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, this has enabled a machine-human symbiosis that allows 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 to 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.

This was a supervised learning type of problem, that required creating a model and training it with labeled data, so that it could predict the labels of unseen data. In this case, the input data were the images from the

SVHN dataset, and the output label was the sequence of digits that each image had. Since this problem input was a big amount of images, it could be specified as a deep learning problem and solved through Convolutional Neural Networks (more on that later).

Metrics

In pursuance of an efficient and useful application that solves the problem stated, it had to be tested against some kind of metric.

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. This metric was used because it was simply to calculate and had to be calculated for every training (batch). At the same time, this metric allowed us to report the behaviour of the model with precision. Other types of metrics like the F1 score could be harder to understand, taking even more time during training, so they were not used.

For this project, num. samples true positives and negatives were defined in terms of correctly classified digits in an image.

II. Analysis

Data Exploration

The SVHN dataset was composed by two different groups of formatted images. The first one contained original images with character level bounding boxes, while the second one contained 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 selected dataset contained three different groups of images, named *training*, *testing* and *extra*, respectively. All three datasets were composed by natural scene images which could contain digits, and a file called '*digitStruct.mat'* that describesd 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 were present in the *.tar.gz* format, and in order to use the image files they had to be extracted. A brief description of each dataset is shown in *Table 1*.

	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 separation structure was kept during the project. The *Extra* dataset was used to add more training samples and generate the *Validation* set, as explained in the ***Methodology*** section. Dataset files were extracted in different folders, according to their dataset name, 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.

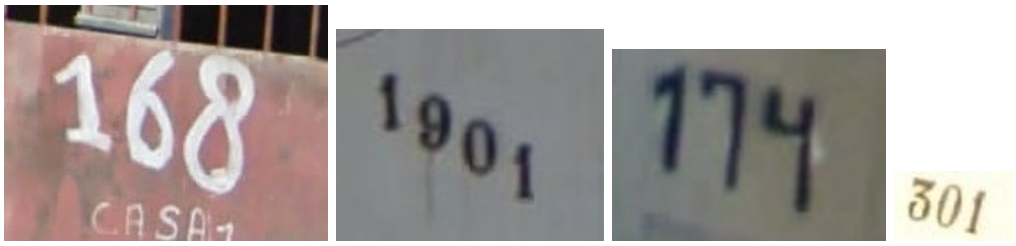

Figure 3: Sample images from the *extra* dataset.

The images in *Figures 1-3* show that the dataset had pictures a variety of dimensions, number sizes, and visibility. This means that to correctly develop a classifier capable of recognizing the digits, a preprocessing stage was required. 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. *Tables 2-3* present the *height* and *width* properties 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 2: 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. Average image height was similar between datasets, but their width varied much more. There was also a big difference between image sizes, having a max. width 49 times bigger than the minimum, and a max. height 43 times bigger than the minimum. Finally, standard deviation was almost as high as the mean for some datasets. These features suggested that image dimensions very different amongst each dataset, and that a well-thought preprocessing strategy was needed (explained in the ***Methodology*** section).

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 weren'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. Solution was focused on identifying numbers with up to 5 digits.

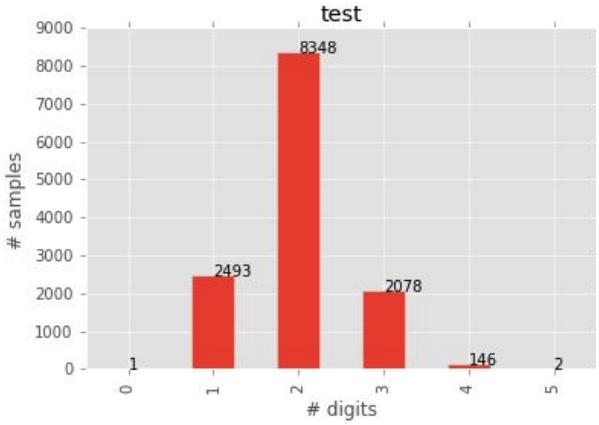

Figure 4: *Test* dataset images digit count

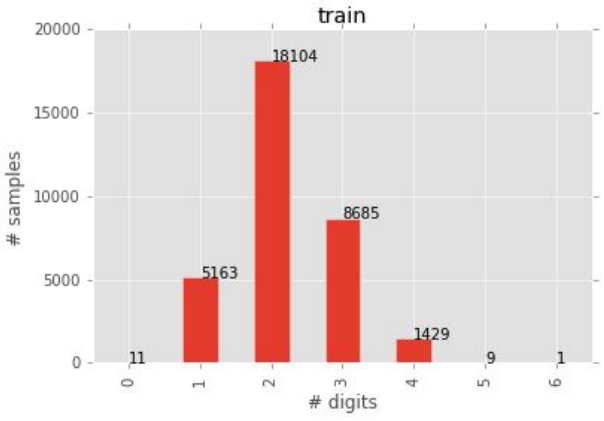

Figure 5: Train dataset images digit count

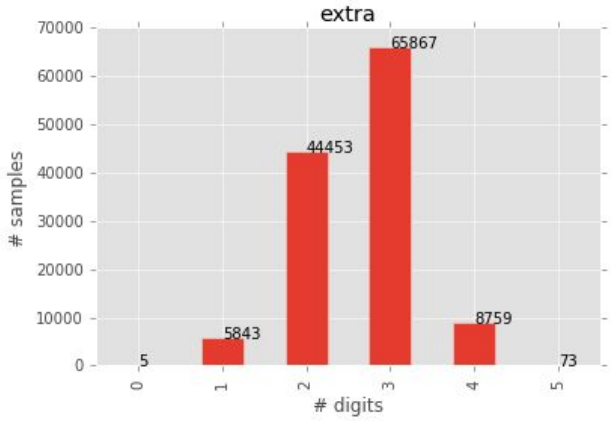

Figure 6: *Extra* dataset images digit count

Exploratory Visualization

Picture visualization was selected as the most useful form of visualization in this project, given that it allows us humans to understand what are the numbers implicit in an image. Otherwise, trying to understand numbers would be harder through visualizing bits and bytes.

An example of the image visualizations is found in the ***Analysis*** section above, and in the **Free-Form Visualization** section (near the end).

Algorithms and Techniques

To train the classifier, a Convolutional Neural Network with multiple softmax classifier outputs was used. 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 CNNs implement one or more kernels that are able to process and abstract the properties of input data. For this project, they 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 was a computationally expensive task, given that processing the convolutions/kernels in the network require many 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 one of these layers was 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 the classification of numbers with less than five digits.

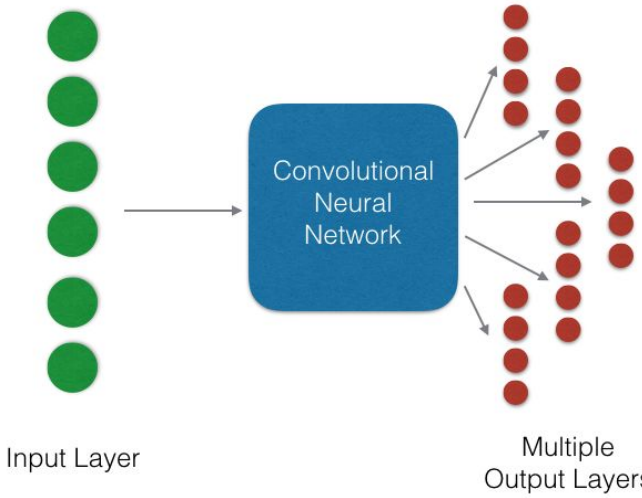

Figure 7: High level design of the CNN (number of nodes is reduced for visualization).

In the particular CNN that was used for this project, the following features were used:

1. **Pooling layers:** Artificial Neural Network layers that are capable of shrinking features (processed images) while retaining the most important information. Their use allow to reduce the computational load and to extract features of an image disregarding their specific locations.
2. **Dropouts:** A filter that that, with some probability, invalidates each feature traveling between two layers. It allows networks not to overfit by allowing the network to generalize better.
3. **Local Response Normalization[1]:** Local normalization scheme (neuronal activation computation) that aids the neural network in generalization.

Details of how these techniques were used are found in the ***Methodology*** section.

Benchmark

According to Google and Stanford[2], human accuracy when recognizing street numbers ranged between 82.0±2.0% and 100%, depending on the size of the images used. They also were able to train a model with an accuracy of 98% for this dataset.

A useful classifier capable of recognizing these digits should be close to what humans are capable of, hopefully better. Given the restrictions in the infrastructure used for this project, the benchmark chosen was set to be 80%, which is the minimum accuracy obtained by a human being for this task. It was a high accuracy benchmark, that allowed to focus on training good models.

[1] http://www.cs.toronto.edu/~fritz/absps/imagenet.pdf
[2] http://ufldl.stanford.edu/housenumbers/nips2011_housenumbers.pdf

III. Methodology

Data Preprocessing

- ### Images to grayscale

  Original images had three channels (Red, Green and Blue). They were transformed to grayscale so that the digit colours wouldn't affect in their recognition and the training could run faster.

- ### Cropping

  Digit sequences were cropped by calculating the surrounding edges of the whole number. A little padding was added before cropping given that not all numbers were equally spaced to the borders.

- ### Reshape

  Previous crops were resized each to a format of 32x32, since image dimensions had to be the same in order to use them as the input of the CNN. This size was chosen given that it has been proven to be successful when creating digit classifying models such as LeNet[3].

- ### Standardization

  Image features were scaled through standardization, which makes the values of each feature to have zero mean and unit variance. This allows the classifier work properly, since each scaled feature will contribute approximately the same as the rest.

- ### Create Validation dataset

  Extra and *Train* datasets were merged and shuffled. Then, a new *Train* and *Validation* dataset were created by splitting the merge. The final dataset sizes are shown in *Table 4*.

Train	230,000
Test	5,754
Validation	13,068

Table 4: Final dataset sizes.

- ### Create label datasets

  Labels were extracted from the '*digitStruct.mat'* files into arrays of digits that represent the number corresponding to each image. Each array was 6 spaces big, so that it could represent all 6 digits that could appear within each picture; only the 5 first digits were used. Every space would contain a digit from 0 to 9, representing that digit in a given position. Additionally, a blank character representing an 'empty digit' had to be used, to allow numbers shorter than 5 digits to be represented. It was modeled by the number '10'.

[3] http://yann.lecun.com/exdb/lenet/

Processed datasets were stored in the *svhn_grayscale.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's design was based on models described in image recognition research papers[4].

All the weight values used in the convolution layers are initialized with the Xavier algorithm[5], which sets the weights to values that allow the network to learn better. Furthermore, all biases used are initialized with the value 1.0.

1. Convolution layer #1

   Receives the preprocessed images as input.
   - Patch size: 5x5
   - Kernel depth: 16
   - Padding type: VALID

2. Activation

   Next, the ReLU function is applied to the previous output.

3. Local response Normalization

4. Pooling layer #1

   Previous output goes through a max-pooling layer.
   - Kernel size: 2x2
   - Stride size: 2x2
   - Padding type: SAME

5. Convolution layer #2

   Receives the preprocessed images as input.
   - Patch size: 5x5
   - Kernel depth: 32
   - Padding type: VALID

6. Activation

   The ReLU activation function is applied to the previous output.

7. Local response Normalization

8. Pooling layer #2

   Previous output goes through a max-pooling layer.

[4] https://arxiv.org/pdf/1409.1556v6.pdf
[5] http://andyljones.tumblr.com/post/110998971763/an-explanation-of-xavier-initialization

- Kernel size: 2x2
- Stride size: 2x2
- Padding type: SAME

9. Convolution layer #3

   Receives the preprocessed images as input.
   - Patch size: 5x5
   - Kernel depth: 64
   - Padding type: VALID

10. Activation

    The ReLU activation function is applied to the previous output.

11. Dropout layer
    - Keep Probability: 1.0

12. Reshape

    Reshapes previous layer into a 64x*num_hidden* layer.
    - Num. Hidden: 64

13. 5 Logit (output) layers

    Receive previous layer as input. Each with shape 11, contains the probability of each digit (0-9 plus '10' for *empty*) in a given position.

- Loss

  The 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. The CNN learned by minimizing the cross entropy of the softmax value between each logit and hot-encoded version of the corresponding training digit's label.

- Optimization

  Adagrad[6] batch gradient descent was applied to optimize the loss function.
  - Learning rate: 0.1
  - Batch size: 64
  - Num. steps: 10,000

- Predictions

  Classifier's image predictions were calculated by applying the Softmax function to the output values of each logit layer, choosing the maximum element in each output (amongst the 11 present), and packing them ordered into an array.

[6] https://en.wikipedia.org/wiki/Stochastic_gradient_descent#AdaGrad

Refinement

A CNN has many hyperparameters that can be tuned to improve it's accuracy, which can also be described in terms of the training, validation and testing datasets. For this project, training accuracy is calculated with the current training batch. The initial scores for the previous network were:

- Training: 81.2%
- Validation: 78.6%
- Test: 72.2%

Various hyperparameters were then modified, picking the best configuration each time, according to the validation and test accuracy. In every refinement the best parameter chosen was set in the model.

Learning rate

Learning Rate	Batch Accuracy	Validation Accuracy	Test Accuracy
0.2	61.2	60.1	66.3
0.05	82.8	83.9	75.4
0.025	80.9	82.2	73.2
0.375	86.9	83.5	75.0
0.0475	82.2	84.4	75.7

Table 5: Learning rate tuning scores.

Best learning rate was 0.0475.

Dropout Keep Probability

Keep Probability	Batch Accuracy	Validation Accuracy	Test Accuracy
0.5	72.5	77.5	71.8
0.75	77.2	81.9	74.2
0.875	80.0	82.8	45.4
0.99	83.8	83.7	75.4

Table 6: Dropout keep rate tuning scores.

Best dropout keep prob. was 0.99.

Num. Hidden

Number features in the only Fully Connected layer (reshape layer).

Num. Hidden	Batch Accuracy	Validation Accuracy	Test Accuracy
256	86.6	87.2	78.2
1024	86.9	87.0	79.1

Table 7: Num. hidden tuning scores.

The best number of fully connected layer features was 1024.

Number of Steps

Then, the amount of steps was increased to 30,000 to allow a deeper training and the results improved.

Num. Steps	Batch Accuracy	Validation Accuracy	Test Accuracy
30000	91.9	89.8	80.5

Table 8: Dropout keep rate tuning scores.

Finally, the previously trained model was allowed to train for 70,000 more steps, while iteratively decreasing the learning rate to 0.005 and the dropout rate to 0.85 every 10,000 or 20,000 steps. Refinement finished when the accuracy improvements were insignificant. The final accuracy obtained is shown in ***Table 9*****.**

Batch Accuracy	Validation Accuracy	Test Accuracy
91.2%	90.9%	81.6%

Table 9: Dropout keep rate tuning scores.

Refinement took approximately 40 hours and included testing several other improvements, which are not described in this document since most them were extensively tested without satisfying results. These other refinements included the use of:

- Exponential decay rates, variating the initial learning rate, decay steps and decay rate in many combinations.
- Variating the image size.
- Using three channels for the images.

Though long, the refinement process was fullfiling and succesful.

IV. Results

Model Evaluation and Validation & Justification

The final refined model accuracy improves upon the initial model implementation by approximately 12.0% in the validation accuracy, and 10% in the testing accuracy. It's predictions are much better than the original model and is robust enough for the problem, scoring above 80% for every single dataset, and above 90% for the training and validation dataset.

Moreover, the accuracy obtained by the final classifier is inside the benchmark proposed. Which means that a well trained classifier was achieved, one which could understand better than humans in some scenarios.

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

This illustration allows to verify how the classifier predicts correctly most of the images. Precision is really good and even incorrectly classified numbers have mostly correctly classified digits. With a little some more training (specially by following the suggestions in the ***Improvement*** section below), the classifier would be able to predict correctly all of them with no error.

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 more than

30 minutes every time with an average laptop. This had to be done several times, so the time consumed doing deep learning was even more.

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. Nevertheless, a greedy approach was used for this project and gave good 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.

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 successfully, though it could do better if trained with a more powerful setup. I am really proud of its final functionality and I'll keep upgrading 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 learning to classify better. In this scenario there weren't enough samples to learn how to classify correctly numbers with more than 3 digits, so 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 250,000 samples were used for training, but to have a model that performs better, more 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.