Multi-Digit detection in Natural Scene using Convolutional Neural Networks

Yogesh Luthra (GTID: yluthra3; Email: yogeshluthra@gatech.edu)

Link to presentation video (Click)

1. Introduction.

This paper describes various techniques and related results to detect, localize and classify a sequence of digits in natural scene, which is font, translation, rotation, scale and lighting invariant. Techniques used to localize and form most-likely sequence, using raw natural image as input to model, are described. This was mainly achieved by using a Deep Convolutional Neural Network (DCNN), along with few simple post-processing algorithms to suppress non-maximum likely detections and localize search to maximum-likely areas within image in a recursive manner.

As described in Goodfellow et al [1], this task was achieved by sharing encoded output from Deep Convolutional Layers (DCL) by 6 parallel Fully Connected Layered (FCL) output networks. Those 6 outputs were (1) N: Number of digits in sequence, and (2)-(6) y_i where $i \in \{1,2,3,4,5\}$. While adopting similar approach, 2 types of architectures were tried where only structure of DCL was changed, while keeping FCL same as in [1]. First architecture (henceforth called Arch1) tried is similar to [1] with 8 DCL followed by 6 parallel FCLs which shared output from DCL. In this architecture a 2x2 max-pool layer was applied after every conv layer, but unlike [1] alternating strides of 2x2 on conv layers were not used due to smaller input image sizes used (32x32x3). Second DCL architecture (henceforth called Arch2) borrows heavily from Simonyan et al in [2], where 13 DCL were used, famously known as VGG16, while keeping output structure similar to Arch1. As further described in [2], max-pooling layers were applied in blocks, instead of after each conv layer as done in [1]. 2 types of trainings were applied on Arch2. (a) Arch2 trained from scratch with weights randomly initialized (henceforth called Arch2-FullyTrained), (b) Arch2 with DCL weights re-used from a pre-trained network trained on ImageNet [3], while only fine tuning by training only 3 final FCLs (henceforth called Arch2-PreTrained). Training data was sourced from SVHN dataset [7] (only dataset 1 was used).

2. Description of methods and algorithms.

Dataset was augmented with (a) 50% as much non-digit crops from [8], as there were crops with digits, (b) random rotations, (c) random image area resizings between 1x-2.5x, (d) random translations in <x,y> directions within 0-10 pixel lengths, (e) Additional Gaussian noise added in 25% randomly selected images. This data augmentation was found crucial in making model robust to aforementioned invariances. Final dataset size was 200k images out of which 15% (30k) randomly selected images were kept as validation set and another 15% as test set.

Each of the DCL and FCL layers shown in Fig 1, were initialized by He uniform distribution [5], followed by a Batch Normalization layer [4], activated by PReLU [5] and then followed by Dropout [6] (0.25 for DCL, 0.5 for FCL was used as dropout probability). Primary reason for using Batch Normalization and Dropout was to make network generalize better as vanilla VGG16 net severely overfits, such that it reaches high training accuracy (>99%) but relatively low training accuracy (<86%).

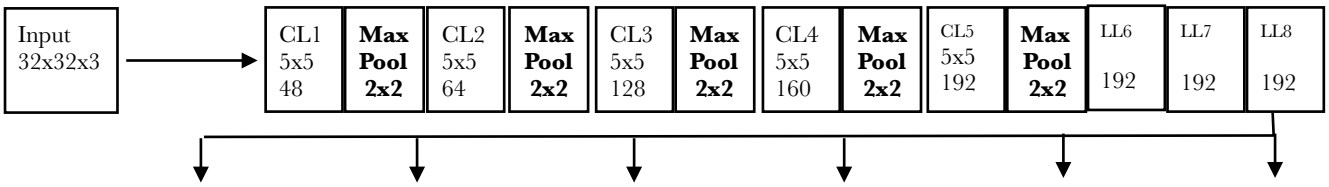
N in Fig 1 represents number of digits with 0-5 classes being N digits and 6 meaning more than 5 digits. y1-y5 each has 11 classes with 0-9 classes representing digits and 10th class meaning non-digit.

Both of these architectures assume individuals digits are independent. Each of the output is a Multinomially distributed random variable. However sequential constraints are imposed by the training distribution, such that (1) if ground truth N=0, then all ground truth values at digit locations are non-digits, (2) if y[i] is non-digit then all y[j] for j>i are also non-digits. Since the network minimizes loss, it learns this constrained ground truth distribution.

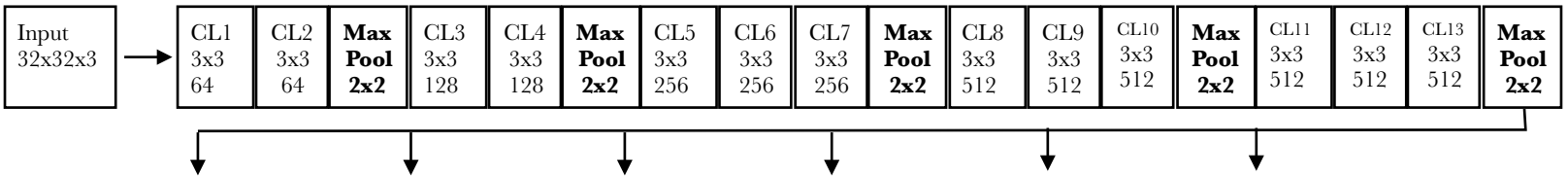
Overall the network tries to maximize following probability distribution, which translates into minimizing cross-entropy loss.

$f(X) = p(N \bigcup y_i \ for \ i \in \{1,2,3,4,5\} \mid X) = p(N \mid X) * \Pi_{i=1}^{5} p(y_i \mid X)$, where X: input image.

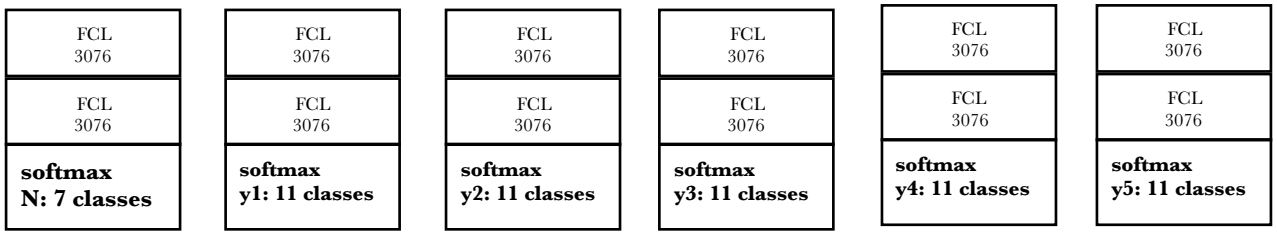
Fig 1 shows high level picture of the architecture used.



(a) Arch1 [1]. LL6-LL8 are locally connected layers.



(b) Arch2 [2]



(c) Output structure as in [1]

Pipeline: Since test images were full scenes and thus much bigger than training images, which were crops from full scenes, sliding window approach was used, along with pyramid approach (atmost 3 pyramids were constructed). Pyramids particularly helped to detected digits in various scales

At first layer of image pyramid: (a) 32x32 crops were constructed from full scene with stride of (4,4), (b) passed to the DCNN, and (c) all possible detections collected * (d) Then Single Link Clustering was performed on all detections, where a cluster is defined as a group of crops in which each of its member is atmost M stride lengths away from at least another member (M=2 was found useful). (e) Then from atmost top 3 densest clusters, 3 different bounding boxes were constructed such that each of corresponding cluster members are enclosed within those bounding boxes.

At second pyramid layer: (a) 48x48 crops at strides (4,4) were constructed from within bounding boxes found from previous iteration. Then steps (b)-(e) were repeated as previously described.

At third pyramid layer: (b) 72x72 crops at strides (4,4) were constructed from within bounding boxes found from previous iteration. Then steps (b)-(d) were repeated as previously described.

As can be guessed, this pyramid method, while being able to detect digits at various scales in the scene, also runs much faster than a brute force search over entire image at each layer, as 2nd and 3rd pyramid layers are basically doing beam search in most likely areas where digits could occur.

All clusterings found from 3 layers of pyramid were concatenated and most dense cluster was extracted from overall detections. Within that most dense cluster, largest sequence length (max N) was considered and each digit location was filled with most occurring digit in that location among cluster members which also had sequence length=max N. This procedure obviously places heavy trust in output N. The reason, as will be shown empirically, that this particular output was found most accurate among all outputs with validation accuracy > 97%.

*: Detections were thresholded such for crops with N>0 found with probability>98% and each digit at index <=N found with probability>40%, were considering at positive hits. Any other detections classified positive by CNN were ignored as part of non-maximum suppression procedure.

Training methods: Stochastic Gradient Descent, along with Nestrov momentum [9] was used for training. Several experiments were performed on optimizers and this turned out to be most robust. Batch size=256 images (each image shape=32x32x3) was chosen as this training was done on Nvidia 1070 Ti, and this size seemed to fit best on VRAM of GPU. Less or more than that caused some degradation in training speed. Learning rate was set at an initial value of 1e-2 and whenever loss stopped decreasing for more than 7 epochs, it was decreased by 1/2. Reducing learning rate usually helped possibly because whenever model got stuck in high eigenvalue regions of Hessian of loss function (meaning steep curvature of loss function), going slower is a better strategy to converge towards local minima. Model was check pointed at last known epoch number at which lowest validation loss was registered. When validation loss stopped decreasing below 0.001 from previous best for 20 epochs, training was Early Stopped.

3. Performance metrics of different approaches:

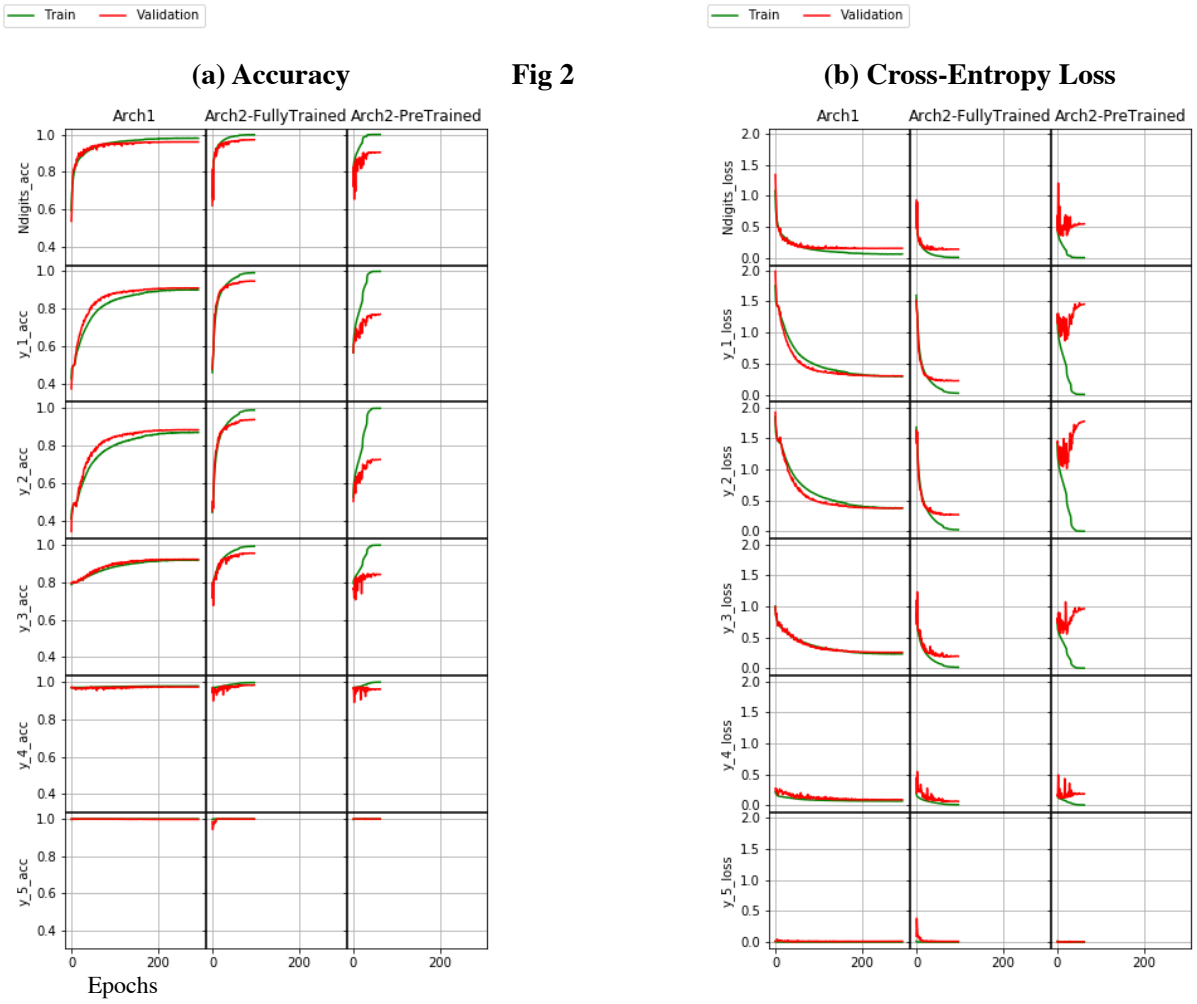


Fig 2

Shown above are performance results of Arch1 [1] and Arch2 (VGG16) [2]. Fig 2 (a),(b) clearly show that VGG16 PreTrained performed worst of all. It clearly overfits the training data as training set accuracy very quickly reaches 100%, while validation set accuracy is stuck at much lower value. In this project, for VGG16 PreTrained exercise, all conv layers, except last 3 layers, were frozen. So last 3 conv layers and FCLs were fine-tuned. This was done because conv layers are shared by all output layers and in this particular architecture, the loss function adds up at shared layers. It was seen that if none of the shared layers were fine-tuned, the loss was even worse than what is shown above.

Best model among all turned out to be VGG16 when trained from scratch. But Batch Normalization and Drop out layers had to be introduced in all conv layers, as otherwise similar overfitting problem was observed. The reason Arch2 learned much better than Arch1, is possible because of much larger capacity owing to much deeper convolutional layers.

Accuracy numbers:

Accuracies	Ndigits	y1	y2	y3	y4	y5
Train	99.58	98.4	98.2	99.7	99.98	99.99
Val	97.12	94.44	94.2	95.1	99.3	99.52
Test	96.79	94.1	94.47	94.98	98.93	99.67

4. Final Results:

Note: For this section, only the best model, that is VGG16 trained from scratch was used.

Video demonstrating that proposed pipeline works well under varying scale, rotation and translation conditions can be found at https://youtu.be/9MyQz5YsBOc

Images: The proposed pipeline, works very well on unseen 1, 2 and 3 digit images. Some examples are shown on the right (Fig 3). However for digit sequences more than 3, it does not work as well, as shown in Fig 4 below.

Fig 3. Where classifier works well

Fig 4. Where classifier doesn't work well

Primary reason for this worse behavior on 4 digit sequence is due to highly skewed dataset distribution (Fig 5), which has lots of 2 and 3 digit images, but relatively very few 4 digit images and almost none 5 digit images. So the classifier was trained with little penalty for misclassifying images with higher than 3 digits.

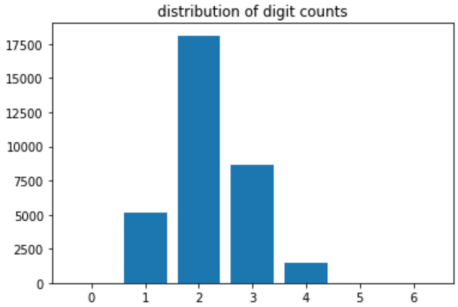


Fig 5. Distribution of digit counts in dataset 1

Conclusion:

According to [10], current state of the art results on SVHN dataset [7] are test accuracy = 99.31% but that appears to be on individual digit classification. Error on sequence is report couple of percent points higher in most papers. Most of reported competitive results also appear to be using ensemble of classifiers instead of a single model as done in this project. Nonetheless, state-of-art results are pretty good.

One simple way to improve accuracy of the proposed pipeline is to use ensemble of simple classifiers which only detect number of digits in a given cropped images. These will generate possible digit hits from those cropped images as explained above and help in forming better clusters. Then beam search could be done on those found clusters using a single or ensemble of heavier classifiers.

Also to makeup for skew in dataset, as discussed in section 4, either more 4 and 5 digit images need to be included or much higher weight needs to be assigned to images with 4 or 5 digits to increase penalty of misclassifying those images. Latter method seems cheaper and will be included in future research.

References:

[1] "Multi-Digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks" , Goodfellow et al. 2014

[2] "Very Deep Convolutional Networks for Large-Scale Image Recognition", Karen Simonyan, Andrew Zisserman 2015.

[3] Pre-trained VGG16 on ImageNet. https://keras.io/applications/#vgg16

[4] Batch Normalization, Sergey et al 2015

[5] Delving Deep into Rectifiers, He et al 2015

[6] Dropout, Srivastava et al.

[7] Street View House Numbers (SVHN) Dataset

[8] Google Street View Data Set

[9] On the importance of initialization and momentum in deep learning, Sutskevar 2013

[10] http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html#5356484e