MACHINE LEARNING ENGINEER NANODEGREE

Capstone Project: Multi-Digit Classification with Convolutional Neural Networks

Harley Jackson

May 16, 2017

I. DEFINITION

Project Overview

Computer vision is a field of elevated importance, now that technology such as self-driving cars, delivery drones, and warehouse inventory robots are fast approaching scales of mass adoption and production. Within the field of computer vision, recognizing multiple digits from pictures or live visual input, has utility in real-world applications. For example, in the case of delivery drones, while the drones would probably locate delivery points with GPS, being able to read addresses on doors, mailboxes, and buildings could ensure that the drone is delivering its payload to the correct unit, more precisely than just GPS coordinates taken from an address database. As Goodfellow, et al, point out in their 2014 paper, in which a multi-digit classifier was trained on the SVHN dataset, recognizing address numbers from photos is also important to mapmaking, in order to automatically process the majority of street-level photographs [1].

My motivation for pursuing this problem was to gain more experience implementing convolutional neural networks using TensorFlow, especially in order to more deeply understand the effects of hyper-parameter tuning, and to gain more insight on how the deep learning network should be adjusted to suit different types of problems. Generally, deep learning has a wide range of applications, including natural language processing [2] and protein structure classification [3] in addition to computer vision, so having expertise in deep learning is valuable as a machine learning engineer. Beyond the machine learning nanodegree, I intend to continue developing skills in deep learning, including Kaggle competitions and other projects, so knowledge gained from the project is valuable to my professional development.

Problem Statement

The problem is to take a photograph depicting numbers up to 5 digits long and correctly recognize how many digits are present, up to 5 (length = 0, 1, 2, 3, 4, 5), and what each present digit is (n = 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, or BLANK). The classifier will only get credit for getting the entire number correct, i.e. predicting each of five digit placeholders correctly to be considered a correct classification, for accuracy measurement purposes. The trained model will be able to accept photographs from an Android phone or other sources and classify multi-digit numbers depicted on them.

The solution will take the form of a convolutional neural network which can input 32x32 images of up to 5-digit numbers and classify them, including the correct value of each of 5 digit placeholders. Its evaluation will be carried out as outlined in the “Metrics” section below.

Metrics

The evaluation metric of the classifier developed in this project was similar to that used to measure the benchmark model. More specifically, the model only got credit for accurately classifying an image if all 5 of its classifiers return the same label as what the picture actually depicts. So, each of the five digit classifiers should register either the digit present or the lack of a digit (meaning 11 possible classifications for each digit placeholder). The accuracy can be represented as:

$$\frac{\sum I_{correct}}{\forall I}, \text{ where } I_{correct} \text{ is an image where}$$

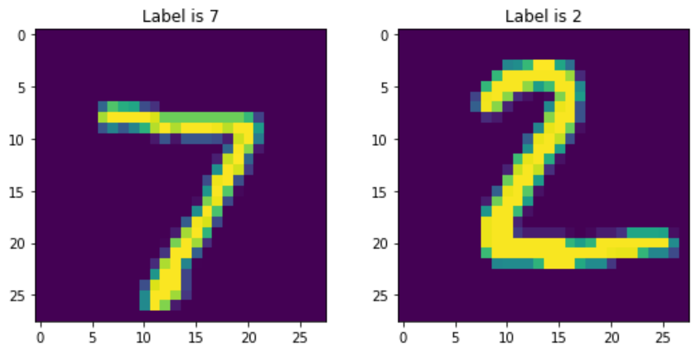
Predictions(Digit1,Digit2,Digit3,Digit4,Digit5) == Labels(Digit1,Digit2Digit3,Digit4,Digit5)

and *I* represents images in the data input to the classifier

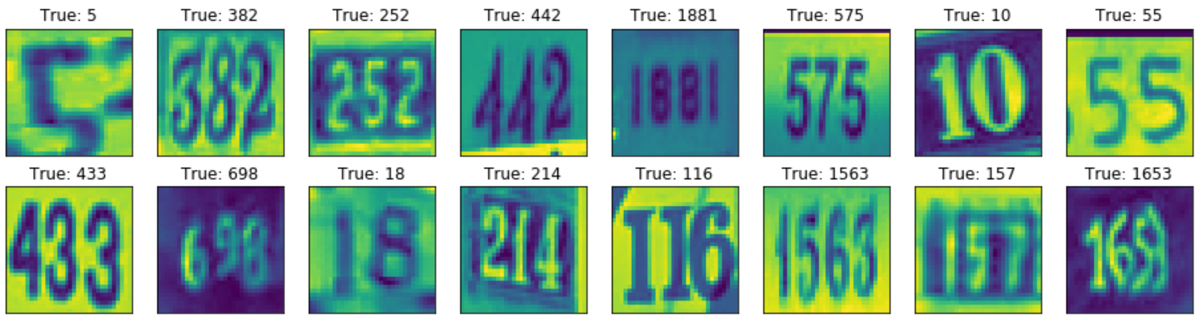
II. ANALYSIS

Data Exploration

The first dataset used is the MNIST database of handwritten digits [4]. The MNIST dataset is an appropriate starting point for this project, because it consists of 70,000 images of hand-written digits, from 0 to 9, split into 60,000 training images and 10,000 testing images. Each MNIST image is 28 x 28 pixels, with a label that can be one of 10 classifications (digits 0 through 9). It seems plausible that a model that can classify these images at a high degree of accuracy could be scaled up to classify multiple digits. Two example images from the MNIST dataset are shown below:



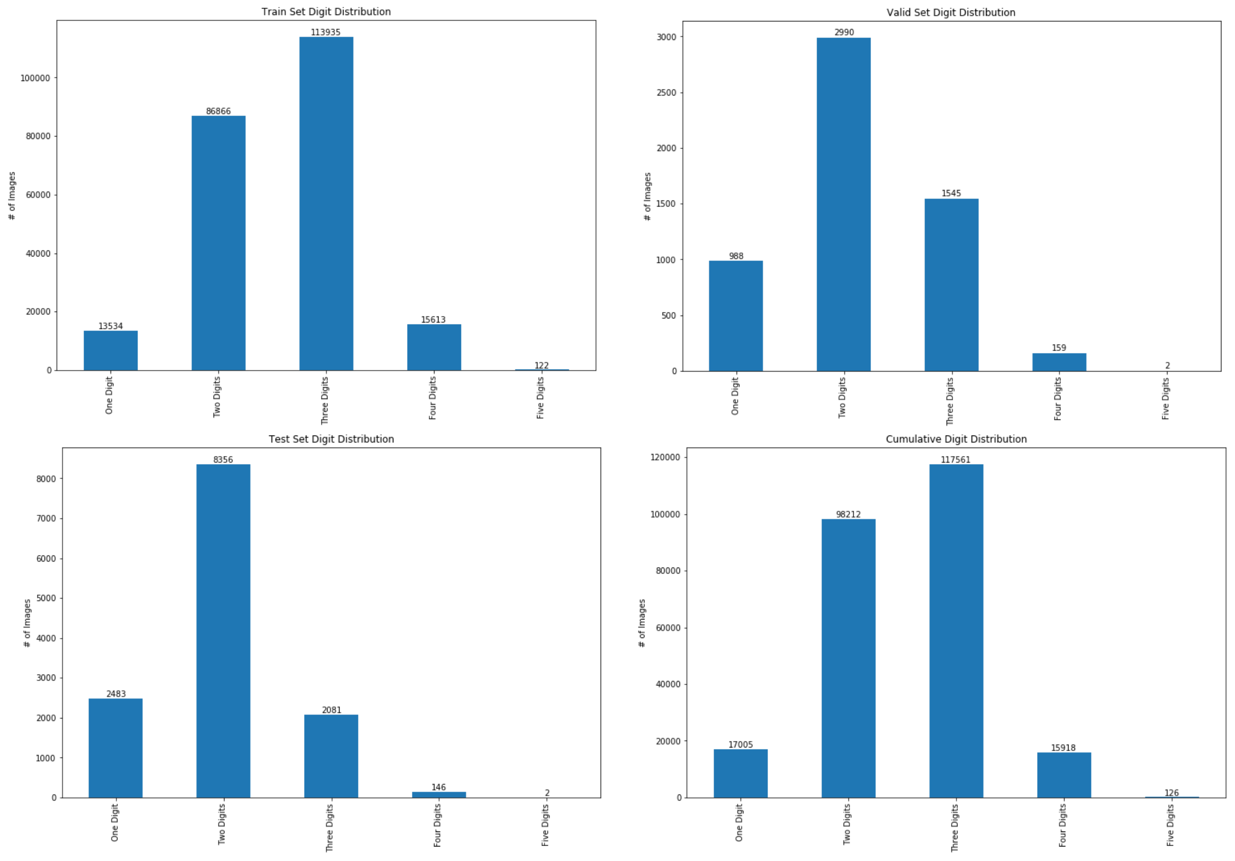
The next dataset used is the Street View House Numbers (SVHN) dataset [5]. The SVHN dataset is a database of over 600,000 digit images taken from Google Street View images, divided into 73,527 images for training, 26,032 for testing, and 531,131 extra images that can be used for additional training. Character level bounding boxes are included, along with labels that specify length in number of digits, and values for each digit. Since these images possess both the characteristics of being real photographs, and containing multi-digit numbers, a model trained to accurately classify on this dataset would presumably be able to correctly classify most pictures taken of numbers. Some examples taken from the SVHN dataset, after preprocessing to maintain multiple digits, but within the bounding box and resized to 32x32 pixels, are shown in the figure below (function for generating visual examples taken from Thomas Almenningen's Github and slightly modified [6]):



The majority of images in the SVHN dataset are from 1 to 3 digits, with 4 and 5 being less common, in declining order. It means that, for example, a high percentage of accuracy could just reflect classifying the numbers with less digits, and few or none of the 4 or 5-digit numbers. To address this imbalance in the data, a weighted penalty on the less common digits was used in the loss function of one version of the model. After building a model that could classify the SVHN images, the model was further tested on additional images, taken from Google images and taken with an Android phone by the researcher.

Exploratory Visualization

As already mentioned in the preceding "Data Exploration" section, there is an imbalance in the SVHN dataset, which is the primary dataset for training the classifier. Namely, there are fewer 4-digit and 5-digit numbers, proportionally, than one, two, or three. Actually, there are less one-digit numbers as well, but because of the labeling method in the SVHN dataset, and the method of backpropagating error for all digit placeholders during training, the model is exposed to more examples of the first digit placeholder than any other. The labeling scheme for the SVHN data set is to place all blank placeholders in the tail end of the label vector. For example, a number 1 would have the label [1,10,10,10,10] (if we are omitting the length value, where 10 represents a blank space). The number 11 would be [1,1,10,10,10], 333 [3,3,3,10,10], and so on. So the first digit placeholder is trained in every case where there are any digits in the image. However, there are, for example, only 122 5-digit numbers in the training data set, which contains 230,070 images in this project. It means that, if the classifier simply classifies digit 5 as Blank every time, it will only miss 122 out of 230,070 of the images for that particular placeholder, or about 0.05% wrong if it gets the first four digits correct. The histograms showing the counts of digits in the training, validation, test, and cumulative data sets are shown below:



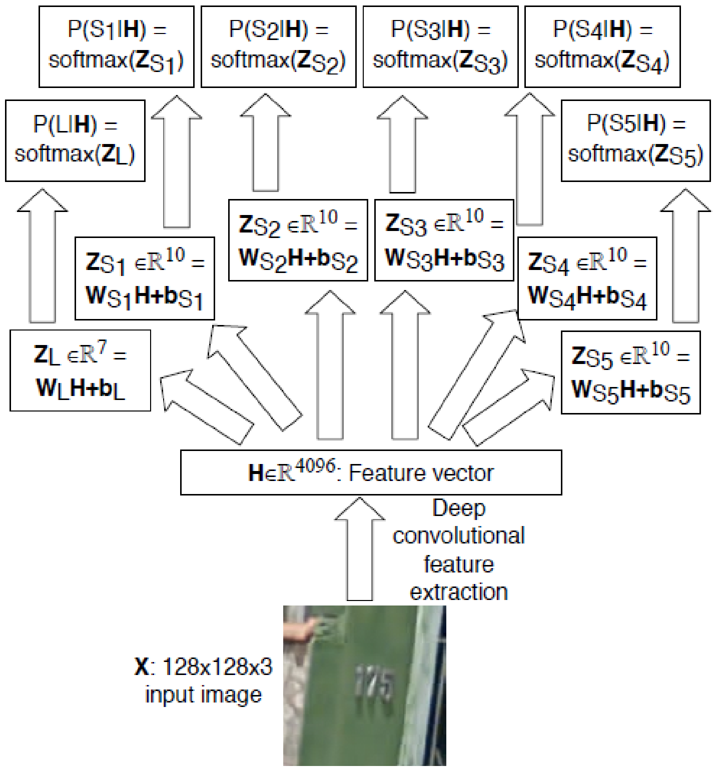
Benchmark

The benchmark model for this project is the multi-digit classifier developed by Goodfellow, et al, in 2014 [1]. The Goodfellow model is a deep learning neural network that utilizes 11 sequentially arranged hidden layers, which are shared by 6 separate classifiers, one for the length and five for each of the digits. Of the 11 neural network layers, 8 were convolutional layers which used 5x5 patches, 2x2 max pooling with strides alternating between 2 and 1, same padding, and RELU rectifiers, except for the first hidden layer, which uses a 3x3 maxout activation. The other 3 layers include 1 locally connected layer and 2 fully connected layers. All layers use dropout, except for the input. Through using this deep network, Goodfellow, et al, achieved 96.03% accuracy – that is, the model accurately classified all length and digits correctly on 96.03% of images - which was just short of their 98% accuracy goal.

Algorithms and Techniques

The core algorithm being used in the project is the convolutional neural network (CNN). CNN's are a variation of neural network that has been particularly useful in image recognition. Fully connected neural network layers can easily grow to unmanageable sizes, in terms of number of weights, for standard resolution images. CNN's address this problem by designating a number of nodes to a predefined patch of the image, for example 5x5 pixels, and doing this repeatedly over the image, moving in strides (number of pixels horizontally and vertically to move before extracting another patch), and outputting the results, where the patches will typically overlap, so that varying levels of abstraction can be represented within the network, allowing the weights to be adjusted over training to classify large objects in images, despite variations, locations, and so on.

The project will start by using the MNIST TensorFlow tutorial to train an established CNN on MNIST data [7]. After the model has been trained on the MNIST data, it will be incrementally improved to gain accuracy on classifying the SVHN dataset, first by just accurately classifying the length, then by adding digit classifiers as well. The Goodfellow benchmark model will be used as a guideline for ways to iteratively improve the model and increase performance. Specifically, the project will employ the benchmark method of using sequential layers, a combination of convolution layers, a flat layer, and fully connected layers, all shared by multiple classifiers (5 in this project, as opposed to 6 by Goodfellow). But each of the classifiers will have their own weights to be trained independently, connecting their logits to the shared hidden layers. An overall mapping of the Goodfellow model can be seen in the figure below [1]:



$P(S_1|\mathbf{H}) = \text{softmax}(\mathbf{Z}_{S_1})$
$P(S_2|\mathbf{H}) = \text{softmax}(\mathbf{Z}_{S_2})$
$P(S_3|\mathbf{H}) = \text{softmax}(\mathbf{Z}_{S_3})$
$P(S_4|\mathbf{H}) = \text{softmax}(\mathbf{Z}_{S_4})$
$P(L|\mathbf{H}) = \text{softmax}(\mathbf{Z}_L)$
$P(S_5|\mathbf{H}) = \text{softmax}(\mathbf{Z}_{S_5})$
$\mathbf{Z}_{S_2} \in \mathbb{R}^{10} = \mathbf{W}_{S_2}\mathbf{H}+\mathbf{b}_{S_2}$
$\mathbf{Z}_{S_3} \in \mathbb{R}^{10} = \mathbf{W}_{S_3}\mathbf{H}+\mathbf{b}_{S_3}$
$\mathbf{Z}_{S_1} \in \mathbb{R}^{10} = \mathbf{W}_{S_1}\mathbf{H}+\mathbf{b}_{S_1}$
$\mathbf{Z}_{S_4} \in \mathbb{R}^{10} = \mathbf{W}_{S_4}\mathbf{H}+\mathbf{b}_{S_4}$
$\mathbf{Z}_L \in \mathbb{R}^7 = \mathbf{W}_L\mathbf{H}+\mathbf{b}_L$
$\mathbf{Z}_{S_5} \in \mathbb{R}^{10} = \mathbf{W}_{S_5}\mathbf{H}+\mathbf{b}_{S_5}$
$\mathbf{H} \in \mathbb{R}^{4096}$: Feature vector
Deep convolutional feature extraction
$\mathbf{X}$: 128x128x3 input image

Computing will be carried out on a free-trial virtual machine instance on Google Cloud, using Debian/Linux, TensorFlow, and Jupyter Notebook with Python 3.5 (due to TensorFlow requirements). In light of computing limitations, images will be preprocessed as size 32x32, and 1 channel will be attempted, 3 if necessary. The model will also attempt to achieve 90% accuracy with less layers than the Goodfellow model. After testing accuracy with the SVHN test set, images from Google Images and pictures taken by the researcher will be used to test the model’s accuracy on photos not included in the SVHN dataset.

III. METHODOLOGY

Data Preprocessing

The steps taken to preprocess the SVHN dataset were very similar to those described in the Goodfellow paper. After downloading the data from the SVHN website, the data were unzipped and converted from Matlab file formats to NumPy arrays, for use with Tensorflow. Each image was cropped based on the following steps:

1) Set the top left point at the uppermost bounding box y coordinate and the leftmost x coordinate.
2) Set the height to the height of the tallest bounding box, plus the difference between the top of the uppermost bounding box and the lowest bounding box top, such that, in the worst case of the tallest bounding box having the lowest top y coordinate, the height would still capture it.
3) Like step 2, set the width such that, in the worst case of the widest bounding box being the rightmost bounding box, the length would start at the leftmost x coordinate and end at the rightmost coordinate.
4) Take 10% of each of the height and length defined in steps 2 and 3, and move the top left point in each of the x and y directions, respectively, further out from the bounding boxes.
5) Move the lower right corner away from the bounding boxes by the lesser of 20% of the height and lengths defined in steps 2 and 3, or to the edge of the image.

The above steps lead to a crop that is about 130% of the length and width between the outermost bounding box coordinates for each image. After the crop, the images are resized to 32x32 pixels, the RGB values are converted to grayscale using the weights 0.2989, 0.5870, 0.1140, and the mean of the image is subtracted. Furthermore, the labels for zero and Blank are inverted, so that blank is denoted by a ‘10’ and a zero is denoted by ‘0.’ One image was found that contained more than 5 digits and was removed. Finally, the extra images and train images were merged, shuffled, and sorted into a train set of 230,070 images and a validation set of 5684. The test set of 13,068 images was not included in this final step of shuffling. Preprocessing steps were carried out using code written by Hang Yao [8].

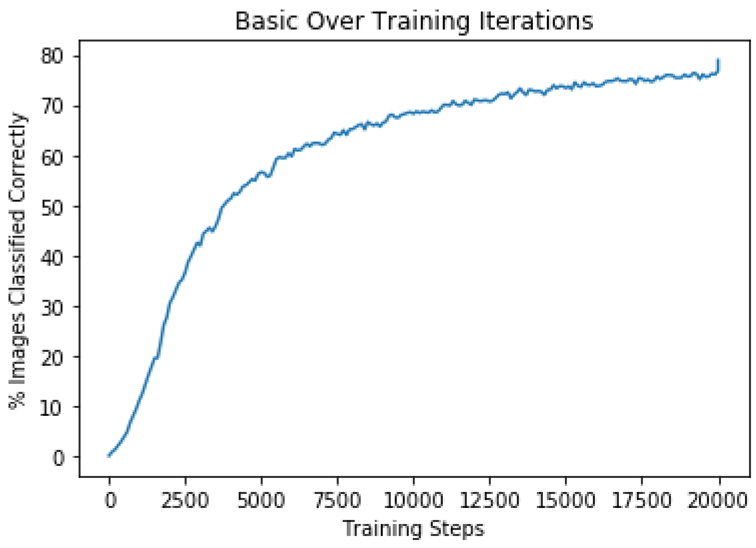
Implementation

The base model implementation used the convolutional neural network from the MNIST Tensorflow tutorial [7], with modifications to account for 5 classifications instead of one, as its architecture. The model achieves 99.23% accuracy on the MNIST data set, which only entails classifying one image on each picture to one of 10 possible classifications, without blanks. The architecture is as follows:

1) A convolutional layer using 5x5 patches, stride = 1, k=32 filters, same padding, and RELU activation

2) Max pooling with 2x2 patches, stride = 2, and same padding
3) Another 5x5 convolutional layer, with stride = 1, k=64 filters, same padding, and RELU activation.
4) Max pooling with 2x2 patches, stride = 2, and same padding
5) A flattening layer
6) A fully-connected layer with 1024 units and dropout with 50% probability

Five individual classifiers, one for each digit, apply their own weights to the output of the fully-connected layer, producing their own logits, one of 11 classifications (0-9 or Blank). The cross entropy is calculated as the sum of the cross entropies of all five classifiers, with the ADAM optimizer being used to minimize this sum cross entropy, with a constant learning rate of 0.0001. After 20,000 iterations, the model achieved an accuracy of 78.9486%, with the training curve depicted below.



Refinement

Two variations of a refined version of the above model were trained on the data in order to attempt increasing the accuracy of the classifier. The layers were altered somewhat to reflect some basic principles used in the Goodfellow paper. For one, the first convolutional layer's filters were increased from k=32 to k=48. Next, the max pooling on the second convolutional layer's output was changed from stride = 2 to stride = 1. Finally, the fully-connected layer's units were tripled, taken from 1024 to 3072. These refined hidden layers were then tested with two different loss functions: one is the same loss function used in the base model, the other loss function has weights applied to the cross entropies of each individual digit classifier, in order to reflect the proportions of each type of image (1-digit, 2-digit, 3-digit, 4-digit, or 5-digit). Specifically, the weights applied a penalty that was inversely proportionate to the percentage of each type of image in the train and validation sets (the test set counts were not included to prevent bleed over of information from the test set into model training), normalized to one. The steps to create the weights were:

1) Divide total number of images by number of i-digit images, for i in range 1 to 5
2) Divide each digit result from step 1 by the sum total of the results from step 1
3) The weights become decimals that add up to 1.0

The weights after the above steps were:

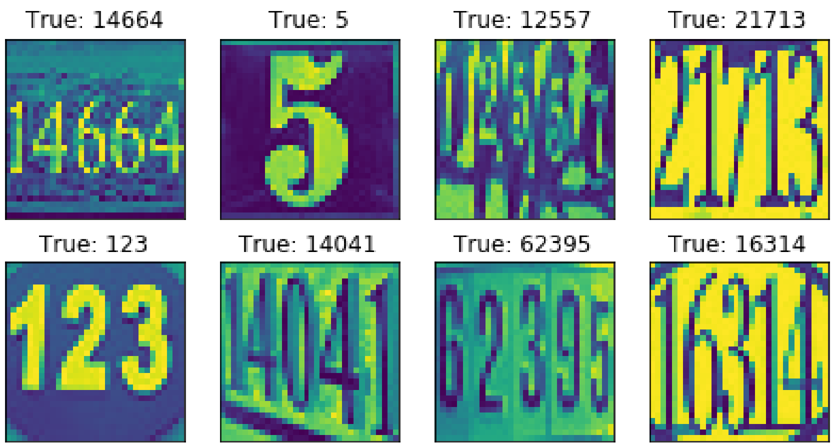
Digit 1	0.0083807532859904
Digit 2	0.0013544482195863667
Digit 3	0.0010539080292617992
Digit 4	0.007716541923608457
Digit 5	0.981494348541553

The purpose behind the weighted loss function was to account for the rarity of certain types of images, especially 5-digit images, by penalizing the model more for getting these rare instances incorrect. Both of these refined models were then trained for 50,000 iterations, as a case-control, to attempt both better accuracy and, in the case of the weighted model, increase accuracy for 5-digit numbers especially.

IV. RESULTS

Model Evaluation and Validation

At the end of 50,000 training iterations, each of the two refined models were tested on the test set, where the models had only seen the training and validation data. The weighted-loss model achieved 79.0174%, and the standard-loss model achieved 80.5632%, using the same metric for accuracy, where the image classification is only counted as correct if all five placeholders are correctly classified. While the model using a weighted loss function was slightly less accurate overall, given the same training iterations, the weighted model was chosen to be tested on 20 images taken from Google images, in which 1 to 5 digits were depicted. The images were preprocessed in the same manner as the dataset: cropped around the digits, resized to 32x32 pixels, converted to grayscale using RGB weights, and having the mean subtracted. A sample of the pictures taken from Google images, after preprocessing, is shown below:



The weighted-loss model classified only 30% of the 20 outside images correctly. The predictions and actual labels are shown below, and it can be seen that the classifier always labeled the fifth placeholder as Blank. To check further, the test dataset was predicted again, and digit 5 was predicted blank in 100% of the 13,068 images, and we know 2 are not blank.

PREDICTED	ACTUAL
[1 2 8 10 10]	[6, 2, 3, 9, 5]
[5 10 10 10 10]	[5, 10, 10, 10, 10]
[2 4 1 10 10]	[6, 3, 4, 1, 9]
[5 4 4 10 10]	[1, 4, 0, 4, 1]
[5 10 10 10 10]	[5, 10, 10, 10, 10]
[1 2 8 10 10]	[1, 5, 5, 9, 5]
[5 5 10 10 10]	[5, 5, 10, 10, 10]
[1 2 3 10 10]	[1, 2, 3, 10, 10]
[7 0 5 10 10]	[7, 0, 5, 10, 10]
[6 3 9 10 10]	[1, 3, 3, 3, 0]
[1 5 10 10 10]	[1, 5, 10, 10, 10]
[5 5 7 10 10]	[1, 9, 1, 7, 7]
[2 1 8 10 10]	[7, 1, 8, 10, 10]
[4 6 0 10 10]	[1, 4, 6, 6, 4]
[1 0 2 10 10]	[1, 2, 5, 5, 7]
[5 5 4 10 10]	[5, 8, 4, 1, 10]
[8 7 8 2 10]	[3, 5, 9, 8, 2]
[2 1 7 5 10]	[2, 1, 7, 1, 3]
[1 3 4 4 10]	[1, 6, 3, 1, 4]
[2 1 10 10 10]	[2, 3, 10, 10, 10]

This evidence suggests that the weighted loss function did not mitigate the imbalanced data, and that the model still ended up always classifying the fifth digit as Blank. Furthermore, it appears to have hindered the accuracy somewhat, over 50,000 iterations, as compared to the model that considered all 5 digit classifiers equally in its loss function.

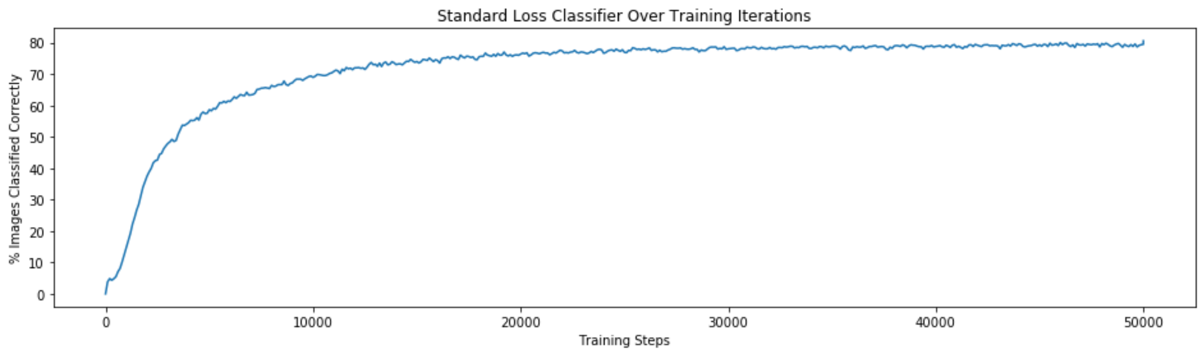
Justification

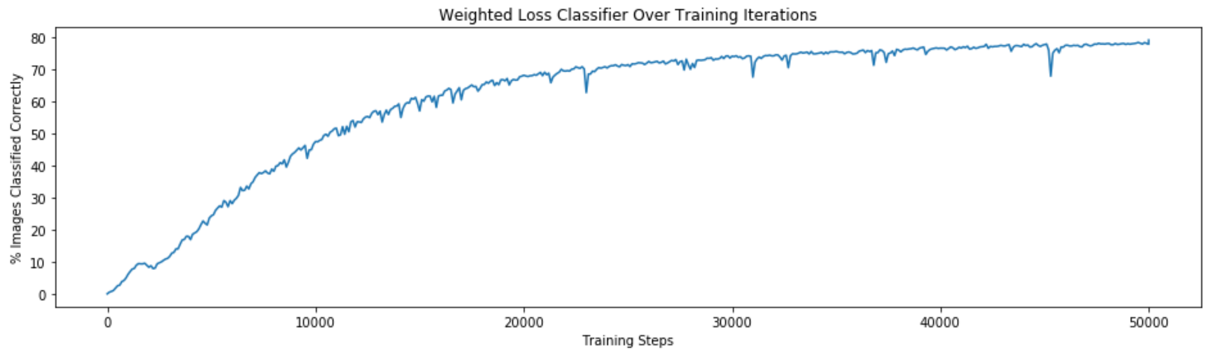
The final result of the project falls short of the benchmark model. The best model from this project, the one that uses no weights in the loss function, classified the test set with an accuracy of 80.5623%, whereas the benchmark model classified the same dataset with an accuracy of 96.03%. In the benchmark model, the classifier had to get six predictions correct to get credit for an image, including length and 5 digits, whereas the final model in this project only had to get 5 digits correct, with the length variable being omitted in favor of including Blank as a classification in each digit placeholder. Overall, the final model of this project only performed at 83.89% capacity of the benchmark model, and the weighted loss version did not address the data imbalance. While the final model could be further tuned and trained longer, the accuracy of the Goodfellow model probably would not be attained without adding more layers. Moreover, at only a little over 80% accuracy, the model is not robust enough, especially if digit length is more than 3. The potential application, computer vision or map-making, requires a much higher rate of accuracy to be useful, specifically 98% in the case of map-making. Classifying 20% incorrectly means that the model is not trustworthy, and a substantial amount of work would have to be put into checking on the classifier's results, which makes its practicality diminish. It also falls short of the goal to attain 90% accuracy with few layers.

V. CONCLUSION

Free-Form Visualization

An interesting trend to observe is the rate at which the two versions of the convolutional neural net learned over 50,000 iterations. We can see the two plots below:





It can be seen that, in the standard loss model's learning curve, the model approaches its asymptote earlier on in the training session, and the curve is smoother and more predictable. In the case of the weighted loss classifier, learning grows at a more gradual rate, and there are perturbations, most likely due to a batch having more 4 and 5-digit numbers in them, resulting in more drastic backpropagations, which would cause a short-term drop in accuracy. However, over the long run, the weighted loss function did not appear to bring up accuracy on the 5th digit. It is possible that the model could have showed superior accuracy given more training iterations, but as it stands the model appeared to converge on the same accuracy as the unweighted version.

Reflection

Summarily, the project entailed training convolutional neural networks to recognize multi-digit numbers in images by first implementing and training an established CNN on a dataset that recognizes single digit numbers in images, identifying a strong benchmark model for reading multiple digits, preprocessing the SVHN dataset in the manner described in the benchmark paper, establishing an imbalance in the dataset, devising a method for addressing the data imbalance, using the benchmark model and devised method to incrementally improve the original MNIST CNN in an attempt at 90% accuracy on multiple digits, and then testing the weighted model on pictures of digits retrieved from outside the original SVHN dataset. In the improved CNN, two versions were trained, with all other hyperparameters held constant, to test whether a weighted loss function could offset the imbalanced data. While the weighted loss function did not appear to improve accuracy, it is also possible that the weighted model simply needed more training iterations, after comparing the two learning curves. The most difficult part for me was the lengthy training sessions for the two final models. Several times, the nearly 24-hour sessions were interrupted by internet connectivity or updates, requiring a restart. Also, testing the images on the restored models revealed that the tensors should be named to be recalled later, which then led to training the models yet again due to user error. Although the final model could predict 1 to 3 digit numbers at a decent rate, it is not reliable enough to be useful in application.

Improvement

To improve on this project, I would make the hidden layers reflect the Goodfellow architecture, as closely as possible. It might be interesting to compare the Goodfellow model to one that uses the weighted loss function for five classifiers, with Blank being a possible choice. The differences would be that the new model would backpropagate errors on all five digit placeholders, since Blank was a choice, but length would not be included as its own classifier. Such a model would have enough layers to extract the details needed to achieve accuracy 96% or more, and would more clearly indicate whether applying weights to the classifiers can increase overall accuracy.

REFERENCES

[1] Goodfellow, I. J., Bulatov, Y., Ibarz, J., Arnoud, S., and Shet, V. Multi-digit number recognition from street view imagery using deep convolutional neural networks. In Proc. ICLR, 2014.

[2] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. ICLR Workshop, 2013

[3] Wang S. Peng J. Ma J. Xu J. Protein secondary structure prediction using deep convolutional neural fields Scientific Rep. 2016 6 18962

[4] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition." *Proceedings of the IEEE*, 86(11):2278-2324, November 1998

[5] 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*.

[6] https://github.com/thomalm/svhn-multi-digit/blob/master/06-svhn-multi-model.ipynb

[7] https://www.tensorflow.org/get_started/mnist/pros

[8] https://github.com/hangyao/street_view_house_numbers/blob/master/
3_preprocess_multi.ipynb