Image Classification and Unsupervised Image Object Removal in the Street View House Numbers Dataset

Group 7 Final Project

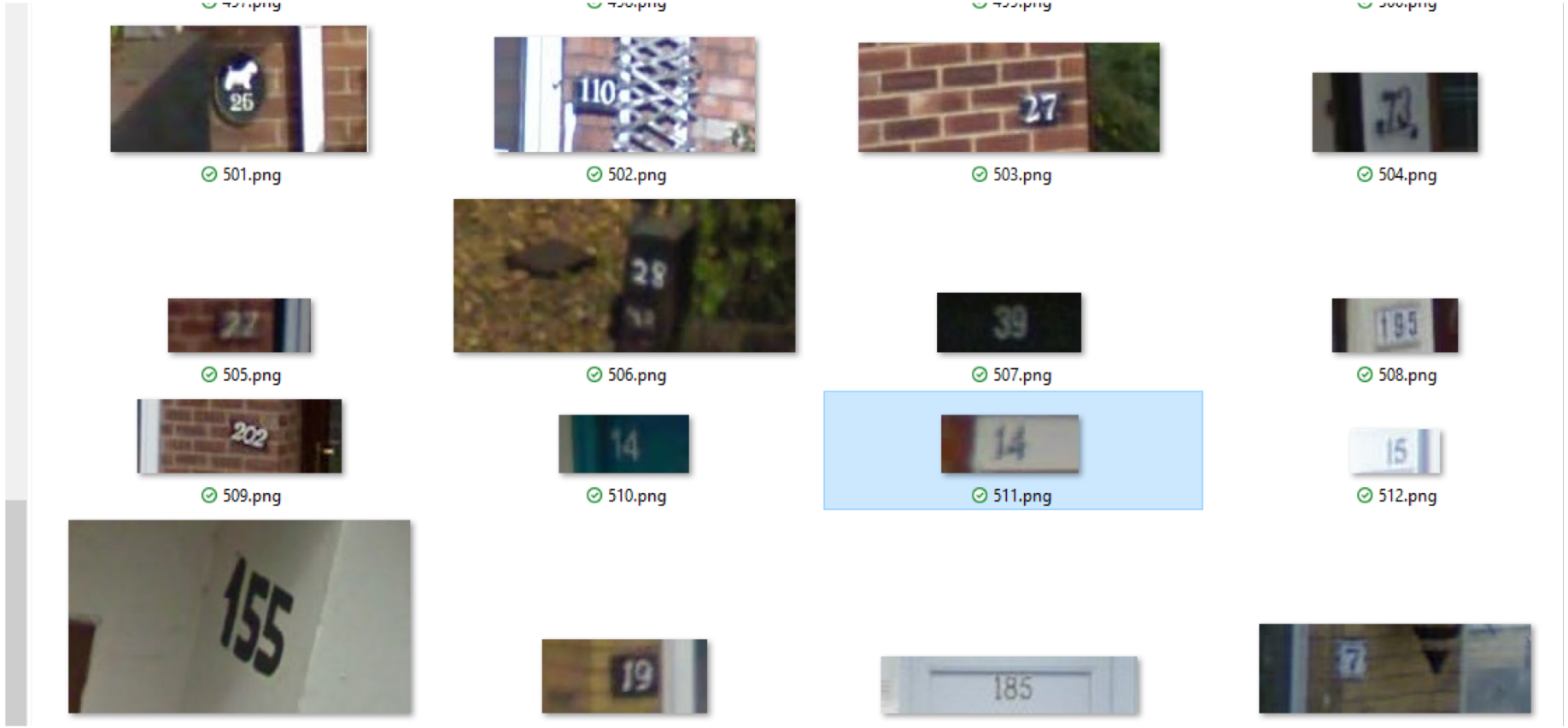
Bill Grieser -- Darshan Kasat -- Shivam Thassu

The Project

1. Use the Street View House Numbers dataset to classify images from the Street View House Numbers dataset
2. Explore using PyTorch and the by-products of an image classification network to remove digits from the test data used in 1)

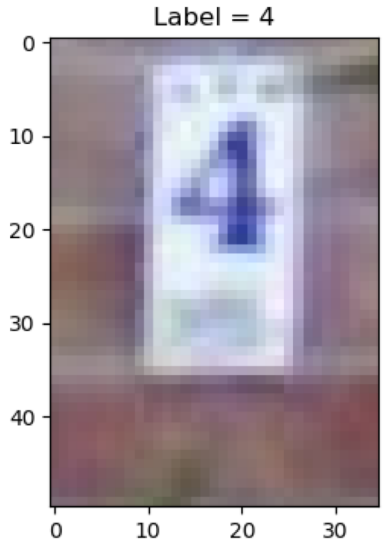
The Data

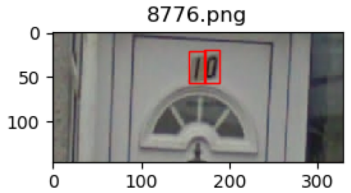
Images from Google Street View that contain digits

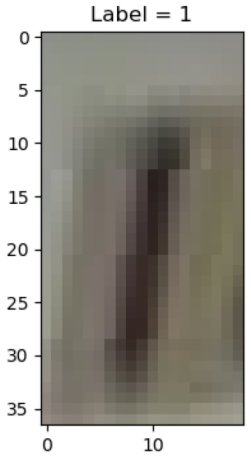


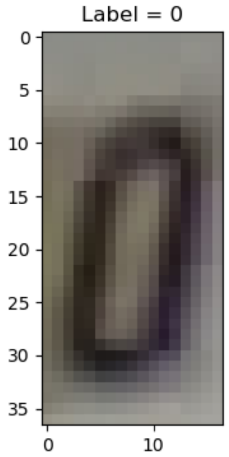
Data Components

- PNG file per parent image
- Bounding Box co-ordinates to identify digits in parent image
- Label (0..9)









Available in two different formats:

Format 1: Original images with bounding box available for each character (may contain multiple characters in same images).

Format 2: MNIST like 32x32 cropped images having single character in each image.

Tasks

- Classify digit from image
  - TensorFlow
  - Caffe
  - PyTorch
- Remove digit from image (experimental)
  - PyTorch

TensorFlow

About TensorFlow

- TensorFlow is an open source software library for numerical computation using dataflow graphs. Nodes in the graph represents mathematical operations, while graph edges represent multi-dimensional data arrays (aka tensors) communicated between them.
- TensorFlow as nothing but numpy with a twist. A major difference between numpy and TensorFlow is that TensorFlow follows a lazy programming paradigm. The advantages of using TensorFlow are:

    It has an intuitive construct, because as the name suggests it has “flow of tensors”. You can easily visualize each part of the graph.

    Easily train on CPU/GPU for distributed computing

    Platform flexibility. You can run the models wherever you want, whether it is on mobile, server or PC.

PREPROCESSING

- In this dataset all digits have been resized to a fixed resolution of 32-by-32 pixels.
- Initial Exploratory Analysis
- Created a Balanced (Startified) 13% of data in Validation Set.

  Splitting to 13% in Val Set as it gives around 9500 data having minimum of 800 instances of each class.

```
('Training Set', (63733, 32, 32, 1))
('Validation Set', (9524, 32, 32, 1))
('Test Set', (26032, 32, 32, 1))
```

- Converting the Label 10's to 0's

  So, we got a target label of [0 1 2 3 4 5 6 7 8 9]

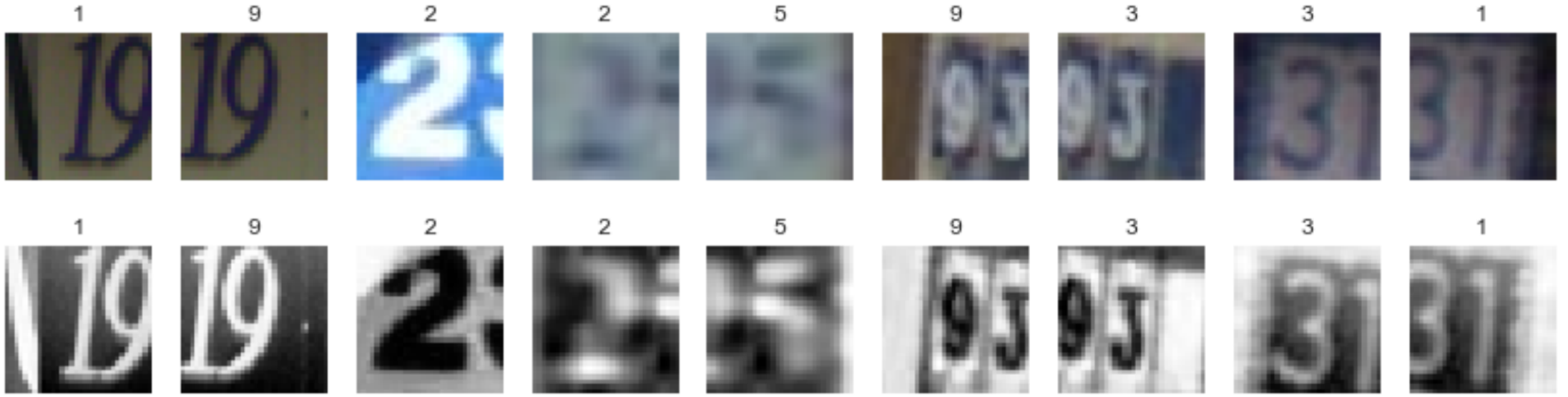
- RGB to Grayscale

  Since we are not concerned about the color of the image. Increase the computation

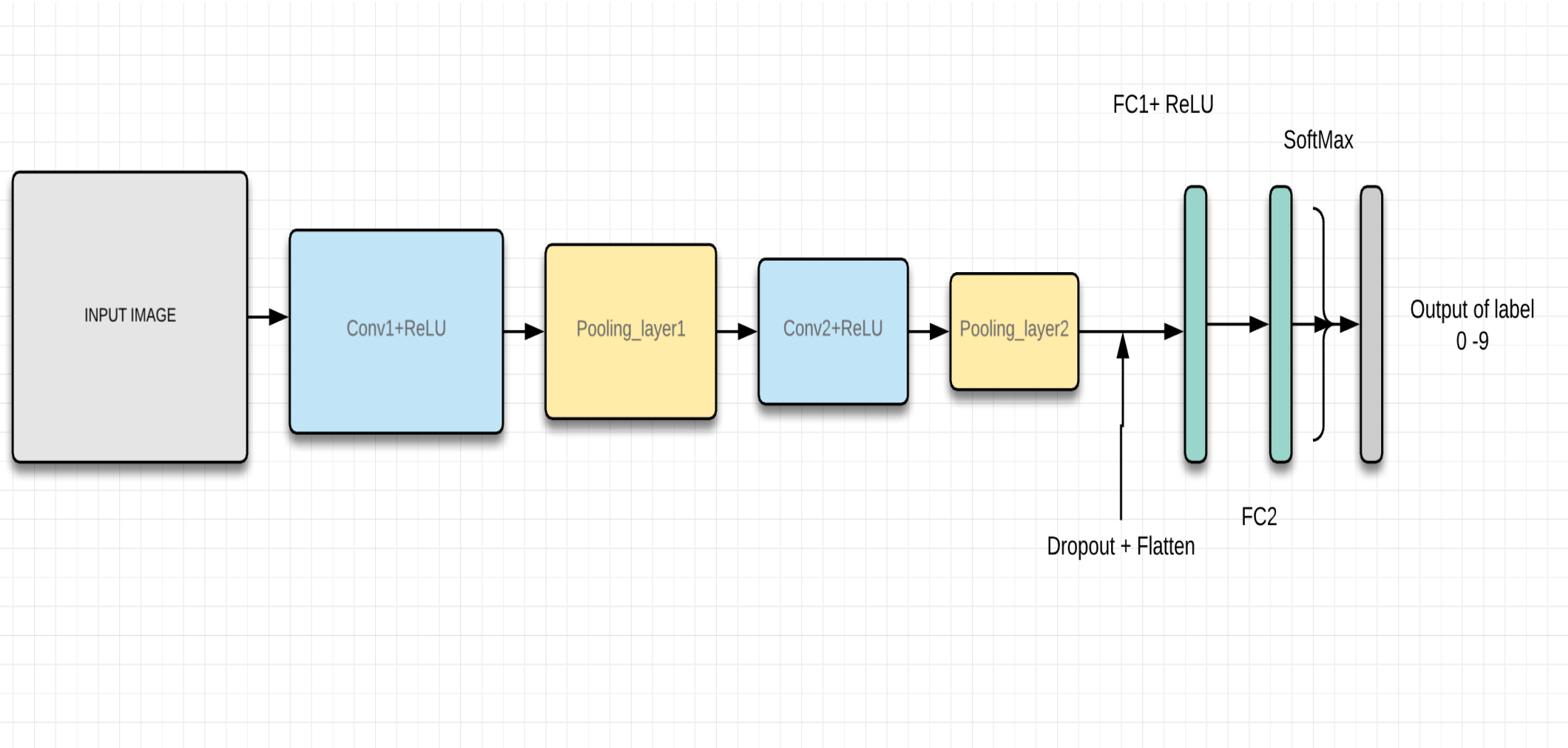
- Normalization & Storing to Disk

  Used h5py package to store the numerical data, so that it can be easily manipulated using NumPy.

IMAGE SAMPLE



MODEL ARCHITECTURE



Bit Info…

- ***Softmax***

The second fully-connected layer estimates how likely it is that the input image belongs to each of the 10 classes. However, these estimates are a bit rough and difficult to interpret because the numbers may be very small or large, so we want to normalize them so that each element is limited between zero and one and the 10 elements sum to one. This is calculated using the so-called softmax function and the result is stored in y_pred.

- ***Calculate Cross-entropy***

To make the model better at classifying the input images, we must somehow change the variables for all the network layers. Comparing the predicted output of the model y_pred to the desired output y_true.

- ***Optimization Method: Adam***

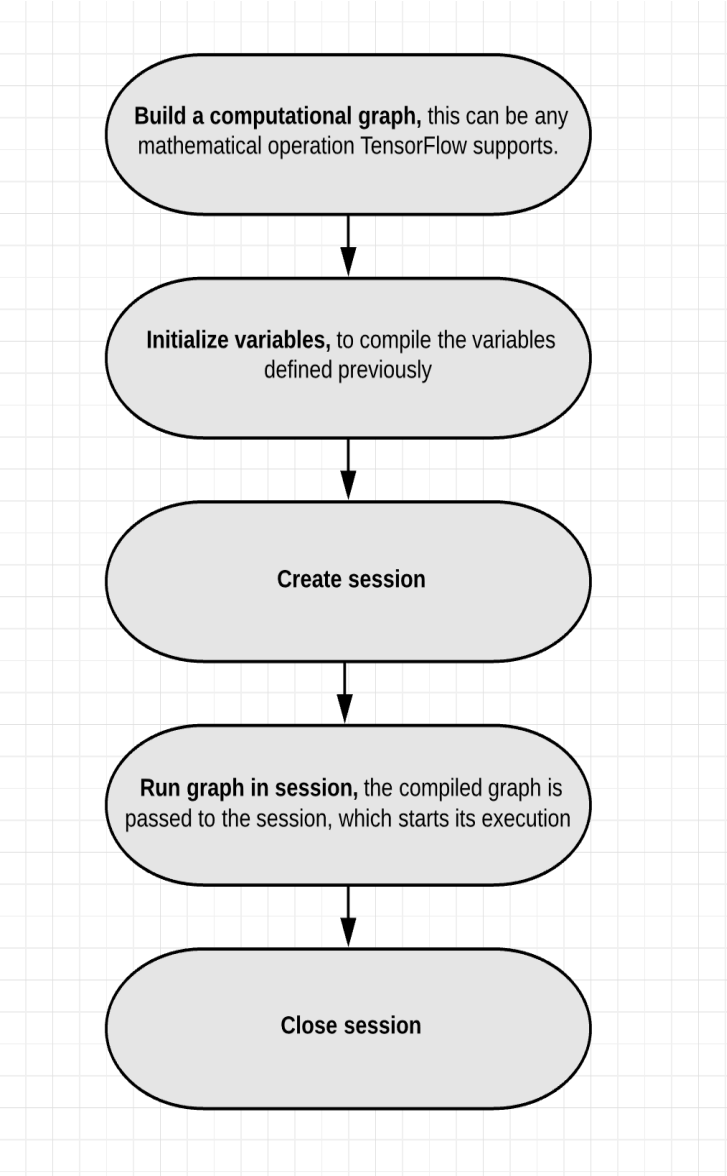
Through experimentation we found out Adam outperformed other optimizers for this problem in terms of convergence speed.

Parameters

Layer	Description
Input Layer	We input a batch of 64 images (64, 32, 32 ,1)
Conv1+ReLU1	32 Filters, 5x5 Kernal size, Stride 1, Zero Padding
Pooling1	2x2, Stride 2
Conv2+ReLU2	64 Filters, 5x5 Kernal size, Stride 1, Zero Padding
Pooling2	2x2, Stride 2
Dropout	0.5
FC	256 nodes

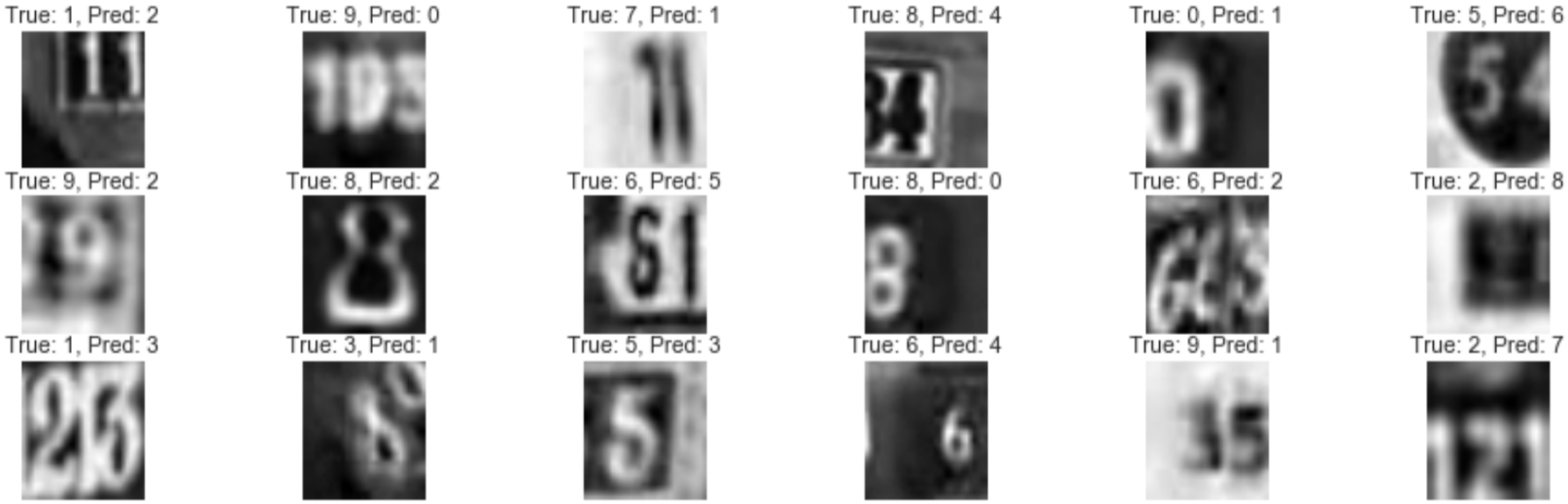
FRAMEWORK WORKFLOW

- write functions for creating new TF variables
- define Placeholder Variables. They serve as the input to the graph that we may change each time we execute the graph
- In order to save the variables of the neural network created saver object which is used for storing and retrieving all the variables of the TensorFlow graph. The saved files are often called checkpoints



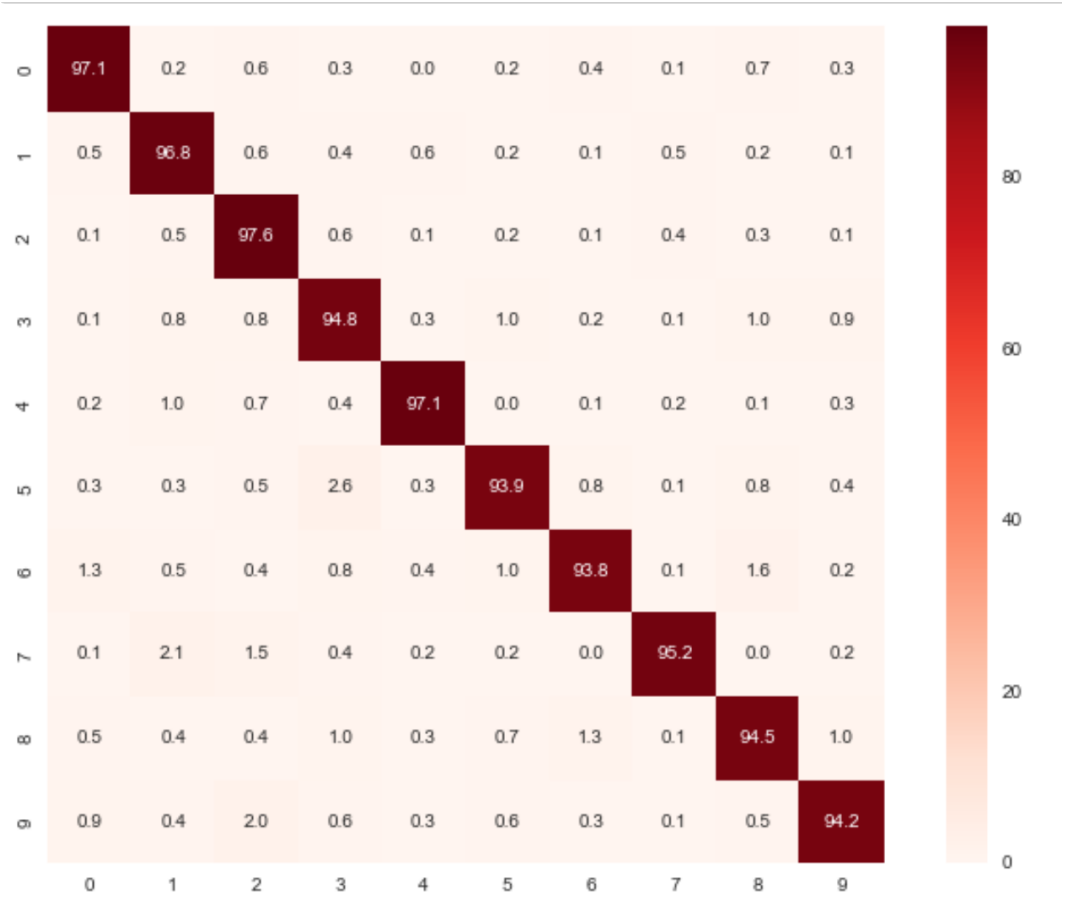
RESULTS

- With the mentioned Architecture I got Test accuracy of **92.96 %** at 50000 iterations.
- I had some of the misclassified images which I found so I dig in deeper to see how good the model works



CONFUSION MATRIX

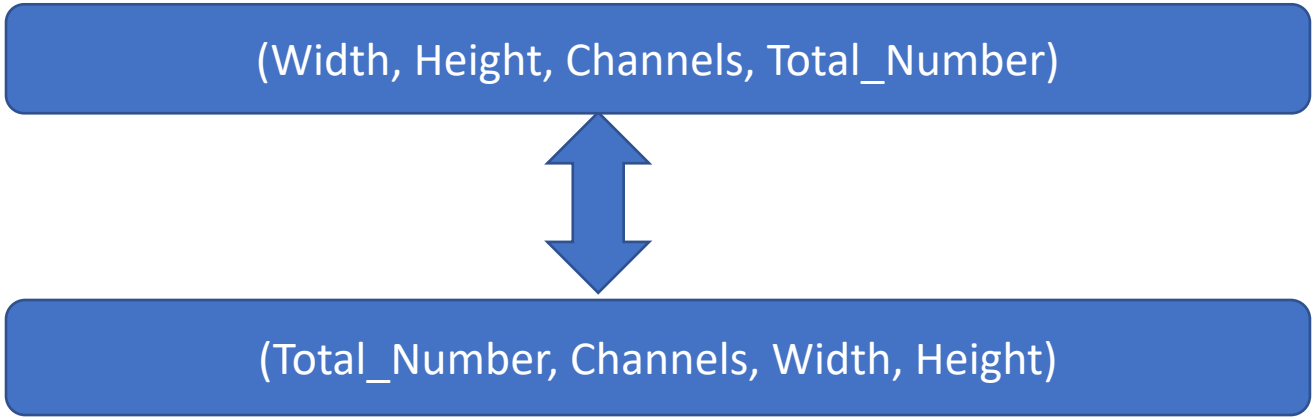
- All digits have accuracy > 90%
- Can say the model does good Classification.




Caffe

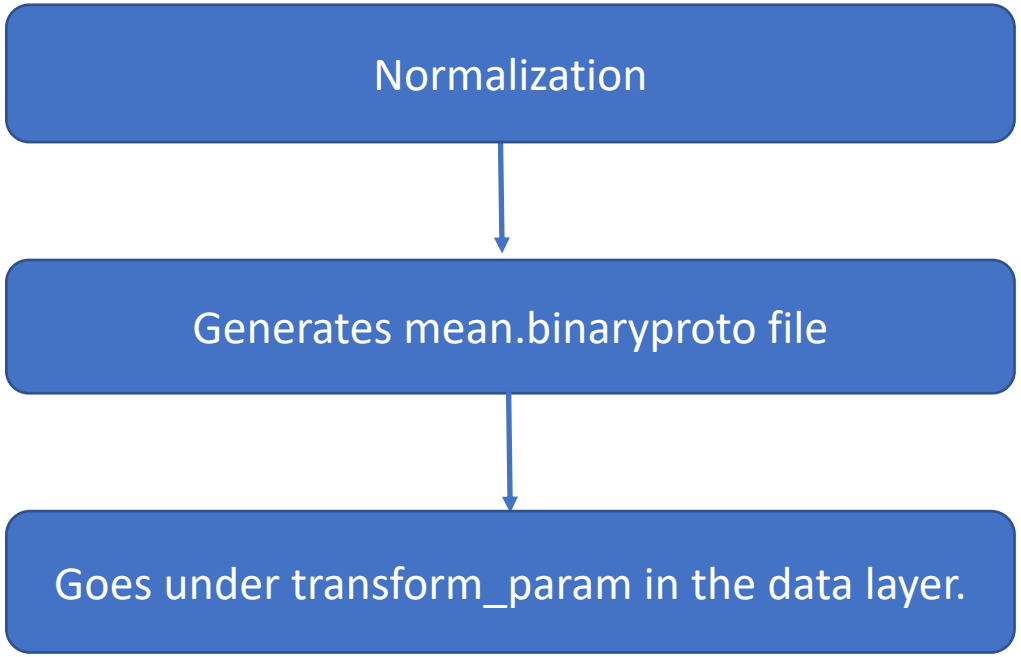
Data Preprocessing

- Caffe works on Lightning Memory-Mapped Database in the back end.
- Data was available in .mat file.
- I used scipy to read the data from mat file
- Data in the form of 4D array:



Data Preprocessing

- Digit "0" was labeled as "10"



CNN Architecture

- The architecture comprises of two conv-relu-pooling-BN layers. The third layer is conv-relu-pooling. It is further connected to a fully connected layer with dropout.
- Network parameters:

  The architecture is a 32-32-64 architecture.

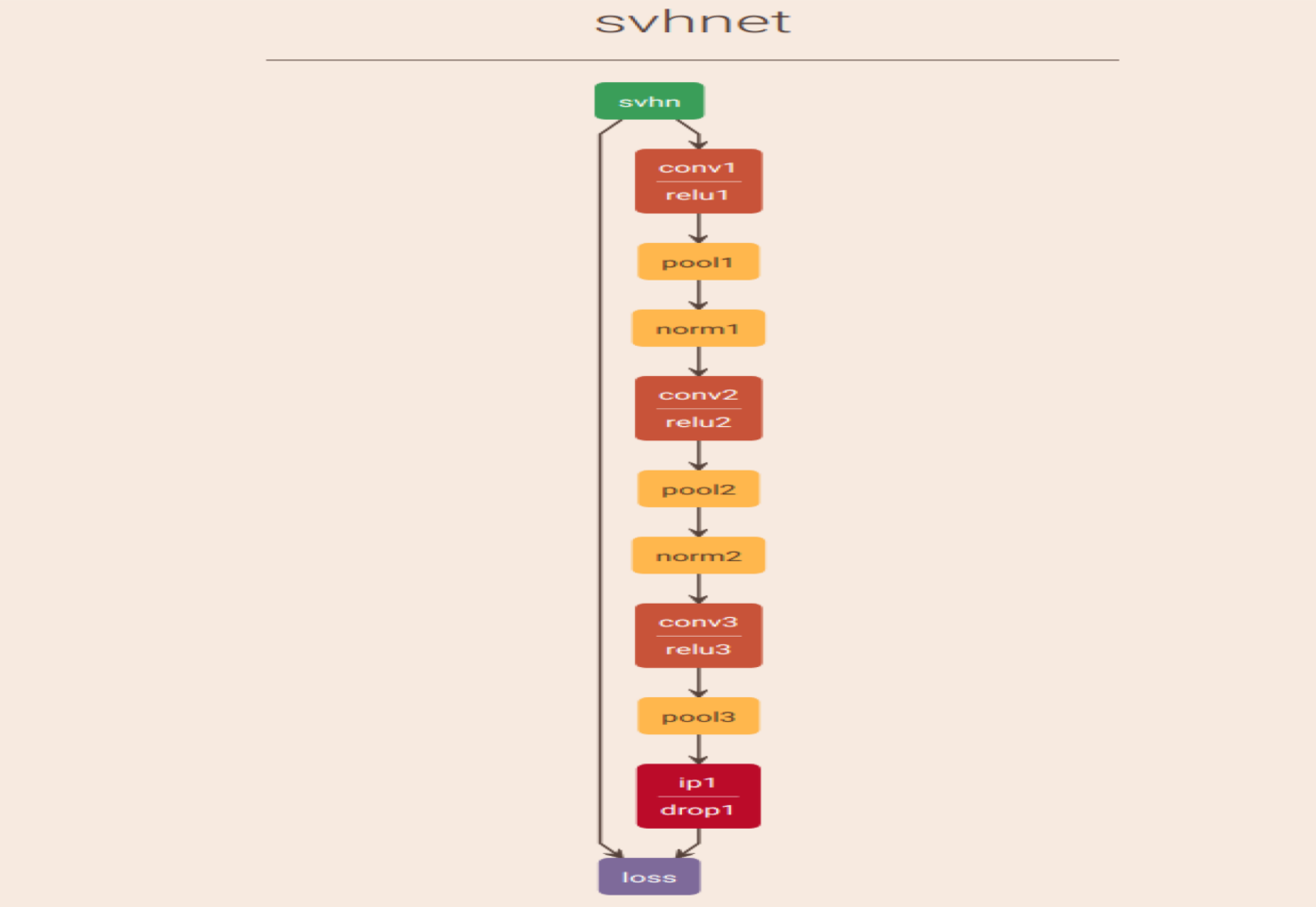
  Kernel size: 5

  Batch_size: 48

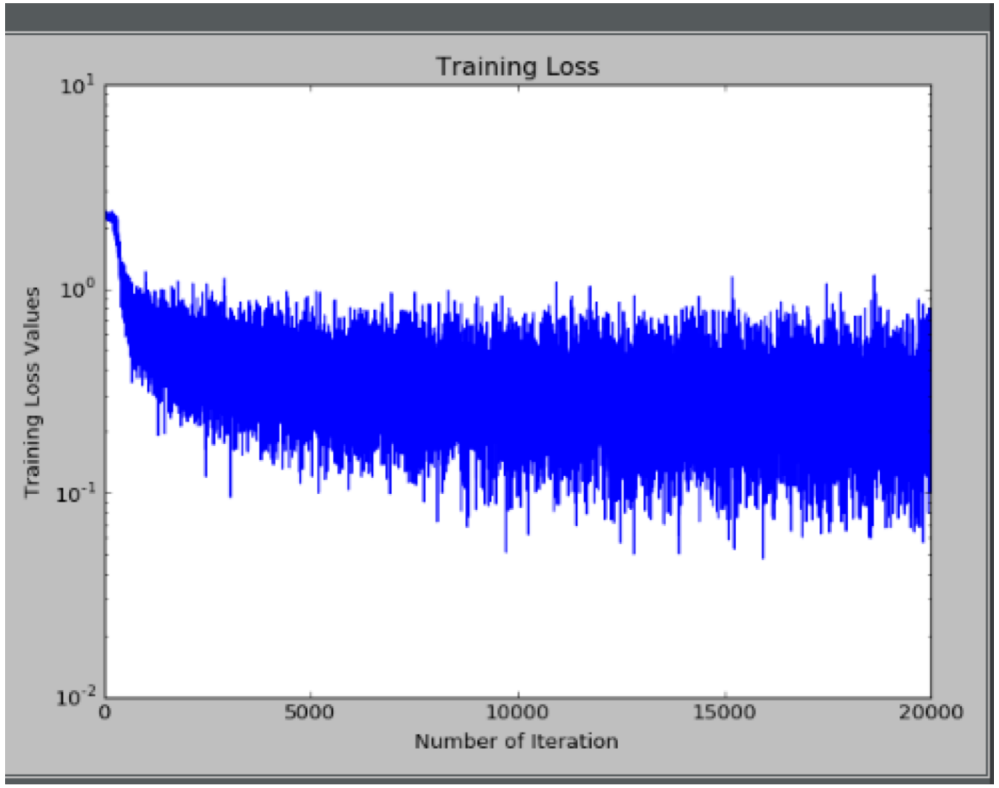
  Dropout: 0.5

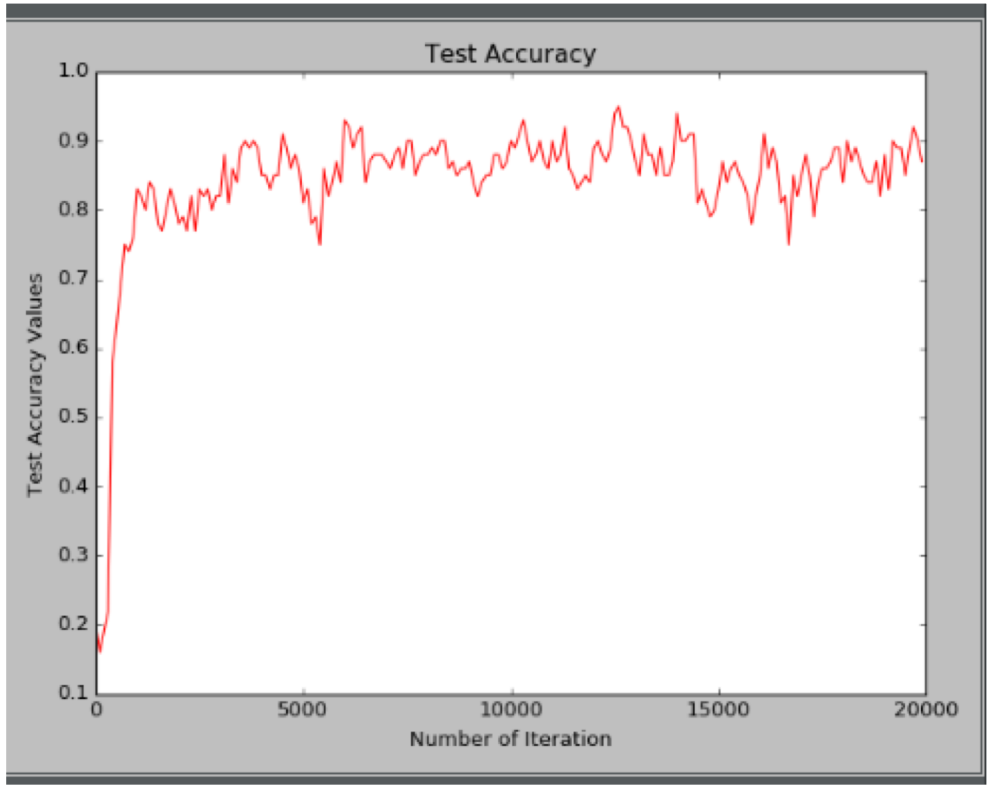
  Optimizer: Adam

CNN Architecture



Results





The model achieved the accuracy of 91%.

Image Classification with PyTorch

- Preprocessing:
  - Resize to 40 x 40
  - Grayscale
- Algorithm:
  - Convolutional Neural Network with 3 Convolution-BatchNorm-Relu-Maxpool, layers, Dropout, and Fully Connected Layer
  - Kernel size 3x3
  - Number of kernels: 48 – 64 – 32
  - Dropout p=0.50
- Optimizer: SGD
- Batch Size: 32
- Learning rate: 0.005

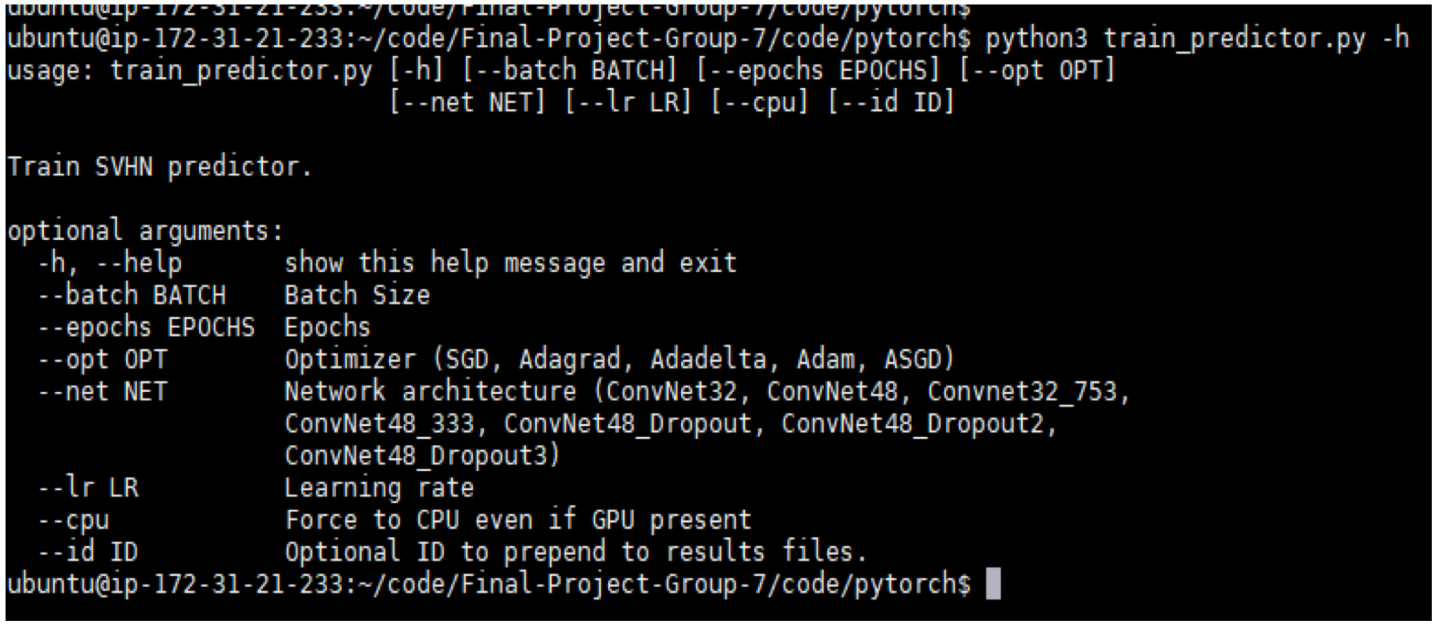
The PyTorch Model

```
279 class ConvNet48_Dropout3(BaseNet):
280     def __init__(self, num_classes, channels, image_size):
281         super(ConvNet48_Dropout3, self).__init__(num_classes, channels, image_size)
282         self.layer1 = nn.Sequential(
283             nn.Conv2d(1, 48, kernel_size=3, padding=1),
284             nn.BatchNorm2d(48),
285             nn.ReLU(),
286             nn.MaxPool2d(2))
287         self.layer2 = nn.Sequential(
288             nn.Conv2d(48, 64, kernel_size=3, padding=1),
289             nn.BatchNorm2d(64),
290             nn.ReLU(),
291             nn.MaxPool2d(2))
292         self.layer3 = nn.Sequential(
293             nn.Conv2d(64, 32, kernel_size=3, padding=1),
294             nn.BatchNorm2d(32),
295             nn.ReLU(),
296             nn.MaxPool2d(2))
297 
298         self.calculate_conv_layer_output_size((self.layer1, self.layer2, self.layer3))
299         self.drop1 = nn.Dropout(0.5)
300         self.fc1 = nn.Linear(self.num_conv_outputs, self.num_classes)
301 
302     def forward(self, x):
303         in_size = x.size(0)
304         out = self.layer1(x)
305         out = self.layer2(out)
306         out = self.layer3(out)
307         out = out.view(in_size, -1)
308         out = self.drop1(out)
309         out = self.fc1(out)
310 
311         return out
```

Experimental Approach

- Main python code accepted command line arguments to select different network architectures and hyper-parameters
- Each run results run written file

- Bash scripts to kick off runs in the background
- Used cloud GPU – way too slow on a CPU-only machine
- Performance measures collected each epoch:
  - Loss
  - Val Accuracy
- Weights saved for later visualization



Overfitting Countermeasures

- Used dropout at 50%
- Monitored performance metrics over epochs to look for tell-tale loss decreasing while validation accuracy increasing

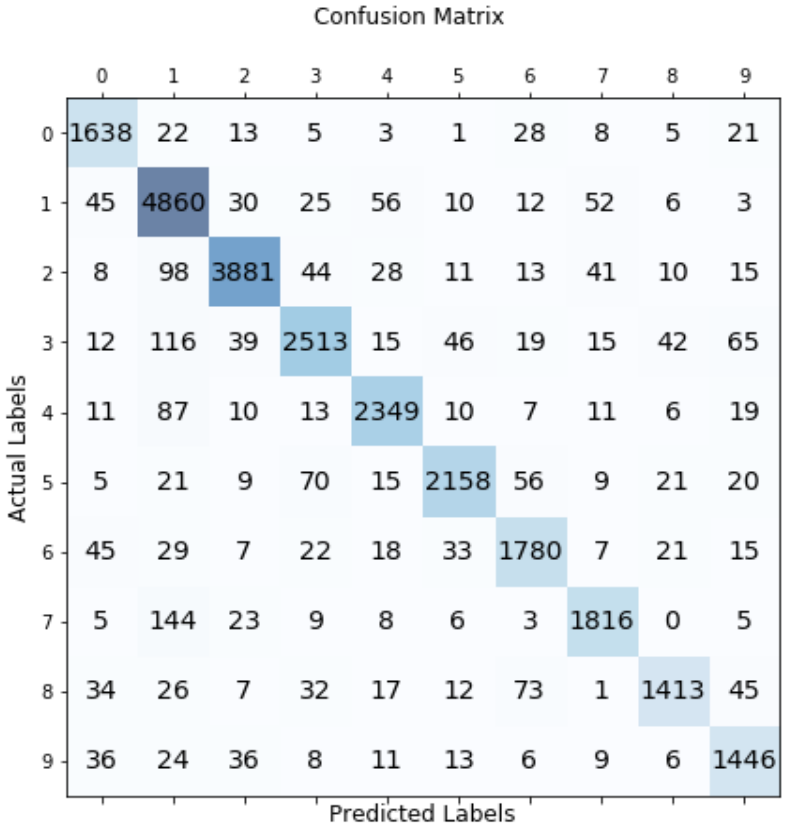
Batch Size

- Used a small batch size of 32 to get frequent weight updates
- Performed as well as 16
- Larger batch sizes ran faster, did not perform as well

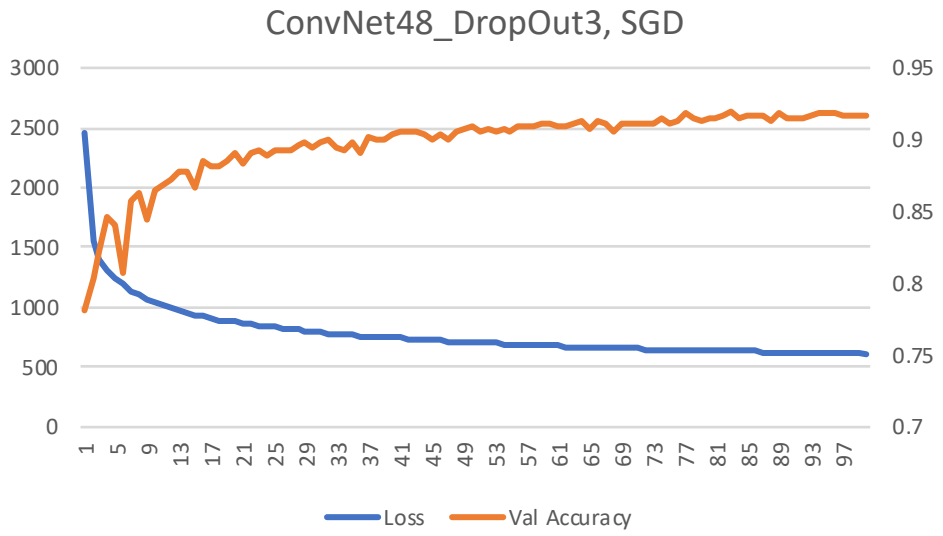
Results

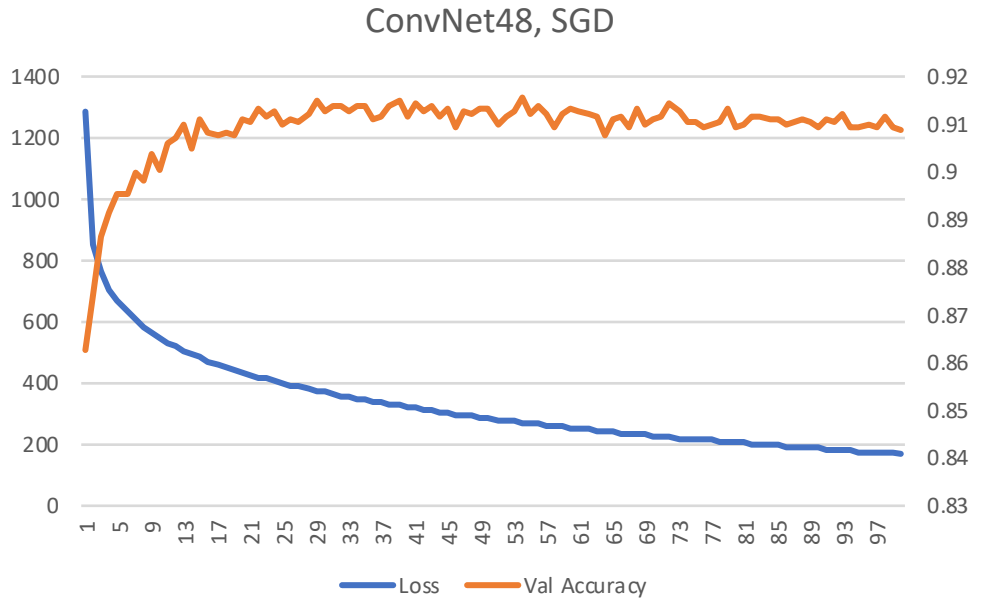
- Accuracy is the performance metric (target .90)
- Achieved 91.6% on the test data
- (Need to try 5x5 kernels w/ Dropout)

Label (The Digit)	Precision	Recall	F1	# Samples
0	0.89	0.94	0.91	1744
1	0.90	0.95	0.92	5099
2	0.96	0.94	0.95	4149
3	0.92	0.87	0.89	2882
4	0.93	0.93	0.93	2523
5	0.94	0.91	0.92	2384
6	0.89	0.90	0.90	1977
7	0.92	0.90	0.91	2019
8	0.92	0.85	0.89	1660
9	0.87	0.91	0.89	1595
avg/total	0.92	0.92	0.92	26032



Monitored Metrics





"Winning" network did not show overfitting signs but runner-up did

Unsupervised Digit Removal from Images

Not supervised: No labeled training data

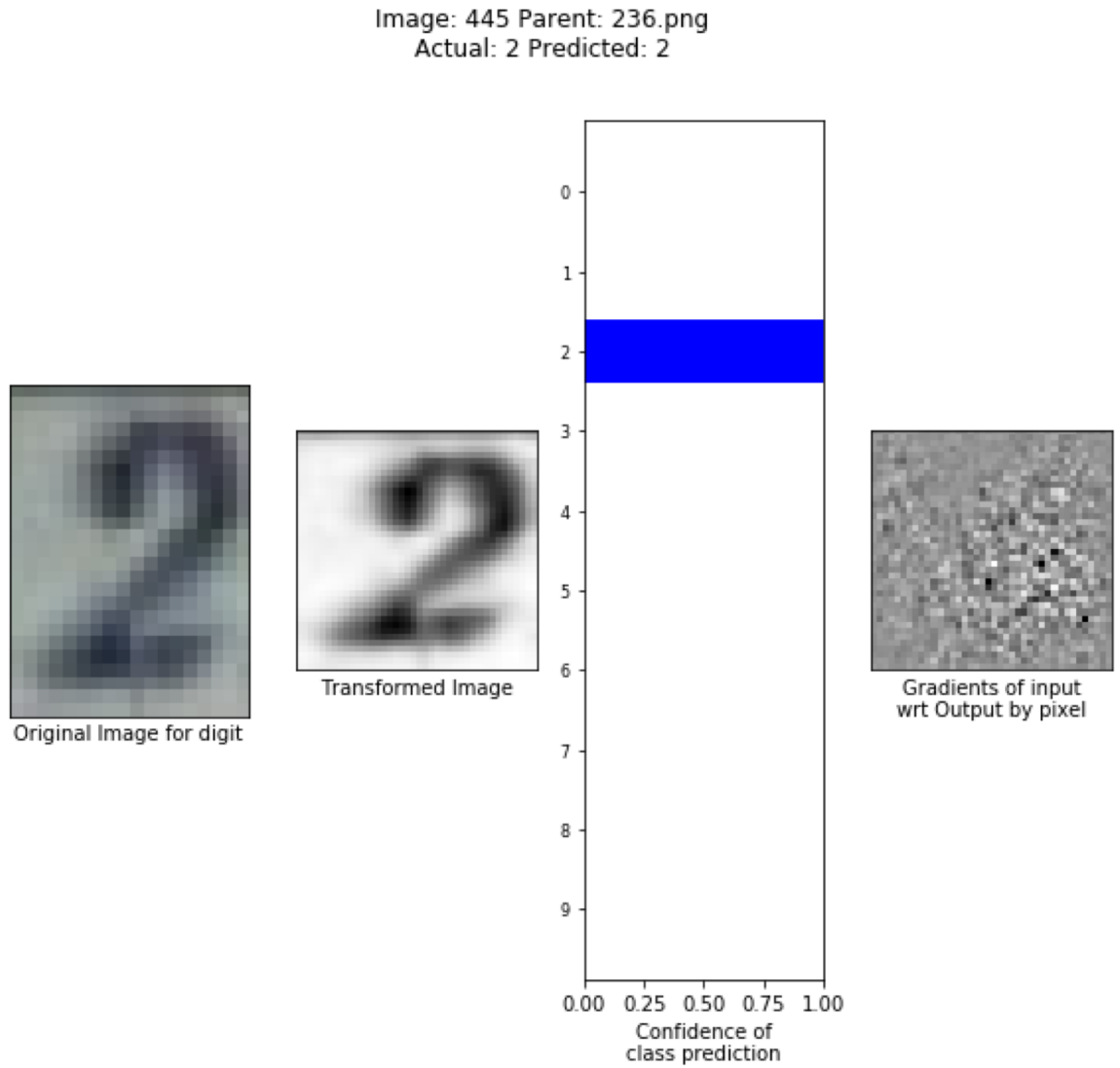
Approach:

- Train a model in PyTorch to classify digit images
- Use by-products of the model to identify key pixels for an image
  - Key Pixel: represents the background without the digit
- Create a Generalized Regression NN, trained with the key pixels, to generate an image

Goal is when the generated image is returned to the parent, a photorealistic image without the digit results

Identifying Key Pixels

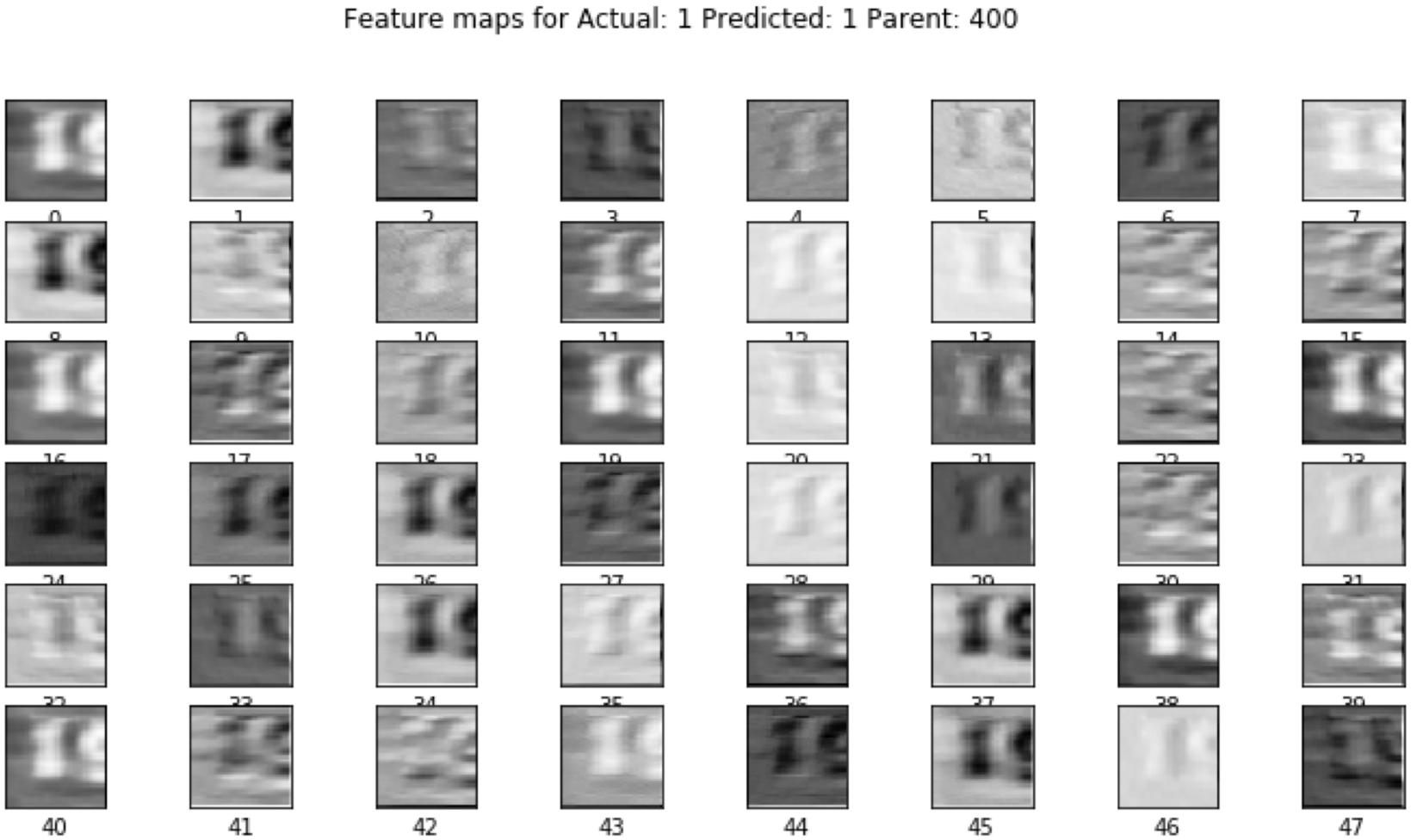
- First thought, gradients
- Very noisy
- Promising, perhaps better suited to a supervised approach



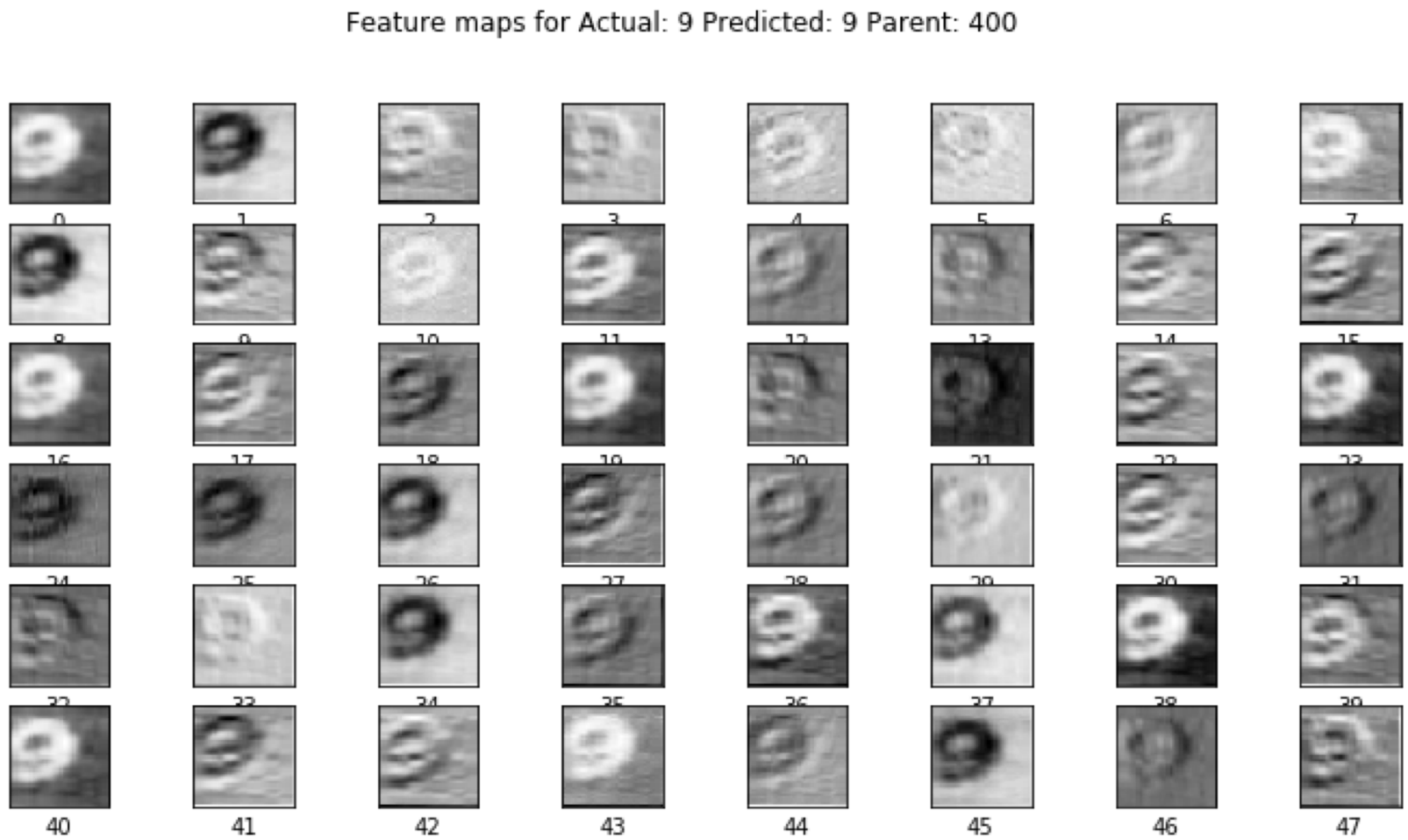
Next Try: Feature Maps from first Conv Layer

- First layer set up with stride, padding to have same Feature Map size as the image
- 48 Feature Maps from the first layer, reduced using mean() and std() across the pixel dimension – results in a tensor dimensioned the same as the image

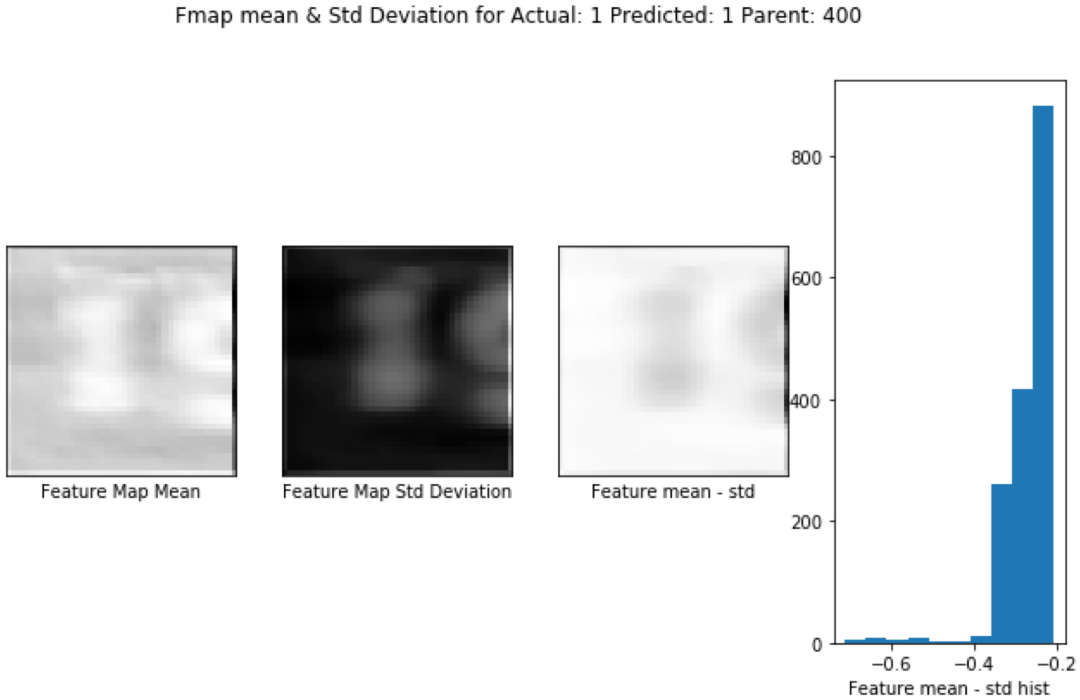
Feature Map for the first digit . . .

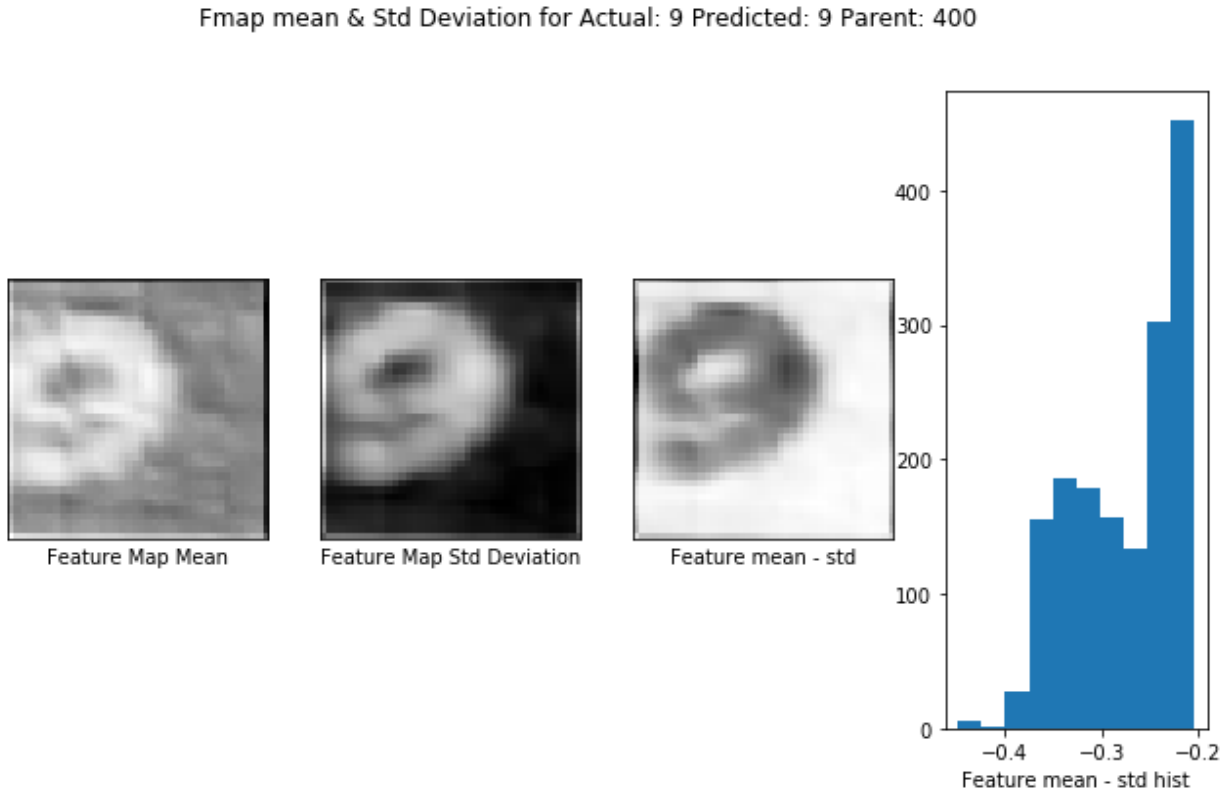


. . .And the second



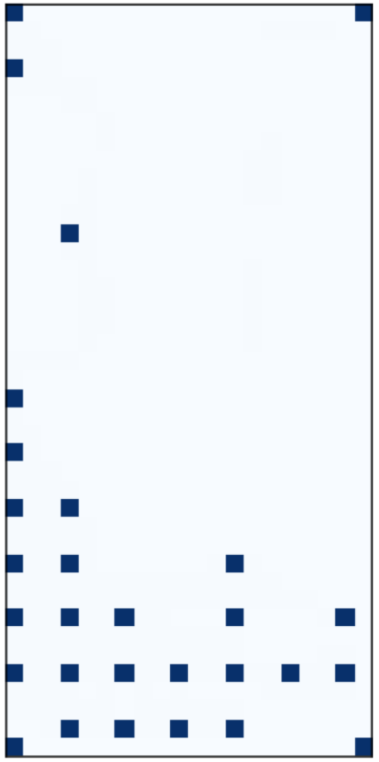
Feature Map Reductions





Key Pixel Identification

Image for digit



Key Pixels

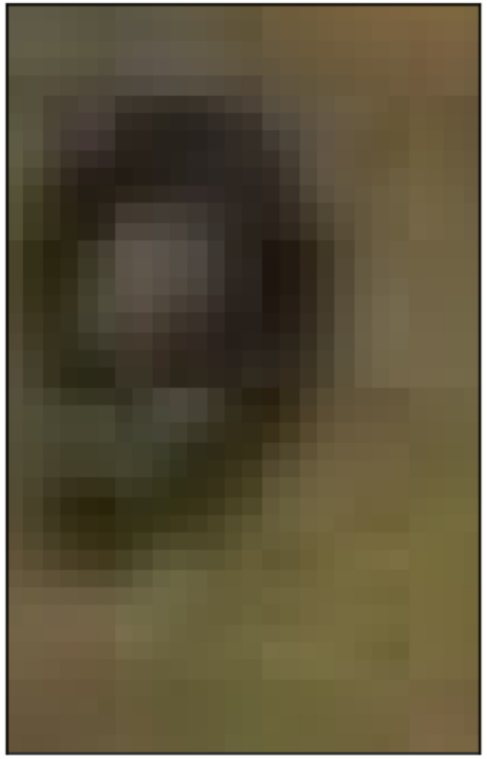
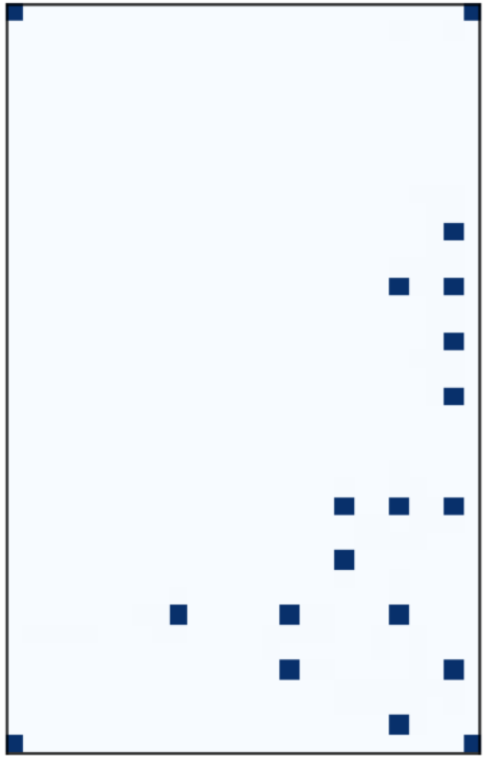


Image for digit



Key Pixels

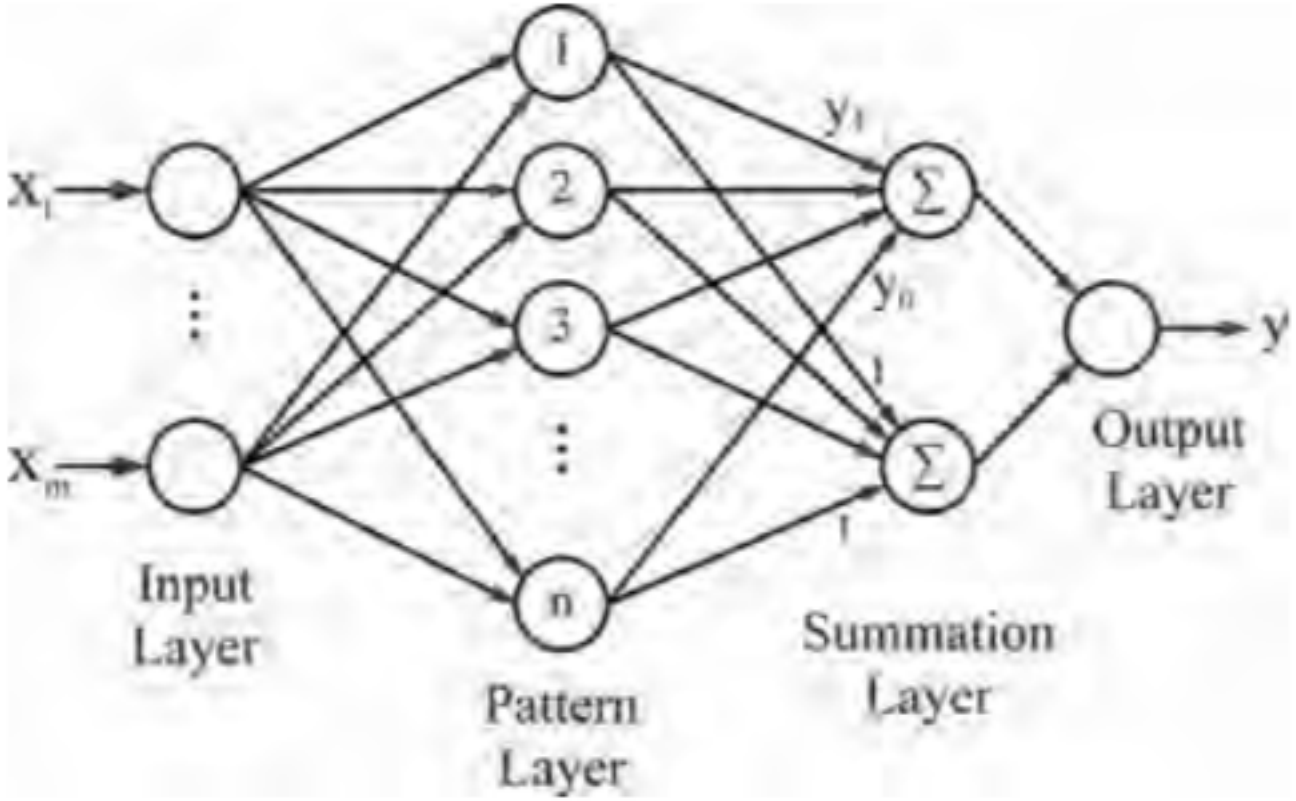
Network to Generate an Image

- Based on a paper about image infill after removing overlays:

Alilou, V., & Yaghmaee, F. (2015). Application of GRNN neural network in non-texture image inpainting and restoration. *Pattern Recognition Letters*, 24-31.

Architecture

- Input: x,y co-ordinate
- Output: Pixel value (range[0..1])
- Pattern layer outputs distance from input X to every node to itself
- Weights between pattern layer nodes and top summation layer are learned
- Weights to bottom note in summation layer always 1
- Output layer transfer function is a radial basis function



Radial Basis Function for summation and output layers

$$y = \hat{f}(X) = \frac{\sum_{i=1}^{n} y_i \cdot \exp\left(\frac{-D_i^2}{2\sigma^2}\right)}{\sum_{i=1}^{n} \exp\left(\frac{-D_i^2}{2\sigma^2}\right)}$$

- Gaussain: Nearby pattern nodes have greater influence than distant one on predicted X
- Sigma – “spread” or “bias: determines how quickly the influence of a pattern node diminishes with distance

PyTorch Module (excerpt)

```
def rbf(self, W2, Dsquared):
    return W2 * (torch.exp(-1.0 * Dsquared / (2.0 * self.sigmaSq)))

def forward(self, X):
    self.X = X
    out = torch.zeros(len(X), dtype=torch.float, device=self.device)

    self.Dsquared = torch.zeros((len(X), len(self.pattern_layer)), dtype=torch.float).to(device=self.device)

    for idx in range(len(X)):

        # Get sqaured distances to all pattern layer points from this X by
        # getting them from the precomputed values
        self.Dsquared[idx] = self.Dsquares[tuple(X[idx].cpu().numpy())]

    self.channel_out = []

    for cidx in range(self.channels):
        self.channel_out.append(self.rbf(self.W2[cidx], self.Dsquared).sum(dim=1) / self.rbf(1, self.Dsquared).sum(dim=1))

    # Repackage the channels so that all channels for a pixel are on the same row
    out = torch.cat(self.channel_out, dim=0).view(self.channels, -1).transpose(0,1)
```

Results

Actual: 1 Predicted: 1 Parent: 400

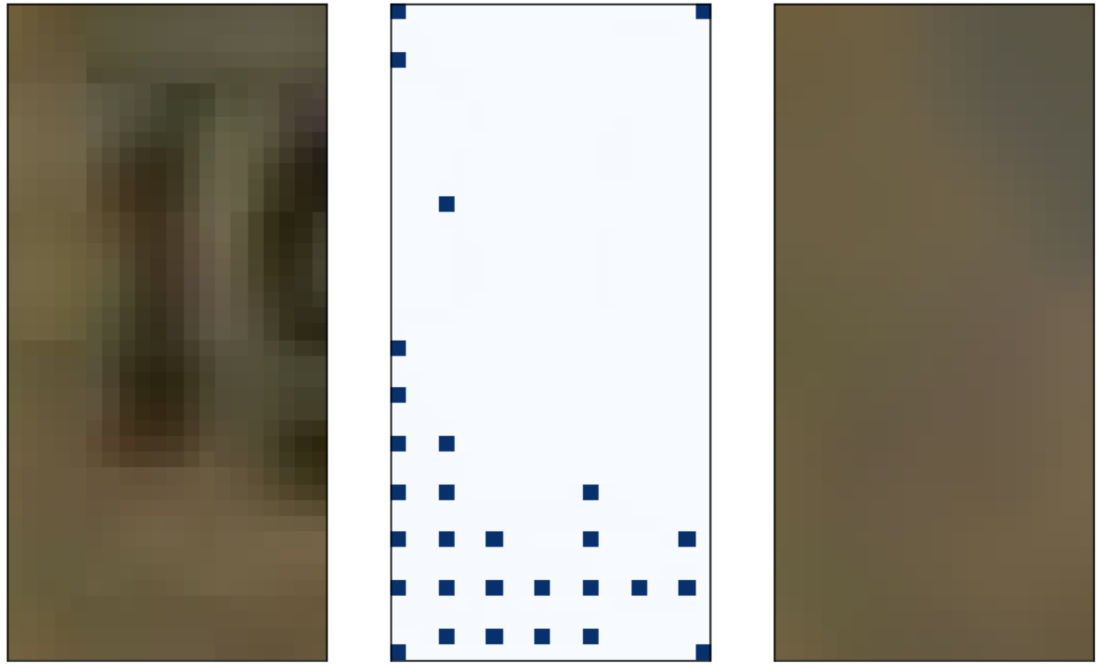


Image for digit | Key Pixels | Filled Image for digit

Actual: 9 Predicted: 9 Parent: 400

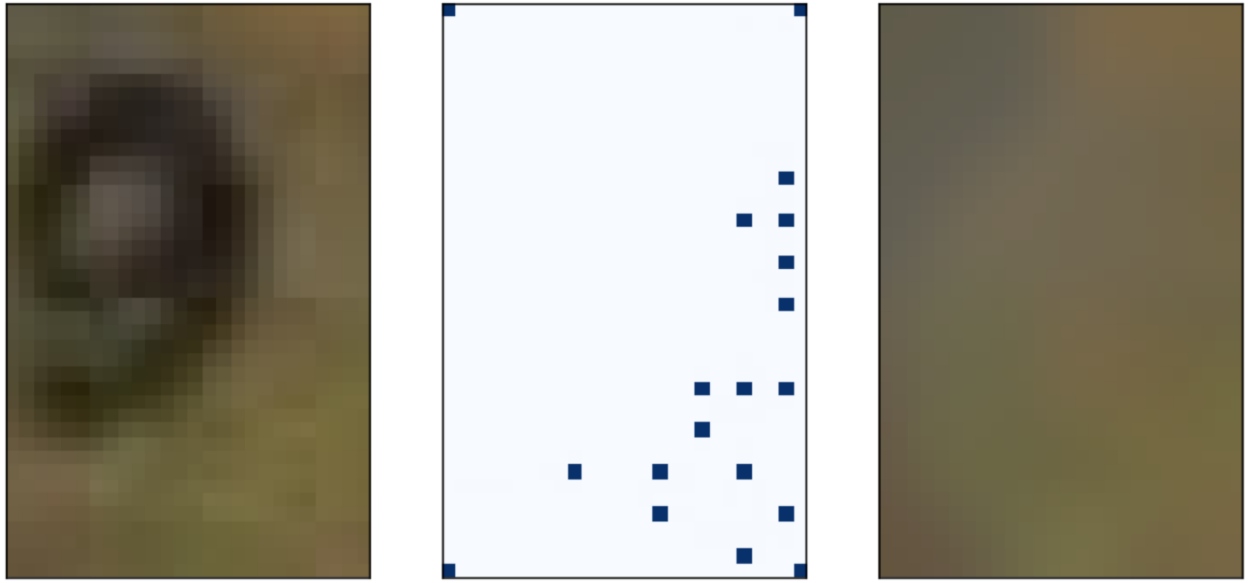


Image for digit | Key Pixels | Filled Image for digit

Final Results

Hits and Misses

Parent Images before and digit replacement (Image 7607)

Original

Altered

Parent Images before and digit replacement (Image 1201)

Original

Altered

Hits and Misses (cont’d)

Parent Images before and digit replacement (Image 577)

Original

Altered

Parent Images before and digit replacement (Image 29)

Original

Altered