Capstone Project

Machine Learning Engineer Nanodegree

Thalles Santos Silva
September 16, 2016

Definition

Project Overview

In the field of computer science, more specific in the sub-field of image processing, image recognition has been one of the most interesting and challenging problem domains. The issue of recognizing multi-character text on images has been studied for a long time, but just recently, with the advances in the field of machine learning, more specific using deep learning models, researchers have been achieving very consistent and relevant results.

Rather than attacking the harder problem of recognizing multi-character text on images, this project aims to describe a computation architecture used to solve the also hard problem of recognizing sequences of multi-digit numbers on street view images. In other words, given a sequence of digits on an image, this document proposes an approach that will analyze these inputs and output which digits that sequence is composed of.

There are some very interesting applications for such models, one of them would be a system that can recognize building/housing numbers from street view images and link these numbers with the street, name and zip code in order to have the full address of that building. Such an application could in turn, be very useful for any kind of Delivery Company such as Amazon or Fedex.

First of all, to implement such a model using machine learning, there are some pre-requisites that must be addressed. To begin with, to tackle such a problem using machine learning, one needs to have data. Machine learning algorithms are techniques that can learn patterns from data, however, just like a regular person needs to see a lot of examples to learn an unknown concept, machine learning algorithms also need to have a lot of examples to base their learning. Particularly, the more data available, the more probably the model will reach consistent results. Also, in order to make the model able to learn from data, there must be some kind of pattern in the data because otherwise, the machine learning techniques will simply not learn anything.

As a branch of machine learning, deep learning also shines upon large amounts of data. That means that in order to achieve good results, this machine learning model would never succeed without a proper dataset with a lot of images. Nevertheless, part of the recent success of machine learning is that suitable datasets with many examples and different conditions including many different real world situations have finally made their way to both academia and corporations.

To make the proposed model able to recognize sequences on street view images, we will use the Street View House Numbers (SVHN) Dataset (Ian J. Goodfellow, 2013). The SVHN is a real-world image dataset for developing machine learning and object recognition algorithms with minimal requirement on data preprocessing and formatting. It can be seen as similar in flavor to the MNIST dataset, but it incorporates an order of magnitude more labeled data (over 600,000 digit images) and comes from a significantly

harder, real world problem (recognizing digits and numbers in natural scene images). SVHN is obtained from house numbers from Google Street View images.

Problem Statement

The goal of this project is to provide a machine learning model that can interpret and recognize sequences of digits in real-world street images.

Strictly speaking, given an input image that contains any sequence up to five digits, the proposed model will output the digits that represent that sequence on the image, see Figure 1. To achieve this goal, this project describes the implementation steps of a deep learning model that can, by operating on the image pixels directly, identify, and recognize sequences up to five digits. More precisely, this deep learning model presented here will be able to extract the main patterns of these kinds of images (its digits) and learn an internal representation that can be used to identify digit sequences on similar real world images.

Figure 1: Given an image with a sequence of street view numbers, the goal of the proposed model is to recognize and output each of the digits that compose the sequence.

To build such a model, we firstly need to gather as many examples that we already know the correct classification, we call it the training dataset. From this dataset, the model will develop the ability to learn the inherited patterns on these examples and as a result, be able to make its own guesses of which digits a sequence is composed.

However, different from a regular digit recognition task, where images containing just a single digit on each one of them are assessed, this model aims to recognize a sequence up to five digits on the same image, and to do that, we need to account for the five possible kinds of sequences we might encounter.

Figure 2 illustrates some of the five possible digit sequence images from the SVHN dataset. As one might see, these different kinds of sequences present some additional hardens to the recognition process. First off, the different sequence lengths add a high scale variability to the images. That is, while some digits fill the entire image, which is typically the case of a one digit image, on the lengthy sequence images, each one of the individual digits looks much smaller or squeezed because they have to fit in to a fixed image size; this example can easily be seen by comparing the first image with only one digit with any with the last two with sequence lengths of four and five respectively.

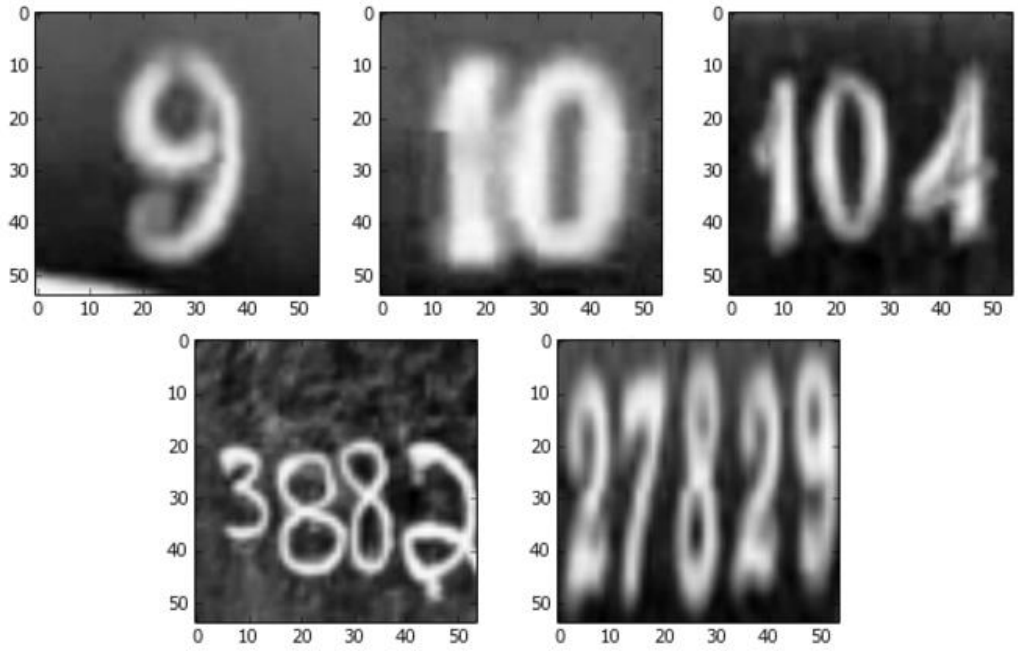
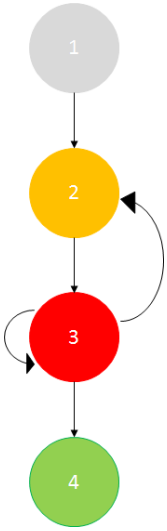


Figure 2: 54 x 54 pixel images representing each of the possible sequence lengths. Sequences of lengths 1 to 5. These input images are already pre-possessed and ready to be fed into the deep learning model.

To conquer our objectives, some of the tasks involved in this process include:



1- Analyze the input proposed dataset (SVHN) and extract some statistics that will help us to understand how to take the maximum advantage of the patterns on this dataset such as the number of samples per class and sequence's length per image.
2- Perform some data processing steps that aim to clear and augment the dataset which will make the model's implementation easier and more robust.
3- Build a machine learning model that will use the processed dataset to learn the patterns on the dataset and produce the desired result.
4- Assess the results using error metrics and compare it with a proposed benchmark.

After implementing these high-level steps, the goal of this document is to demonstrate a model that will be able to tackle the problem of street view sequence recognition with high success rate.

In practice, the process was concentrated more on the step 3, where various models were created and tested, but sometimes, it went back to step 2 for implementing new data processing techniques for instance.

Metrics

To measure the final performance of the model, some metrics such as Error, Recall and Precision will be calculated and analyzed so that the final results will be thoroughly analyzed and reasoned.

Error is a straight forward measurement that captures the proportion of data points classified as incorrectly. To calculate it, we do as follows:

$$Error = \frac{number\ of\ images\ classified\ incorrectly}{total\ number\ of\ images}$$

However, as a performance measurement, this simple measure has a significant drawback, especially when the data points are skewed towards one class over another. For example, in a simple binary classification problem, i.e. a model that outputs true or false based on some input, supposing we had a dataset with 100 samples and only one sample was classified as false, we could create a model that always outputs a positive result and the expected error would be 1/100 = 0.1 or 1 percent. While this is a very low error, it does not capture how bad this model is. This problem arises when we measure the error globally across the set of all classes, rather than taking a granular view at the classes' level. Let us take as an example a two class classification problem. When the model outputs a positive result, there are two possible conclusions: 1- the model was correct (True Positive) or 2- the model was incorrect, (False Positive), the same can be derived for when the model outputs a negative result (True Negative and False Negative). If we place the number of occurrences of these four values in a two by two matrix we have a decision matrix, as depicted in Figure 3.

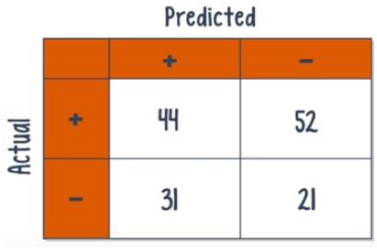


Figure 3: Confusion matrix for a classic 2 class classification problem.

Using the confusion matrix, we can come up with some different metrics that overcome the problems of the Error metric. For example, if we look only at the positives in the dataset we might ask two questions, 1- "What percentage of these positives was the model able to classify?" that is called the Recall, which is defined as:

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Positives}$$

Additionally we can look at the number of times the model predicted positive and ask "What percentage of these predictions were actually positive". That is the precision metric and it is defined as:

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Negatives}$$

Although similar, these two metrics answer two different questions, the recall regards how many of the data points was the model able to recall and precision accounts for how precise the actual predictions were.

Another interesting measure is the F1 score, which can be expressed as the harmonic mean of Precision and Recall.

$$F1 = \frac{2TP}{2 * TP + FP + FN}$$

The Figure 4 shows the case of a multi-class classification problem. In this scenario, the definitions of precision and recall remain the same however, because a data point, now, can be misclassified in multiple ways, the False Positives and False Negatives must be summed over all the possible misclassification pairs.

Figure 4: Confusion matrix for a multi-class classification problem.

Precision is the ration of the number of true positives over all points classified as positives and recall is the ration of the number of true positives over the total number of positives.

For this project, we will pay more attention to the Error and Accuracy metrics because these are the two most common metrics used in the literature for this kind of problem. Also, the benchmark values defined as the goal of the proposed solution are in terms of error/accuracy. However, a complete summary regarding all of the metrics discussed in this section will be presented.

Analysis

Data Exploration and Visualization

The SVHN dataset, used to train and evaluate the model presented in this report consists of approximately 600,000 training digits and roughly 26,000 digits for testing. According to the dataset website, the training set part of the SVHN dataset is divided in two sets, the first one, contains 73257 digits for training while the second has 531131 digits. The testing set on the other hand, is composed of precisely 26032 digits for testing. The dataset have 10 classes, 1 for each digit, so the digit '1' has label 1, '9' has label 9 and '0' has label 10. The website provides the datasets in two flavors, the first consists of the original (various size) images with character level bounding boxes, and the second is a MNIST-like 32-by-32 dataset with images centered around a single character (many of the images do contain some distractors at the sides). Because the goal of this project is to develop a model capable of multi-digit sequence recognition, we chose the first one for our final architecture.

Figure 5 shows some of the images and its bounding boxes. According to the dataset website, each tar.gz file (three in total) contains the original images in png format, together with a digitStruct.mat file, which can be loaded using Matlab. The digitStruct.mat file contains a structure called digitStruct with the same length as the number of original images. Each element in digitStruct has the following fields: **name** which is a string containing the filename of the corresponding image and **bbox** which is a struct array that contains the position, size and label of each digit bounding box on that image.

Figure 5: Original, variable-resolution, color house-number images with character level bounding boxes.

One interesting thing about the dataset is that it is not well balanced among the class labels. Figure 6 shows the distribution of class labels (digits from 1 to 10, where 10 accounts for the digit 0) across the training dataset with 73K digits.

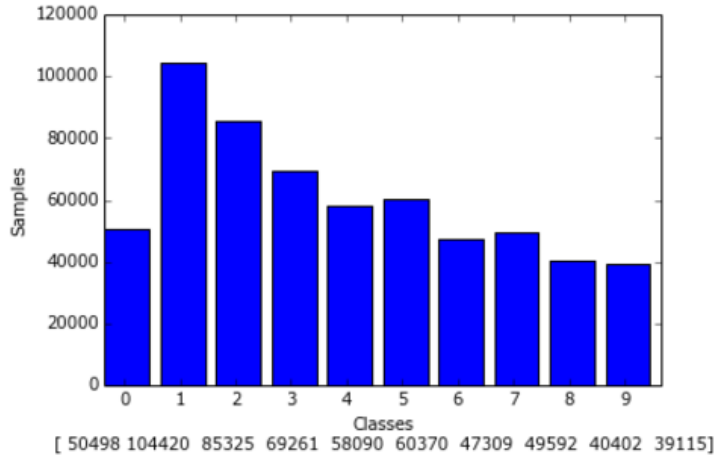


Figure 6: Distribution of digits in the original SVHN training and extra datasets with over 604.000 digits. The y-axis represents the number of images with one particular digit and the x-axis the distribution of digits across the images.

As one can see, the distribution of digits is not well balanced across the SVHN training datasets. There are much more images of class label 1 than any other. Indeed, we notice that there are more than twice as much images with the digit '1' than with digits '0', '6', '7', '8', and '9'. In order to have a relative measure of class imbalance, we can define the Degree of Class Imbalance (DCI) as described in (Walter Daelemans, 2008), using the equation on Figure 7.

$$DCI = \frac{1}{N}\left(\frac{1}{(K-1)}\sum_{k}\left(|c_k| - \frac{N}{K}\right)^2\right)^{\frac{1}{2}}$$

Figure 7: Degree of Class Imbalance (DCI)

This is the standard deviation of the cluster with respect to "the expected value" of the size of cluster c_k in a balanced problem, divided by N, the number of examples. Using this equation, a DCI closer to 0 means that the problem has a well-balanced distribution among the classes. As we run this equation using the training set, we get a DCI * 100 of 3.43, and as we may infer, it is not close to zero, which confirms our visual findings that the classes are not well balanced. To have a better feeling about that, the MNIST dataset is a fairly well balanced dataset with DCI * 100 of 0.57 (Walter Daelemans, 2008).

Knowing whether or not the working dataset is well balanced among its classes is a real asset when dealing with machine learning models because not only it helps to evaluate how good the final solution is, by providing significant insights about metrics we should choose, but it also gives a sense of how difficult the problem will be and it helps to make some decisions regarding the complexity of the model or if techniques for data augmentation will need to be taken for instance.

Another relevant insight we may ask concerns the distribution of images with different digit sequence lengths. In other words, how many images with a four digit sequence there are? How about two and three? That is what Figure 8 shows. We can see that the majority of images have sequences of length 2 and 3 while images with sequence lengths of one, four and especially five (with only 124 examples) are somewhat rare.

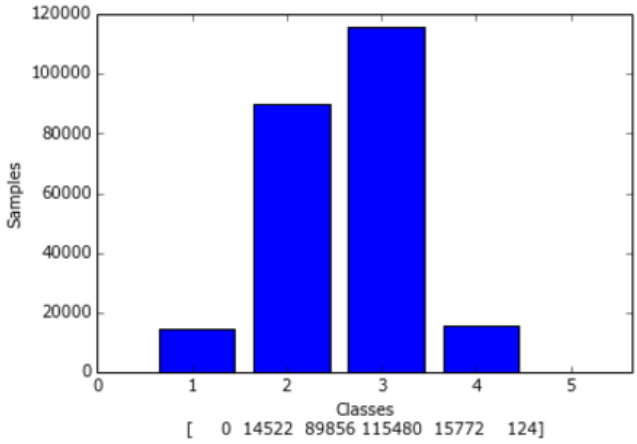


Figure 8: The distribution of images with different sequence lengths across the original SVHN training and extra datasets with over 604.000 digits. The y-axis represents the number of images, and the x-axis the sequence length of each image.

Also, the distribution of images with different sequence lengths depicted in Figure 8 resembles a regular normal distribution which indicates that the majority of data points lies on the middle of the distribution and centered in the standard deviation which are the images with 2 and 3 digit sequence length.

Another rather interesting fact about the SVHN dataset is that not all of the images are corrected labeled. Such abnormality challenges the capacity of the model because not only it will need to learn from the correct examples but it will also have to develop some intuition regarding the examples wrongly classified and know that these abnormalities are values that do not represent the entire population. Figure 17 shows some of these cases and how they challenge the models capacity of recognizing such patterns.

As for the testing dataset, Figure 9 and Figure 10 presents the distribution of sequences' length and digits.

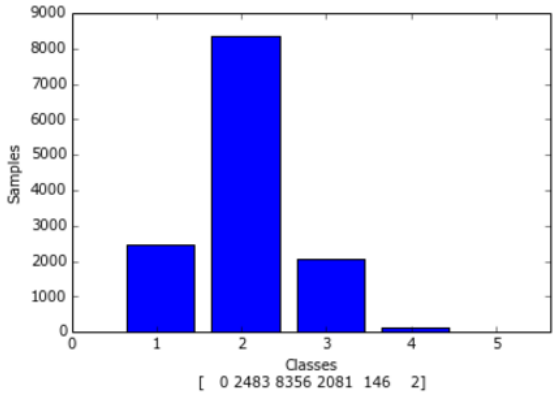


Figure 9: The distribution of images with different sequence lengths across the original SVHN testing nearly with over 26K digits. The y-axis represents the number of images per sequence's length, and the x-axis the length size of each image

Note that the distribution of sequences' length across the testing dataset differs from the training dataset. While the training dataset distribution resembles a normal distribution, the testing data's is more skewed to the right. Another interesting point is that there are only 2 images with a sequence length of 5 digits.

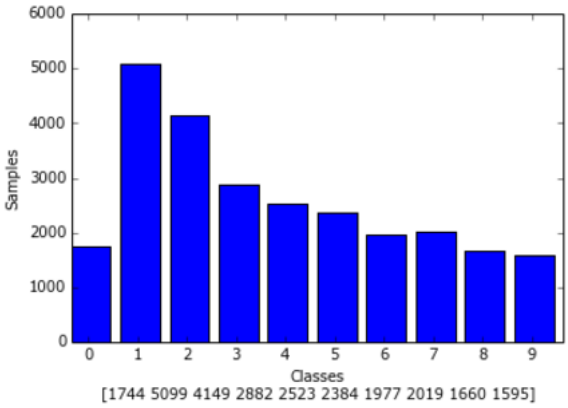


Figure 10: Distribution of digits in the original SVHN testing dataset with over 26k digits. The y-axis represents the number of images with one particular digit and the x-axis represents the possible digits on an image.

Regarding the digits distribution on the testing set, we acknowledge a very similar digit spreading when compared to the training dataset.

Algorithms and Techniques

To solve the sequence recognition problem, we propose a system that implements a deep convolutional neural network. This network receives as input 54 x 54 image pixels and outputs five values – one for each of the 5 possible digits that can compose a sequence.

Convolutional networks are, with no doubt, one of the main reasons behind the flourishing of deep learning. Pioneered by Yann LeCun (LeCun, 1998), Convolutional Nets (CNNs), are the principal force behind the recent successes in image, object and speech recognition. To have a clearer idea about the success of that technique, since 2012, every single winner have used CNNs for their implementations in the famous ImageNet challenge competition (Li, 2013).

CNNs are networks that share their parameters across space. In the case of object recognition for instance, for each input image we have a width, height and a depth which are typically the Red, Blue and Green channels. Suppose we select a small patch of that image, let's say a 5x5 small window and call it a kernel. A convolution is the action of taking that kernel and running a tiny neural network on this patch with k outputs. Further, we can repeat this operation by sliding the kernel window over the input image which will output an image with a different width, height and depth (k). Basically, by doing this operation, the net is looking for the same pattern (digits in our case) on different positions of the image.

In short, a Convolutional Neural Network is a stack of convolutions which increase the depth of the output while decrease its spatial dimension – width and height – in such a way that the first layers of a CNN learn simple patterns such as edges while the farther layers take these basic features and lean new and more complex shapes. These characteristics plus some other operations that include RELU functions as the activation units and pooling layers that are usually responsible for the dimensionality reduction previously mentioned represent the common structure that makes CNNs appropriate techniques for image/object recognition.

To build and foremost to make this model able to perform the proposed task, we firstly are going to perform some pre-processing algorithms on the dataset used to train the model. These steps are performed to make sure that the data meets the specifications that will make the model learn the most it can. Some of these pre-processing steps include image rescaling and resizing, feature reduction, and image normalization. For the case of the SVHN dataset, image rescaling and resizing will make sure that the images look all the same in terms of the digits position and final image size, which is very important to guarantee that the input data has a clear pattern to be learned by the model.

Feature reduction will help the model to save some important processing time by allowing it to only focus on the images' features that are necessary for the learning process whereas image normalization has been used and it is considered a very good practice when training image networks. In particular, the problem comes in when some values with very different ranges must be computed in some manner. To avoid that, we will normalize the image pixel values to have approximately 0 mean and standard deviation of roughly 0.5.

In addition, to make the model able to learn the most it can we need to provide it with the most examples we have. However, some of the patterns in the SVHN dataset are not well distributed, which could in turn hurts the model capacity of recognizing a pattern that it did not learn well enough for a lack of examples to learn from during training. To alleviate on this issue, we are going to perform some data augmentation techniques. These techniques will increase the variance on the training dataset by randomly applying image processing algorithms which will slightly change the images so that they will act as new examples to the model. For each set of examples used to train the deep learning model, some algorithms such as random cropping, random contrast, brightness and blur will be applied on the images so that they will never look the same from the models perspective. These small and random variations to the training set will help the model to foresee examples not yet seen so that it will be able to generalize the concepts learned from the training data to a broader universe.

As for the network itself, we intend to use Google's TensorFlow machine learning library (Abadi, 2015). In short, TensorFlow™ is an open source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them. To train the model, we are going

to use the built-in Adam optimizer (Diederik Kingma, 2014) and to evaluate how good we are during the training process and to minimize this discrepancy, we will use the built-in cross-entropy routine from TensorFlow.

The Adam optimizer is an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments. Although relatively new, the method has gained some popularity in the field of deep learning because it is straight forward to implement, is computationally efficient and makes the learning phase less sensitive to hyper parameter tuning.

The TensorFlow built-in Adam algorithm uses Kingma and Ba's Adam algorithm to control the learning rate. Adam also offers several advantages over the simple Gradient Descent Optimizer such as moving averages of the parameters (momentum). In short, this enables Adam to use a larger effective step size, and the algorithm will converge to this step size without fine tuning. The main down side of the algorithm is that Adam requires more computation to be performed for each parameter in each training step (to maintain the moving averages and variance, and calculate the scaled gradient). However, it takes out most of the hyper parameter tuning such as learning rate decay and momentum which makes the model easier to train.

In a very high level, the job of the Adam optimizer algorithm is to minimize the amount of error that the network produces when learning from the data. During the training phase, the model is going to take some amount of data (images) as input, process it and produce some partial predictions, then it will compare these predictions with the actual values (cross-entropy) and produce some values, that we call the loss, which basically is the average cross-entropy over the input data. Than the model will take the derivative of this loss, which will be back propagated through the network so that the weights and biases – learning variables – will be slightly adjusted in order to minimize the overall loss. However, we are going to do that with a very small amount of data, rather than using the whole dataset every time. Since the model has a complex input space (54 x 54 input images), it has been proved that for models with a lot of data to train, it is not only much more cheaper (in terms of performance) but it is also more efficient to do these operations using small amounts of data (BATCHES) at a time and taking these estimates in order to converge to a more consistent model. For this model, the BATCH size used consists of a chunk of 86 examples randomly chosen.

In addition, because overfitting is always an issue that must be on the developers mind when dealing with machine learning algorithms, some regularization techniques will be used to make the model able to learn as much as it can and still be able to generalize what it has learned to new data. The first regularization technique used is the L2 regularization. The ideal behind that is to add another term to the loss which will penalize large weights. The second technique is called dropout. The insight behind it is to randomly destroy some of the activations that go from one layer to the other so that the network can never rely on any given activation because they might get removed at any time, which in turn, will prevent the model to overfit since it will be forced to learn a redundant representation of everything in order to lean the patterns on the data.

Benchmark

In order to find a suitable benchmark for comparing the results of our model, we first looked over for an established digit sequence recognition competition using the SVHN dataset. Once such completion was not found, it was decided to compare the results of the proposed model with the ones reported at

(Ian J. Goodfellow, 2013), which presents a baseline for street view digit sequence recognition using a deep convolutional network.

This baseline reports a sequence transcription accuracy of 96.03% and a character-level accuracy of 97.84%. Although the conditions are not totally equal, since our proposed solution based some of the decisions on this baseline, compare the results found here with the ones reported on this work is the fairest comparison to make.

The results will be compared using two metrics. The transcription accuracy and the character-level accuracy. The transcription accuracy is a very strict metric that considers a prediction correct if and only if the two conditions apply. The first condition regards that each digit of the sequence has to be corrected predicted, while the second states that even if the first condition applies, the sequence length has to be corrected predicted in order to consider a final corrected prediction, therefore, a sequence is evaluated by considering both, its per digit accuracy and its length.

Finally, as a benchmark comparison, the proposed model has to achieve a transcription accuracy that is compared to the one reported at (Ian J. Goodfellow, 2013), in a **±3** percent range. In other words, our model has to achieve a minimum of 93.03% sequence transcription accuracy. As for the per character-level accuracy, since it is a slightly easier problem, our model has to achieve a final accuracy that lies between a **±1** percent range, therefore it has to accomplish a minimum of 96.84% character-level accuracy on the test dataset to be considered successful.

It worth noting that, to make the comparison as fair as we could, the model is allowed to use only the two training sets available on the SVHN website and leave the testing set untouched during training, thus, using it just for evaluation.

Methodology

Data Preprocessing

In order to make the model as much lighter and precise as we could, some image preprocessing steps such as crop, resize, feature scaling, and feature normalization as well as some image enhancement algorithms that that include brightness, contrast and blur distortions were implemented. Also, some abnormalities on the dataset had to be addressed to make the model more consistent.

First off, because the images in the SVHN dataset have different sizes, the first step was to make them as close in look to one another as possible. As can be seen in Figure 5, most of the images do not present the digits as the central object of the images. Actually, the digits' positions varies considerably depending across the samples. To fix that, when feeding images for training our model, the first step was to crop the images by their digits bounding boxes so that the digits would become the central part of each image. However, that bounding box was calculated in such a way that it would include all the individual bounding boxes plus some scale variance that would make the mode more robust. More precisely, first we found the smallest rectangle that contains all of the individual character bounding boxes. After, we expanded that rectangle in order to make it the more squared as it could be. For instance, if the smallest bounding box that contains all individual digits of a sequence has dimensions 50 x 35 pixels (width x height), firstly, an effort to expand this rectangle's height respecting the original image's dimensions and ratio is taken

until it reaches the 50 pixels width so that it would be a perfect square (50 x 50). Finally, we expanded the resulting bounding boxes equally in both direction as much as 30% of its current size, rotate each image to a randomly chosen degree that extends from ±20, then cropped each image to their bounding boxes and resize the crop to a 64 x 64 shape size.

Moreover, in order to synthetically increase the size of the training dataset, we applied random brightness, contrast, and blur enhancements to each image. For Brightness and Contrast enhancement, two random numbers from 0.4 to 1.0 are generated, where in both cases, values closer to 1.0 result images closer to the original while a smaller values represent more changes in brightness or contrast. The same applies to blur distortions with the exception that the range now is from 0.2 to 1.0.

Since the network receives 54 x 54 image pixels as input and we currently have a 64 x 64 image shape, during training, a routine to crop 54 x 54 pixel image from a random location within the 64 × 64 input image is applied. This means we generated several randomly shifted versions of each training example, and along with the previous changes, these routines synthetically increase the size and variance of the dataset which consequently make the model more robust upon changes on the training data. However, note that this operation is only performed during training.

To create the testing set, most of the procedures used for the training set was followed with some minor exceptions. Since the model accepts 54 x 54 pixel images, we decided to create two testing sets. To the first testing set (54 x 54 image pixels) we decided to not include the 30% expanding step since this was designed to provide the model with a more consistent dataset for training only. It is worth noting that this is the official testing set and the final results are reported upon this dataset.

The second testing set, also 54 x 54 pixel images, follows the exact pattern applied to the official testing but it includes the 30% bounding box increasing step, so the only difference between this testing set and the official one is the 30% expanding step. The reason behind this change is just to measure how robust the proposed model is by measuring its performance under small perturbations (changes) in the input space to appropriately see how greatly this would affect the final results. Also, it is important to note that neither of the image enhancement algorithms are applied to the testing sets since it was designed to only provide the model with more robust data for training only.

Now that we have images with the same size and centered in the digits, we decided to implement some feature scaling and normalization. To begin with, since the job of the model is to recognize digits in street view images, the color of these images do not play an important role to define the digits' shapes. Indeed, the more complex the input features are, the more it will suffer from the curse of dimensionality that states that when the number of features grow, the number of samples need to grow exponentially to make the model able to converge. Therefore, by removing the RGB channels of the images and transforming it in plain greyscale ones, we are reducing the size of the input and consequently the amount of images needed to train the model.

To achieve the feature reduction, for each image we applied the following formula:

```
L = R * 299/1000 + G * 587/1000 + B * 114/1000
```

This formula is known as the ITU-R 601-2 luma transform and R, G and B represent the Red, Green and Blue channels of the original images.

Lastly, we normalize the pixel values to have approximately 0 mean and standard deviation of roughly 0.5, we do that by subtracting each pixel value by 128 and dividing the result by 255.

```
Scaled = (pixel_value - 128.0) / 255.0
```

The final result is displayed on Figure 2.

One last step was to convert the labels of the images with 0s on it to have label 0 instead of 10 as the default.

Implementation

The proposed final solution to solve the sequence recognition problem is a convolutional neural network that receives as input 54 x 54 pixel image(s) and outputs, for each image, a collection of N random variables S1,…,SN, where N is 5, representing the possible sequence digits on each image. Each of these variables that the model outputs has a set of possible outcomes therefore, each Si has 11 possible results each, ranging from 0 to 10 where 0 represent the digit 0, 1 represents the digits 1 and so fourth with 10 representing no digit. For instance, when analyzing the image with the sequence 316 (first image on the third row of Figure 2), the desired output of the model will be the sequence [3, 1, 6, 10, 10].

The best convolutional architecture we implemented consists of a stack of 7 convolutional hidden layers, three fully connected layers, and at the top of the model, we have a set of softmax classifiers that receive as input the features calculated from the previous layer of the model and outputs the set of values Si – the digits of the sequence. Each layer performs a rectified linear unit operation and has a set of biases to account.

Each convolution operates in a windows size of 5 x 5 with stride of 1 and the set of spatial dimension implemented on the first 7 convolutional layers are [16, 32, 48, 64, 80, 96, 112]. Each of the three fully connected layers has 1024 neurons each. Each convolutional layer implements max pooling and Local Response Normalization, however, the order in which each of these operations are executed is alternated. The max pooling window size is 2 × 2, with a stride that alternates between 1 and 2 at each layer, so that each half of the max pooling operations reduces the spatial dimension of the representation by half where the other half preserves it. All convolutions and max pooling operations use zero padding (SAME padding) on the input to preserve representation size while increasing the depth. Figure 9 presents an overview of the proposed model.

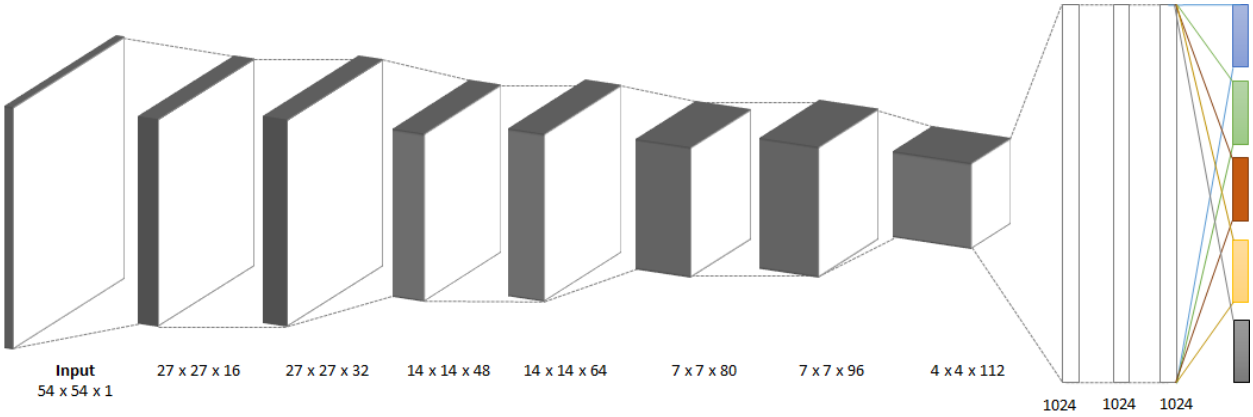


Figure 11: Convolutional deep learning model to solve the sequence recognition problem on image street images. The first 7 layers are convolutional layers with Max Pooling and normal and Local Response Normalization followed by three fully connected layers with 1024 neurons each. On top the mode model there are 5 Softmax classifiers, one for each possible digit of a sequence – up to a maximum of 5 digit length.

To train the proposed network, we decided to use the Adam optimizer algorithm over small image batches of size 86. The loss function that the Adam optimizer will minimize is the vanilla cross-entropy function, where this loss is actually the sum of all five individual losses (one for each digit Si). Also, dropout is applied to all layers but the input. The dropout probabilities to the total 10 layers are [0.85, 0.86, 0.87, 0.88, 0.89, 0.9, 0.91, 0.92, 0.93, 0.94].

Refinement

During the implementation process, many techniques were tried before reaching the final model architecture. At first, the idea was to implement a simple digit recognition convolutional model that would be able to only identify one street view digit per image, and then apply a separate segmentation algorithm to make the whole model able to recognize sequences. Indeed, the strategy of separating the recognition from the segmentation is one of the most used in the literature and it has achieved success over passed studies. To do that, we used the SVHN MNIST like dataset described on the Data Preprocessing section, and then built a convolutional model that rapidly was able to get roughly 90% accuracy on the test dataset.

However, when the segmentation part of the algorithm had to be implemented, many problems with this initial approach were faced. The first approach to solve the segmentation problem was to segment the image from left to right by cropping the first ¼ of the image and feed it to the network, if the score was high enough the model would assume that the segment contained a digit and it would re-segment the image from that quarter in order to find more digits. If the score was not high enough, it would increase more ¼ of the original image to the first segment and try again.

Although the strategy seemed reasonable at first, the results turned out not to be as good as expected, mainly because the network was being constantly misled when fed with the segments. In short, it was always returning a high digit probability value even for segments where there was no digit on it. After a lot of trials training this net, I gave up on this strategy and decided to implement a model that would do the whole process, segmentation and recognition, based on this published base line model (Ian J. Goodfellow, 2013).

After reading the model's paper I got an ideal of how I could implement that on my way. The most challenging part at first, was to make the classifiers softmax backprop nothing on examples for which one specific digit of the sequence was not present. The first strategy to address this issue was to use five different optimizers, each one would minimize its own loss, and thus I would also have five losses, one for each digit when training the model. With that, I could sort the images based on the number of digits on it, so I had five training sets, one for images with only one digit, another one for images with two digits and so on. Then, I could train the net by feeding to the appropriate optimizer the right batch of images in such a way that the optimizer for the first digit was trained with images that only contained one digit, the optimizer for the second digit would get batch images with sequences of two digits and so on and so forth, and the goal was to make each optimizer to minimize the loss for each digit on the sequence.

When I first started to train the model, the first thing I noticed was that the time required to the train the net had gotten substantially bigger. For images with only one digit, only the optimizer for the first digit was triggered. However, for images with two digits, I had to call the optimizer of the first and the one for the second digit, therefore, for images with five digits, all five optimizers was being called and that added a lot of processing time. Apart from these issues, the model started to converge and things started to get working. However, one thing noticed in the learning process was that every time the model was able to minimize the loss for a particular digit, the loss for the others suffered, especially for the fourth digit in relation to the first three. Every time the model was able to make the loss for the fourth digit less, (which implies better accuracy from what was being predicted to what it actually was), the losses for the first three digits was getting higher (which means worse accuracy for these digits).

Because this model tried to learn each digit of the sequence separately, the problem was that when the model was able to get the estimates for the fourth digit better, by adjusting the weights and biases of the network, these changes were messing up the work done by the optimizers from the first digits to get them accurate. Then I realized that in order to make it better, a better approach would be to combine the losses for each digit and make a total loss, so that the optimizer would try to minimize all losses together and not in detriment of one another. In addition, it would also implies a single optimizer function and by doing so, not only the new model managed to converge to better results but it did it in less time.

In addition, even with this new model that minimizes a total loss, the learning rate was one of great concern during the many training attempts. The fact was that the model was starting to converge very soon, however, when it reached a certain loss/accuracy value, it just could not get it better. In other words, it seemed that the model was always getting stuck in a local minimum. One of my strategies was to minimize the learning rate, as it is mentioned on the Udacity Deep Learning Course, sometimes, with a higher learning rate, the model seems to learn faster but it usually falls in a local minimum, on the other hand with a smaller learning rate, the model might take more time to start converging, however, it tends to converge to better results as the training proceeds.

This more prominent model had 10 layers (7 convolutional layers plus 3 fully connected). It was able to accomplish a nearly 14% error on the testing dataset, and a per digit accuracy of 82, 80, 75, 74 and 70 percent for each digit of the sequence respectively. Since 14% error was not an acceptable mark, I decided to implement a different model.

This new and final model main concept lied on having a new possibility for each digit of the sequence. Now, each digit of the sequence could output one of 11 possible results – the usual 0 through 9 plus the label 10 which indicates that there was no digit. With this modification, we improved our training routines

since we no longer have to distinguish between the image sequences length, therefore not needing to have the five datasets described above. Another noticed improvement was that now, the model was not suffering to learn a digit of the sequence in detriment of forgetting another.

The Table 1 summarizes the error rates for each of the discussed model's approaches.

Table 1: Error rates for the different model's approach

	Characteristics	Error (testing set)
First Model	Single digit recognizer	10%
Second Model	Multi-Digit recognizer with multiples optimizers	14%
Final Model	Multi-digit recognizer with single recognizer	5.8%

Results

Model Evaluation and Validation

In order to make sure that the model was able to generalize well when tested with unseen data, we used cross-validation techniques such as the creation of a separate validation dataset to be used during learning and only deploying the testing set after the learning process. It is important to note that for the validation set, the image enhancement algorithms, described in the section Data Preprocessing, such as random brightness, contrast, cropping and blur were not applied. However, the 30% increase bounding box routine was applied to the validation set because we wanted to see how the model would behave when fed with images with a little more variance. In short, the validation set follows the same rules described for the second testing set. For more information, please see the Data Preprocessing section.

The final model was trained in an Intel(R) i7 8 Core(TM) with 4.00GHz CPU along with a NVIDIA Titan X featuring 3072 Cuda cores. The whole training process took roughly 12 hours to be completed.

Regarding the hyper parameter tuning, one of the main concerns when training this model was the learning rate. Since the optimizer function used for training, the Adam Optimizer, already implements learning rate decay as well as moving average (momentum), the most important hyper parameter to set was the initial learning rate.

Figure 12 shows the loss during training across different initial learning rates. Although the losses plateau to almost the same value, they behave differently in the beginning of training. While the model with an initial learning rate of 5e-5 started to learn faster – steeper curve, the one with learning rate of 3e-5 took more time to learn. However, to the contrary of what I was expecting, the model with the 5e-5 learning rate (Blue curve) was able to generalize better over the testing set reaching the maximum transcription accuracy of 94.2% while the others two, converged to nearly 93% transcription accuracy.

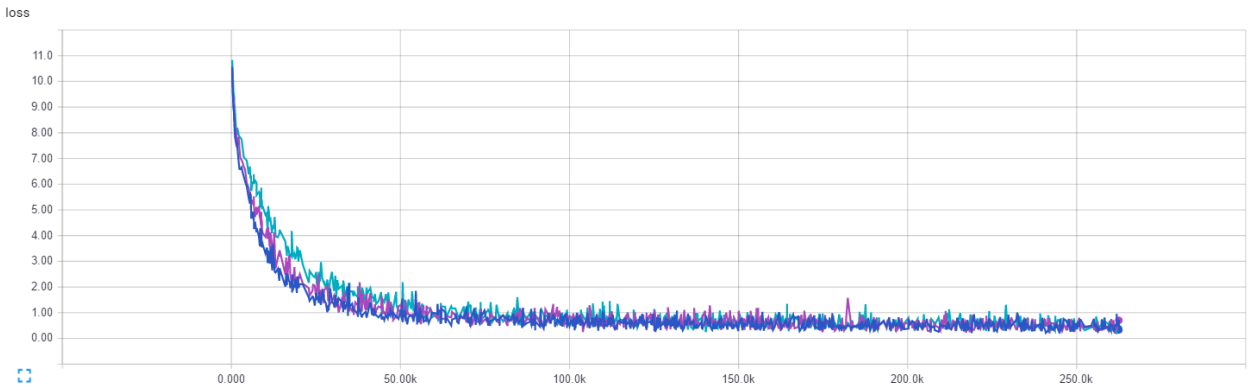


Figure 12: Model's learning behaviors under different learning rates. The plot shows the Loss (y-axis) of the batch training sets over training steps (x-axis) over three different initial learning rates. The Blue curve was produced with an initial learning rate of 5e-5 (0.0004), the Purple curve had 4e-5 and the Cyan curve was generated with an initial learning rate of 3e-5.

When it comes to the character-level accuracy, in order to better understand the results presented here, let us first analyze the accuracy of each digit placeholder on a sequence. In other words, since this model is able to predict sequences up to 5 digits, and it learns each of these 5 placeholders, considering each one separately might give us a better overview of the final results.

Starting on Figure 13 we can see three very important measures regarding the first 2 placeholders of a 5 digit long sequence. The first is the confusion matrices that show the corrected and wrongly predicted classifications for all digits that appeared as the first and second digit of a sequence. Secondly, we have the standard percentage error accomplished by the model, for that digit placeholder, and lastly, a table overviewing the precision, recall and f1 score for each of the possible 10 labels – 0 through 9 – that may appear as the first two digits of a sequence.

As can be seen by looking at the matrices diagonals, the model was able to get most of the digits right, though it made some mistakes along the way. The same reasoning can also be reinforced by looking at the error percentage – 2.8 and 3.7 for the first and second digits placeholders respectively – as well as the precision, recall and f1 score where most of the values lie above the 95% mark. Also, these two errors are the greatest among all 5 digits placeholder measures, which makes sense since most of the images contain at least a 2 digit sequence – refer to Figure 9.

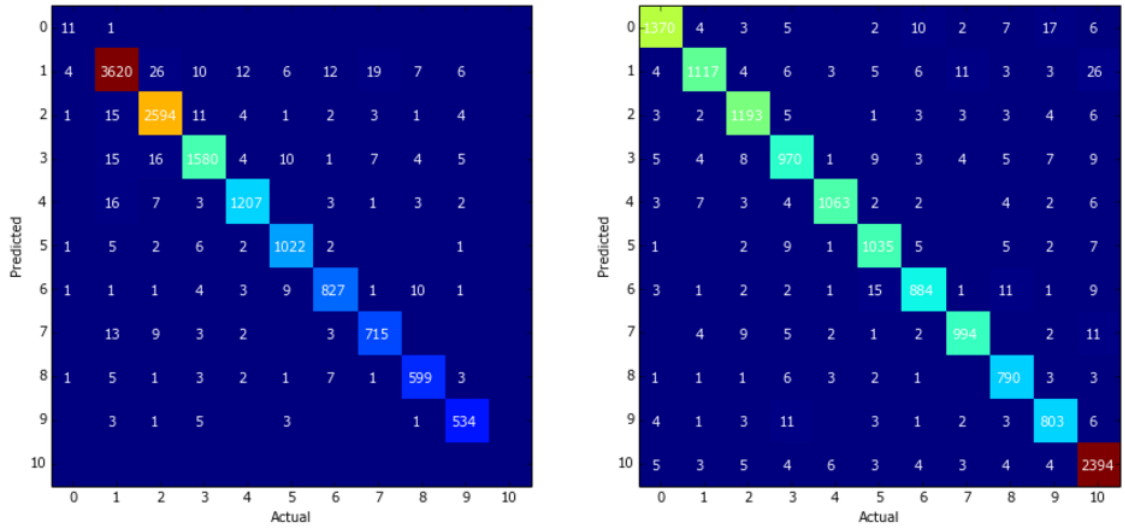
Additionally, the same arguments can be made to the third and fourth digit sequence placeholders shown on Figure 14. However, the confusion matrices now show that most of the values on these two placeholders were the label 10, which in this model architecture means that there is no digit there. In other words, these images were of sequences with at most 2 or 3 digits long. Still, the diagonals exhibit high accuracy for the cases where there were actually digits on these two placeholders.

The fifth placeholder, however, presents a different scenario. If we just look at the error – 0.01% – we might be tempted to conclude that the model was able to get an astonish mark of success. Nevertheless, if we refer back to Figure 9, one sees that in the testing dataset, there are only 2 images with a 5 digit long sequence and indeed, the model failed to classify both of them. This failure, which can also be perceived by looking at the precision, recall, and f1 score results – all 0.0%, can be explained by looking at Figure 8 where among the over 130.000 images used to train this model, just 128 of them are images with sequences of 5 digits long. This discrepancy emphasizes how important it is to have a very consistent and

well balanced dataset for training deep learning models because when we do not provide the model with enough data of a particular class it cannot create a robust representation of that concept.

The confusion matrices also expose that the second digit of a sequence is the hardest to get corrected classified, which not only had the greatest error rate (3.48%), but as demonstrated in its confusion matrix, although most of the examples were corrected classified (the diagonal), this scenario presents the most cases of wrong predictions.

On the other hand, the fifth digit of a sequence is considered to be the easiest one to predict. This can be explained for the fact that there are only 2 images with a 5 digit sequence length, which makes the majority of the examples free from a fifth digit – label 10.



Error: 2.74716865626

	0	1	2	3	4	5	6	7	8	9
Precision	0.92	0.97	0.98	0.96	0.97	0.98	0.96	0.96	0.96	0.98
Recall	0.58	0.98	0.98	0.97	0.98	0.97	0.96	0.96	0.96	0.96
F1 Score	0.71	0.97	0.98	0.96	0.97	0.97	0.96	0.96	0.96	0.97

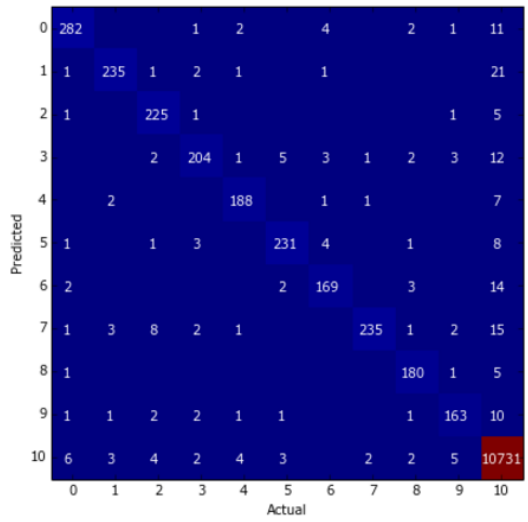
Error: 3.4817875727

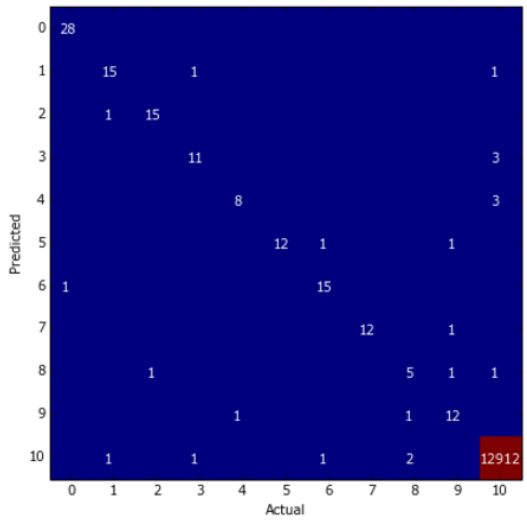
	0	1	2	3	4	5	6	7	8	9
Precision	0.96	0.94	0.98	0.95	0.97	0.97	0.95	0.97	0.97	0.96
Recall	0.98	0.98	0.97	0.94	0.98	0.96	0.96	0.97	0.95	0.95
F1 Score	0.97	0.96	0.97	0.94	0.97	0.96	0.95	0.97	0.96	0.95

Figure 13: Confusion matrices for the first two digits placeholders a sequence may have along with the standard error plus precision, recall and f1 score. Each confusion matrix shows the correct and wrong predictions for these two placeholders.

To calculate the final character-level accuracy for the testing set, we can take the average accuracy from each one of the 5 placeholders a sequence might have which yields 98.37% accuracy.

The model also demonstrated a very well behavior even when tested under slightly different conditions. Using the second testing set for evaluation, which implements a 30% bounding box increase making the digits of the sequence slightly smaller, the results turned out to be acceptable. The transcription accuracy reached 88.3%. Therefore, even with a decrease in the transcription accuracy on the second testing set, we can conclude that the results on both testing sets show that the model is robust enough for solving the digit sequence recognition problem.





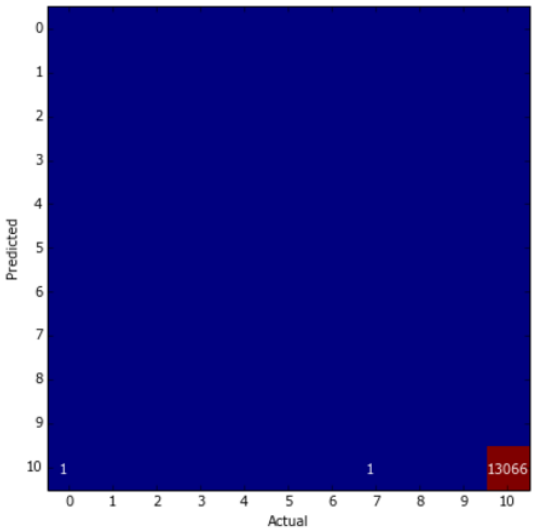
Error: 1.7217630854

	0	1	2	3	4	5	6	7	8	9
Precision	0.93	0.90	0.97	0.88	0.94	0.93	0.89	0.88	0.96	0.90
Recall	0.95	0.96	0.93	0.94	0.95	0.95	0.93	0.98	0.94	0.93
F1 Score	0.94	0.93	0.95	0.91	0.94	0.94	0.91	0.93	0.95	0.91

Error: 0.17600244873

	0	1	2	3	4	5	6	7	8	9
Precision	1.00	0.88	0.94	0.79	0.73	0.86	0.94	0.92	0.63	0.86
Recall	0.97	0.88	0.94	0.85	0.89	1.00	0.88	1.00	0.63	0.80
F1 Score	0.98	0.88	0.94	0.82	0.80	0.92	0.91	0.96	0.63	0.83

Figure 14: Confusion matrices for the third two forth placeholders a sequence may have along with the standard error plus precision, recall and f1 score. Each confusion matrix shows the correct and wrong predictions for these two placeholders.



Error: 0.0153045607591

	0	1	2	3	4	5	6	7	8	9
Precision	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Recall	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
F1 Score	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Figure 15: Confusion matrix for the fifth placeholder a sequence may have. Also, the standard error along with the precision, recall and f1 score for the fifth placeholder is displayed.

Justification

In order to evaluate the final solution, we stablished benchmark values for both, the transcription and character-level accuracy. These benchmark values are based on the proposed baseline (Ian J. Goodfellow, 2013) that has almost the same goals discussed here. On the section Benchmark, we stablished that our proposed model had to achieve a minimum of 93.03% accuracy on the transcription category and 96.84% accuracy for the character-level category.

As expected, the proposed model reached the desired levels in both classification metrics. For the transcription problem, with a 94.2% accuracy, our model ended bellow the results of (Ian J. Goodfellow, 2013) by roughly 2.5%. Although 2.5% is equivalent to nearly 325 different sequences that our model could not get all correct, it is worth noting that this model is very smaller than the one reported by (Ian J. Goodfellow, 2013) and consequently reached these values in a much shorter time and demanding fewer computational resources.

As for the character-level accuracy, our model managed to accomplish a total average of 98.37% accuracy on the official testing set. Although this result have exceeded the benchmark, which is 97.84%, some differences in how the model was designed and evaluated can explain such deviation. Because our solution not only outputs the actual predictions for each digit, but it also considers the results where a digit was not there and it was corrected or incorrected predicted, this measure might not be well fit for comparison. For instance, upon evaluating an image with the sequence “123”, the proposed solution would output its prediction in the following way, [1,2,3,10,10] – considering a correct prediction scenario. Therefore, the per digit evaluation routine will account positively for all 5 digits, even though the image is a three digit sequence. As a result, because most of the images do not have a fourth or fifth digit sequence, if the model is able to get most of them corrected (label 10) it would increase the final percentage values. However, it is important to note that it would also account negatively if the prediction was [1,2,3,5,10] for instance.

Although the solution deviates from the benchmark in this point, I believe it is important to account for the scenario where the digit is not there and the model predicted it correctly or incorrectly because it shows that the model not only knows that a digit is there but it also can respond to the opposite situation.

Conclusion

Free-Form Visualization

Upon evaluating the proposed model, some characteristics about the model, the dataset used to build it and the results were found and kept my attention. Figure 16 presents some of the testing images that the model was able to classify correctly. One very interesting point is the required robustness to recognize very hard sequences that can be presented in very different conditions. One of these conditions is the various angles that a sequence might have when dealing with street view digit sequences. As can be seen in Figure 16, many example present random diagonal layouts as well as many different light conditions which makes the entire process much harder.

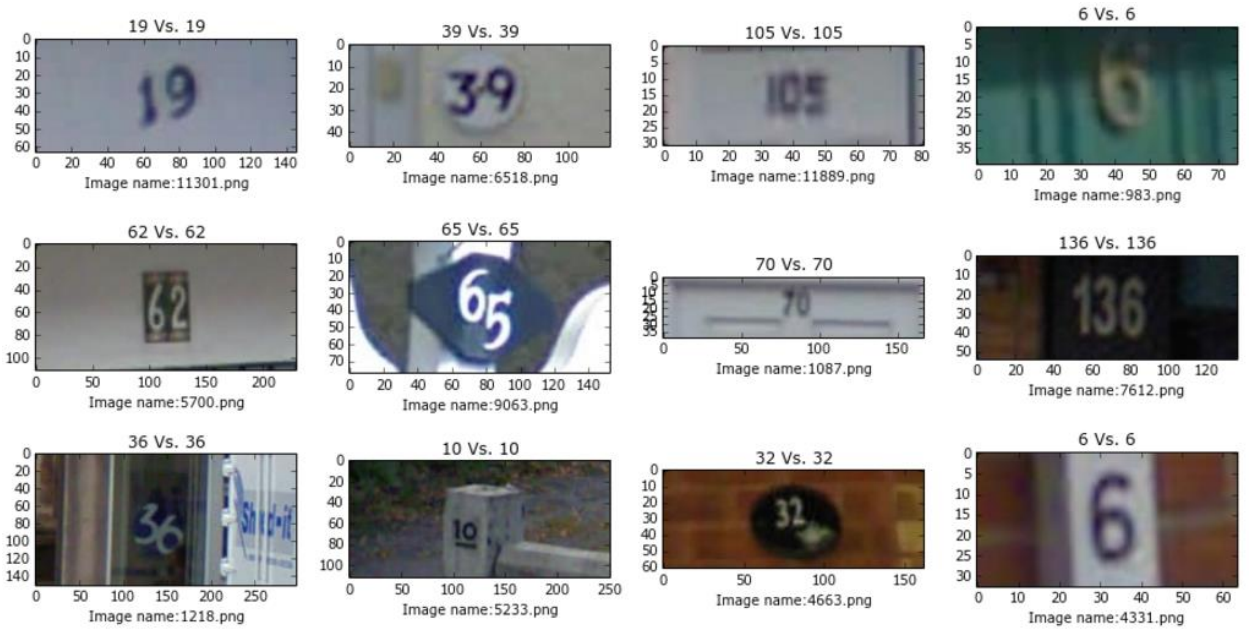


Figure 16: Difficult but correctly transcribed examples from the SVHN testing dataset. (Predicted transcription Vs. Ground Truth). Some of the examples present additional hardens for recognizing sequences such as different layout angles.

I believe that part of that robustness was gained when we decided to add noise to the training data such as random rotation and image enhancements because such changes were designed to mock this situations and make the model aware that it could face these examples in real time. Furthermore, that proves that our solution was able to generalize the knowledge experienced during training to real live situations where new unseen examples were presented.

Despite the accuracy achieved by our solution, some important characteristics about the model's results and the dataset utilized in this project are worth pointing. First off, the dataset seems to have some of its images wrongly classified as demonstrated in some of the images on Figure 17. Also, one can see that in some of the cases, the model's predictions seem to have more sense than the dataset real classification. The ground truth labels in this dataset are quite noisy, as is common in real world settings. However, the model seems to adapt itself well when some of the conditions varies which proves why it can derive a correct label even from incorrected classified ones.

We believe that some of these mistakes could have been avoided if the original bounding boxes expansion step was not carried out during training since this step added more noise to the sequences such as sided digits that were not part of the sequences' original classification label. On the other hand, this step proved to be more beneficial because it allowed the model to learn the patterns from more complex examples which contributed positively to its generalization ability.

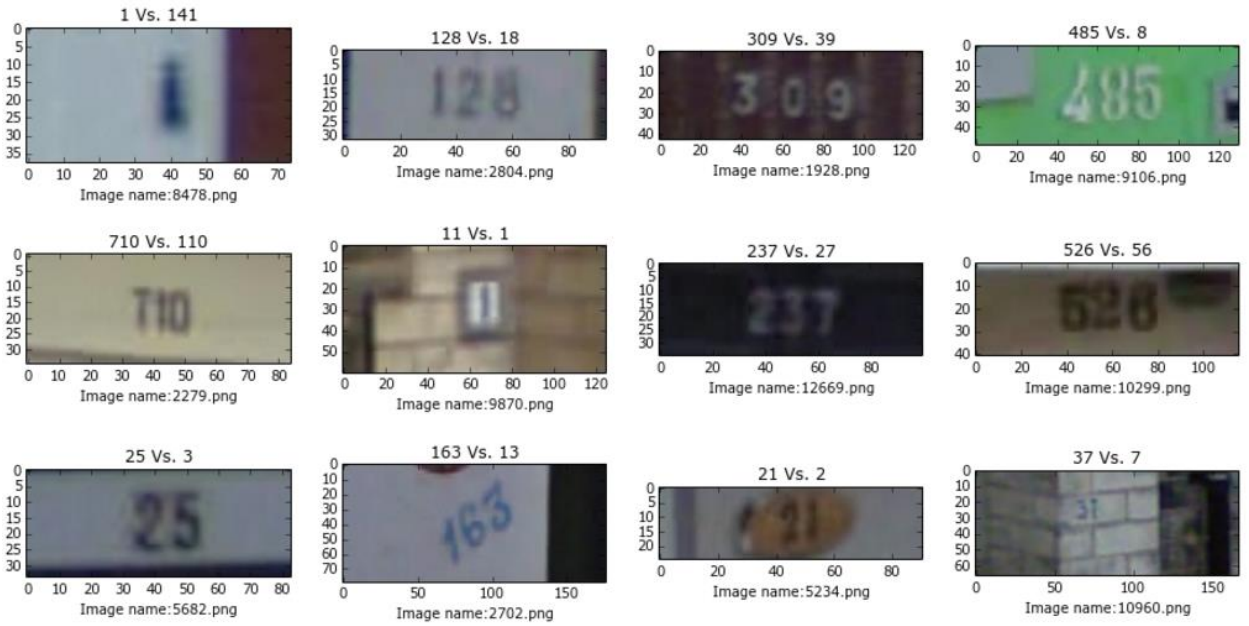


Figure 17: Very difficult test cases where the proposed model could not predict the correct sequence. (Predicted transcription Vs. Ground Truth). Note that for some of these, the "ground truth" is also incorrect while the models predictions seem to be correct.

Reflection

To solve the problem of sequence recognition on street view images, we proposed a deep learning model that could learn from examples and generalize its knowledge when faced with new real world examples. Since a good machine learning model depends heavily on the data used to build it, the first and most important step to take was to choose the most appropriate dataset. Based on the problem's aspects and characteristics, we found that the SVHN dataset (Yuval Netzer, 2011), was a good fit for the problem and decided to use it as the main source of training and testing examples.

However, although the dataset presents itself in a very concise and organized manner, to make our model capable of learning the maximum from it, some preprocessing steps that would standardize and increase the dataset synthetically were done. Some of this procedures included, image processing techniques such as bounding box cropping, image resizing, feature reduction and normalization. Also, some image enhancement procedures such as randomly brightness, contrast and blurring adjustments were implemented so that the dataset would look more heterogeneous.

Once the dataset routines were all implemented, the model needed to be developed in order to learn the patterns found on these images. To do that, a deep convolutional neural network built using the Google's TensorFlow library (Abadi, 2015) was developed and trained to accomplish the final goal of recognizing digit sequences on street view images. This model features 7 convolutional layers, 3 fully connected layers and a set of softmax classifiers on top of it that is able to take 54 x 54 image pixels as inputs, and outputs predictions referring to the sequences on these images.

Undoubtedly, one of the most challenging parts of this project was to develop such as model to attain good results on the testing phases, and after considering many architectures, data processing algorithms, and parameters combinations, I was able to find a suitable solution. To ensure that these results were

valid and trusted, the most famous techniques in the field of machine learning such as cross entropy was carefully implemented so that the results could be demonstrated safely. Also, we considered many of the most common metrics to measure the model's results and compare it with high benchmark results publicly available and trusted.

In the end, the model was able to accomplish the goals stablished on the benchmark with some advantages such as its size and time required to train and process inputs. Surely by evaluating the benchmarks achieved on this project, I am assured that it does meet the expectations I had at the beginning of the project and it does would be of a great asset on solving problems that need such expertise.

Improvement

Although the proposed solution attained the benchmark values defined as thresholds for its success, I believe that some improvements could result in even better results. First off, one thing that did play an important role when developing this model and improving over trials was how deep the model became. As mentioned in (Ian J. Goodfellow, 2013), and many other resources regarding neural networks, in most of the cases, going deeper (in number of layers) offers more advantages as opposed to increase the size of each layer. Overall, this scenario presented to be true in this model's development since the model started off with only 3 convolutional layers and 1 fully connected and ended up with 7 convolutions and 3 fully connected. The reason behind this increase in the model's complexity was solely the fact that as the model got deeper, the results were getting better. Following that reasoning, I believe an even deeper model could reach better solutions, and one example for that is the model published at (Ian J. Goodfellow, 2013) that has more complex and numbered convolutional layers to solve the same problem.

Another point that might improve upon the current solution is the method used to train the network. We decided to use the Adam optimizer algorithm as the main optimizer. This optimizer already implements learning rate decay and momentum for us, which makes the training process less sensitive to hyper parameter tuning. However, a finer tuned Gradient descent with momentum and learning rate decay could reach better final results. Also, since there are many others optimizers to choose from such as Adagrad, Adadelta and RMSProp, a further investigation has to be done in order to find the one that best fits to the problem.

Another point that could increase the robustness of the predictions would be to develop a technique that would make the model able to better recognize when an image with no digits is being assessed. This model is able to do that when dealing with images with at least one digit. Because of the way it was trained, it knows that when an image with only one digit is presented, the places for the second, third, fourth and fifth digits are empty. However, because the dataset does not contain images with no digits on it, the classifier will always try to recognize a first digit on an image even though this digit might not exist. Luckily, the score, which tells how sure the model is about the existence of a digit in such an image might be small.

References

Abadi, M. (2015). *TensorFlow: Large-scale machine learning on heterogeneous systems*. Retrieved from tensorflow.org

Diederik Kingma, J. B. (2014). Adam: A Method for Stochastic Optimization.

Ian J. Goodfellow, Y. B. (2013). Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks. *CoRR*, 13.

LeCun, Y. (1998). Gradient-based learning applied to document recognition. *IEEE*.

Li, F.-F. (2013). *ImageNet: crowdsourcing, benchmarking & other cool things*. Retrieved from ImageNet : http://www.image-net.org/papers/ImageNet_2010.pdf

Walter Daelemans, K. M. (2008). *Machine Learning and Knowledge Discovery in Databases.* Springer.

Yuval Netzer, T. W. (2011). *Reading Digits in Natural Images with Unsupervised Feature Learning*. Retrieved from The Street View House Numbers (SVHN) Dataset: http://ufldl.stanford.edu/housenumbers

Yuval Netzer, T. W. (2011). Reading Digits in Natural Images with Unsupervised Feature Learning. *NIPS Workshop on Deep Learning and Unsupervised Feature Learning*.