Final Project
Group 1: Akshay Kamath. Cesar Lopez, Purvi Thakor

Convolutional Modeling with The Street View House Numbers Dataset

1. Description of the data set.

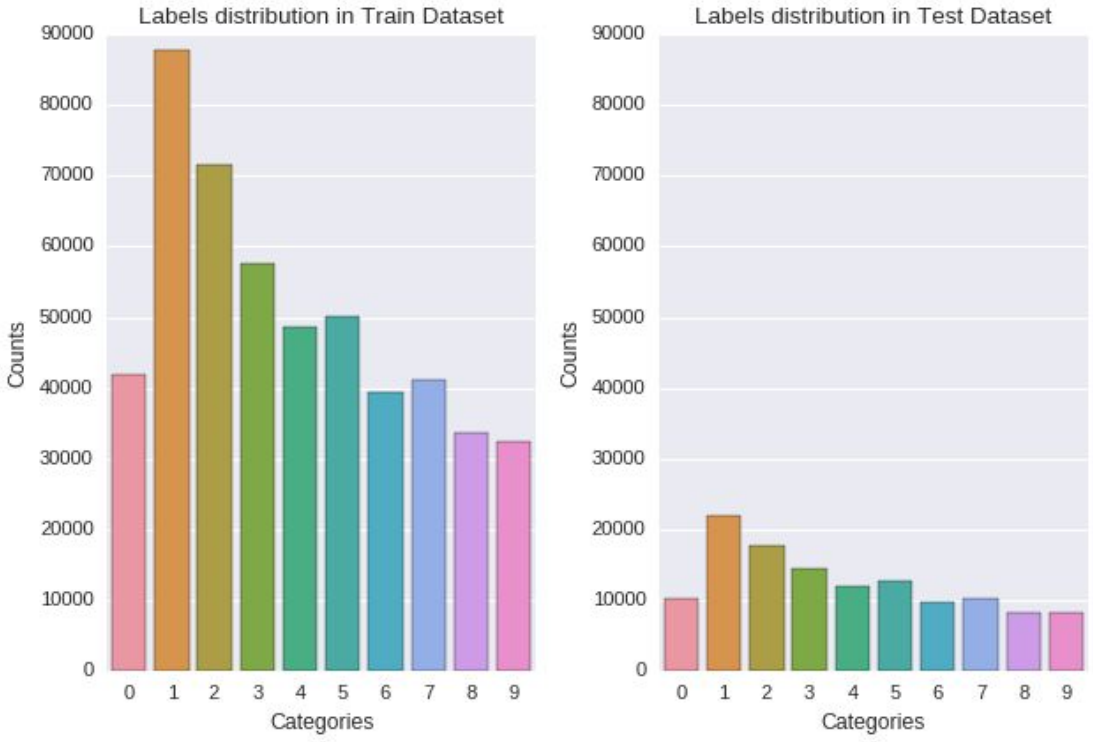
We used the Street View House Numbers (SVHN) Dataset for our final project. This is a real-world image dataset for developing object detection algorithms. It is similar to the MNIST dataset; however, it contains a significantly greater amount of labeled data (over 600,000 images).
The data has been collected from house numbers viewed in Google Street View. This dataset is 1.4 GB. Thus, it is large enough to train a deep network. The SVHN dataset has:

- 10 classes (unique labels)
    - One class per each digit. Digit "1" will have label 1, Digit "9" will have label 9. But digit "0" will have label 10.
- 73,257 digits for training.
- 26,032 digits for testing
- 531,131 additional samples for extra training.

It comes in two formats:

1. Original images with character level bounding boxes.
2. MNIST-like 32-by-32 images centered around a single character.

We decided to use the 32-by-32 images for our models. The format we selected came in a MATLAB file that we will later discuss how we used it. We did a bit of EDA on the dataset & found that the classes are unevenly distributed. There is a greater amount of digit "1" in both, training and test, datasets. In the graph below, you can see the distribution of all the categories.



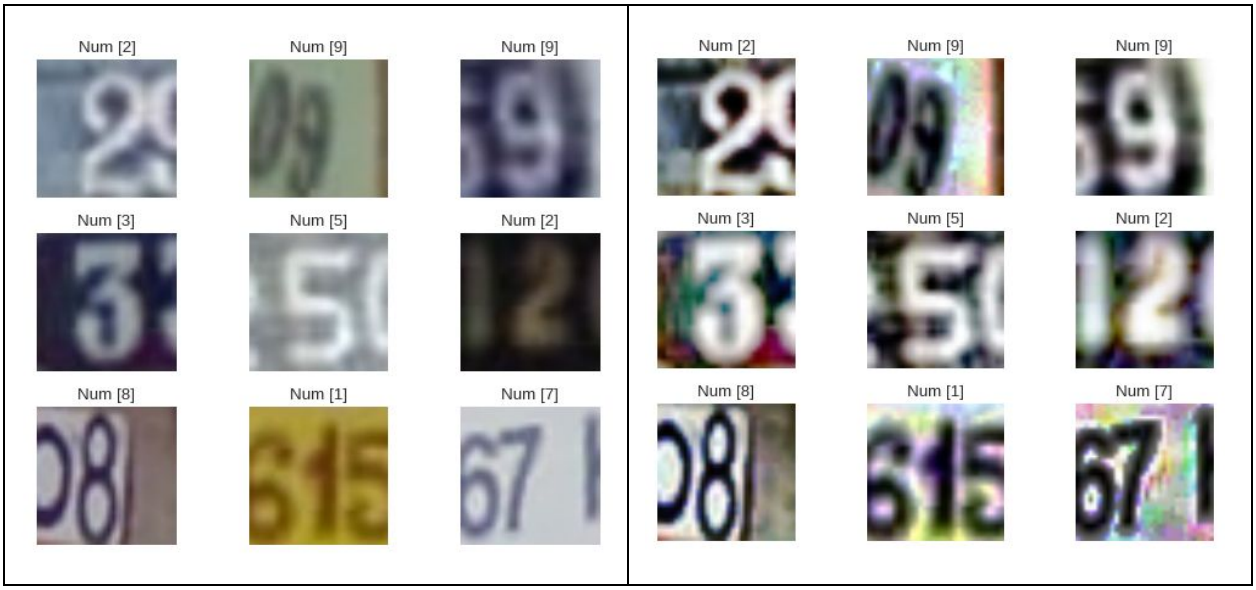
Final Project
Group 1: Akshay Kamath. Cesar Lopez, Purvi Thakor

++

2. Description of the deep learning network and training algorithm. Provide some background information on the development of the algorithm and include necessary equations and figures.

Since we are working with images and our goal is to predict the labels for each image, implementing convolution neural networks to this data would be best. We had 3 datasets training, testing, validation. The validation data contained less difficult images that might be useful for the training process. Therefore, we decided to concatenate the image data and labels from the three datasets respectively together into one single dataset. This resulted in having a dataset with all the images and labels. We then preprocessed the data by reshuffling it, splitting the data 80-20 and finally used histogram equalization to improve the contrast of our images. This may help the model because of edges being more defined from the histogram equalization. Below are sample images before and after the histogram equalization. The final step was to export the outputs into test and train ***.mat*** files.

#Effect of Histogram Equalization - (contrast improvement)



++

3. Experimental setup. Describe how you are going to use the data to train and test the network. Explain how you will implement the network in the chosen framework and how you will judge the performance. Will you use mini batches? How will you determine the size of the mini batches? How will you determine training parameters (e.g., learning rate)? How will you detect/prevent overfitting and extrapolation?

Once we the data was split, each of us decided to work on different frameworks. Our goal was to compare model performances across frameworks. Purvi & Akshay decided to work on Caffe & Pytorch. Cesar worked on Tensorflow and Keras.

Final Project
Group 1: Akshay Kamath. Cesar Lopez, Purvi Thakor

Our convolutional network architecture consists of these layers:

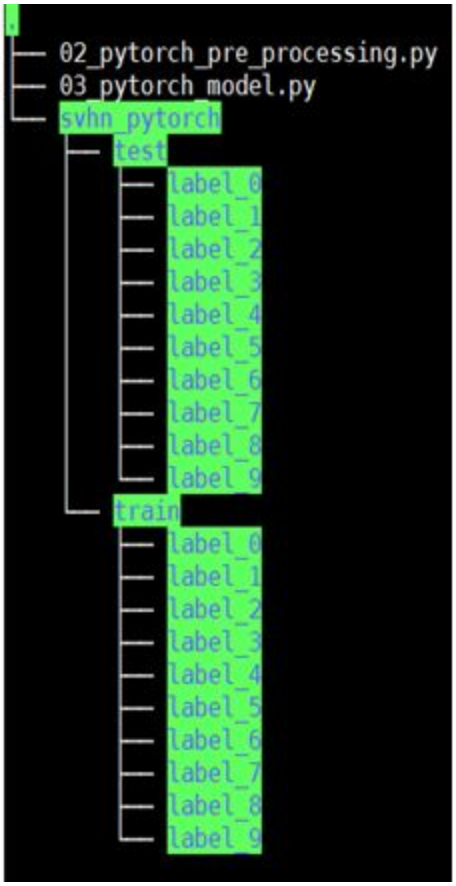
- Input Layer
- Convolutional Layer
- Pooling Layer
- Dropout Layer
- Fully-connected Layer

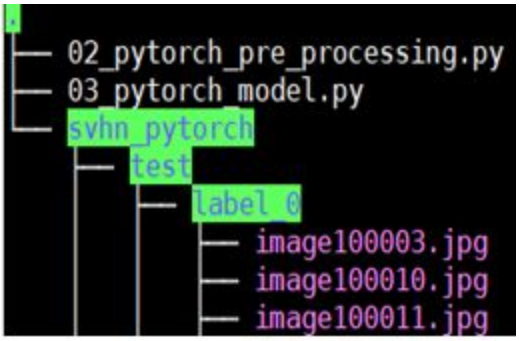
To start running the convnet, it was critical to understanding the shapes of the images. There are 4 dimensions of data must have the number of images or batch size, pixel height, pixel width, and channel. Different frameworks take these dimensions in a different order. The following are the frameworks we implemented along with the dimension order they take as inputs.

The initial idea was to try and build models across different frameworks. Keeping all other hyperparameters constant, we wanted to see if results could be reproduced across different frameworks.

PyTorch: [batch_size, channels, height, width]

PyTorch is a python based library that is closely related to the python's scientific library - numpy. It is easy to learn due to its similarity to numpy. PyTorch uses Tensors which are alike numpy arrays that can be run on GPUs. Pytorch requires that the directories be set up in the following structure so that it can associate the images with the correct label.





Final Project
Group 1: Akshay Kamath. Cesar Lopez, Purvi Thakor

Data loading in Pytorch is easy. First, we applied a data transformation to convert the scale of the image from [0,255] to [0,1] and also convert it from .jpg to a torch tensor in a single step using 'transforms.ToTensor()'. Next, we loaded the train and test images using ImageFolder. Dataloader basically takes care of loading the images in the specified batch sizes which is later used to enumerate through the train and test loader.

I trained the 504,336 training images using two layers comprising of Convolution, ReLU, and Max pooling. Using the below architecture with 10 number of epochs, a batch-size of 500, and a learning rate of 0.001.

```
class CNN(nn.Module):
    def __init__(self):
        super(CNN, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Conv2d(in_channels=3, out_channels=16, kernel_size=5, stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2, padding =0))
        self.layer2 = nn.Sequential(
            nn.Conv2d(in_channels=16, out_channels=32, kernel_size=5, stride=1, padding=1),
            nn.ReLU(),
            nn.MaxPool2d(kernel_size=2, stride=2, padding =0))
        nn.Dropout(p=0.2),
        self.fc1 = nn.Linear(in_features=6*6*32, out_features=10)

    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = out.view(out.size(0), -1)
        out = self.fc1(out)
        return out
```

Here, in_channels in the first layer mean the depth of the channels (i.e 3 for color images), out_channels are the number of feature maps produced, in the second layer these out channels are the input to the second convolution layer.

The choice of using Cross-Entropy Loss function was straightforward because it combines Log Softmax and Negative Log Likelihood Loss function. The way that Cross-Entropy works in Pytorch is that it converts the outputs to probabilities using the log softmax function. The Loss functions expect a class index in the range of 0 to C-1 where C is the number of classes. This was also one of the reasons to convert the label 10 in our data to label 0.

I liked working on Pytorch mainly because one can explicitly define the network parameters. Understanding how the math works during forward propagation is crucial to supplying those values in the CNN network.

Caffe: [batch_size, channels, height, width]

For using the Caffe Framework, the data needs to be in a database. Formats acceptable are LMDB, & HDF5. Our dataset was initially in a ***.mat*** format.

The files were of MATLAB 3 format. I tried using MATLAB to convert the ***.mat*** files to the MATLAB 7.3 format. However, when I restructured the data to the format which our models needed, I found that the transformation was not working. I eventually found out that MATLAB files can be read as Numpy arrays in python. So using the Scipy.io package in python, I managed to get the data in python. Since we had 3 datasets (Training, Testing & Validation) we decided to group them together into one single dataset. I then helped preprocess it, (i.e - reshuffled the data, split it into train and test, used histogram equalization to improve the contrast of our images) & exported the outputs into test and train ***.mat*** files.

In order to convert the processed ***.mat*** files to an LMDB database format, I needed to convert the arrays to images *(.jpg)* and save it in the following file folder structure.

root/train/label_0/img0.jpg
root/train/label_0/img1.jpg
root/train/label_1/img0.jpg

I used the PIL (Python Imaging Library) for this very process.
Then using Dr. Jafari's 03-CreateLMDB.py code & tweaking it a bit, I created LMDB databases for the Train and Test sets which were then used in a CNN architecture to train the model.

Tensorflow [batch_size, height, width, channels]
This framework was used to explore its use benefits of computational graphs visualization in deep learning. Originally, the approach I took in building convnets with this framework was with many functions. I realized that I kept typing the same code over and over for different layers, so I made functions to reduce the code clutter in the file. However, later on, I realized that I need to set up my code in such a way that it can make the tensorboard effective for data analysis.

I decided to build my convolutional neural network model using the following parameters:

1. Two Convolution layers followed by max pooling layer after each convolution.
2. Convolution Layer 1: Weight Size 5 x5, Stride: 1, Feature maps output: 16
3. Max Pooling layer: pool window size 2 x 2, stride 2
4. Convolution Layer 2: Weight Size 5 x5, Stride: 1, Feature maps output: 32
5. Max Pooling layer: pool window 2 x 2, stride 2
6. Flatten Max Pool output
7. Dense layer 1: Weight shape: [flatten Max x 512], 500 neurons.
8. Dropout: hold probability: 0.5
9. Dense layer 2: Weight shape: [500 x 10]

Final Project
Group 1: Akshay Kamath. Cesar Lopez, Purvi Thakor

Keras: [batch_size, height, width, channels]

I wanted to try Keras for this project as well. I wanted to see what was so great about. Keras is a wrapper around tensorflow. I followed the same procedure I used for tensor with Keras. Keras is user-friendly. It just took a couple lines of code for it to run a model. The output of it was descriptive. Below is a sample program I ran with a small portion of our data. It tells you the accuracy and error of both the training and testing. It updates live as the program is running.

.

```
50000/50000 [==============================] - 8s 159us/step - loss: 0.9724 - acc: 0.6869 - val_loss: 0.4360 - val_acc: 0.8788
Epoch 2/10
50000/50000 [==============================] - 2s 47us/step - loss: 0.3803 - acc: 0.8900 - val_loss: 0.3358 - val_acc: 0.9070
Epoch 3/10
50000/50000 [==============================] - 2s 48us/step - loss: 0.3092 - acc: 0.9099 - val_loss: 0.3084 - val_acc: 0.9148
Epoch 4/10
50000/50000 [==============================] - 2s 47us/step - loss: 0.2661 - acc: 0.9221 - val_loss: 0.2834 - val_acc: 0.9220
Epoch 5/10
50000/50000 [==============================] - 2s 48us/step - loss: 0.2314 - acc: 0.9319 - val_loss: 0.2673 - val_acc: 0.9300
Epoch 6/10
50000/50000 [==============================] - 2s 49us/step - loss: 0.2064 - acc: 0.9390 - val_loss: 0.2583 - val_acc: 0.9316
Epoch 7/10
50000/50000 [==============================] - 2s 47us/step - loss: 0.1885 - acc: 0.9443 - val_loss: 0.2532 - val_acc: 0.9306
Epoch 8/10
50000/50000 [==============================] - 2s 46us/step - loss: 0.1652 - acc: 0.9512 - val_loss: 0.2474 - val_acc: 0.9324
Epoch 9/10
50000/50000 [==============================] - 2s 47us/step - loss: 0.1491 - acc: 0.9555 - val_loss: 0.2575 - val_acc: 0.9330
Epoch 10/10
50000/50000 [==============================] - 2s 47us/step - loss: 0.1344 - acc: 0.9604 - val_loss: 0.2436 - val_acc: 0.9376
```

++

4. Results. Describe the results of your experiments, using figures and tables wherever possible. Include all results (including all figures and tables) in the main body of the report, not in appendices. Provide an explanation of each figure and table that you include. Your discussions in this section will be the most important part of the report.

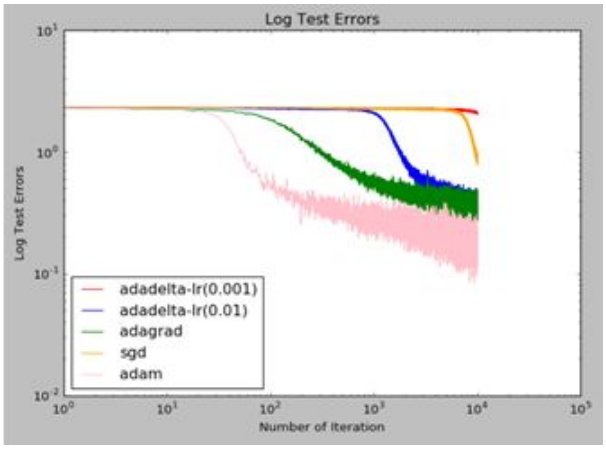
Pytorch

Optimizers:

Comparing the various optimizers, it turns out that Adam optimizer converged faster than others. To update the network weights, I chose the Adam (Adaptive Moment Estimation) optimizer algorithm. Compared to Stochastic Gradient Descent which maintains a single learning rate for weight updates, the Adam algorithm is based on adaptive learning rates which is practice requires less tuning and gives better, quicker results.

Final Project
Group 1: Akshay Kamath. Cesar Lopez, Purvi Thakor

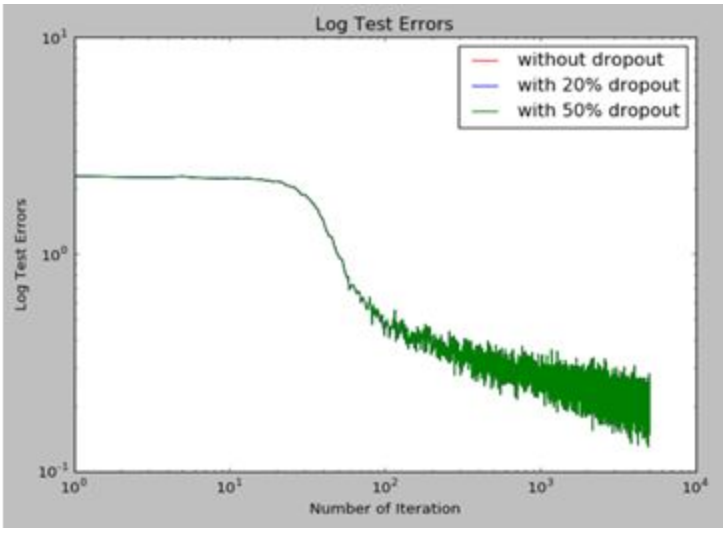
Optimizer	Test Accuracy	F1 score	Time(mins)
Adam	95%	0.948	27.19
Adagrad	89%	0.891	26.8
Adadelta(lr = 0.01)	89%	0.892	26.65
SGD	77%	0.748	26.61
Adadelta	29%	0.136	26.62



Dropouts:

Drop out is done to prevent over-fitting and dependency of neurons on each other. Drop-out means to ignore or shut off some neurons (along with their connections) during the training phase. During dropout, these nodes are not considered in forward or backward pass. Nodes are dropped out with a probability of 1-p or kept with a probability of p. This leaves a training network with less number of neurons in the network. Testing is done with the full network without dropout nodes.
As can be seen in the below figure, adding a dropout of 20% and 50% keep probability did not affect the log errors. This meant that the model was not overfitted.

Dropouts	Test Accuracy	F1 score
0.2	95%	0.948
0.5	95%	0.948
0	92%	0.918



Confusion Matrix:

The confusion matrix is such that for every element in **ith** row and **jth** column, the number is equal to total number of images that belong to label i but are classified in label j. Thus, the diagonal elements give the number of images
that are correctly classified by the model and off diagonal elements represent misclassifications. The actual labels
are represented as rows and the predicted values are represented as columns.
Predictions are based on 4 classification parameters:

1. TN / True Negative: case was negative and predicted negative

2. TP / True Positive: case was positive and predicted positive
3. FN / False Negative: case was positive but predicted negative
4. FP / False Positive: case was negative but predicted positive

Precision: Accuracy of correctly classifying a label
Precision = TP/(TP +FP)

Recall: Fraction of labels that were correctly classified.
Recall = TP/(TP+FN)

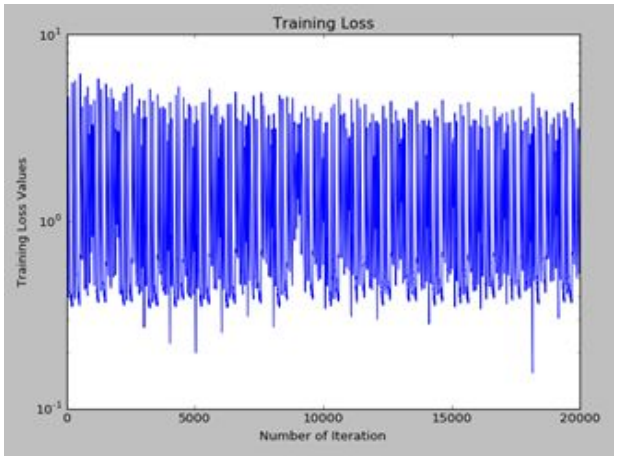
		PREDICTED										
		label 0	label 1	label 2	label 3	label 4	label 5	label 6	label 7	label 8	label 9	RECALL
TARGET	label 0	9868	133	39	17	53	12	137	21	24	114	0.95
	label 1	55	21235	75	51	228	32	44	164	23	27	0.97
	label 2	22	107	17217	79	102	40	30	69	53	111	0.97
	label 3	9	128	135	13532	66	239	76	57	100	79	0.94
	label 4	11	224	74	31	11771	16	51	33	13	48	0.96
	label 5	10	50	41	185	40	12004	176	11	33	41	0.95
	label 6	60	51	19	60	65	121	9233	15	93	15	0.95
	label 7	10	338	157	51	59	11	16	9593	14	29	0.93
	label 8	36	68	48	139	55	67	152	12	7744	94	0.92
	label 9	77	82	74	65	59	53	26	28	62	7667	0.94

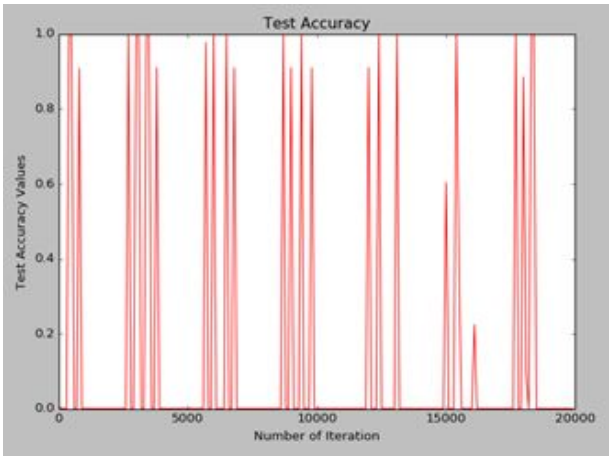
Our dataset had class imbalances and probably that is why some of the labels have higher recall as compared to others.

Caffe

Initially, I had used Dr. Amir's solver files check if my model was executing correctly. On executing that I figured that the losses were all over the place & the accuracy was really poor. I then modified the train_test & solver files with a different set of layer combinations and hyperparameters. However, the model accuracy was still pretty poor.

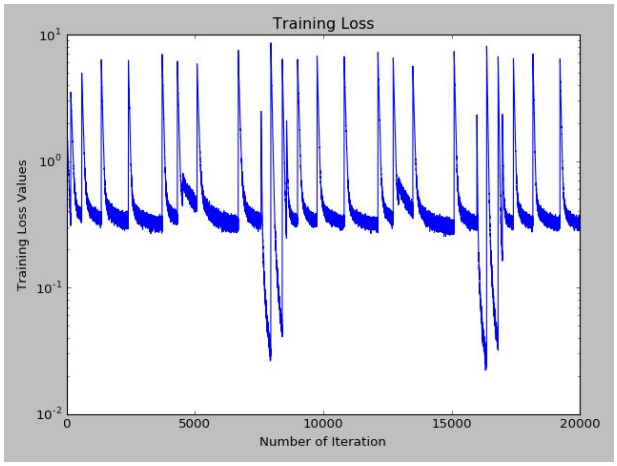
(Initial Model)

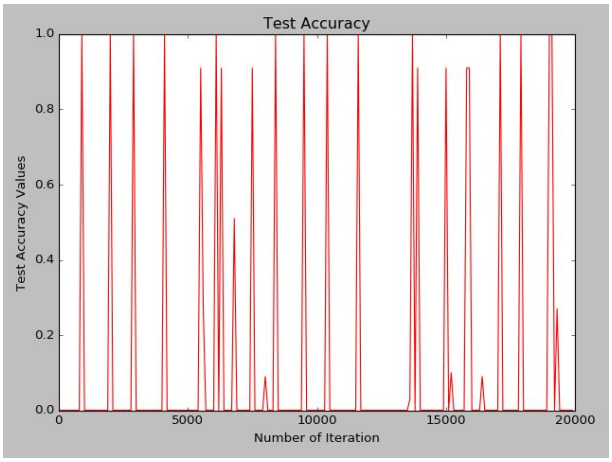




(WIP Model)

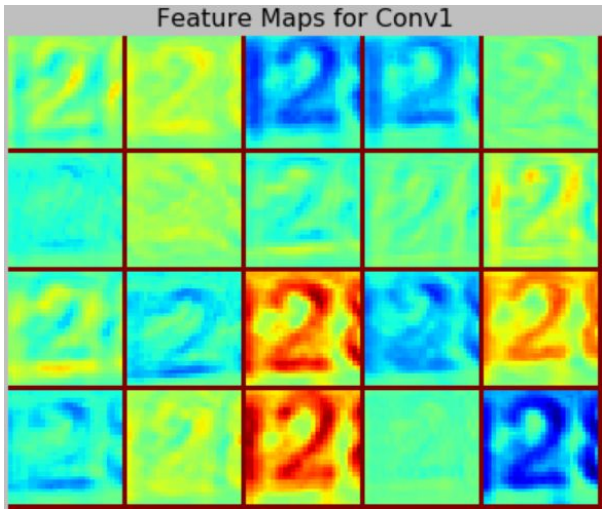
Final Project
Group 1: Akshay Kamath. Cesar Lopez, Purvi Thakor

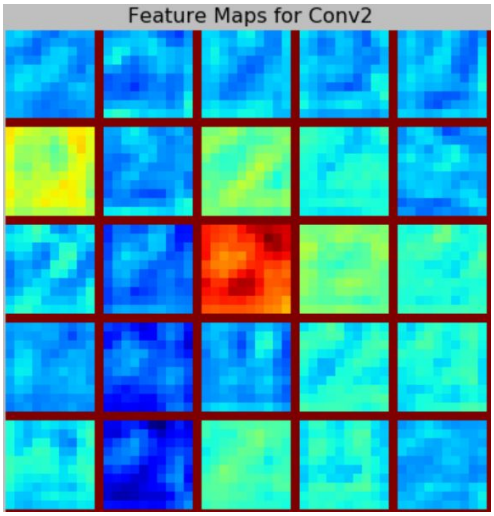




As we can see, the initial graph on the bottom left has losses that tend to drop over specific intervals, but when a new batch enters the model, the loss tends to go up high. This was a bit baffling to me as well. After having a chat with Dr. Jafari, we came across 2 scenarios.-

- The data was perhaps not loaded correctly onto the LMDB. However, after debugging the code which created the databases, we concluded that the LMDB was created correctly.
- According to the Caffe documentation, I had rescaled my dataset from the range of 1-255 to 0-1. However, this in not really Normalization. It is just rescaling the values. In order to ***correctly*** normalize the datasets, I needed to add a LRN layer (to do a batch normalization) to the batch_size.





Tensorflow

I decided to work on Tensorflow and Keras. The reason behind this was because we never got to do homework on with these frameworks and they are important to know in the field.

First thing I dealt with was the data preprocessing. I am familiar with Matlab, so I originally wanted to change the image format to jpeg, then start using python. However, I learned about the SciPy python library. This enabled me to get the data I needed from the Matlab files using python.

Final Project
Group 1: Akshay Kamath. Cesar Lopez, Purvi Thakor

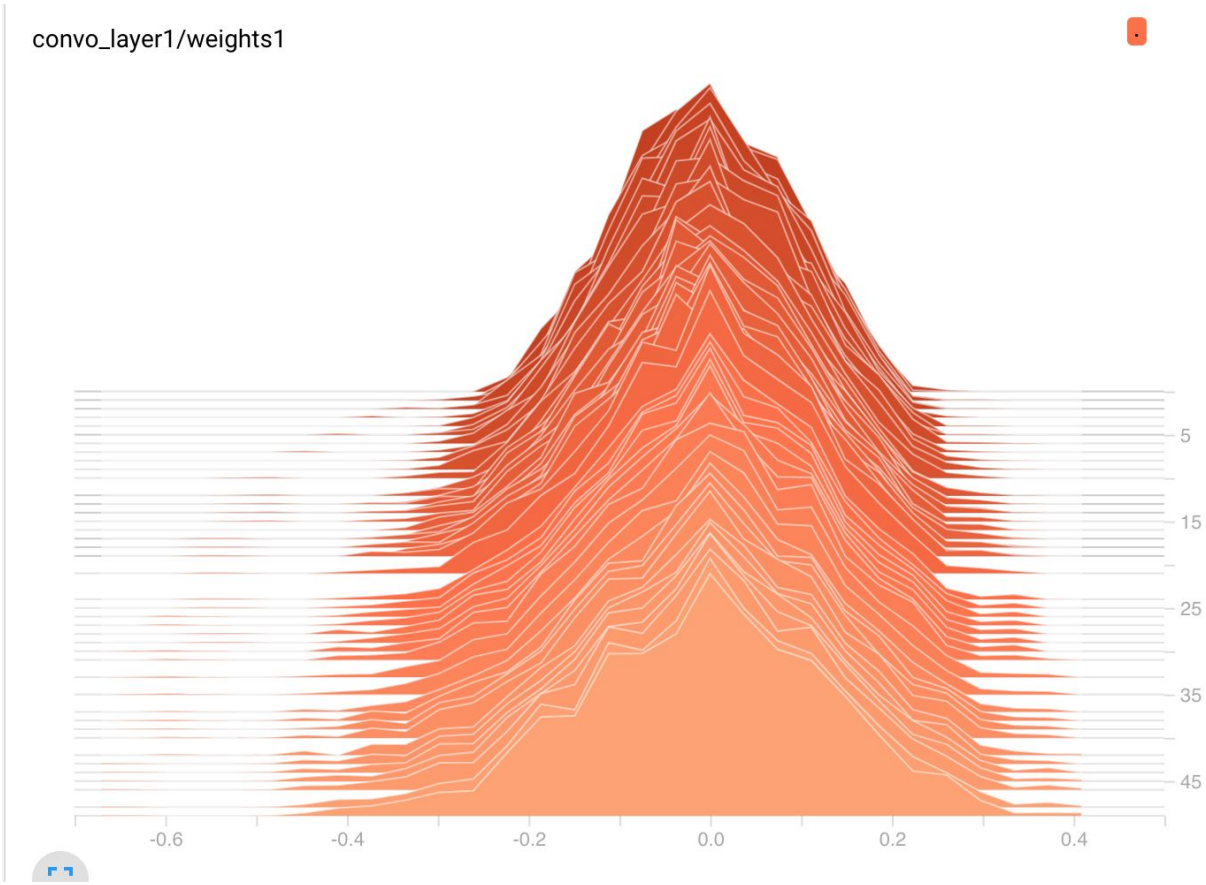
I looked around on what to use to download the Matlab files from the website. I wanted to use request, but wget (python version) seemed easier to use. I implemented my style of coding to use it.

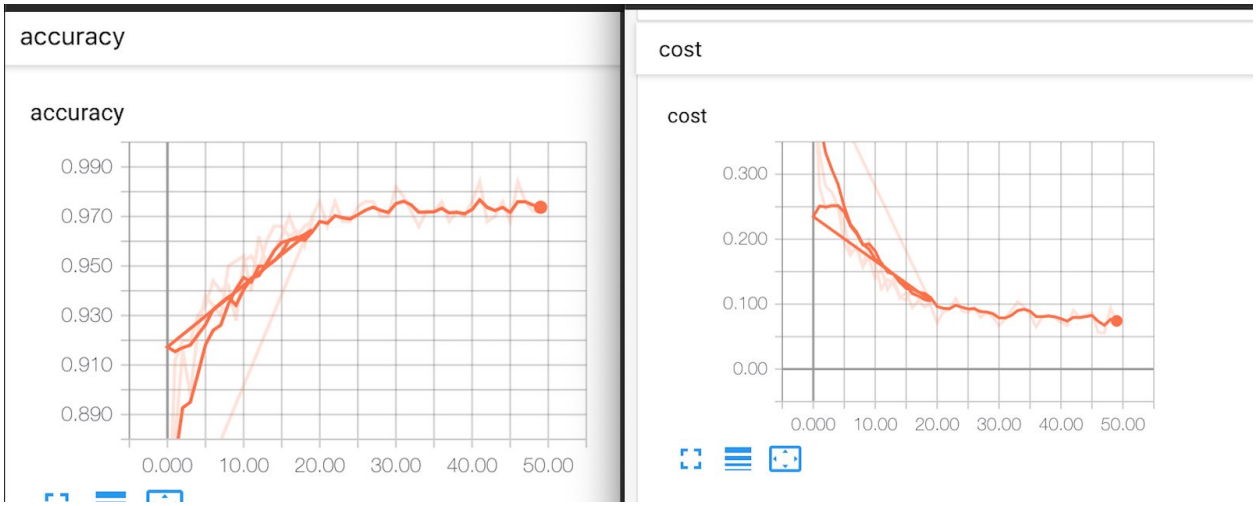
I also wrote code to create a directory for the data. It will check if it exists first. If it does not, it will create one. I used a similar method with checking if the Matlab files existed to prevent multiple downloads when running the code. A screenshot of this code is below.

```
folder_name = 'svhn_data'
filename_list = ['test_processed.mat', 'train_processed.mat']

print('\nChecking if ' + folder_name + ' directory exist...\n')
try:
    os.makedirs(folder_name)
    print('Directory does not exist. Creating ' + folder_name + ' directory now...\n')
    print('Directory ', folder_name, ' Created\n')
except FileExistsError:
    print("Directory ", folder_name, ' already exists\n')

print('downloading svhn data files...')

for filename in filename_list:
    filepath = './svhn_data/' + filename
    if not os.path.exists(filepath):
        print('\nDownloading ' + filename + ' file')
        url = 'https://storage.googleapis.com/1_deep_learning_final_project_group_1/processed_files/' + filename
        wget.download(url, filepath)
        print('\n')
    else:
        print('\nfile ' + filename + ' already exist.')

print('\n' + '*'*10 + 'Downloading Done' + '*'*10 + '\n\n')
```

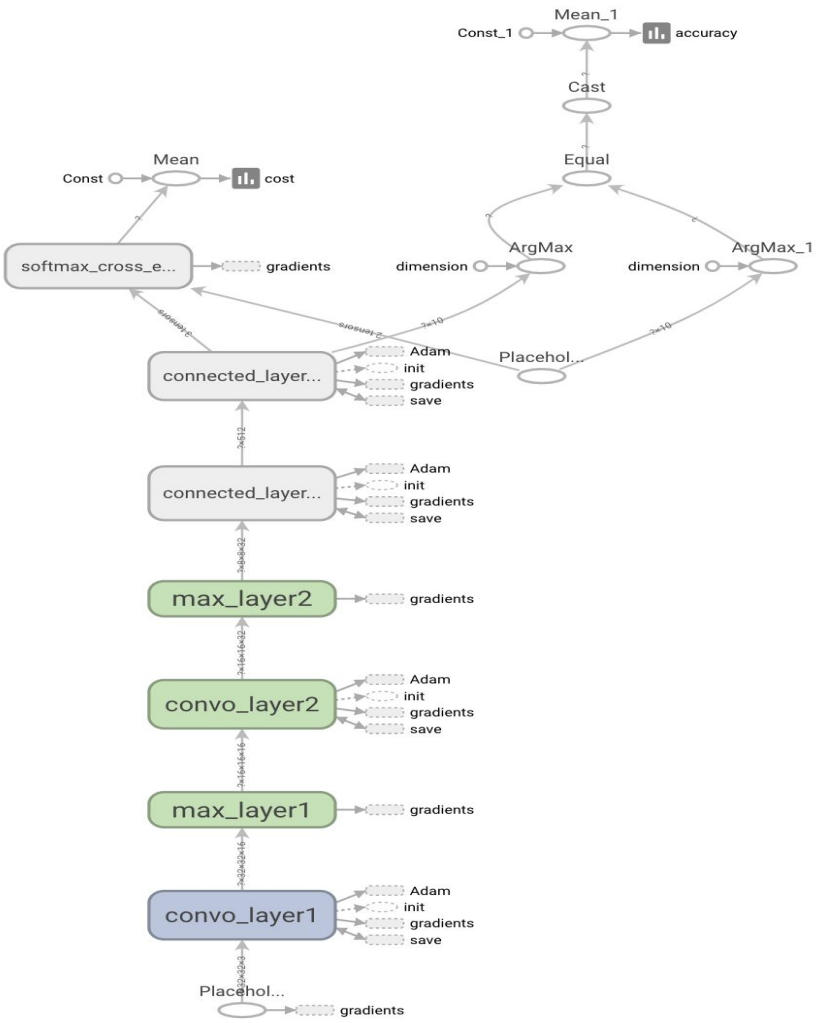
I had a Resource Exhausted Error in tensorflow when I tried running the code. I went online and searched for people having the same issue. The only solution I found was to decrease my epoch or get a better GPU. I decided to train half the data. I used 300,000 training data and 45,000 testing data. It ran with no problem. I ran tensorboard on it to the histograms of the weights, loss and accuracy plots, and the network graph. Below are the three images respectively.

Final Project
Group 1: Akshay Kamath. Cesar Lopez, Purvi Thakor





Final Project
Group 1: Akshay Kamath. Cesar Lopez, Purvi Thakor



My accuracy is going up and my cost is going down. Therefore the model is running well. You can see that the kernels in the first convolutional layer get more normally distributed after every epoch. The network graph you can see the layers. If you click on a layer, it will open up the variables in that layer.

++

6. Summary and conclusions. Summarize the results you obtained, explain what you have learned.

Pytorch:

Because of the way Pytorch is written, users have the ability to easily change different hyper-parameters and fine-tune their models.

Caffe:

Caffe Community is pretty sparse as compared to some other of the frameworks. Help is not easily available and the documentation is not too informative. Being built run on top of C++, it is a bit clunky to use. However, being built on C++, the execution time is extremely low as compared to some other frameworks.

Tensorflow:
This framework is difficult to debug in because the graphs do not run until the session in initiated. One would have to use tensorboard to look at the graph and its parameter to see what is wrong with the code. My results using Tensorflow was about 95% after 10 epochs.

Keras:
Keras is easy to use. This would be good to use to check out quick model, but it would be better to use frameworks like PyTorch or TensorFlow.

++

7. Suggest improvements that could be made in the future.

Pytorch:
This project was indeed a good learning experience to apply all the things I learned this semester. Though my part on this project seems similar to the exams, there were quite a few challenges that we came across when working on a dataset of our choice. The first and foremost was pre-processing the data. It took us quite some time to come up with appropriate pre-processing that made sense to us. Once we managed that, loading a custom data in Pytorch was something that I had never done before (We used built-in datasets in the exams). Nevertheless, that seems like an easy part now. I still need to have a better understanding of how each loss and optimizer function works.

Since, MNIST and SVHN both are datasets about Digit Classification, I would really like to try working on using weights from MNIST and use those on the SVHN data. I would also like to explore more architectures that could help to classify these images.

Caffe:
Once this model is fixed, (i.e image normalization), I would like to transfer the (test) weights from this model onto another dataset & keep track of the model performance.
Also, working on grayscale images would be better & getting a higher model accuracy would have been easier since we would have to deal with only a single channel, thus the number of parameters would decrease by a lot.

Tensorflow:
Much time was spent on setting up the architecture taking the shapes into account. Another issue I had was the bias. The project would have been better if I learned more tensorboard features. Image embedded is a great one that I wanted to use, but it had some input that I do not understand yet. This feature could illustrate how our model groups each label together.

Keras:
I did not spend too much time exploring keras. Keras also uses tensorboard and it adds everything for you. Since it does this for you. It would be a good approach to add the data differently. We can let Keras split the data to train and test. We may get different results. We can save those models and use tensorboard to find the best parameters for that best model.
++

8. References. In addition to references used for background information or for the written

Final Project
Group 1: Akshay Kamath. Cesar Lopez, Purvi Thakor

portion, you should provide the links to the websites or GitHub repos you borrowed code from.

Pytorch:

https://discuss.pytorch.org/t/using-your-own-data-for-pytorch/20193/3
https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/
http://ufldl.stanford.edu/housenumbers/nips2011_housenumbers.pdf
https://discuss.pytorch.org/t/convolution-input-and-output-channels/10205
http://cs231n.github.io/convolutional-networks/
https://www.analyticsvidhya.com/blog/2018/02/pytorch-tutorial/
https://codetolight.wordpress.com/2017/11/29/getting-started-with-pytorch-for-deep-learning-part-3-neural-network-basics/
https://stackoverflow.com/questions/38135950/meaning-of-weight-gradient-in-cnn
https://pytorch.org/docs/master/torchvision/datasets.html#imagefolder
https://github.com/xpzouying/animals-classification/blob/master/animals-classification.py
http://adventuresinmachinelearning.com/convolutional-neural-networks-tutorial-in-pytorch/

Caffe:

http://caffe.berkeleyvision.org/tutorial/
https://github.com/BVLC/caffe
https://github.com/amir-jafari/Deep-Learning/blob/master/Caffe_/3-Create_LMDB/create_lmdb_tutorial.py#L133
http://caffe.berkeleyvision.org/tutorial/net_layer_blob.html

TensorFlow:

https://stackabuse.com/download-files-with-python/
https://docs.scipy.org/doc/scipy/reference/tutorial/io.html
https://www.tensorflow.org/tutorials/

++