Capstone Project

Machine Learning Engineer Nanodegree

Diego Gómez
December 20th, 2016

I. Definition

Project overview

Artificial Intelligence research has focused for decades in allowing machines to perform tasks that only humans are capable of. One of such tasks is visual recognition, the capacity to recognize objects or symbols gathered with visual receptors such as the eyes and cameras. A common application of this task, which has been studied thoroughly, is digit recognition. It has been intensively tested in scenarios with single digits and white backgrounds. By creating machines with these capabilities, the automatization of various tasks has been permitted. Thus, enabling a machine-human symbiosis that enables humans to employ their time in different tasks, while machines are used to recreate what people had to do by themselves.

A more complex task involving digit recognition is the recognition of multi-digit numbers in real world images. In this scenario, machines need not only to learn about single digits, but they have got t11o recognize what is and what is not multi-digit number, and not to confuse other objects as numbers. Developing a machine like this would allow automatic digital mapping, while also benefitting kids in order to learn and recognize numbers, blind people recognize note/bill denominations, and complex systems to solve understand mathematical equations. These are only some of the endless unthought advantages a machine with number recognition in real world images could bring to society.

Knowing the benefits multi-digit number recognition could bring, a machine learning based classifier capable of distinguishing numbers was created. Specifically, it was built as a deep neural network, and was trained images from natural scenarios that contain. The dataset used to create such a program the Street View House Numbers dataset (SVHN), which is a large-scale dataset of house numbers in Google Street View Images.

Problem Statement

The objective was to create an image classifier, capable of recognizing numbers in real world pictures, which makes accurate predictions. In order to complete the job, the following tasks had to be solved:

1. Download the SVHN dataset.
2. Explore the dataset.
3. Preprocess the downloaded dataset.
4. Train a classifier with the training samples taken from the dataset.
5. Refine classifier.

Metrics

In pursuance of an efficient and useful application that solves the problem stated, it had to be tested against metrics. A brief description of the metrics used is found below.

Accuracy: The proportion of correctly classified samples.

$$\frac{true\ positives + true\ negatives}{num.\ samples}$$

A higher accuracy means that the classifier is performing better at recognizing the value of the numbers showing in the input images. It was important to train a model that scores a high accuracy, so that users of the application receive correct feedback from what is going on in their surroundings.

For this project, an accurate prediction made by the classifier was only valid if every digit from the number predicted is exactly the same as the number found in the image.

II. Analysis

Data Exploration

The SVHN dataset is composed by two different groups of formatted images. The first one contains original images with character level bounding boxes, while the second one contains 32-by-32 cropped images centered around a single character. Given that the first set of images could provide more information when training the classifiers, it was the one used in this project.

The first format of images contains three different groups of images, named *training*, *testing* and *extra*, respectively. All three datasets are composed by a group of natural scene images which may contain digits present, a file called '*digitStruct.mat*' that describes the bounding boxes for each digit, and the number that appears in each image. Additionally they contain a Matlab file that allows users to visualize images with the bounding boxes, surrounding the digits.

All three image datasets are present in the *.tar.gz* format, so they needed to be extracted in order to use the image files. A brief description of each dataset is shown below:

	Size (bytes)	Number of Images	Number of digits
Testing	356,832,724	13068	26032
Training	626,161,412	33402	73257
Extra	3,504,414,984	202353	531131

Table 1: SVHN sub-datasets properties

Testing and *training* datasets were already separated, and this structure was kept during the project. The *Extra* dataset contained too many files to be able to process it with the current infrastructure, nevertheless it could be used to extract a set of samples to generate the *Validation* set.

Files were extracted in different folders, according to their containing folder, and some were randomly sampled to visualize them and understand their structure. Figures 1, 2 and 3 show some samples found in each dataset.

Figure 1: Sample images from the *training* dataset.

Figure 2: Sample images from the *testing* dataset.

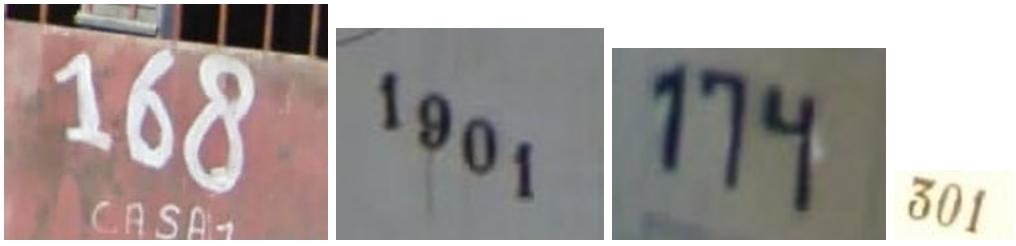

Figure 3: Sample images from the *extra* dataset.

The images presented above show that the dataset contained images with different dimensions, number sizes, and visibility. This means that a preprocessing stage was needed in order to correctly develop a classifier capable of recognizing the digits. Additionally, a detailed analysis of the image properties was made, so that it was easier to understand the characteristics of the dataset on a bigger scale. Below, images *height* and *width* properties was analised for each subset of images.

	Mean (pixels)	Max. (pixels)	Min. (pixels)	St. Deviation
test	71.566	516	13	52.752
train	57.213	501	12	36.180
extra	60.800	415	13	29.747

Table 1: Image height description for each sub-dataset.

	Mean (pixels)	Max. (pixels)	Min. (pixels)	St. Deviation
test	172.583	1083	31	122.586
train	128.285	876	25	80.546
extra	100.389	668	22	49.786

Table 3: Image width description for each sub-dataset.

SVHN dataset image representation was found to be very heterogeneous. Regarding image sizes, height average between datasets is similar, while width varies much more. Images height range between 12 to 516 pixels, while width ranges between 22 and 1083 pixels. Additionally, the standard deviation is very close to the mean for the *train* and *test* dataset, and almost half the mean for the *extra* dataset.

Finally, in the training dataset digit amount in image numbers range between 1-6. Majority of the numbers length range between 1 and 4, with most of the images having only two digits. Because there aren't many samples for numbers with 4 to 6 digits, it was projected that the results for predicting numbers with these many digits wasn't going to be as successful as with the other kind of images. Given this data, solution was focused on identifying numbers with up to 5 digits.

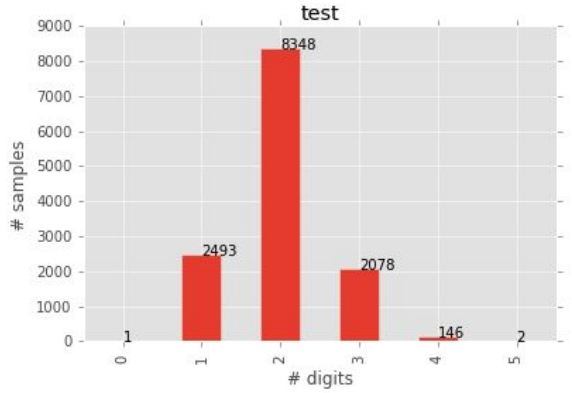

Figure 4: *Test* dataset images digit count

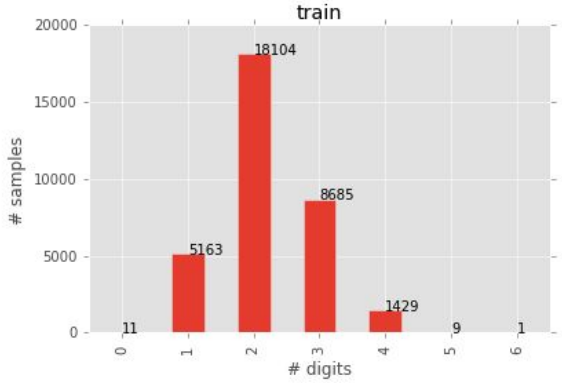

Figure 5: Train dataset images digit count

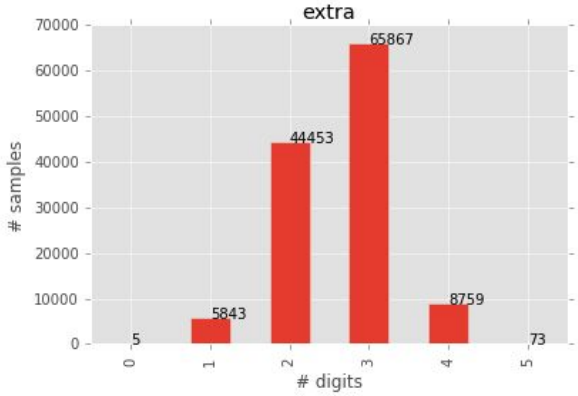

Figure 6: *Extra* dataset images digit count

Exploratory Visualization

In this scenario, picture visualization was selected as the most useful form of visualization. this approach is beneficial, since it allows us humans to understand what are the numbers implicit in an image. Trying to understand images through bits and bytes is not an easy task, since the numbers are found in natural scenarios. An example of the image visualizations is found in figures 1,2, and 3.

Algorithms and Techniques

In order to solve the problem stated, images needed to be processed, so they became homogeneous and the classifier can train over relevant information. One technique for processing the data is cropping the digits from the images, adjusting their sizes and using a filter to reduce their depth dimensions.

To train the classifier, a convolutional neural network was used, with multiple softmax classifier outputs. Convolutional neural networks (CNNs) are a specific kind of artificial neural networks that have been proved to be useful when solving image-processing problems. This is because convolutional neural networks implement one or more kernels that are able to process and abstract the properties of input data. In this project, CNNs were used to characterize multi-digit patterns, so that number recognition was possible. Approaching the solution in this way provided better results than using previously known algorithms, nevertheless it could be much more expensive computationally, given that processing the convolutions/kernels in the network require high computational resources.

The CNN created took the preprocessed training data and minimized its own loss function, which allowed it to predict better new results. Validation data was used as an input to understand how the network managed unseen samples, and training data helped to determine the accuracy of the classifier with new data (which hadn't been used during the training phase).

It's important to clarify that instead of using a single output layer, five different softmax layers were implemented. Each layer was is used to recognize the digit, for each position, that the number present in an image had. An additional *blank digit* value was used to represent the lack of a digit. This, to allow numbers with less than five digits to be classified.

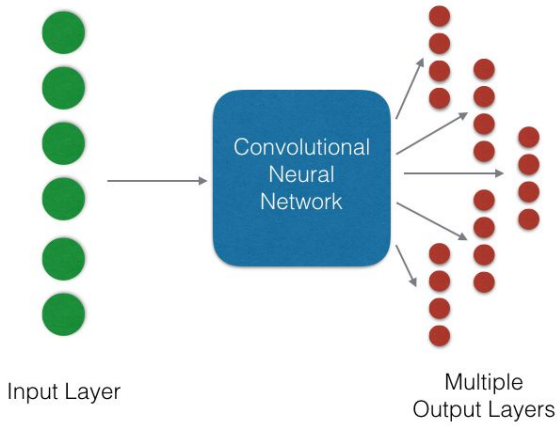

Figure 7: Design of the CNN (number of nodes is reduced for visualization)

Various methodologies were tested to train the neural network in the best way. Weight regularization, dropouts, rectifier linear units and decaying learning rate were some of the different techniques used to generate the best model possible in this project.

Benchmark

Classification models trained during the project were compared against each other in accordance to their accuracy when classifying the unseen testing dataset. The higher the accuracy, the better.

According to Google and Stanford[1], human accuracy when recognizing street numbers range between 82.0±2.0% and 100%, depending on the size of the images used. They also predict the accuracy of correctly detecting the numbers found in the dataset as 98%. A useful classifier capable of recognizing these digits should be close to what humans are capable of, hopefully better. So a benchmark of 95% was established as the lower bound for a successful classifier.

III. Methodology

Data Preprocessing

To begin with, a *Validation* dataset had to built with in order to train the neural networks. In order to create it, a random group of 2000 images were selected from the *Extra* dataset, and were added to the new *Validation* dataset. It is important to mention that only 125,000 out of 202,353 images could be extracted from the *Extra* dataset, and these were used to create the *Validation* one.

Given that images had different size formats, each had to go through a preprocessing phase, where they were changed to a proper formatting for the neural network to use. To begin with, digit sequences were cropped from the images using the edge information of the boxes that surrounded them. Each cropped image was resized to a 28x28 size, based on the image format used in LeNet's architecture. Having now images with dimension (28, 28, 3), a feature scaling process was applied to reduce the range of their RGB values from [0, 255] to [-1, 1], so that it was easier for the classifiers to learn. Additionally, because colour was not as important as figures in digit feature extraction, the RGB values of each image were merged through finding their average and replacing the depth of the image to this value. Finally, each image was transformed into a *numpy*. As a result, each final image dataset was now composed by images containing the sequence of digits of their former images, being 28px width, 28px height, and 1 value for its colour.

On the other hand, label datasets extracted from the *.m* files contained an array of digits to represent the number corresponding to each digit. Each array was 6 spaces big, so that it could represent all 6 digits that could appear within each picture. A blank character had to be used, represented by digit 0, in order to allow numbers shorter than 6 digits long. Other digits range from 1 to 10, each representing the same digit in the picture, except for number 10, which represents digit 0. Information for each number's digits included 6 digits even though for training purposes only 5 digits were used.

[1] http://ufldl.stanford.edu/housenumbers/nips2011_housenumbers.pdf

Processed datasets were stored in the *svhn.pickle* file.

Implementation

In order to classify the data, a convolutional neural network was designed, implemented, and trained with the data generated in the preprocessing phase. It was designed taking into account LeNet5 architecture, which classifies single digit recognition, but changing the parameters and output layer numbers. This neural network received images with dimensions (28, 28, 3) as inputs, and its outputs were the probabilities of every digit being at a given position. It is worth noting that it was trained with the training dataset, which contained 33402 images, was validated with the validation dataset which contains 2000 images, and was tested against the test dataset, which contains 13068 images.

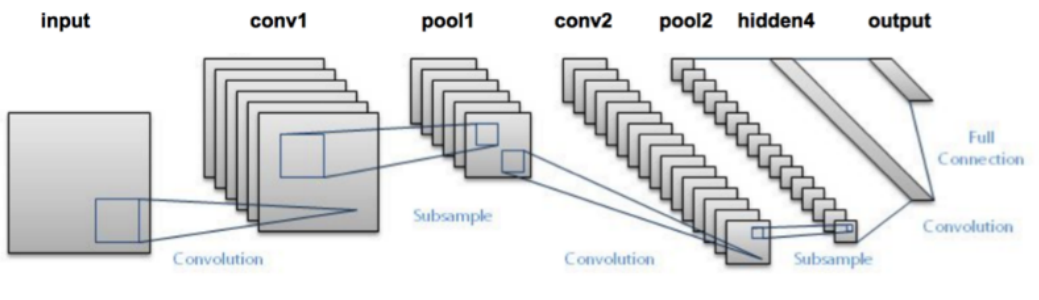

Figure 8: LeNet5 architecture diagram. Used as base for this project's CNN.

The first hidden layer of the convolutional network consisted of a convolution applied to the input, using patch sizes of 5 pixels (both width and height), an output depth of 16 pixels, a stride length of 1 pixel, and VALID as the padding type. This means that striding didn't allow the kernel to go past the edge of the image. The output of this layer was then passed through a max-pooling layer, which reduced the extent of the feature maps. That layer used both a kernel size and a stride size of 2 pixels (both width and height). Next layer in the network was a convolutional network with the same characteristics of the one mentioned above (only difference is the input size), which is followed by a max-pooling layer with same properties as the second hidden layer mentioned previously.

Resulting outputs were then transformed into a normal 'artificial neural network' layer, by reshaping each into an array of dimension 784, generating a layer of dimensions ('num_input_samples', 784). Given layer was used as the input of 5 different classification mini neural networks (part of the CNN). Each mini neural network was used to learn how to classify each one of the first five digits present in every image sample. Mini neural networks had a hidden layer with size 256, which then output to a layer of shape 11, meaning it learned which one of the 11 possible values for every digit was the most probable one. The result of these final layers are called logits.

A loss function used to optimize the weights of the neural networks consisted of the sum, amongst all output logits, of the sparse softmax cross entropy between the correspondent training label digit and the logits. This means that the neural network learned by minimizing the cross entropy of the softmax value between each logit and hot-encoded version of the corresponding training digit's label. Finally, batch gradient descent was applied to optimize the loss function, using 30000 batches of size 16. Learning rate was set to a value of 0.005 given that it was a neat value that gave good first results and didn't make the loss value diverge after multiple iterations.

To obtain the probability of each value in a given position, for a given image, the softmax function was applied to the corresponding output logit. Choosing the value with highest probability within each logit, and then packing each selection into an array resulted in the most accurate prediction by the neural network of the number present in that image.

It is worth mentioning that all the weight values used in the convolution phases of the network are sampled from a truncated normal distribution with std. deviation of 0.1. Moreover, all biases used in those layers are initialized with zeros. Conversely, the mini neural networks hidden layers and output layer weights are initialized with values samples from the truncated normal distribution with std. deviation of 0.1, while its biases are initialized with the value 1.0.

Refinement

The implementation described above gave good results. The accuracy obtained when predicting the test dataset was 74.3%. Nevertheless, tuning up the many hyperparameters that a convolutional neural network allowed some improvement on its final performance. Additionally, various advanced methods were be used to boost the correctness of the neural networks. New neural networks were generated when adding modifications, and each was fully trained and tested with the same data as the original one. Also, every setup was tested with various hyperparameter configurations, and the best results obtained are described below:

Alternative configuration #1:

This model was very similar to the first solution stated, with only two major changes. The first one is that for each hidden layer that comes before the output softmax layers (in the mini 'neural networks'), a dropout filter was added with a keep probability of 50%. Additionally, the learning rate was modified so that it had an exponential decay, instead of being constant, as previously. The exponential decay had an initial learning rate of 0.05, 10000 decay steps, and decay rate of 0.96 which followed a staircase function.

Training this model resulted with a final test accuracy of 73.1%.

Alternative configuration #2:

Configuration #2 was the same as the alternative configuration #1, but adding an additional regularization factor to the loss function. The regularization rate used in this scenario was 0.05 and was applied to the weights and biases of the hidden layer of each mini neural network.

It scored a final test accuracy of 75.8%.

Alternative configuration #3:

This implementation was the same as the alt. configuration #1, with the only differences that convolution kernel striding type is done with VALID as the striding type. Having such kernel striding means that no padding type is when striding through the images.

It scored a test accuracy of 74.2%.

After trying with different configurations and tuning different hyperparameters, the best model obtained was the alternative configuration #3. It did better than other alternative configurations and also better than the base configuration built. This affirmation is based on the accuracy obtained by each model.

IV. Results

Model Evaluation and Validation

The final model chosen is the alternative configuration #3 described previously. It was the classifier trained that performed with the highest test accuracy, which was 1.8% higher than the one obtained with the basic (first) neural network modeled.

It achieved the following results after training:

- Training set:
  - Minibatch loss: 3.516663
  - Minibatch accuracy: 83.8%
- Validation set:
  - Accuracy: 62.2%
- Testing set:
  - Accuracy: 76.1%

Its predictions are accurate for approximately 3 out of 4 different images which is somewhat a good accuracy, but is not suited for using as a replace of humans. So using it under a supervised scenario is recommended. It would be also reasonable to use this model as a base point to improve over it and reach a classifier with a higher accuracy.

The model is not robust enough for the problem. Though It is much better at classifying numbers with less digits, it isn't accurate enough for usage.

Justification

The accuracy obtained by the final classifier is approximately 20% lower than the benchmark used. It is a low score, given that using it for automatic mapping purposes requires it works just as or better than a human. Nevertheless it is a good enough classifier that allows insights of the data which is used as input.

It does solve the problem of generating a model capable of classifying images with different digits on it. An accuracy of 76.1% allows it to recognize more than half of the images presented to it. Nonetheless, it is not good enough as a human being and therefore it shouldn't replace one on its job.

Even though the classifier may recognize incorrectly a number, it is capable of recognizing some of the digits in the image. Probably a bigger and more diverse training dataset would allow higher accuracy for every digits, which means a higher general accuracy score.

V. Conclusion

Free-Form Visualization

After training the classifier, it was tested by picking random samples from the testing dataset and visualizing how the prediction compared with the real labels. Real labels are visible in the images, while predictions are found above each image.

Figure 9: Sampled images with respective predictions made by final classifier

After careful examination, it was found that the classifier is fails the most with images that have more than two digits. Since the input dataset contains many mostly images with numbers of two digits or less, it is reasonable understandable that there aren't many samples that allow the classifier to learn numbers with more digits.

It also confuses some digits when they look pretty similar, such as the 5 and 9, the 3 and 5, and some more. Nevertheless its prediction does approximate to the real value if the images, missing the correct value for one or two of its digits, like in the following examples.

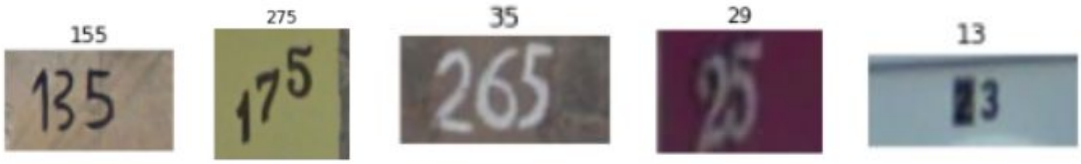

Figure 10: Sampled images with predictions showing how classifier correctly labels some digits

It seems like using a bigger training dataset would allow the classifier to recognize independently each digit better, and thus giving an overall higher accuracy.

Reflection

Using convolutional networks to solve image recognition problems is an approach that nowadays is possible, and more popular because of the development of better and cheaper processors. Furthermore,

using a neural network with multiple output softmax layers is a unique approach that allows multiple digit recognition without the use of more than one classifier. This allows a faster classification after training, which enables uses for real-time predictions; this wasn't possible beforehand with the techniques known.

It was necessary to understand what would the best design for the convolutional neural network be. Researching on known state of art neural network models allowed to start with a base of something known to be useful, and from then on playing with multiple hyperparameters allowed to improve the results. It was a time-consuming task to do this, given that training each network with 30.000 samples could take between 15 and 30 minutes every time with an average laptop. Knowing the best design for a network in each scenario is a task with non-deterministic results, not only because the dataset is different every time, but because variable initialization can also be random. This leads to different results with similar or equal training processes, which makes it harder to have consistent results.

Feature/data processing is also another task that needs contemplation. Thinking about the best way to retain extract information in order to use it later on in a classifier takes time. With images it might be easier, since whatever it is we need from the images, we probably can see it. On the other hand, dealing with sets of data with multiple dimensions requires to know what information is relevant; otherwise learning could take much longer and results wouldn't improve.

Having the possibility to observe how the results changed based on small and big changes was very interesting. Besides, I realized that training the model was not as easy as thought, after obtaining unexpected results when modifying the neural network.

Finally, working on a machine learning problem, independently and from bottom up, is something that gave me satisfaction. The final model does solve the problem with a reduced amount of accuracy, and it doesn't fulfill my initial expectations. Still, it is the best model that I could make with my current skills and I am proud of its final functionality. In a later opportunity I'll be able to upgrade it to the point it classifies digits better than humans.

Improvement

There are multiple ways in which this project could be improved. First of all, using a non-skewed dataset would allow the correct learning of numbers with multiple amount of digits. In this scenario there weren't enough samples to learn how to classify correctly numbers with more than 3 digits. Either obtaining more real samples or creating an artificial are valid approaches to have more data.

Furthermore, training the neural network with many more samples could allow the neural network to score a higher final accuracy. In this project approximately 40.000 samples were used for training, but to have a model that performs better at least 200.000 samples should be used. This could make the learning take longer, which means also that using a better infrastructure is a must. For this project a laptop with two processors were used, but better solutions use multiple GPUs to accelerate the process.

One final possibility for improvement would be the use of a different type or structure of neural network. A valid example would be to try with recurrent neural networks which in theory could allow the prediction of images with an infinite amount of digits on it.