Using Deep Neural Networks for Object Detection in Images: A Paradigm Using the Google SVHN Dataset

Avraam Th. Tolmidis[1,]

Keywords: Deep Learning, Image recognition, Convolutional Neural Networks

1. Scope

In this document, the results of the Deep Learning Capstone Project of the Udacity Machine Learning Nanodegree will be presented. The aim of the project was to select a field where the knowledge acquired during the Nanodegree can be used to solve a real-world problem. The interest of the author was primarily Deep Learning, thus the project described was selected. The specific problem has been studied in the past, its current study however, provides hands-on experience with a real-world problem, and promotes deeper understanding of the topic. In addition, the specific project choice builds the foundations for working later on more complex and novel applications.

2. Introduction

The problem of identifying digits from street images is still considered an open problem. In the recent years there have been advances, mostly due to the use of Deep Learning architectures. Deep learning is a field in Artificial Intelligence that emerged during the recent years, building on the concept of Neural Networks. Neural networks in principle try to emulate the information processing capabilities of human neurons. They try to process complex data, and are used for both regression, as well as classification problems. However, due to the large number of computations that are required, there were limitations on the use of Neural Networks on large-scale problems, as it was not possible to use networks with many layers. As more efficient hardware became recently available, capable of performing large amounts of calculations, the use of Deep Neural Networks for solving difficult problems, such as the Image Recognition studied in the present, became more common. Image recognition using Deep Neural Networks yields an accuracy comparable to that achieved by humans, e.g. 97.84% (per-digit identification in images) [1].

Email address: atolmid@gmail.com (Avraam Th. Tolmidis)

3. Object Detection in Images

Object detection in images and videos, or their classification, is very important lately, as they enable a plethora of applications. It includes detecting objects and recognising patterns in sequential images/video frames. The use of such applications, and the demand for them has increased significantly during the last years, for many reasons, including:

- The massive use of cameras (including their integration in mobile phones), that makes endless sources of image and video available.
- The availability of more efficient hardware than in the past, capable of processing big amounts of data.
- The use of new/refined image processing techniques.

Some of the most common methods that have been used so far for this purpose are described in the following paragraphs.

Optical Flow [2] calculates the optical flow field of an image. Then, using the optical flow distribution characteristics of the image, it performs clustering. It is used for distinguishing a moving object from the background, achieving a relatively high accuracy. However, it is not considered a method suitable for real-time object detection, since a lot of calculations are required, and it is additionally sensitive to noise.

Point detectors try to find points in images that have distinct characteristics, relevant to their locality [3] [4]. They are points that do not get affected by changes in the point where the image was taken from, or illumination

In *Background Subtraction* we work with video images. Implementing the method, we follow in principle the following steps: We construct a representation of the background model, and look for deviations stemming from the model for each next frame in the video. If there is a change in a part of the image that is noteworthy, we consider this as a moving object. The pixels this image region consists of, are then processed further. There are two types of algorithms, *Recursive*, and *Non-recursive*.

- *Recursive Algorithms* [5] [6] update a single background model, based on each input frame. Since they do not use a buffer, they require less storage than non-recursive techniques. However, the model is influenced by images that extend to the more distant past, and this means that any errors in the model are propagated to future instances.
- *Non-Recursive Algorithms* [5] [6] on the other hand, use storage, whose requirements can sometimes be high, however, they are immune to the influence of frames that extend deeply in the past. A buffer is stored, that contains The L previous frames. By considering the changes in the frames of the buffer with time, an estimation of the image is performed. A common method is *Simple Background Subtraction* [4] [7] [8], where we try to find a motion detection mask D. In order to do this, we take the absolute difference between the current image, and a reference background image, which is usually the first frame of the video, without containing any foreground objects.

Frame differencing [9] detects moving objects by calculating the difference between two consecutive images.

Temporal differencing [10] on the other hand, detects moving regions by examining the pixel-wise difference between two or three frames in a video. It is a relatively effective method, with a somewhat lower effectiveness though, when the object moves slowly, or its texture is uniform [11] [12]. The object is no longer detected when it stops moving, since there is no difference between the consecutive frames.

For a more detailed study of Object detection in Images, the reader can refer to [13]

4. Deep Neural Networks for Image Object Recognition

In the present work we study the use of Deep Learning models in order to decode sequences of digits from natural images. Deep Neural Networks are used lately more often in object detection, due to their characteristics, which prove very convenient. In principle, we do not need to define specific features in order to detect objects and classify them. This will be performed by the network itself, through the update of the weights of the connections between the network layers.

The most limiting factor until recently was the restriction with regards to the number of layers and nodes that could be used. The large amount of computations required, was making it impossible to use more than a few number of layers. The evolution of the available hardware though, and the use of GPUs capable of performing a large number of computations per second, enabled the use of many more layers, up to tens of them. Furthermore, the growing availability of large datasets provides the conditions to train neural networks with a larger amount of data, hence improving their accuracy. Additionally, the use of concepts like *Convolutional Neural Networks* [14] [15], improve the accuracy of the detection to a level comparable with the capabilities of a human.

A *Neural Network*, consists of layers of *neurons*. A single neuron can be seen in figure 1

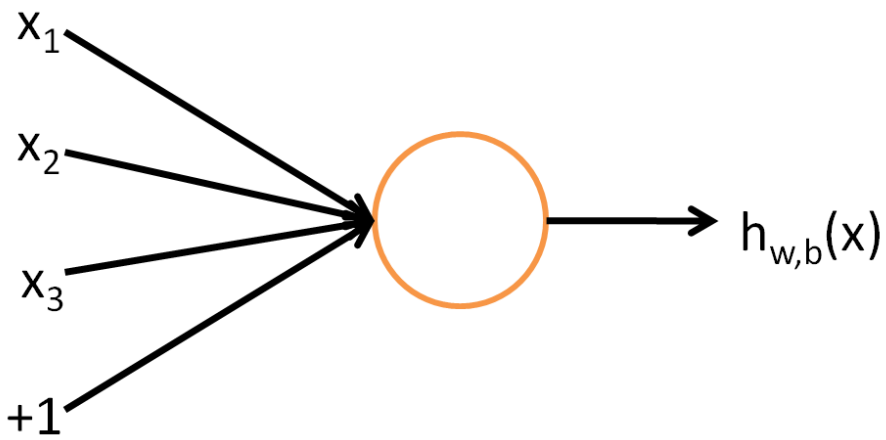

Fig. 1: A single neuron

The neuron has a number of inputs (x1,x2,x3,...) as well as an additional intercept term. Its output is $h_{W,b}(x) = f(W^T x) = f(\sum_{i=1}^{3} W_i x_i + b)$, and it's called the *Activation Function* Common activation function choices are the *sigmoid function*:

$f(z) = \frac{1}{1+\exp(-z)}$

or, the tanh, function:

$f(z) = \tanh(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$,

The tanh(z) function gives us the opportunity to have a function that has an output range centred at zero, since its output range is [1,1] instead of [0,1] that the sigmoid has.

Thus, the neural network is in principle a combination of layers of nodes, which can have multiple inputs, as well as multiple outputs. These layers are the *fully connected layers* - or the *feed forward neural network*. An example of a simple neural network is presented in figure 2

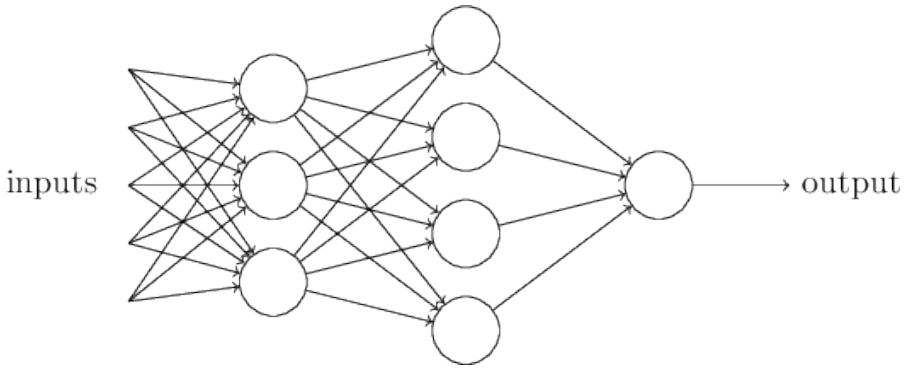

Fig. 2: A single neuron

A *Convolutional Neural Network (CNN)* is a network that can contain several functions, like convolution, pooling, non-linear functions, normalisation and fully connected layers. In most cases, there are convolutional layers, that are followed by normal, fully connected layers, like the ones described above.

CNNs are mostly used for image recognition, and each of their layers acts in practice as a filter, detecting specific features, or patterns. The first layers detect simplest features such as lines or curves, while each consecutive layer detects more abstract features, that may combine features from the previous layers. Using the data from the previous layers, the last layer is usually used in order to perform a classification task.

Each Convolution layer of the has a specific filter, which is able to detect features in the image, regardless of the position of the feature in the image. The filter is applied on the image data, and shifted multiple times, until it has been over the whole image. For the image that is represented as a matrix of pixel values, the filter will be a matrix of a smaller dimension (usually 3x3 or 5x5), Following at least one of the convolution layers, is usually a pooling operation, where the filter slides over the image data matrix by a number of pixels (called the *stride*). An element wise multiplication is performed between the two matrices, and the outputs are then added, and they provide the elements of the (final) output matrix or *Activation Map*. The Convolutional layers are

common, meaning the same parameters are shared by all features. A visualisation of the above process can be seen in figure 3.

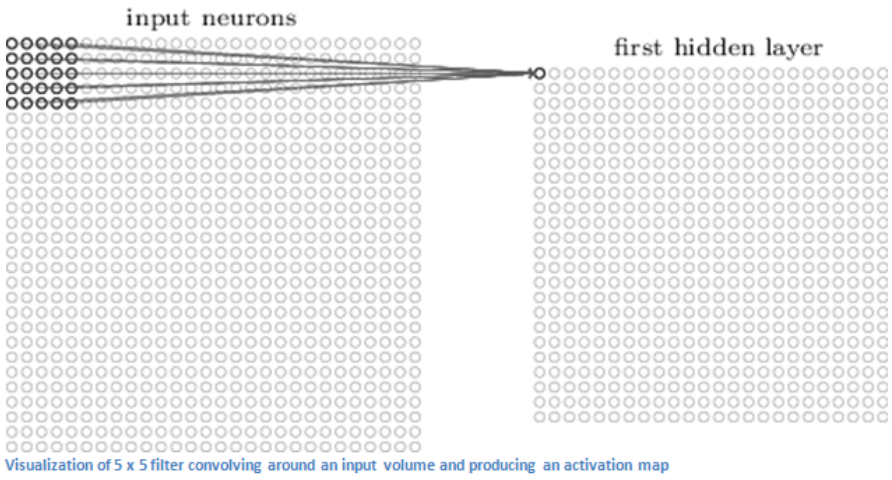

Fig. 3: Activation Map Creation Process

A commonly used function to introduce non-linearity to the model is the *Rectified Linear Unit (ReLU)*. It is usually used after each convolution, and provides as output the maximum between its input, and zero. the operation can be seen in figure 4.

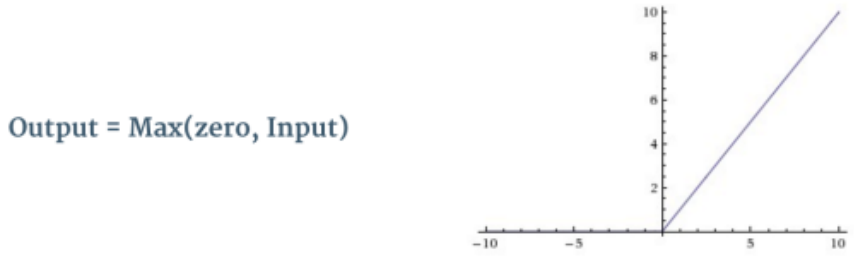

Fig. 4: ReLU Operation

The pooling, or *subsampling* function, is used to reduce the dimensionality of the data. There are different types of pooling operations, such as sum, max, average, etc. In principle, neighbouring elements in a matrix are summarized to one element in a new matrix, of lower dimensions. The new dimension is chosen as part of the model design, as well as what will be the pooling operation used. A Max Pooling operation is displayed in figure 5, which is taken from the material of the Stanford *CS231n: Convolutional Neural Networks for Visual Recognition* course.

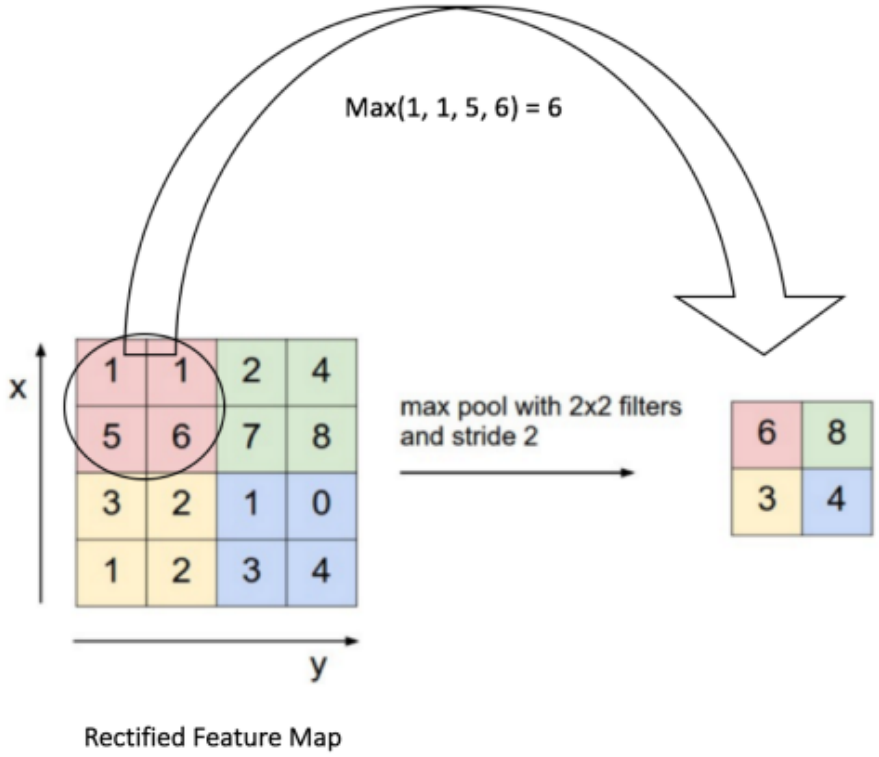

Fig. 5: A Max Pooling Operation

To avoid the effect of *overfitting* the data (learning to recognise the data used to train the network, but failing to accurately predict new data), there are several methods used, one of which in specific (that will also be a subject of study in the present work) is *dropout* [16]. With dropout, a percentage of the inputs to each neuron in a layer that it is applied, is randomly ignored. This results in higher accuracy levels when classifying new data, in the case where overfitting occurs. However, the network then requires more training, otherwise the desired result might not be achieved.

An example of a Neural Network with convolution layers is shown in figure 6, where the LeNet network is presented [15]. There are two Convolution layers, each followed by *max-pooling*, with two fully-connected layers, that provide the outputs.

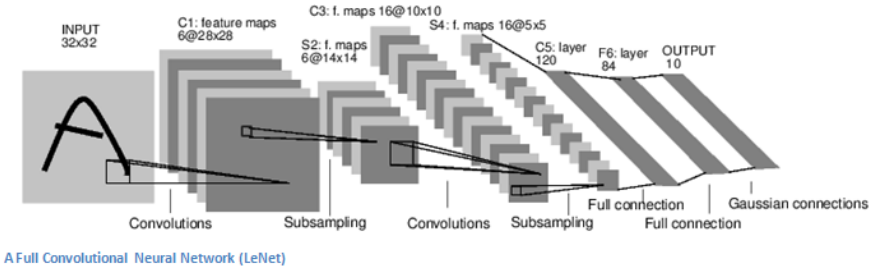

Fig. 6: The LeNet Architecture

5. Problem Definition

The purpose of this work is to study the behaviour of Deep Learning models when used in order to decode sequences of digits from natural images. For this task, Convolutional Neural Networks were used. The behaviour/performance of the model will be studied, with different network parameters (optimizer, techniques like dropout, etc.). It is compared to the naive approach where we only have a simple regression with a Gradient Descent optimizer. The implementation of the project will be a Python application, using different Python scripts for independent parts of the application.

6. Datasets and Inputs

The data that used was that from the Google Street View House Number dataset [17]. SVHN is a real-world image dataset for developing machine learning and object recognition algorithms. The images are of small cropped digits (over 600,000) from natural scene images. The dataset is obtained from house numbers in Google Street View images.

An issue with the images in the dataset is that they are not all the same size. Depending on the process used to detect and identify the content, this can be a problem during the development of the algorithm. To mitigate this issue, during the preprocessing phase, the output of the images was resized: All images, after preprocessing, have a size of 128x128. More details on the preprocessing of the images can be found in chapter 7.

The dataset contains 10 classes, 1 for each digit. Digit '1' has label 1, '9' has label 9 and '0' has label 10. The maximum number length is considered to consist of 5 digits. Therefore, to simplify the problem, the number of digits will be considered fixed to 5. To account for the numbers of smaller length, an additional class with label 0 will be added to the dataset. It will represent the digits that need to be added in the front of the number, in order to make its length equal to 5.

7. Selected Solution

As already discussed,in order to solve the problem, Deep Neural Networks were used, and more specifically Convolutional Neural Networks [14] [15]. Convolutional

layers are followed by two fully connected layers. There are 5 features predicted, each of them corresponding to one digit of the number we are trying to identify (0 was used as a 'no digit' indicator, in numbers that have length smaller than 5 (the maximum considered).

In order to solve the problem, that data has to be preprocessed. Initially images are converted to grayscale, as they do not need to be coloured for the specific problem of digit recognition. This reduces significantly the size of our data. Pixel values are then centered around zero, and they are normalised. Images are cropped, discarding all other areas except bounding boxes. Finally, the labels are also processed, in order to conform with the design selected (length of five digits, zeros denoting the absence of a value). Approximately 10% of the original training set is used as validation test. The models are created and trained using python [18] and tensorflow [19]

The base model includes five convolution layers with stride one, followed by max pooling, and then two fully connected layers. The optimiser used is Adam. The cost function used is softmax_cross_entropy_with_logits, included in the Tensorflow library. Other configurations that will be tested include the use of dropout [16] after the convolution layer, and for some after the hidden layer, and some also a decay of the learning rate.

The first step was to pre-process the data. The images were processed, as well as the data available with them. Included in the data were the label for each image, as well as the data defining the bounding box, in which the numbers in each picture can be found.

Initially, images were scaled down, so that calculations needed to be performed only over a more limited amount of pixels. RGB images were be converted to grayscale, as colour information does not affect the identification of the digits themselves. Converting the image to grayscale will allowed working on arrays with fewer dimensions.

As already mentioned, the dimensions of the data throughout the dataset varied. Thus, the next step was to first select and crop the area defined as bounding box for each image, so we would work with the relevant information. The images were cropped, so we can keep only this part.

Next, the image data was centred around zero, and normalized. After that, the dataset was created, creating a list for each image with the labels in order, from the most significant, to the least significant digit. 90% of the 'train' data was used as a training dataset, and the remaining was used as a validation dataset. the complete dataset was stored in python dictionaries, having different key-value pairs for image data and labels, for both the training, validation, and test set. Then, they were stored altogether in a file, so that they can be reused multiple times, and the preprocessing process does not need to be repeated.

Regarding the actual neural network, there were two basic structures tested: Initially, a setup consisting of a simple regression, where there are just the inputs and the outputs was tested, in order to use as a benchmark. Then, a Convolutional Neural Network was created, which consisted of four convolution layers, followed by two fully connected layers. The number of neurons in the hidden layer was initially 1024. The network was trained using two different values for the number of training instances, with a batch of data of size 32, used as input during each training instance. The Adam optimizer was used, with initially a steady learning rate of 1e-4. The final Fully Con-

nected layers are different for each one of the digits. The effect of using dropout, or choosing different network parameters was studied, and the results from the different versions that were tested , were compared.

The cost function used was softmax_cross_entropy_with_logits, included in the Tensorflow library. The models were created and trained using python [18] and tensorflow [19]. Weight initialisation was performed using the tensorflow truncated_normal distribution with a standard deviation of 0.1. Biases were initialized to zero. Convolutions had a stride of 1, while for padding "SAME" was used, meaning the dimensions of the activation map for each layer were the same as the input.

After the last convolution layer, a max pooling operation was used, reducing the output dimension in half. The size of the processed images used was initially chosen to be 128x128 pixels.

The steps taken during the project implementation were the following:

- The convolutional network was trained for 10000 steps.
- Then, the same network was trained for 40000 steps.
- The number of neuron in the hidden layer was reduced to 500, and dropout was added after the pooling operation, with keep probability of 0.6.
- Then, the keep probability of the dropout was increased to 0.9.
- Next, a learning rate decay was introduced, with the initial learning rate set to 0.5, and a reduction inversely analogous to the number of training steps, or *epochs*.
- Following that, a second dropout function was included after the hidden layer.

After that, some of the configurations were tested once again, this time using images of size 64x64. the configurations used were the following:

- Double dropout with a learning decay, with initial learning rate 0.01, and keep probability 0.9, trained for 40000 steps.
- Double dropout with a learning decay, with initial learning rate 0.5, and keep probability 0.9, trained for 40000 steps.
- Double dropout without a learning decay, with learning rate of 0.001, trained for 30000 steps.

Finally, the number of the hidden layers was reduced to 250, and a configuration with double dropout without learning decay, with learning rate of 0.01, was tested, after being trained for 40000 steps. The various configurations are summarised in table 1, where probability refers to the probability of keeping an input.

Table 1: Network configurations used.

Configuration No	Optimizer	Number of Hidden Nodes	Dropout	Learning rate	Training Steps	Image Dimensions
1	GradientDescent	500	No	0.5	40,000	128x128
2	AdamOptimizer	1024	No	1e-4	10,000	128x128
3	AdamOptimizer	1024	No	1e-4	40,000	128x128
4	AdamOptimizer	500	Once, Prob.:0.6	1e-4	40,000	128x128
5	AdamOptimizer	500	Once, Prob.:0.9	1e-4	40,000	128x128
6	AdamOptimizer	500	Twice, Prob.:0.9	1e-4	40,000	128x128
7	AdamOptimizer	500	Once, Prob.:0.9	1e-2/decay	40,000	128x128
8	AdamOptimizer	500	Twice, Prob.:0.9	1e-2/decay	40,000	64x64
9	AdamOptimizer	500	Twice, Prob.:0.9	0.5/decay	40,000	64x64
10	AdamOptimizer	500	Twice, Prob.:0.9	1e-2	30,000	64x64
11	AdamOptimizer	250	Twice, Prob.:0.9	1e-2	40,000	64x64

8. Benchmark Model

The Benchmark for the application was the naive model, where we only have the inputs, and regression outputs. Efficiency criterion was the detection accuracy. The accuracy is defined as described in chapter 9. The various configurations used were examined, and compared to our naive model (in this case, a simple regression model). In the ideal case, the goal would be to achieve an accuracy comparable to - or not dramatically worse than the one achieved by the configuration the authors of [1] used (96.03% for sequence transcription), or that of humans (98%). The accuracy they achieved is provided, so a comparison is also easy to make.

9. Evaluation Metrics

As already mentioned, the evaluation metric will be the detection accuracy. For each of the numbers, the accuracy will be evaluated as shown in Algorithm 1:

Algorithm 1: Estimation of accuracy %

for *all digits i* **do**
  prediction: $p[i]$=*argmax*(*predictions*);
  label: $l[i]$=*argmax*(*labels*);
end
for *every image* **do**
  successful digits = sum(p[i] == l[i])
end
$accuracy(\%) = 100 * count(successfuldigits == 5)/numberofimages$

A successful detection is only that which includes a correct identification of all five digits (including zero for the lack of one). The percentage of all images, for which all five digits were correctly identified gives us the accuracy.

10. Experimental Results

As can be seen in figure 7, The accuracy of the naive model that was used as baseline (the simple model using regression) had a prediction accuracy that was only 0.7%. The model was underfitting the data, as the accuracy measured on the training set was also very low, at most around 10%. Using the model with the convolutions and the fully connected layers previously described, resulted in a significant increase of the accuracy up to 10,000 epochs, and a smaller one when training continued up to 40,000 epochs, reaching 37%. However, the model was clearly overfitting the data, since the training set accuracy reached levels of 100%. Configuration No 4, introduced dropout, with a keep probability of 0.6. In addition, the number of hidden nodes were reduced to 500. This produced a model that had a very low accuracy on the test set (2.9%) and a bit higher on the training set, but not significantly.

The highest accuracy observed was when the keep probability of the dropout was increased to 0.9, and then a second dropout function was introduced (configurations 5,6). The model with double dropout had an accuracy of 47.7% on the test set, while the one with the single dropout slightly less. Both of them though were still overfitting the data, since their training accuracy reached 100%. Introducing a learning rate decay reduced significantly accuraxy to 2.4%.

Reducing the dimensions of the input image to 64x64 did not yield better results: the contrary. The accuracy of models 8-11 did not exceed 13.7% (Configuration No 11, the one without a learning rate decay). In cases 7-10 it was around 2.5%, and once again underfitting the data.

The situation seemed to improve a bit when the number of hidden neurons was reduced to 250, however the results were still far from those witnessed with configurations 5 and 6.

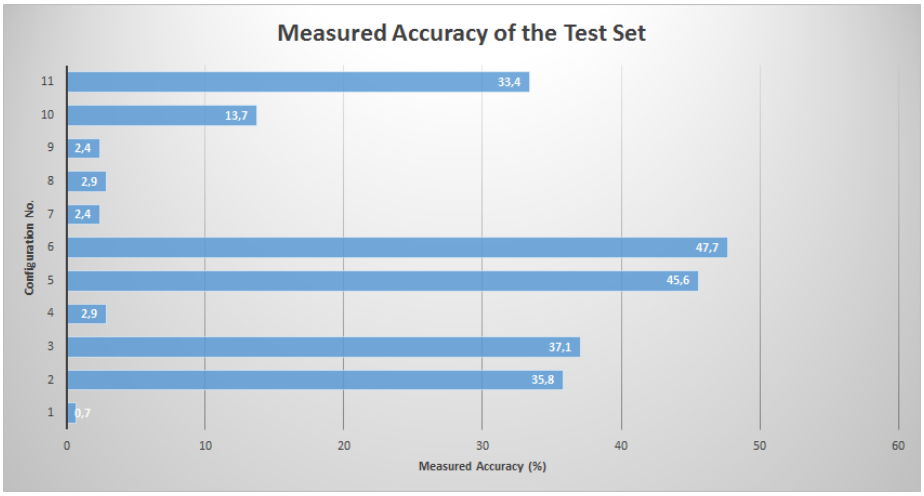

Fig. 7: Test Set Accuracy

11. Discussion and Future Work

As can be deducted from the results presented above, the solution selected, with the for convolution layers and the two fully connected layers, can provide results that may not be comparable to the state of the art, however are significantly superior to the naive approach used as benchmark. This of course, provided the right hyperparameters are selected, like the number of hidden nodes, dropout keep probability, etc.

The state of the art solutions that provide accuracy comparable to that of the human, have mostly many more layers than the models tested. In the environment where the tests took place, adding more layers would not be efficient, since the processing power of the system used is limited. Even the currently tested architectures took at least a couple of days to train each, depending on the number of steps, as well as image size, etc. Therefore, even though testing architectures with more layers would be interesting in order to compare results with the ones currently used, the required time for performing all tests would range from weeks to months (using the hardware used).

One think that would definitely help would be to use data augmentation techniques, in order to generate additional data. This is expected to be extremely important, especially in the case of the models that are overfitting the data. Since the models that displayed the best results on the test data were actually overfitting, the use of additional data is expected to improve the accuracy of the model further. Such techniques have been explored, and it is planned as a next step to use functions that are built in scikit-learn and tensorflow, in order to create further data, and examine if the accuracy of the test set predictions will be improved, and to what extend.

A second direction to be pursued in the future is to perform the same tests, using the same models, only this time using the original images. Meaning, instead of cropping the SVHN images and keeping only the parts contained in the provided bounding boxes, to use the complete provided images. A final step would be to use the original images, and simultaneously try to output the bounding boxes, besides the digits in the images themselves.

However, the most useful probably adjustment would be just detecting digits using the original images, and adjust the implementation, so that real time video can also be used. This can then find many applications, for example be used in systems like autonomous vehicles, in order to identify road signs, or detect in real time licence plates of cars, etc.

References

[1] I. J. Goodfellow, Y. Bulatov, J. Ibarz, S. Arnoud, V. D. Shet, Multi-digit number recognition from street view imagery using deep convolutional neural networks, CoRR abs/1312.6082.
URL http://arxiv.org/abs/1312.6082

[2] A. K. Chauhan, P. Krishan, Moving object tracking using gaussian mixture model and optical flow, International Journal of Advanced Research in Computer Science and Software Engineering 3 (4).

[3] W.-T. Lee, H.-T. Chen, Histogram-based interest point detectors, in: Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on, IEEE, 2009, pp. 1590–1596.

[4] R. K. Rout, A survey on object detection and tracking algorithms, Ph.D. thesis (2013).

[5] S. C. Sen-Ching, C. Kamath, Robust techniques for background subtraction in urban traffic video, in: Electronic Imaging 2004, International Society for Optics and Photonics, 2004, pp. 881–892.

[6] K. Srinivasan, K. Porkumaran, G. Sainarayanan, Improved background subtraction techniques for security in video applications, in: 2009 3rd International Conference on Anti-counterfeiting, Security, and Identification in Communication, IEEE, 2009, pp. 114–117.

[7] C. Kim, J.-N. Hwang, Fast and automatic video object segmentation and tracking for content-based applications, IEEE transactions on circuits and systems for video technology 12 (2) (2002) 122–129.

[8] C. Zhan, X. Duan, S. Xu, Z. Song, M. Luo, An improved moving object detection algorithm based on frame difference and edge detection, in: Image and Graphics, 2007. ICIG 2007. Fourth International Conference on, IEEE, 2007, pp. 519–523.

[9] R. S. Rakibe, B. D. Patil, Background subtraction algorithm based human motion detection, International Journal of scientific and research publications 3 (5).

[10] K. A. Joshi, D. G. Thakore, A survey on moving object detection and tracking in video surveillance system, International Journal of Soft Computing and Engineering 2 (3) (2012) 44–48.

[11] N. Paragios, R. Deriche, Geodesic active contours and level sets for the detection and tracking of moving objects, IEEE Transactions on pattern analysis and machine intelligence 22 (3) (2000) 266–280.

[12] S. C. Zhu, A. Yuille, Region competition: Unifying snakes, region growing, and bayes/mdl for multiband image segmentation, IEEE transactions on pattern analysis and machine intelligence 18 (9) (1996) 884–900.

[13] T. Bushra, M. H. Khan, A survey on object detection and classification methods from video stream.

[14] K. Fukushima, Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position, Biological Cybernetics 36 (4) (1980) 193–202. doi:10.1007/BF00344251.
URL http://dx.doi.org/10.1007/BF00344251

[15] Y. Lecun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based learning applied to document recognition, in: Proceedings of the IEEE, 1998, pp. 2278–2324.

[16] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: A simple way to prevent neural networks from overfitting, J. Mach. Learn. Res. 15 (1) (2014) 1929–1958.
URL http://dl.acm.org/citation.cfm?id=2627435.2670313

[17] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, A. Y. Ng, Reading digits in natural images with unsupervised feature learning, in: NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011, 2011.
URL http://ufldl.stanford.edu/housenumbers/nips2011_housenumbers.pdf

[18] G. van Rossum, Python Reference Manual, 2nd Edition (Mar. 2006).
URL http://docs.python.org/ref/ref.html

[19] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Man, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Vigas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, X. Zheng, Tensorflow: Large-scale machine learning on heterogeneous distributed systems (2015).
URL http://download.tensorflow.org/paper/whitepaper2015.pdf