Machine Learning Engineer Nanodegree

Capstone Project: Street View House Numbers Recognition using Deep Convolutional Neural Networks

1. DEFINITION

Domain Background

Optical character recognition (OCR) is highly challenging, but can help solve many problems. One of OCR problems is to identify house number digit classification based on photos from Google Maps. The difficulty arises due to the size of internal features, such as fonts, colors, styles, orientations, and character arrangements, as well as external features, such as lighting, shadows, resolution, motion, and focus blurs.

In this project, we propose using Convolutional Neural Networks[1] to solve real-world image dataset obtained from house numbers in Google Street View images, known as Street View House Numbers (SVHN[2]) dataset. Street View House Numbers (SVHN) is a similar problem as MNIST digits recognition. Given an image, the task is to identify the number in the image.

It is extremely important to have at least human level accuracy. Users of maps find it very time consuming and frustrating to be led to the wrong location, so it is essential to minimize the amount of incorrect transcriptions entered into the map.

Problem Statement

The Street View House Numbers (SVHN)[3] dataset is a real-world image dataset for developing machine learning and object recognition algorithms with minimal

[1] Fukushima, 1980; LeCun et al., 1998

[2] Source: http://ufldl.stanford.edu/housenumbers/

[3] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, Andrew Y. Ng Reading Digits in Natural Images with Unsupervised Feature Learning *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*

requirement on data preprocessing and formatting. It can be seen as similar in flavor to MNIST (e.g., the images are of small cropped digits), but incorporates an order of magnitude more labeled data (over 600,000 digit images) and comes from a significantly harder, unsolved, real world problem (recognizing digits and numbers in natural scene images).

To solve this problem, we will examine the input of SVHN dataset. SVHN has 10 classes (1 for each digit. Digit '1' has label 1, '9' has label 9 and '0' has label 10). The dataset consists of 73257 digits for training, 26032 digits for testing, and 531131 additional, somewhat less difficult samples, to use as extra training data.

Unlike MNIST dataset, SVHN images consist of large variations in font, sizes, color, light conditions, angle, etc. We will preprocess the input data to before we train our covnet model. Example of SVHN images is as follows:

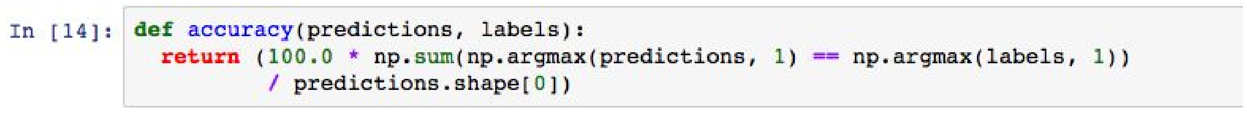
During data preprocessing stage, we will split train and valid dataset, turn images from 3-channel RGB into 1-channel grayscale, and reshuffle dataset in case our train data are sorted in certain orders.

After our dataset is ready, we will build our Covnet model using Tensorflow graphs. For this project, we will examine 3 different models so we can compare which covnet model will perform the best. We will run Tensorflow session to train and test our covnet models. We are going to fine-tune our models using several CNN techniques, such as decay learning rate, dropout, and AdamOptimizer.

Evaluation Metrics

We are going to use “accuracy” as our evaluation metrics. Accuracy is defined in percentage point and calculated based on how well our model can predict the actual result. We deemed the use of “accuracy” metric is more appropriate than “F-measure” because the occurrence of each class label should be well balanced.

We define accuracy method in our model as follows:



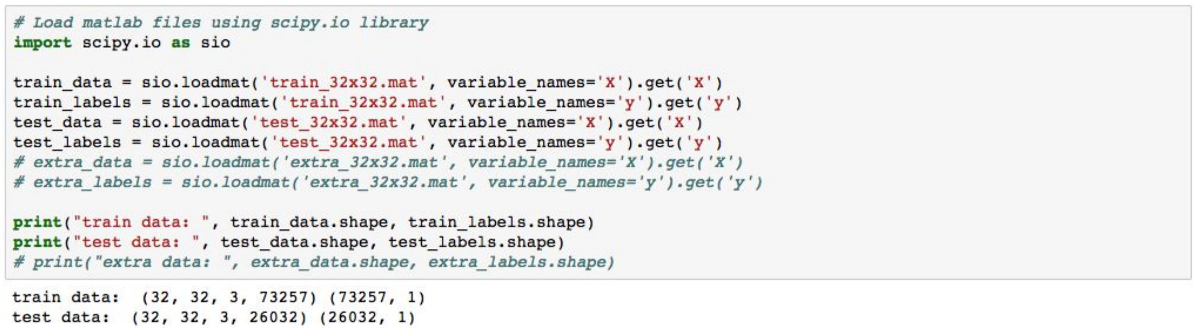
```
In [14]: def accuracy(predictions, labels):
           return (100.0 * np.sum(np.argmax(predictions, 1) == np.argmax(labels, 1))
                   / predictions.shape[0])
```

II. ANALYSIS

Data Exploration

The format and size of Matlab files for the raw cropped SVHN images is as follows:

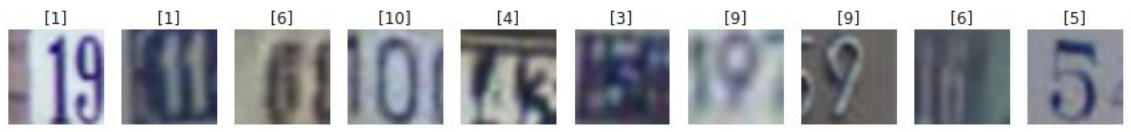
- Each image is cropped into 32x32 pixels format with 3 channels (RGB).
- Train data consists of 73,257 images and labels.
- Test data consists of 26,032 images and labels.



```
# Load matlab files using scipy.io library
import scipy.io as sio

train_data = sio.loadmat('train_32x32.mat', variable_names='X').get('X')
train_labels = sio.loadmat('train_32x32.mat', variable_names='y').get('y')
test_data = sio.loadmat('test_32x32.mat', variable_names='X').get('X')
test_labels = sio.loadmat('test_32x32.mat', variable_names='y').get('y')
# extra_data = sio.loadmat('extra_32x32.mat', variable_names='X').get('X')
# extra_labels = sio.loadmat('extra_32x32.mat', variable_names='y').get('y')

print("train data: ", train_data.shape, train_labels.shape)
print("test data: ", test_data.shape, test_labels.shape)
# print("extra data: ", extra_data.shape, extra_labels.shape)
```

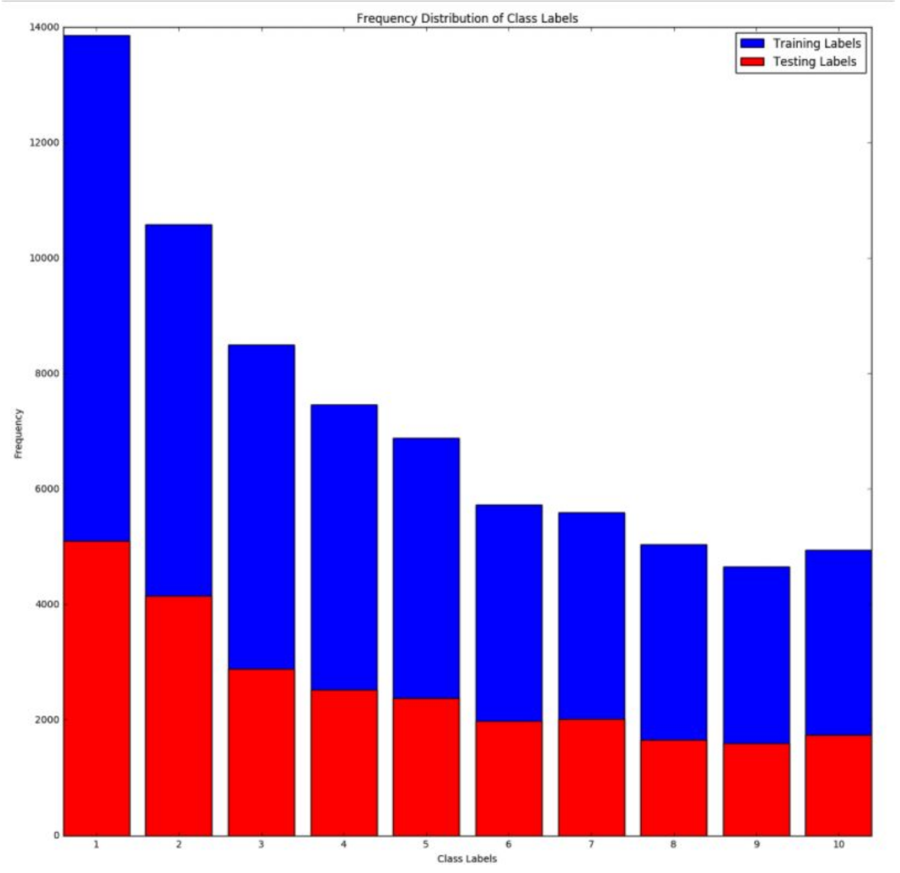
```
train data:  (32, 32, 3, 73257) (73257, 1)
test data:  (32, 32, 3, 26032) (26032, 1)
```

10 samples of raw RGB image with its label on top:



Exploratory Visualization

Looking at the plot of single-digit distribution from the train and test data labels, we can examine the distribution is relatively evenly-distributed for digit 3 to 10. Digit 1 and 2 have slightly higher frequency distribution. The result from bar chart below confirms our selection to use “accuracy” evaluation metric over “F1-score” since the distribution is relatively well balanced. That means the occurrence of each digit to show up has equal probability.

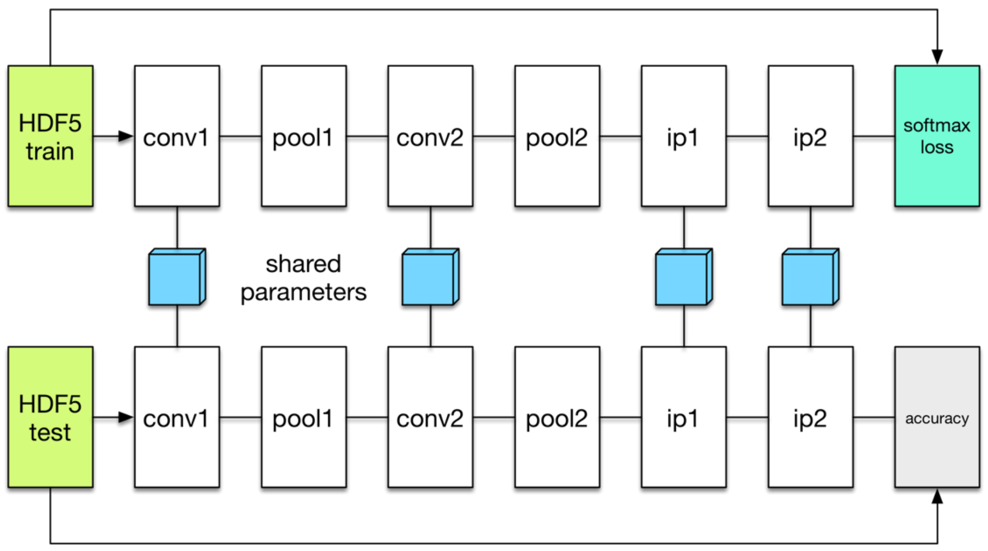


Algorithms and Techniques

The Street View House Number Dataset (SVHN) is similar as MINST Dataset. Machine learning researchers have found that a deep convolutional neural network (also known as Covnet) can be achieve reasonable performance on hard visual recognition tasks.

Deep learning allows computational models that are composed of multiple processing layers to learn representations of data with multiple levels of abstraction. These methods have dramatically improved the state-of-the-art in speech recognition, visual object recognition, object detection and many other domains. Deep learning discovers intricate structure in large data sets by using the backpropagation algorithm to indicate how a machine should change its internal parameters that are used to compute the representation in each layer from the representation in the previous layer. Deep convolutional nets have brought about breakthroughs in processing images, video, speech and audio, whereas recurrent nets have shone light on sequential data such as text and speech[4].

Example of a CNN architecture:



Techniques:

- Convolutional layer: consists of a set of kernels that compute the entries of the filters and inputs and use RELU to activate some specific features.
- Max_pooling: helps to reduce the memory usage after the activations are triggered by the convolutional layer.
- Fully connected: this layer connects all the neurons after several convolutional layers and pooling.
- Dropout: helps to reduce overfitting by eliminating some neurons during training iterations.
- Softmax: is the probability of categorical distribution to predict the output.
- AdamOptimizer optimization: uses moving-averages of the parameters. This optimizer controls the learning rate and it will converge without fine tuning.
- Exponential decay learning rate: to lower the learning rate as the training progresses.

Benchmark

[4] Yann LeCun, Yoshua Bengio & Geoffrey Hinton, Deep Learning, Nature May 28, 2015

According to this website[5], the top 5 SVHN dataset benchmark are:

Method	Summary	Error Rate
Generalizing Pooling Functions in CNN: Mixed, Gated, and Tree[6]	Seek to improve deep neural networks by generalizing the pooling operations.	1.69%
Competitive Multi-scale Convolution[7]	Introduce a new deep convolutional neural network (ConvNet) module that promotes competition among a set of multi-scale convolutional filters.	1.76%
Recurrent CNN for Object Recognition[8]	Demonstrate the advantage of the recurrent structure over purely feed-forward structure for object recognition.	1.77%
Batch-normalized Maxout Network in Network[9]	Employ maxout MLP to learn a variety of piecewise linear activation functions and to mediate the problem of vanishing gradients that can occur when using rectifier units.	1.81%
Deeply-Supervised Nets[10]	Simultaneously minimizes classification error while improving the directness and transparency of the hidden layer learning process.	1.92%

The above benchmark uses “error rate” evaluation metric. However, we use “accuracy” evaluation metric. To compare our model performance with the above metrics, one can convert error rate to accuracy by using a simple formula:

accuracy = 1 - error rate

For example, The above benchmark method (“Generalizing Pooling Functions in CNN:

[5] http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html#5356484e

[6] Chen-Yu Lee, Patrick Gallagher, Zhuowen Tu, AISTATS 10 Oct 2015

[7] Zhibin Liao, Gustavo Carneiro, asXiv 18 Nov 2015

[8] Ming Liang, Xiaolin Hu, CVPR 2015

[9] Jia-Ren Chang, Yong-Shen Chen, arXiv 2014

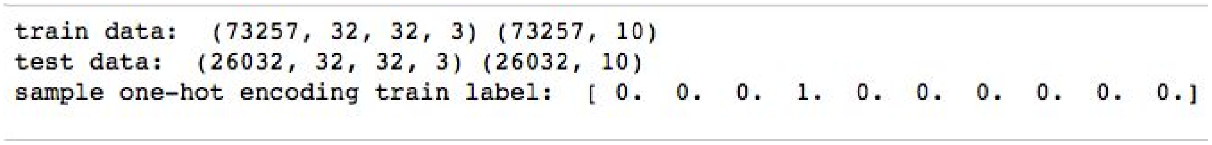
[10] Chen-Yu Lee*, Saining Xie*, Patrick Gallagher, Zhengyou Zhang, Zhuowen Tu, arXiv, 2014

Mixed Gated, and Tree") has the accuracy of 98.31 (= 100% - 1.69% error rate).

III. METHODOLOGY

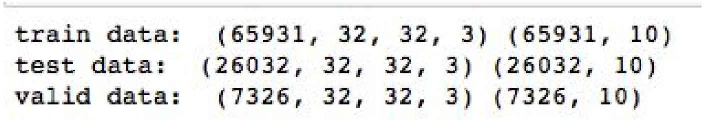
Data Preprocessing

1. Normalize train and test data
2. Apply one-hot encoding to train label



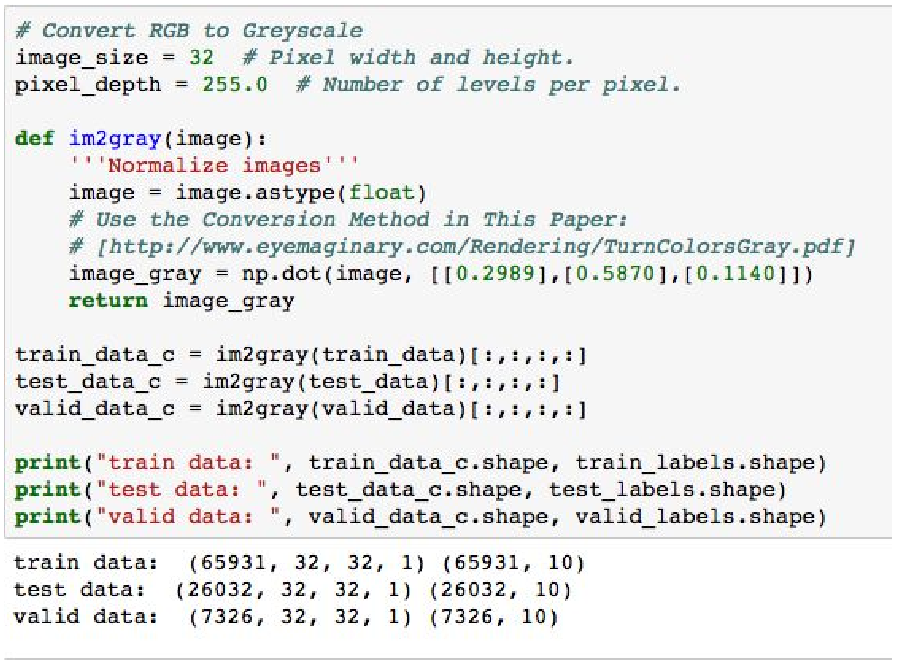
```
train data:  (73257, 32, 32, 3) (73257, 10)
test data:  (26032, 32, 32, 3) (26032, 10)
sample one-hot encoding train label:  [ 0.  0.  0.  1.  0.  0.  0.  0.  0.  0.]
```

3. Split train dataset to create a subset of valid dataset
4. Shuffle train, test, and valid data to remove the possibility SVHN images may be in sequential order



```
train data:  (65931, 32, 32, 3) (65931, 10)
test data:  (26032, 32, 32, 3) (26032, 10)
valid data:  (7326, 32, 32, 3) (7326, 10)
```

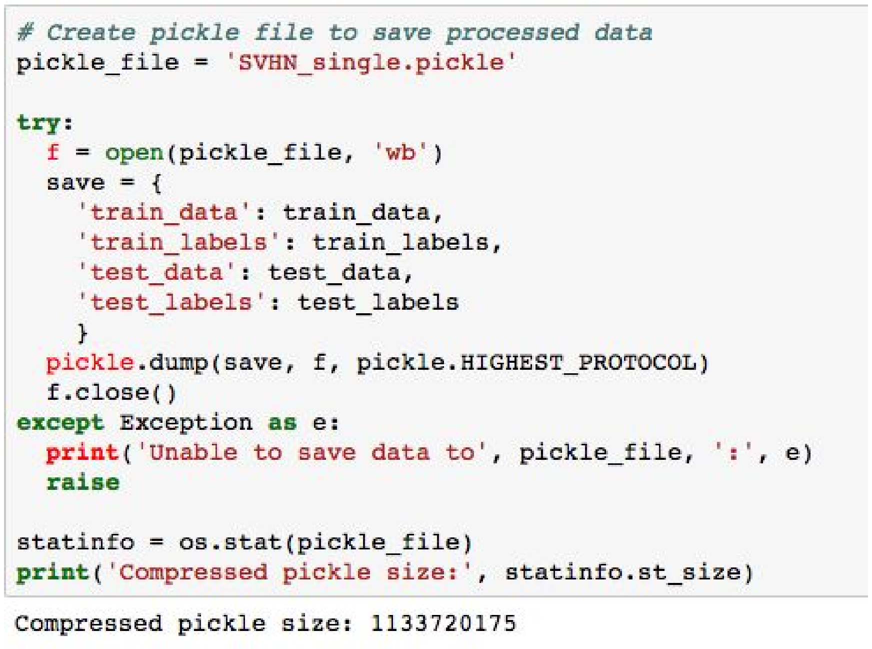
5. Convert RGB (3 channels) to grayscale (1 channel)



```
# Convert RGB to Greyscale
image_size = 32  # Pixel width and height.
pixel_depth = 255.0  # Number of levels per pixel.

def im2gray(image):
    '''Normalize images'''
    image = image.astype(float)
    # Use the Conversion Method in This Paper:
    # [http://www.eyemaginary.com/Rendering/TurnColorsGray.pdf]
    image_gray = np.dot(image, [[0.2989],[0.5870],[0.1140]])
    return image_gray

train_data_c = im2gray(train_data)[:,:,:,:]
test_data_c = im2gray(test_data)[:,:,:,:]
valid_data_c = im2gray(valid_data)[:,:,:,:]

print("train data: ", train_data_c.shape, train_labels.shape)
print("test data: ", test_data_c.shape, test_labels.shape)
print("valid data: ", valid_data_c.shape, valid_labels.shape)
```

```
train data:  (65931, 32, 32, 1) (65931, 10)
test data:  (26032, 32, 32, 1) (26032, 10)
valid data:  (7326, 32, 32, 1) (7326, 10)
```

6. Save preprocessing dataset into pickle file



```
# Create pickle file to save processed data
pickle_file = 'SVHN_single.pickle'

try:
  f = open(pickle_file, 'wb')
  save = {
    'train_data': train_data,
    'train_labels': train_labels,
    'test_data': test_data,
    'test_labels': test_labels
    }
  pickle.dump(save, f, pickle.HIGHEST_PROTOCOL)
  f.close()
except Exception as e:
  print('Unable to save data to', pickle_file, ':', e)
  raise

statinfo = os.stat(pickle_file)
print('Compressed pickle size:', statinfo.st_size)
```

```
Compressed pickle size: 1133720175
```

Implementation

We implemented 3 CNN models and compare the results as follows:

- Model 1[11]: Conv1/Pool1/Conv2/Pool2/FC1/dropout/FC2/softmax CNN architecture with grayscale images
- Model 2[12]: Conv1/Pool1/Conv2/Pool2/FC1/dropout/FC2/softmax CNN architecture with RGB images
- Model 3[13]: Conv1/Pool1/Conv2/Pool2/Conv3/Pool3/FC1/dropout/FC2/softmax CNN architecture with RGB images

Model 1 above is our base model. The input is 1-channel grayscale SVHN images.

The CNN architecture base model is as follows:

1. First convolutional layer with 32 filters of 5x5 pixels and add 32 biases
2. Max_pooling with 2x2 pixels
3. Second convolutional layer with 64 filters of 5x5 pixels and add 64 biases
4. Max_pooling with 2x2 pixels
5. First fully connected layer to convert 4,096 nodes into 1,024 nodes
6. Dropout 50% of nodes
7. Second fully connected layer to convert 1,024 nodes into 10 nodes
8. Softmax layer

---

[11] Model 1 Github repo: SVHN+Capstone_single+digit-greyscale.ipynb

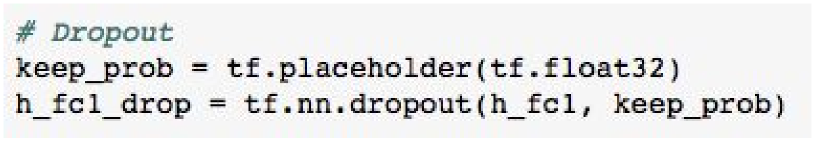
[12] Model 2 Github repo: SVHN+Capstone_single+digit-RGB.ipynb

[13] Model 3 Github repo: SVHN+Capstone_single+digit-RGB-deeper+layer.ipynb

Refinement

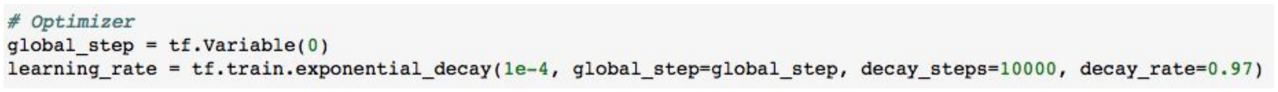
To refine our data, I used the following techniques:

- Set 50% dropout to allow more degree of freedom



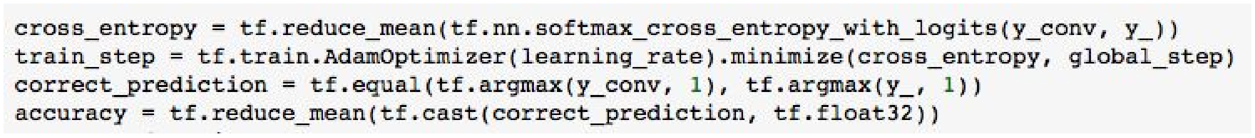
```
# Dropout
keep_prob = tf.placeholder(tf.float32)
h_fc1_drop = tf.nn.dropout(h_fc1, keep_prob)
```

- Set exponential decay learning rate



```
# Optimizer
global_step = tf.Variable(0)
learning_rate = tf.train.exponential_decay(1e-4, global_step=global_step, decay_steps=10000, decay_rate=0.97)
```

- Use AdamOptimizer instead of Gradient Descent



```
cross_entropy = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(y_conv, y_))
train_step = tf.train.AdamOptimizer(learning_rate).minimize(cross_entropy, global_step)
correct_prediction = tf.equal(tf.argmax(y_conv, 1), tf.argmax(y_, 1))
accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.float32))
```

- Deeper CNN architecture (by adding one more convolutional layer as seen in model 3)

IV. RESULTS

Model Evaluation and Validation

Model	Test Accuracy
Conv1/Pool1/Conv2/Pool2/FC1/dropout/FC2/softmax CNN architecture with grayscale images	17.04%
Conv1/Pool1/Conv2/Pool2/FC1/dropout/FC2/softmax CNN architecture with RGB images	88.93%
Conv1/Pool1/Conv2/Pool2/Conv3/Pool3/FC1/dropout/FC2/softmax CNN architecture with RGB images	86.90%

Our model 1 using grayscale images produced very poor result with test accuracy of 17.04%. During early iterations, both training and validation accuracy seems to be stuck at 20%. Toward the end of iterations, the training accuracy improved slightly to 30%, while validation accuracy dropped to 17%. The final test accuracy is similar to validation accuracy at 17%. We can see there is a sign of over-fitting as our training accuracy improved slightly, but both validation and test accuracies perform poorly.

We obtain a significant improvement in model 2 by changing the raw input from 1-channel greyscale to the original 3-channel RGB images. The test accuracy using model 2 jumped to 88.93% from 17.04% from model 1. At the early training iterations, model 2 managed to quickly achieve training accuracy around 90% and validation accuracy around 80%. Toward the end of iterations, the training accuracy achieve 100% and validation accuracy around 91%. Our final result with test accuracy is 88.93%. Despite the significant improvement from model 1, our model 2 shows a strong sign of overfitting.

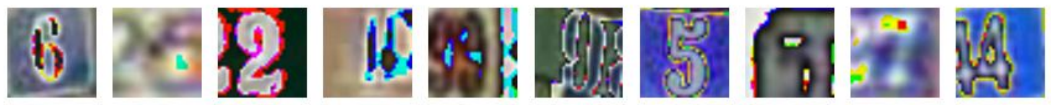
Our model 2 result shows satisfying improvement, but still 10 points below our benchmark. In our attempt to obtain a better test accuracy, we build model 3 by adding another convolutional layer. The CNN architecture of our model 3 is as follows:

1. First convolutional layer with 16 filters of 5x5 pixels and add 16 biases
2. Max_pooling with 2x2 pixels
3. Second convolutional layer with 32 filters of 5x5 pixels and add 32 biases
4. Max_pooling with 2x2 pixels
5. Third convolutional layer with 64 filters of 5x5 pixels and add 64 biases
6. Max_pooling with 2x2 pixels
7. First fully connected layer to convert 4,096 nodes into 256 nodes
8. Dropout 50% of nodes
9. Second fully connected layer to convert 256 nodes into 10 nodes
10. Softmax layer

Model 3 shows that our overfitting problem in model 2 is no longer presence. Model 3 training accuracy is able to achieve 89%, while the validation accuracy follows closely at 88.38%. However, the test accuracy shows a small drop to 86.9% in model 3, compared to 88.93% in model 2.

Justifications

1. Although model 2 achieved the highest test accuracy at 88.93% test accuracy, we will argue that model 3 is the best model since it eliminates overfitting problem that appeared in model 2.
2. We were surprised by our model 1 due to very poor performance. The test accuracy for the same CNN architecture using the grayscale images is very low at 17.04%. The low quality of grayscale (1 channel) images could be the root cause of the low test accuracy. Upon closer examination on 1-channel grayscale images, we observed these images are barely readable by human. The poor test accuracy shows that computer is unlikely able to read these images as well.



3. Despite the deeper CNN architecture, model 3's test accuracy of 86.90% is no better than that for model 2. We assume deeper neural network can improve test accuracy as our network can add more information to find patterns. We wary that adding deeper neural network may increase overfitting problem. However, the result contradicts our assumption. The results suggest that the strength of deep learning may arise in part

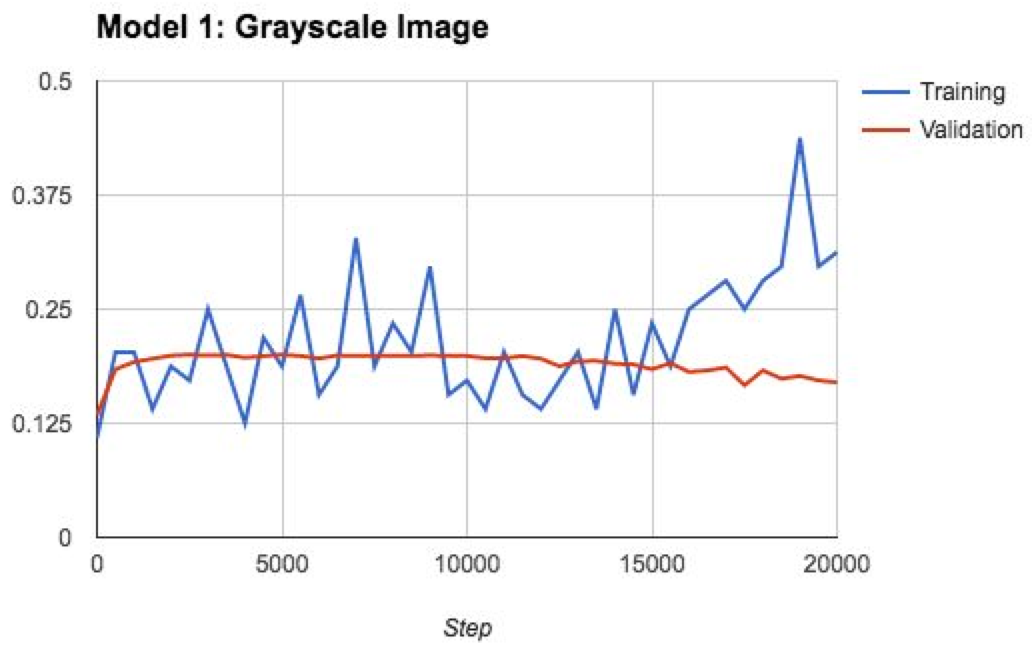
from a good match between deep architectures and current training procedures[14].

V. CONCLUSIONS

Reflections

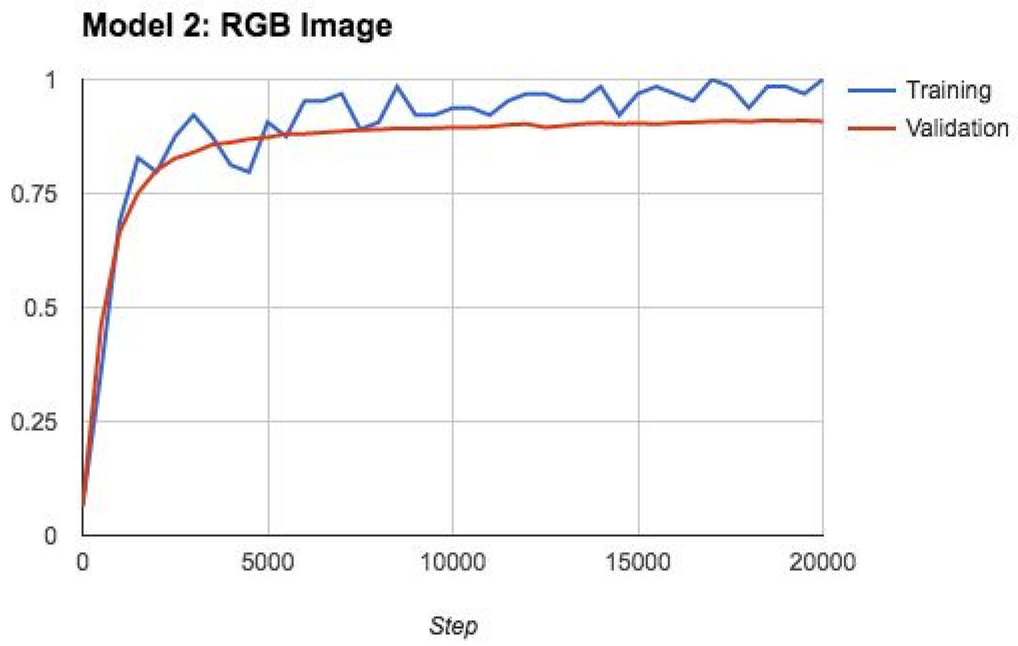
In this project, we examine SVHN dataset and apply convolutional neural network to predict the digits from the images. Researchers have experimented different CNN techniques and achieved ~98% accuracy, which is higher than human's accuracy at ~90%. In this project, we tried 3 different models using SVHN dataset and the best result is 88.93%, which is close to human's level of accuracy. While the result is satisfactory, it's still below the benchmark which apply more sophisticated techniques and powerful GPUs.

The training and validation accuracy from the three models are plotted below:

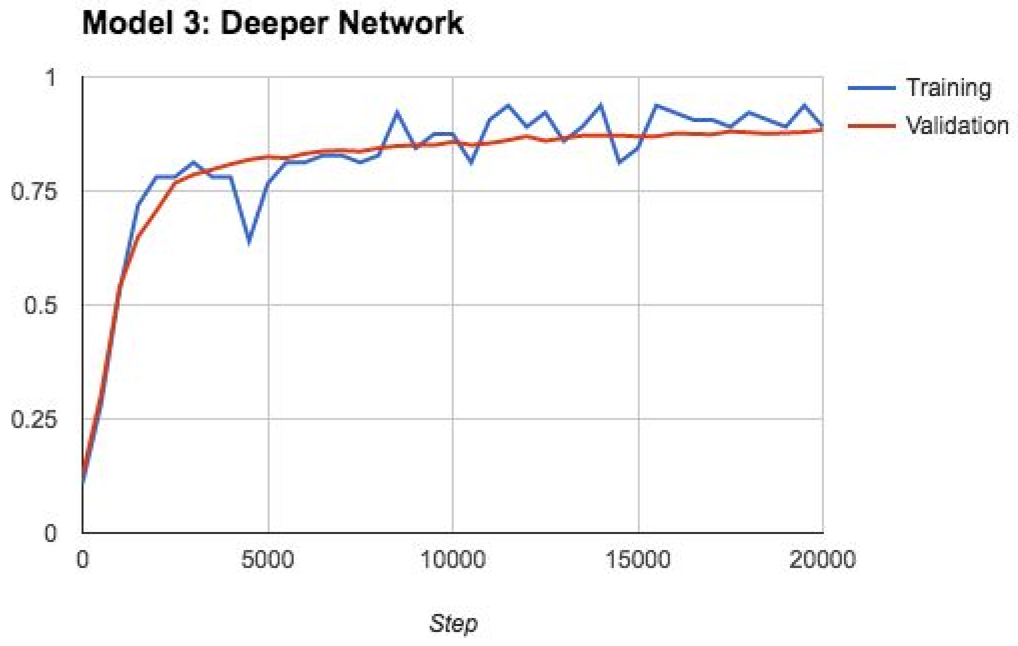


Model 1 is our worst performer model. As we can see from the chart above, the training and validation accuracies are below 40%. Our model 1 applies 5-layer covnet with optimization techniques, such as decay learning rate and dropout. The low accuracies may result from the poor input into our model using 1-channel grayscale. As mentioned above, the grayscale input smooshed the colors. Computer, just like human eyes, may have difficulty to understand the input as the edge detections and contours of the image are not visible.

[14] Lei Jimmy Ba and Rich Caruanna, "Do Deep Nets Really Need to be Deep?", University of Toronto, 11 Oct 2014



Model 2 above shows much improvements in training and validation accuracies. The covnet architecture and optimization techniques emulate those from Model 1. The above chart shows that the original 3-channel RGB images are much better input for our model. While the training and validation accuracies improve smoothly as number of iterations, we notice the training accuracy hit 100% in step 17,000 and 20,000. However, the validation accuracy seems to reach the maximum at around 90%. Therefore, we may conclude this model 2 may have overfitting problem, which translate to highly accurate prediction on training dataset but perform slightly poorer on validation and test dataset.



In model 3, we added one extra convolutional layer from model 2 covnet architecture and use the same 3-channel RGB input. The plots on the training and validation data seems to move at the same rate. Despite slightly lower validation accuracy result, ie. 88.38% from model 3 vs 90.81% from model 2, we have much higher confidence our model 3 due to lack of overfitting evidence in the training accuracy.

Improvements

These are some of the improvements we can think of to solve this problem:

- Due to hardware and computation power limitations, it took a long time to run an epoch of training data. Therefore, we couldn't test more sophisticated models, such as ResNet, GoogLeNet, VGGNet. We can improve the results if we use higher GPU power by removing the memory bottleneck.
- We can also experiment with image preprocessing. In this project, we use im2gray method to smoosh the 3-channel RGB image into 1 channel. There are many other preprocessing techniques that can be applied to improve the result, such as edge detection, Gaussian blurring, rotations and contour selections. Scikit-learn, Numpy, OpenCV, and Matplot libraries have the built-in functions to manipulate the images to make the image more readable by computers.
- Tensorflow library can now be embedded into mobile devices and Raspberry-pi IoT devices. We can implement our trained model into mobile and IoT devices with camera to detect images and videos and create an AI application using computer vision using real-time data.