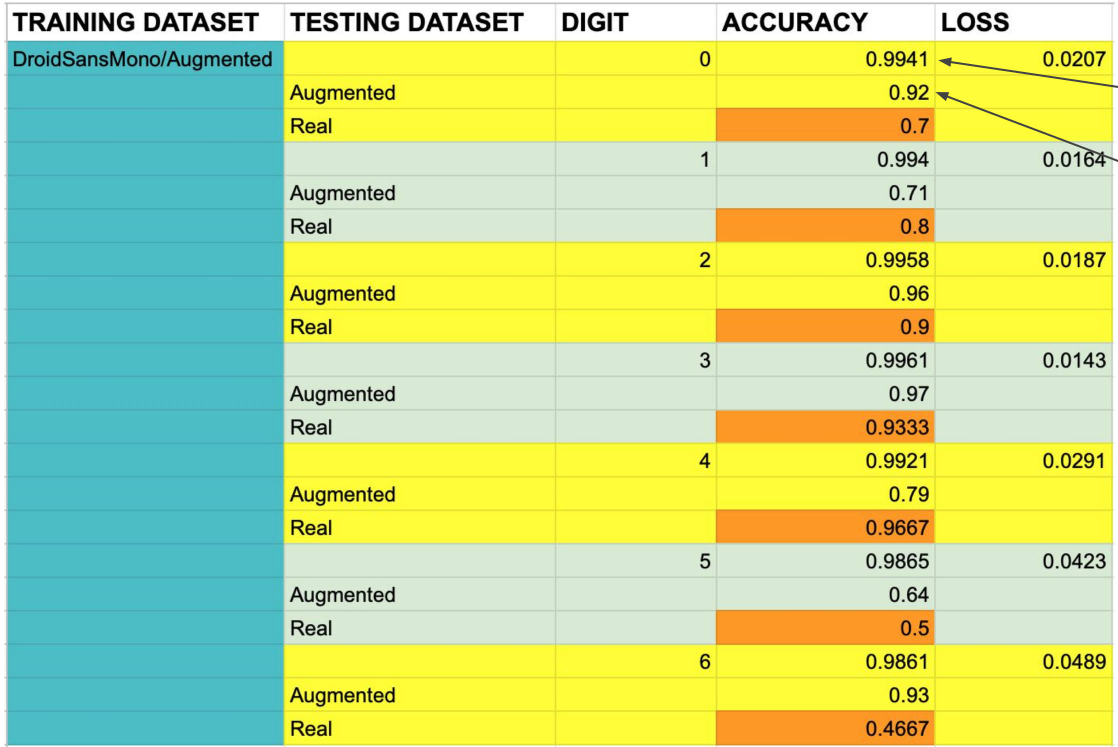
Loss vs Epoch (for *epochs* = 20)



Training data size :

- 1,000 O
- 5,000 O
- 10,000 O
- 15,000 O
- 20,000 O

Table with accuracy and loss for each digit - One font

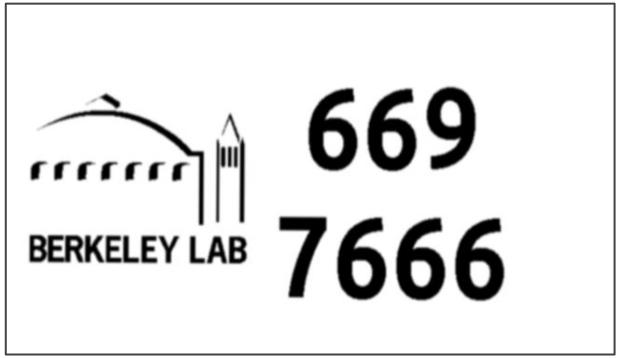


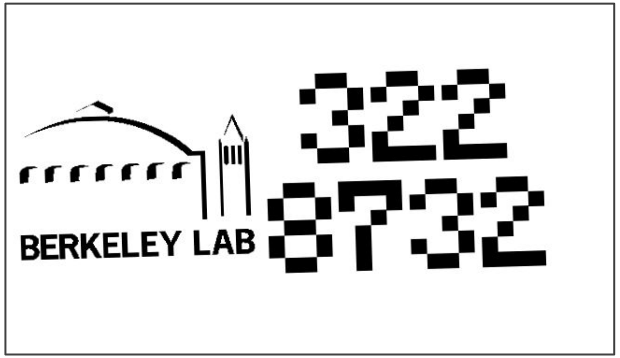
TRAINING DATASET	TESTING DATASET	DIGIT	ACCURACY	LOSS
DroidSansMono/Augmented		0	0.9941	0.0207
	Augmented		0.92	
	Real		0.7	
		1	0.994	0.0164
	Augmented		0.71	
	Real		0.8	
		2	0.9958	0.0187
	Augmented		0.96	
	Real		0.9	
		3	0.9961	0.0143
	Augmented		0.97	
	Real		0.9333	
		4	0.9921	0.0291
	Augmented		0.79	
	Real		0.9667	
		5	0.9865	0.0423
	Augmented		0.64	
	Real		0.5	
		6	0.9861	0.0489
	Augmented		0.93	
	Real		0.4667	

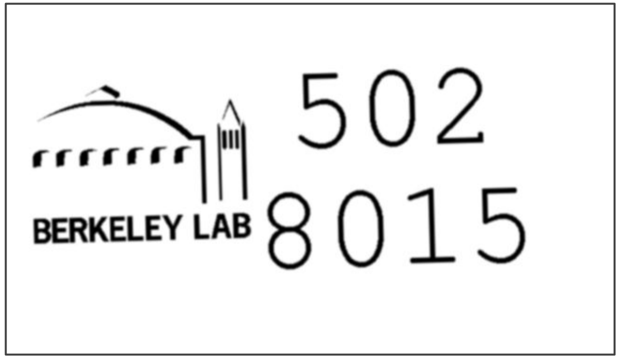
Training Dataset

Testing Dataset

Augmented images - other fonts







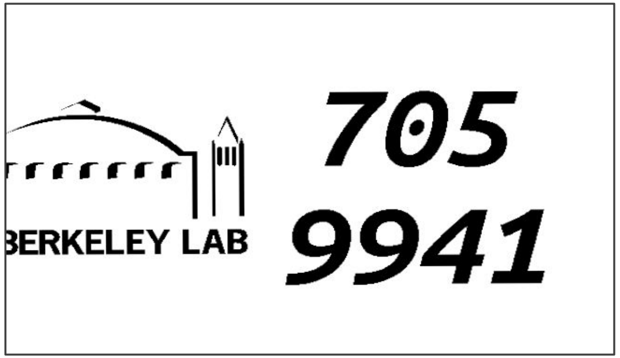
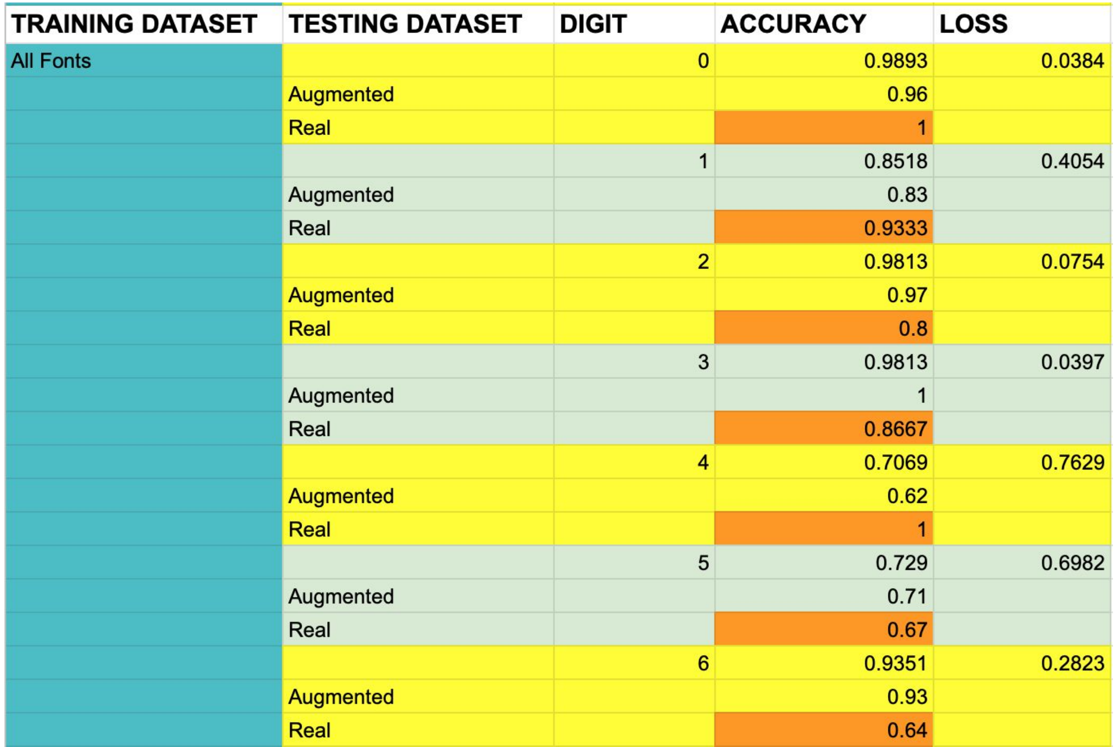
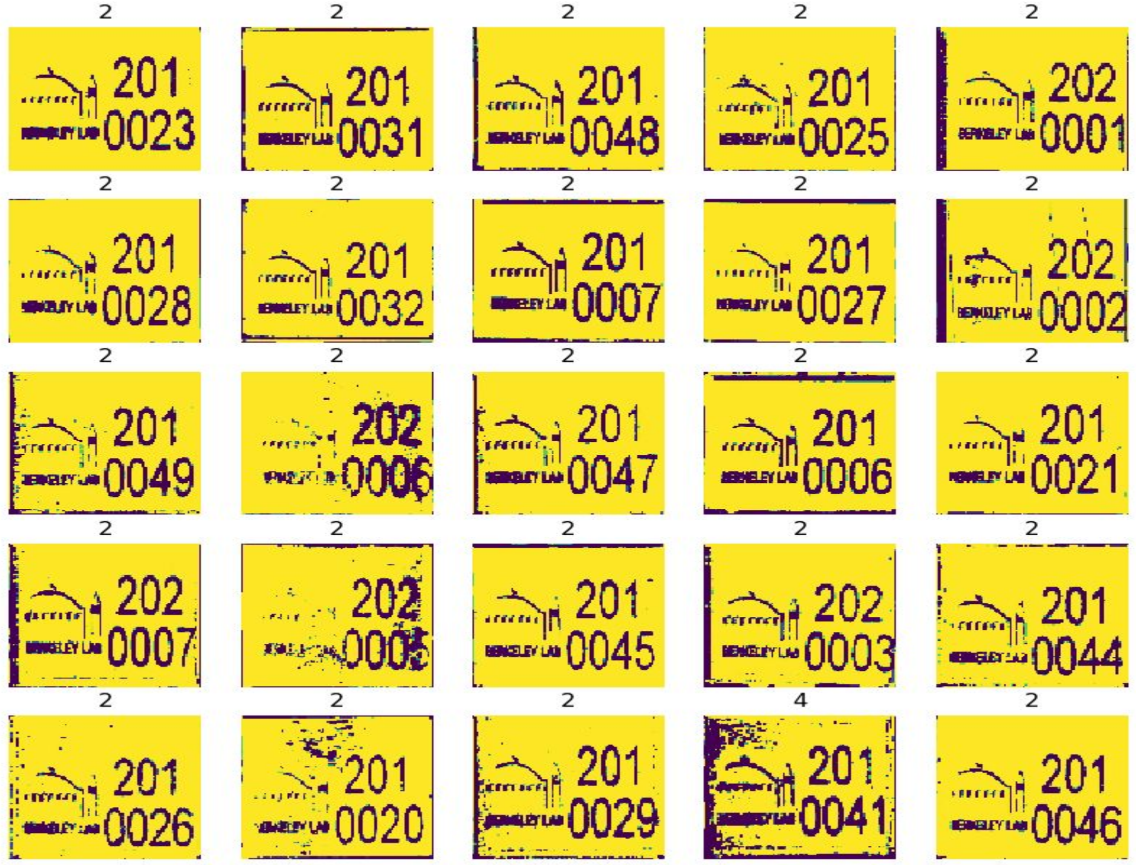


Table with accuracy and loss for each digit - Multiple fonts

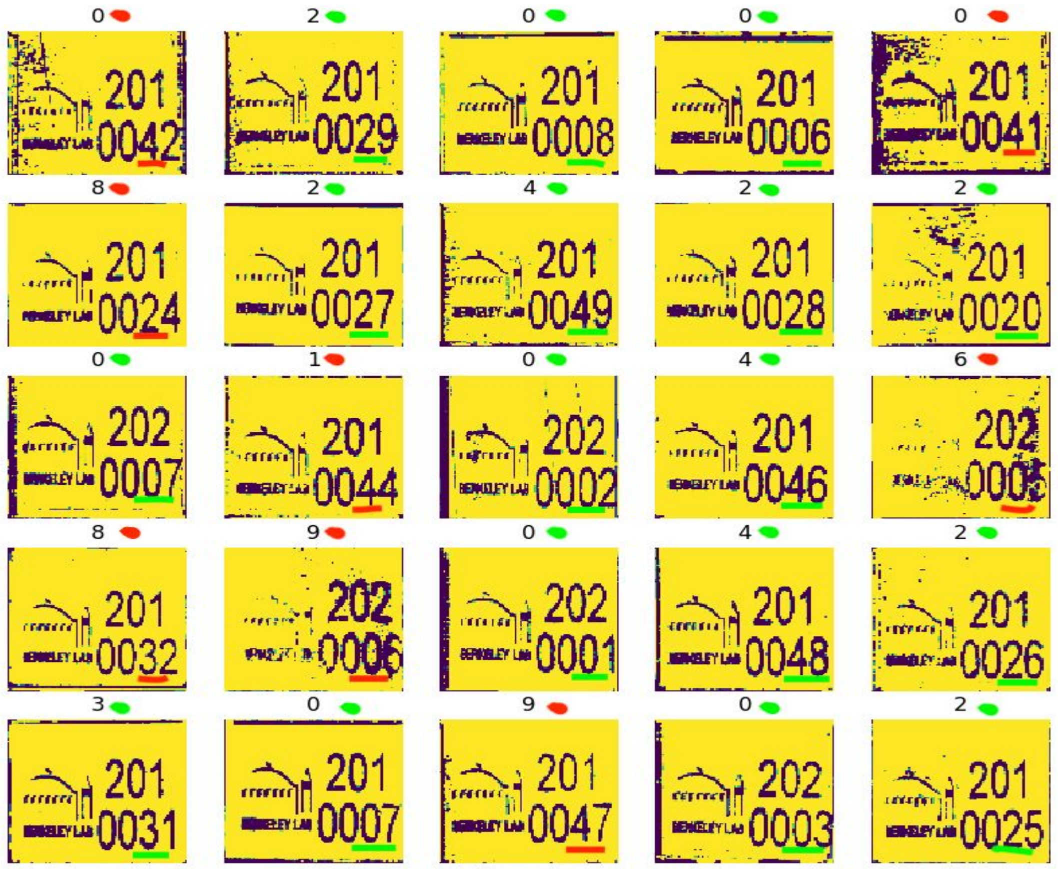


TRAINING DATASET	TESTING DATASET	DIGIT	ACCURACY	LOSS
All Fonts		0	0.9893	0.0384
	Augmented		0.96	
	Real		1	
		1	0.8518	0.4054
	Augmented		0.83	
	Real		0.9333	
		2	0.9813	0.0754
	Augmented		0.97	
	Real		0.8	
		3	0.9813	0.0397
	Augmented		1	
	Real		0.8667	
		4	0.7069	0.7629
	Augmented		0.62	
	Real		1	
		5	0.729	0.6982
	Augmented		0.71	
	Real		0.67	
		6	0.9351	0.2823
	Augmented		0.93	
	Real		0.64	



Testing for Digit 0:

~ 100%



Testing for Digit 5:

~ 70%

Conclusion

We have a system setup for training and testing a model for recognizing digits from real images. It's working better than random, but still not perfect. That's what you need for production system.

Next steps:

- Play around with size of neutral network.
- Remove Berkeley logo from the images to simplify input vectors
- Try to improve thresholding algorithms to better remove noise
- Identify when the NN prediction is unsure (look at score) and notify user that manual check might be needed.
- Suggestions welcome!

Thank you!