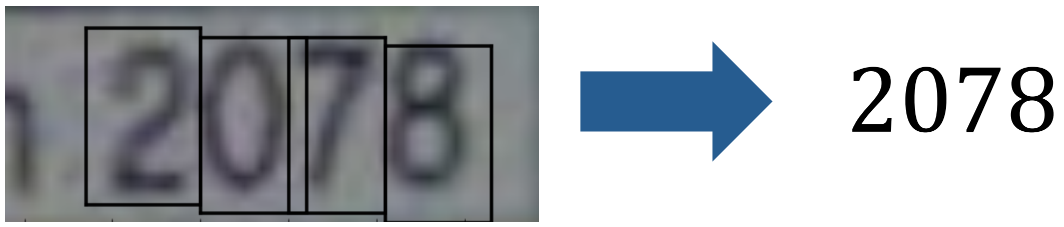
Deep Learning Capstone

Machine Learning Engineer Nanodegree

Scott Burns
November 1, 2016

Definition

In this project, I built and trained several multi-layer neural networks that can accurately label a sequence of digits captured in an image. The networks receive a picture of numbers and predict the actual numbers that the picture represents, as indicated in the figure below.



With my final network, I aimed to achieve over **95% accuracy** in predicting image labels, which represents a substantial improvement on the predictive power of 'off the shelf' machine learning tools. In their work on similar challenges, Google researchers have achieved 96% accuracy in recognizing complete street view address sequences (see Goodfellow et al 2014), and I have hoped to approach this target, while recognizing constraints in the computing power and time resources I have available to build predictive networks.

To clarify my accuracy target – if my model is presented 100 pictures with sequences of digits of length 2 to 5, I look to have it correctly recognize the number of digits shown in total for the sequence, as well as which digits are shown, for 95 of those 100 pictures. For example – if shown an image of the address sequence 2165, a label prediction from the network of 3165 would not be receive an accuracy score of 75%, *it would receive an accuracy score of 0%.*

Building the models presented in this project, I draw from two widely used datasets with letters and numbers represented in images with a variety of shapes and skews. Each dataset offers labels for each character image or sequence of characters image that we can use to create associations in our model between image characteristics and the actual characters in the image.

notMNIST dataset example – ***the letter D***

SVHN dataset example - ***the number 260***

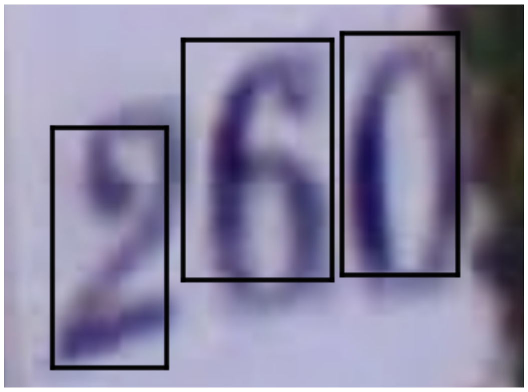


Image recognition is a common application of machine learning and neural networks, with digit recognition being one of the most typical objectives in using this technology. The field is widely covered in academic and commercial research on neural networks because it has high relevance to significant real-life problems. Through image recognition technology, we are now able to more seamlessly deposit checks (with ATMs able to read handwritten numbers), track down people running red lights (reading license plates from stoplight cameras), and convert documents to digital format.

This project draws on notable work researchers have done in the field of image recognition, as well as open-source projects with similar aims.

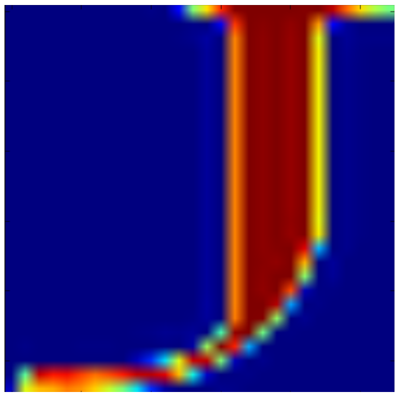
Over several runs my designed networks reached over 95% accuracy in predicting test set labels, while demonstrating that multi-layer convolutional networks do provide a substantial boost to digit recognition accuracy over simple fully-connected networks, with a standard 3-layer convolutional network adding between 4 and 5% percentage points in testing accuracy after 100,000 step training runs based on my observations.

Analysis

In preliminary research and subsequent training and testing, I used two datasets:

- **notMNIST**: a set of images made available by Yaroslav Bulatov, which includes 500,000 32x32 visual representations for the letters A – J with a wide variety of letter shapes, along with a label for each letter. Description and data available here notMNIST dataset example – ***the letter E***

notMNIST dataset example – ***the letter J***



As part of initial exploration, and preparation for this capstone as part of the Udacity *Deep Learning* course, I prepared training, validation and testing datasets from notMNIST 'large' and 'small' folders – building a shuffled set of 200,000 images for training and 10,000 for validation from *notMNIST_large.tar.gz* and 10,000 images for testing from *notMNIST_small.tar.gz.*

After preprocessing, the stored feature datasets can be summarized in the following tables. As expected, mean and standard deviation statistics for each dataset remain around 0 and 0.45 respectively.

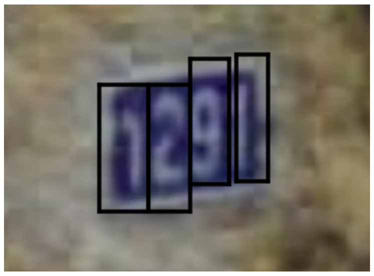
NotMNIST dataset summary statistics

	Rows	**Mean**	**Standard Deviation**
Train	200,000	-0.082	0.454
Validate	10,000	-0.075	0.458
Test	10,000	-0.083	0.453

Labels for each are of dimension 10 - with index number corresponding to the letter in the sequence from A to J. We note that labels are roughly evenly distributed across the classes, at between 52910 and 52912 for each letter.

- **Street View House Numbers (SVHN):** images of physical address markers from houses as collected by Google's Street View service. Includes over 600,000 total images, with 73,257 for training, 26,032 for testing and over 531,000 'extras' for additional training. Images in the set include a bounding box around each presented digit, which allows us to normalize their associated visual data. Description and data here

SVHN dataset example – ***1291***



SVHN dataset example – ***312***

As part of preprocessing the streetview data, which are offered in Matlab *.mat* format, I referenced code used by researchers working with SVHN data (Brian Lester and Hang Yao) and prepared a dataset for training and testing which can be summarized in the following table.

SVHN dataset summary statistics

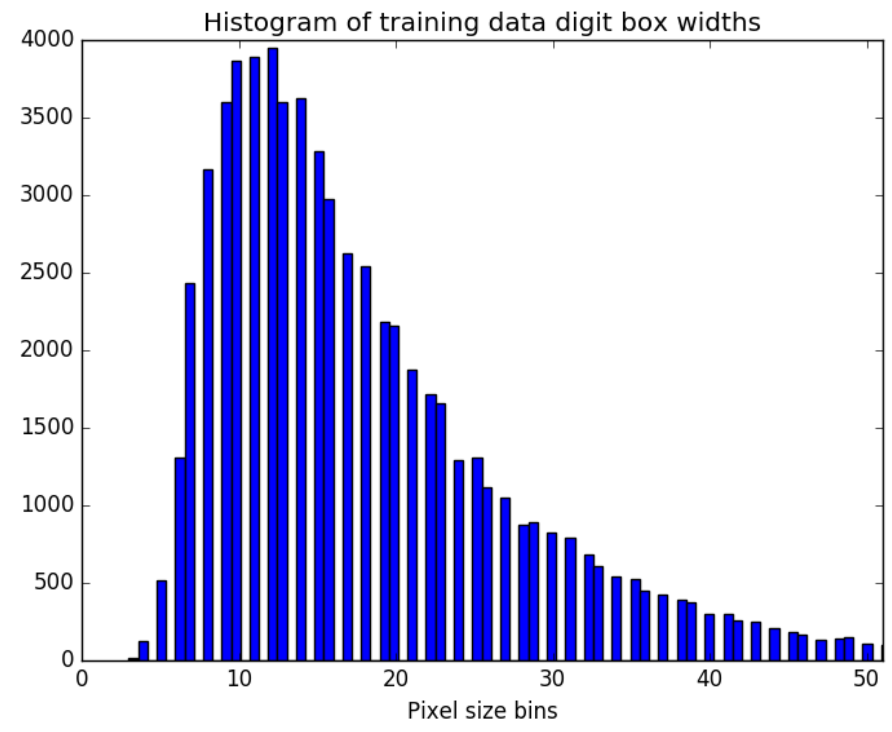
	Rows	**Average Digits**	**Digit Distribution**				
			1	2	3	4	5
Train	230,070	2.57	13,534	86,866	113,935	15,613	122
Validate	5,684	2.15	988	2,990	1,545	159	2
Test	13,068	1.99	2,483	8,356	2,081	146	2

Luckily, because the dataset has been so widely used and vetted, it is remarkably clean, and requires minimal wrangling to prepare it for training our model. As a small update to training labels and the corresponding dataset elements, I deleted a row listed as having 6 digits, as we only have 5 digits in our label array. No array elements were found to be blank during summary analysis.

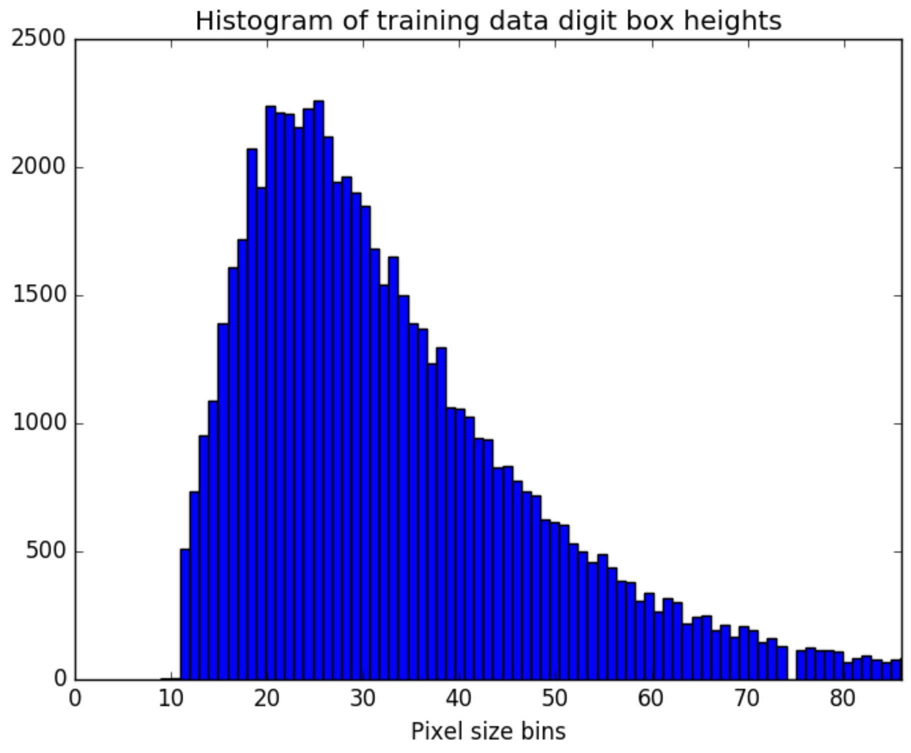
I also performed various spot checks of the dataset, viewing a variety of images and sampling various label arrays from the dataset to ensure that the listed number of digits in the first array position were consistent with the actual digits shown in later label array elements. I didn't uncover other major issues requiring resolution as part of this project's scope.

To better understand the images used in training, I explored the distribution of digit box dimensions, and looked to represent this distribution in a pair of histograms, along with summary statistics.

The widest digit box of each image sequence in SVHN data averages about 19 pixels, but their values are skewed higher, with some boxes extending to 207 pixels, and the 98th percentile falling at 51 pixels (the x-limit of the graph below).



The tallest digit boxes of each image sequence in SVHN data are distributed similarly, with an average height of 34, but a max height of 403. Our graph below shows heights up to 86 – the 98th percentile for digit image boxes.



The final model used for training on SVHN data and testing on arbitrary images applies deep neural networks and convolutional neural networks to form associations between our image data and label data. It passes feature data through 3 convolutional layers with a patch size of 4, strides of [1,1,1,1] and max_pooling, then a fully-connected neural network of 3 hidden layers and 1 output layer using rectified linear units.

The final neural network used for the visualization in the previous section incorporates a number of techniques to improve model predictive accuracy and avoid overfitting on test datasets, including L2 regularization, dropout, local response normalization, Xavier Initialization and Gradient Descent Optimization, with a learning rate that decays over time.

Descriptions of the techniques uses and their justification are featured below

Technique	**Description / Justification**
Convolutional neural network (Convnet) layers	Tiles of input from an image, linked to carry output from a region to the next network layer, shifting network dimensions from 'wide' to 'deep'. Convnet layers help to ensure translational invariance in images and help to capture grouped features and feature hierarchies, with neurons from common regions sharing weight and bias parameters [Reference: Deep Learning Udacity course]
Max pooling in convolutional layers	Downsampling of an image by taking the maximum values from non-overlapping regions for activating nodes in the next layer of the network [Reference: Stanford CS231n]
Fully connected neural network layers	Layers of neurons receiving image input, from which each layer neuron output is received by each neuron in a following layer as input. This process is repeated until output layer values are generated and compared against actual dataset labels to assess a loss function that will be minimized as part of the learning process. [Reference: Stanford CS231n]
Rectified linear units (RELUs)	Activation units defined by the function $f(x) = \max(0, x)$. Introduced to receive input for neurons as part of a fully-connected layer, these introduce non-linearities that have been shown to improve network performance [Reference: Nair and Hinton 2010]
Tensorflow batch training	Processing of data subsets to generate loss function values to optimize in the training process and testing against validation sets for tuning hyperparameters [Reference: Tensorflow tutorial]
L2 regularization	A term in the loss function, adding the sum of the squared layer weight values. Including L2 regularization helps prevent overfitting by ensuring that any large weights contribute substantially to model performance. [Reference: Neural Networks and Deep Learning book]
Dropout	The process of randomly excluding a subset of activations during the training process. Through dropout a network is forced to learn redundant representations of image data, making it more robust and less prone to overfitting. [Reference: Deep Learning Udacity course]
Local response normalization (LRN)	Process of normalizing a neuron's activation value by the activation value of adjacent neurons. This helps highlight neurons that contrast with their 'neighborhood' and reduce the importance of those which are uniform over a layer region. In widely referenced studies, like the reference here, LRN has been shown to increase networks' predictive accuracy. [Reference: Hinton et al 2012]

Technique	Description / Justification
Gradient Descent	An iterative process to find a local minimum, by calculating the gradient at a specific point in a function, then applying a given 'step-size' to the negative of the gradient, thus moving 'most steeply' toward the minimum. For neural networks we apply Gradient Descent to minimize our loss function with training data. Over many iterations we adjust weights and biases for the network to come close to minimizing the loss function. [Reference: Machine Learning for Mastery]
Loss function with sparse softmax cross entropy	The *sparse_softmax_cross_entropy_with logits,* function in Tensorflow calculates the sparse softmax cross entropy between the logits created from training data and dataset label values. The function operates on unscaled logits and doesn't take values already transformed through the *softmax* function. As the project here requires finding the summed entropy for 6 classes (5 digits and 1 sequence length indicator), this function is more appropriate than Tensorflow's *softmax_cross_entropy_with_logits* function. [Reference: Tensorflow Github README]
Xavier initialization	Method to initialize weights appropriately, so that signals are fully passed through the network. Under it, the variance of the distribution of each neuron in a layer is inversely proportional to the number of neurons feeding into it. This approach prevents cases in which the initialization variance is set too low and the signal shrinks until it's too small to be useful in the output layer, as well as the case in which the distribution variance is initialized too low the signal becomes to large to effectively generalize in the model. The name of the process is taken from the author of the paper proposing the process. [Reference: good overview blogpost]

Key parameters for model features are outlined below.

Technique	Key parameter	Theoretical impact
Convolutional layers	Patch size: 4 Depth of layers 1 – 3: 16, 32, 64 Convolutional stride: [1,1,1,1] Max pooling strides: [1,2,2,1] K- Size: [1,2,2,1]	Convolutional layers help to ensure translational invariance and feature hierarchies in digit images. Smaller patch size chosen for greater resolution in network, which also requires greater computing resources.
Local response normalization (LRN)	LRN alpha: 0.1 LRN beta: 0.75	Inclusion of LRN helps to prevent overfitting and ensure robustness in trained models. Parameter values were initially chosen to match those used by Hinton et al 2010, but updated after finding better results with a slightly larger beta.
Hidden full-connected layers	Nodes of hidden layers 1 – 3: 128, 64, 16	The architecture allows the network to train complex, non-linear models, with 3 hidden layers, a large number of initial nodes in the first, and RELU activation functions.
Loss function	L2 Beta: 0.005 Learning decay rate: 0.98 Initial learning rate: 0.05	Including L2 losses helps us avoid overfitting, and the final L2 beta value was selected after trying a range of values in early tests. The relatively low initial learning rate was chosen to avoid hitting a

Technique	Key parameter	Theoretical impact
		premature local minimum as part of our Gradient Descent. This choice also causes our network to achieve higher validation accuracy rates more slowly.
Dropout	Keep rate: 0.8	Avoids overfitting and increases network robustness through requiring replication redundancy. Parameter determined through trial-and-error.
Batch session feeding	Batch size: 100	Batch size chosen after trial runs with various sizes to balance required training periods with trained network accuracy.

As discussed in the project overview, I targeted a classification **accuracy rate of at least 95%** from my final model. To re-iterate – my measure of accuracy would require 95% of digit sequences to be correctly labeled *in totality* – any mislabeled digit in a sequence would require the entire sequence to be ruled inaccurate.

While this benchmark is somewhat arbitrary, I saw it as a target that was reasonable under my resource constraints, reflective of performance significantly better than could be achieved with 'off-the-shelf' machine learning tools, and relatively close to Google research teams' model performance of 96% - using the same metric of accuracy I have adopted as a gauge of my network's performance.

To justify the 'off-the-shelf' performance comparison, I reference the outcomes I achieved in training a simple logistic regression model on *notMNIST* data as part of the Deep Learning Udacity course. Using a LogisticRegression classifier from the *Sklearn* library without any parameter tuning, and under 20 lines of code applied to a training set of 20,000 examples, I was able to label individual letters from letter images with 89% accuracy. Using the same techniques on SVHN data would likely yield lower accuracy, given its multi-digit classification requirements, and the natural, more arbitrary nature of the dataset's images.

The difference between simpler single layer neural networks (roughly equivalent to logistic regression) and multi-layer networks is also apparent in the testing I conducted to produce my concluding visualization (see Conclusion). While it is omitted from the final chart because it falls outside the chart ranges that fit for other layer levels, a single-layer neural network achieves only 60% accuracy after 100,000-step training.

Similarly, a review of published work and results achieved by other Udacity participants with this dataset suggests that **95%** represents a non-trivial challenge.

Methodology

Preprocessing the data from the SVHN dataset requires 6 essential steps:

1. Download data in tarball format
2. Extract folders and images from downloaded files
3. Set processed dataset features and labels using raw dataset metadata (including cropping according to bounded boxes)
4. Prepare images for training
    a. Cropping areas beyond a box around all digit boxes
    b. Converting to grayscale
    c. Normalizing all pixel saturation by the mean and standard deviation of saturation for all image pixels
5. Add 'extra' data to processed datasets
6. 'Pickle' (serialize) processed data for easy storage and loading for network training

Through this process we are able to scale our images uniformly and better maintain signals from our data through a multilayer network for training while helping avoid overfitting.

I wasn't familiar with the *h5py* library used in transferring data in-memory and for serialization with pickle, so extracting the data from online sources in *.mat* format required extensive reference to other people who have worked with the SVHN dataset in the past, including Hang Yao – a research also exploring the SVHN data.

After several rounds of network testing at around 20,000 steps, batches of 50 and a series of fully-connected layers, I looked to improve my accuracy scores through increasing training resources through 1) more steps, 2) bigger batches, 3) convolutional layers and 4) more carefully tuned parameters.

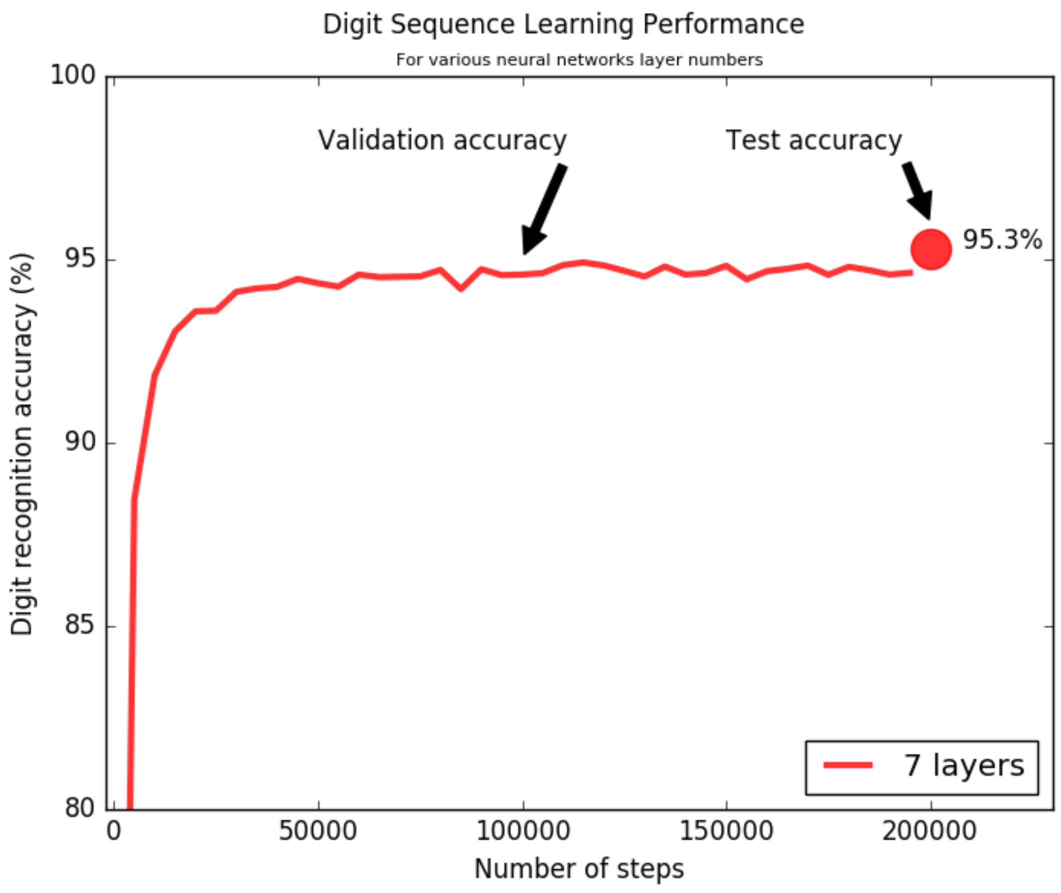
After adding convolutional layers and trying a range of dropout `keep_rates`, learning rate exponential decay and initial parameters, L2 loss beta and patch size parameters, at tests ranging form 20,000 to 50,000 steps, I decided on the values used in my ultimate model, with an L2 beta of 0.0005, an initial learning rate of 0.05 with 0.98 exponential decay, and a 0.8 keep rate for dropout. These parameters allowed the model to achieve the test accuracy scores above 95% with training periods of 100,000 steps and greater, as pictured in the charts below.

Similarly, adding greater training intensity and resolution with features like greater batch sizes (increasing from 20 to 50 to 100) and smaller patch sizes (down from 5 to 4) elevated test performance.

To satisfy a curiosity about the impact of additional layers and convolutional layers on test accuracy, I set up a loop in my script to train and test at total network layer counts of 2 through 7. Weights and variable dimensions were correspondingly shifted at each stage to accommodate changing total layer values. The chart in my conclusion shows the impact of additional layers, which is significant, particularly with convolutional layers being added (solid lines – with 5 – 7 layers). With these layers test accuracy rises from under 90% to over 95% for 100,000 step training sessions as seen in my conclusion section.

Results

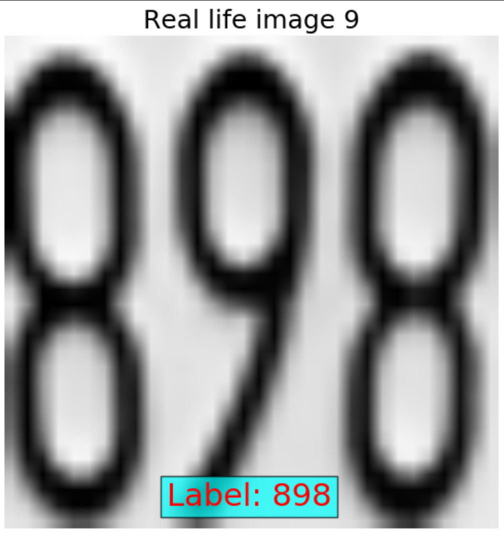
The final model was chosen for the theoretical benefits of included features, as well as the increasing performance I found in providing complexity like convolutional layers, an exponentially decaying loss functions and more computational power with deeper depths, larger batch sizes and smaller patch sizes.

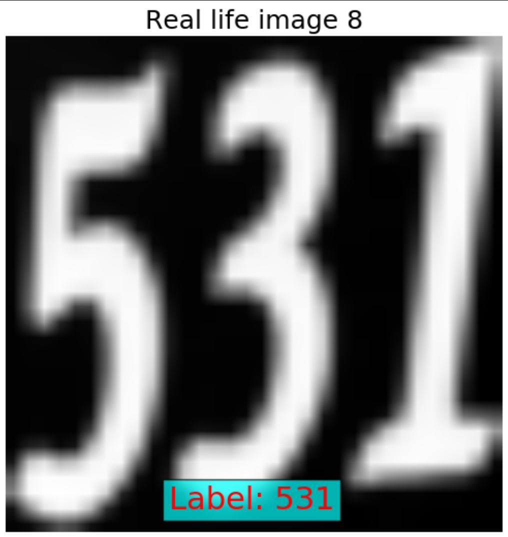


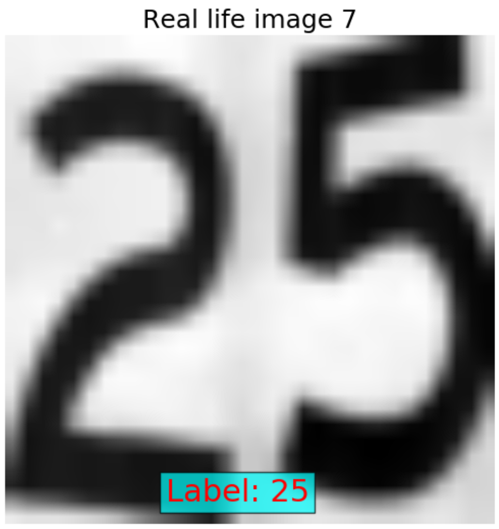
Besides testing the model against data in the SVHN test set, I also chose to feed 'real-life' data I gather by photographing numbers displayed in my neighborhood – including addresses, business phone numbers, handwritten notes and license plates. When testing a

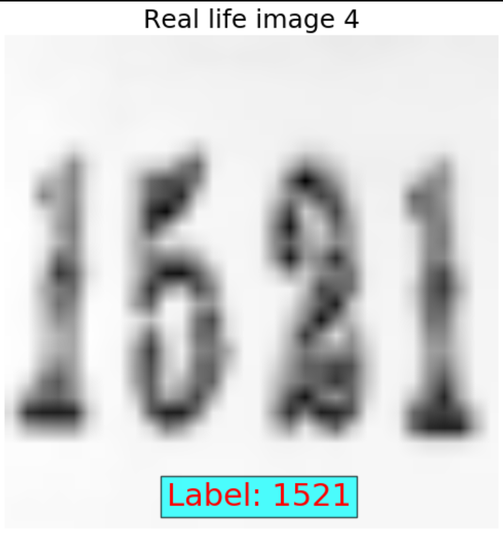
7-layer model trained over 200,000 steps, I found that out of 10 number images - ranging from 2 to 4 digits - the model was able to correctly identify 4 of them.

Example correctly labeled 'real life' digit sequence images



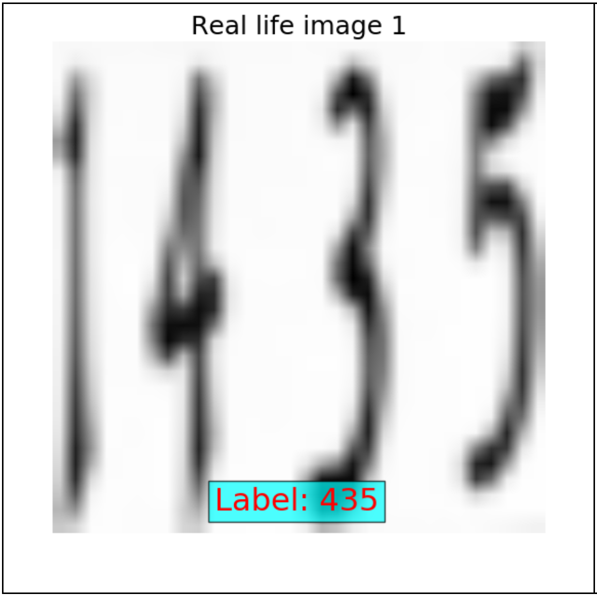


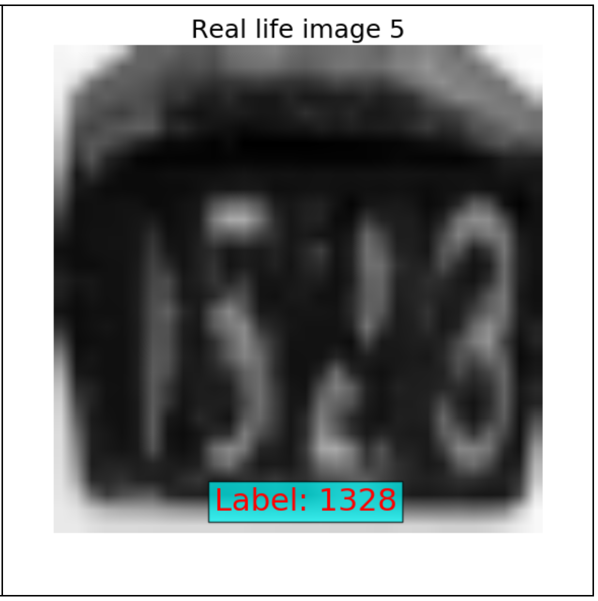




While the accuracy achieved was lower than I had hoped in testing on 'in the wild' data, I also recognize that the presented digits were significantly more challenging than the typical numbers presented in SVHN training data, as well as being potentially less favorably cropped than images from the dataset. In several cases, digits close to the edge of the frame of the image were neglected in the network's predictions, while interior digits were correctly labeled.

Example incorrectly labeled 'real life' digit sequence images





Note that the above digit sequence would correctly be labeled 1435 and 1523

The final model achieves a target accuracy of 95.3% after 200,000 steps - above my benchmark of 95%. While the model doesn't achieve human-level transcription accuracy (Google researchers benchmark this at 98% accuracy in Goodfellow et al 2014), it demonstrates substantial improvement over 'off-the-shelf' algorithms, and could potentially reach 'human-transcription' accuracy more computing power and more time.

In particular, I could see model accuracy improving with the addition of smaller patch sizes in convolutional networks as well as larger node sizes in the hidden layers of the neural network. The changes would lengthen processing time significantly, which in the current 'train_and_tests_svhn_data.py' script already runs for over 7 hours at 100,000 steps.

Network performance would suggest that it has used fairly well-tuned parameters, and can effectively generalize over a large test set that is representative of relatively 'clean' real-life address digit sequences.

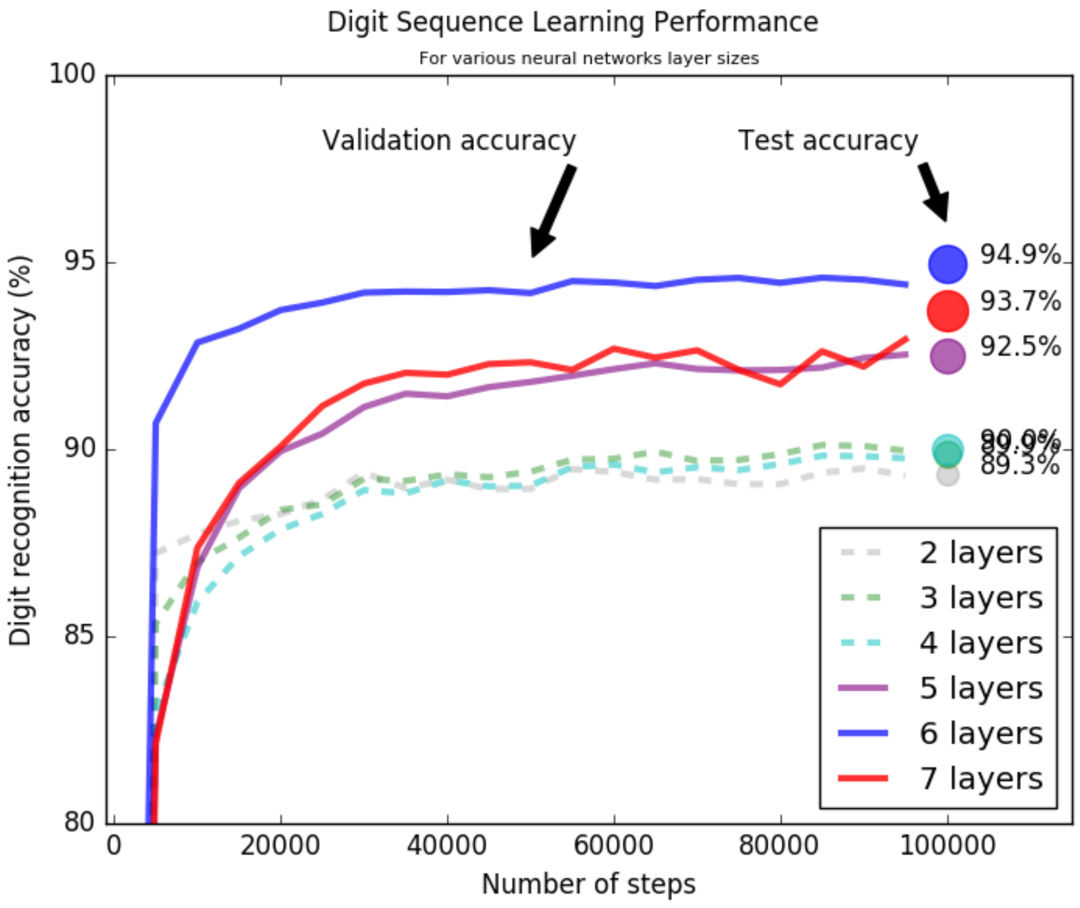
Parameter tuning for the exercise is described in the Refinement section. The train-test sessions for various parameters suggest that the model selected here is among the best specified options possible for the time/computational power that it requires to run. There are likely opportunities to further optimize the model, but controlled parameter variation I did early in my project explorations suggests that significant improvements would require a prohibitive amount of further testing.

This model is robust to restrictions to the training dataset.- as batches are randomly shuffled, and over the course of dozens of trial runs, networks with similar parameter values achieved consistent test accuracy values.

Conclusion

Concluding this project, I was struck by two findings: 1) complex non-linear neural networks add significant predictive power to image recognition models and 2) machine learning technology that was recently only conceptual or restricted to a very small part of the population is now available to amateur enthusiasts, along with a wealth of open source examples and references to help them build powerful models.

You can see the power of additional, and convolutional networks in the chart of predictive accuracy over 100,000 steps. While networks composed of multiple fully-connected layers (marked by dotted lines) achieve an impressive 90% test accuracy after this period of training – which represents an achievement that would have been remarkable in academic institutions even a few years ago – it is still far short of the predictive power that networks with convolutional layers achieve, reaching over 95% test accuracy.



A useful baseline to consider for comparison is our basic 2-layer fully-connected (non-convolutional) network, marked with gray dotted lines in the figure above, which features node sizes of 128 and 64 in the connected layers. Note that it achieves accuracy of only

89.3% - further highlighting the enhancements that additional fully-connected and convolutional layers offer in a network. As noted above, I've omitted a chart of a single layer model (of 128 nodes) I also tested as part of my exploration, as it achieved only around 60% test accuracy following 100,000 steps of training, and would not be captured on the plot above.

It is gratifying to have used this technology via a standard-issue personal computer, and to have achieved results close to those that Google researchers generated as part of widely read papers only two years ago.

Obtaining these results and presenting them clearly for a project reviewer was not easy. Including the background research I conducted as part of completing the Deep Learning Udacity course, I spent about 3 months on this project – more than double the time required on any other projects in this Nanodegree.

The most challenging aspects of the project included 1) learning how to build networks in tensorflow – which required a lot of conceptual reflection and review, as well as hours of practical trial and error and 2) extracting data from various image formats, which required a substantial amount of wrangling.

As I am relatively new to neural networks, I could see substantial room for improvement if I were to approach this challenge again.

Some approaches I would consider to improve performance in future:

- Explore ways to automate and optimize hyperparameter selection
- Remove some higher-digit number dataset elements from our training set to make our digit distribution more like the test sets
- Check for highly-similar images in the dataset and potentially remove close duplicates
- Look to more carefully architect my classes and scripts for better model extensibility and logic re-use
- Incorporate more advanced techniques like Adagrad Gradient Descent optimization

References:

Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks, Ian J. Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha Arnoud, Vinay Shet (2014) https://arxiv.org/pdf/1312.6082.pdf

ImageNet Classification with Deep Convolutional Neural Networks, Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton (2012), https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

Udacity Deep Learning Course - https://www.udacity.com/course/deep-learning--ud730

Tensorflow tutorials - https://www.tensorflow.org/versions/r0.11/tutorials/index.html

Stanford cs231n - http://cs231n.github.io/convolutional-networks/#layers

NeuralNetworks and Deep Learning Book http://neuralnetworksanddeeplearning.com
Definition of Rectified Linear Units:
http://machinelearning.wustl.edu/mlpapers/paper_files/icml2010_NairH10.pdf

Explanation of Local Response Normalization:
https://prateekvjoshi.com/2016/04/05/what-is-local-response-normalization-in-convolutional-neural-networks/

Aymeric Damian examples: https://github.com/aymericdamien/TensorFlow-Examples/blob/master/examples/3_NeuralNetworks/multilayer_perceptron.py

Unique counts in numpy: http://stackoverflow.com/questions/28663856/how-to-count-the-occurrence-of-certain-item-in-an-ndarray-in-python

Brian Lester - Multidigit Recognition Github repo
https://github.com/blester125/multi_digit_recognition

Hang Yao – Street View House Numbers Github repo
https://github.com/hangyao/street_view_house_numbers