Classification and Detection with Convolutional Neural Networks

Binu Enchakalody
binuen@gatech.edu

Abstract:

Automatic detection of digits and numbers is a task where recent work in neural networks and computer vision has shown a lot of promise. The goal of this project is to replicate earlier results [2][1] using multiple Convolutional Neural Network (CNN) models to predict a sequence of numbers. The Street View House Number (SVHN) data set which has ~250,000 labelled images were used in this study. About 150,000 samples from this dataset were used to train three different CNN models: designed architecture, VGG-16, Pre-Trained VGG-16 to predict a sequence of up to four digits. The reported train and test sequence accuracies are 91.53% Train and 85.4% Test for own architecture, 96.6% Train and 91.24% Test for Pre-Trained VGG-16, 90.8% and 82.1% train and test accuracies for the VGG-16 model. The bench mark is a reported 96% test accuracy on the entire data set [1]. The pre-trained VGG-16 model which uses the 'ImageNet' weights showed the best accuracy. This model is tested on real-life detection scenarios using a 10-level image pyramid with sliding window detection to tackle digit sequence variance with size and scale.

Methods:

The SVHN dataset has two sets of data: fully cropped individual (MNIST) digits and sequence of one to six digits. The latter set was chosen for this project. This data has around ~33 K and 13 K labelled train and test images along with an extra set of 200 K images. The provided digit bounding box data was used to crop a 30 % padded, minimum bounding box around the image [1]. Each image was then resized and saved as 48 x 48 pixel BGR and grayscale formats. Almost 60 K images were cropped from the region outside the bounding box as true negative data. There was only one six digit and <25 five digit sequence images in the set. I chose to remove these from the set to avoid a class imbalance. The model was designed to predict a sequence of maximum of 4 digits along with the number of digits and a 'digit classifier'. The original labels have '0' labelled as '10'. These were relabeled and each label was converted to a string of 6

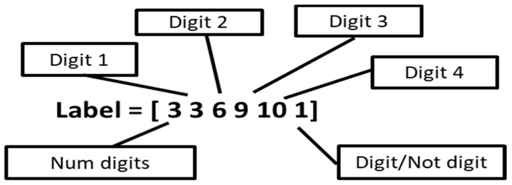


Image normalization:

Mean subtraction is done for all training and test samples [1][2]. This is followed by a feature normalization [4] using the mean and standard deviation (sd). The mean for each feature is subtracted from the training set. This is then divided by the sd. The same mean and sd are used on all the test images. A train, test and validation set were used here. Approximately 150,000 images were used for the training set which includes 30,000 true negatives. The test set contains ~13,000 samples and a validation set is randomly chosen from 10% of the training set.

Typical mini-batch sizes are 16, 32, 64, 128, etc. The mini-batch size used here is 64(32 and 128 were tried). Each pass allows the model to take one step in the gradient descent or one update. The descent may appear noisy but in runs much faster especially when using a big data set. Each batch was shuffled to ensure the samples were split randomly.

For the optimizer, Adaptive Momentum Estimation (Adam) is used in all 3 models. Adam is a very effective optimizer which is a combination of both momentum (exponential weighted average) and Root Mean Square (RMS) prop. Momentum can reduce the oscillations before convergence by accounting for the gradients from the previous steps. The hyper parameters for learning rate (α), momentum (β1) and RMS(β2) are recommended to be used at their default values of 0.001, 0.9 and 0.999. A decreasing learning rate when value loss plateaued was experimented with using SGD optimizer. Adam adapted better without any α tuning for the same.

The weights for each layer were randomly initialized [3][4] by Keras to ensure the weights are not uniform. I experimented with using l1 and l2 norm regularizers to avoid the effects of over-fitting in the first few models. This was soon replaced by Drop outs after every other layer. This method randomly drops hidden units and measures their performance. Dropouts make the neurons less sensitive to the activation of specific neurons because that other neuron may shut down any time [3]

Early stopping is used if the model loss plateaus for more than 5 epochs. This was experimented with validation loss as well. Each layer is batch normalization which like normalizing the input, helps the convergence [3][4]. This is applied before each max. pooling layer.

Training CNN Models:

For creating the deep learning models, Keras was used with a TensorFlow(nightly-gpu v.1.9) backend on Python 3.5. Keras API class was preferred over sequential to assign the SoftMax(SM) layers. Multiple models with smaller number of layers and filter sizes were experimented until this was finalized. The activation function used was ReLu for all layers expect the final dense layer. Six parallel SM outputs are connected to the final layer. The first SM represents the number of digits, followed by the four digits. The last SM represents a binary digit or not output. This is to quickly identify true negatives. Similar methods [2] were used for 6 digits and number of digits. Figure shows the architecture of the designed model. Sparse categorical cross-entropy loss was used to avoid hot-encoding the output. The model loss, model validation loss and accuracy for each SM output were measured per epoch. Since there was no model accuracy, I created a custom 'sequence accuracy' by combining the predicted of the four digit SMs to the ground truth [2]. This would have been easier to implement as the loss in the model by using TensorFlow without Keras.

The VGG-16 and pre-trained VGG-16 networks were easier to set up using the Keras libraries. The pre-trained network used the '*ImageNet*' weights which were trained on over 1000 classes. Dense levels were added to both models.

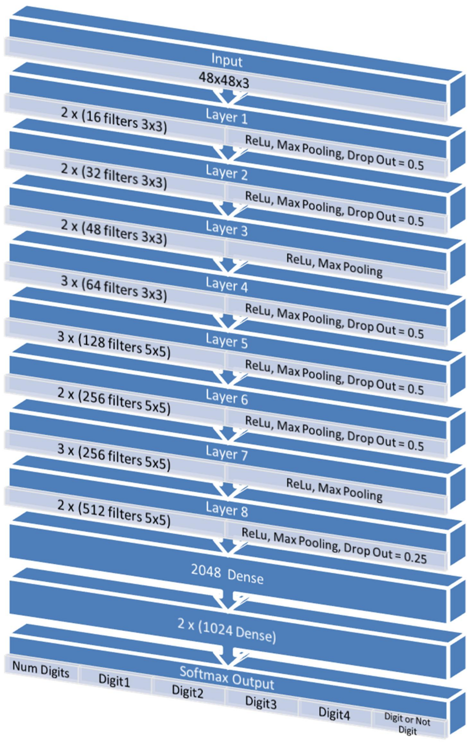


Figure 1 Designed Architecture for SVHN

Detection algorithm: Scaling, localization and sliding windows:

Using sliding windows through the entire image at different image levels can be slow. To ease this process, localization is performed using image gradients and magnitude to create a binary mask. This mask ignores parts of the image without any significant gradient changes and thresholds objects that are below and above a certain size. The image and binary image is then scaled to 10 levels from the original size where each level is 80% the size of its predecessor.

The sliding 48x 48 window runs through the entire image at steps of 5(smaller at lower levels) avoiding windows without any significant gradients. For each candidate, the classifier eliminates window candidates where there is a high likelihood of "num digits" = 0(not a number) and "not a digit" = 1. Another check is done for the overall confidence of being a digit < 70%. For candidates that pass these classifiers, the local bounding box and predictions with likelihood is saved.

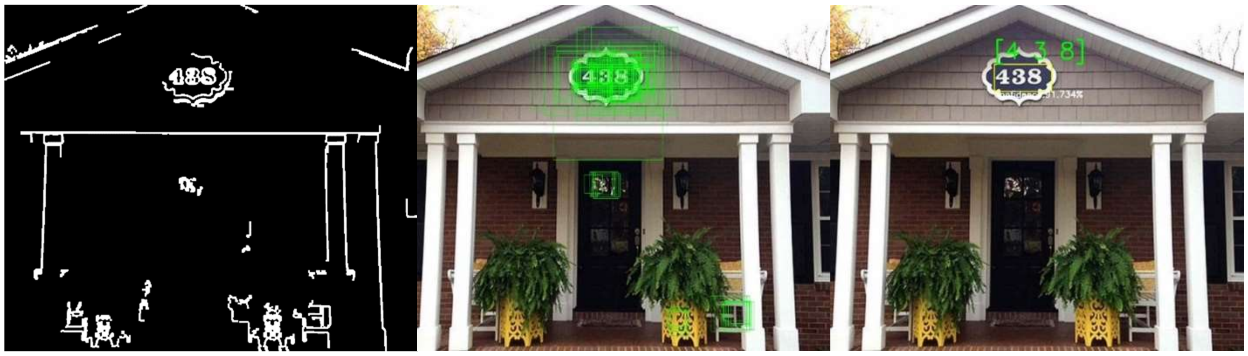
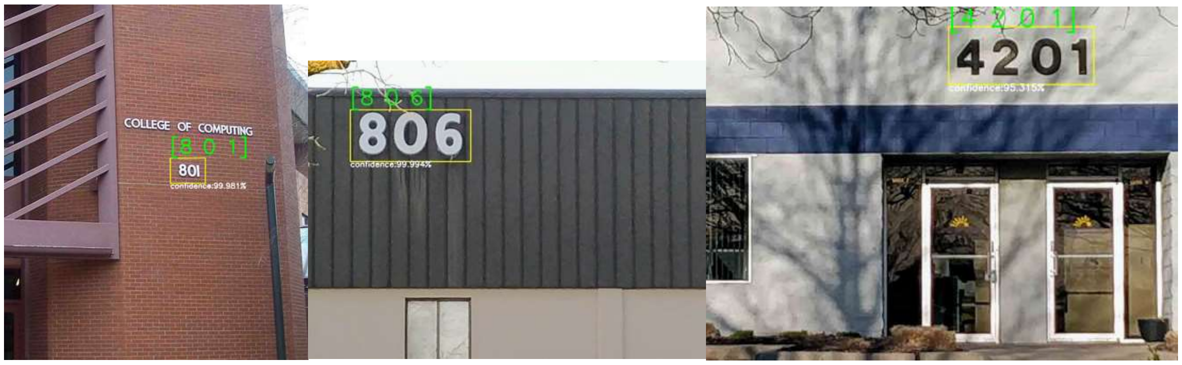


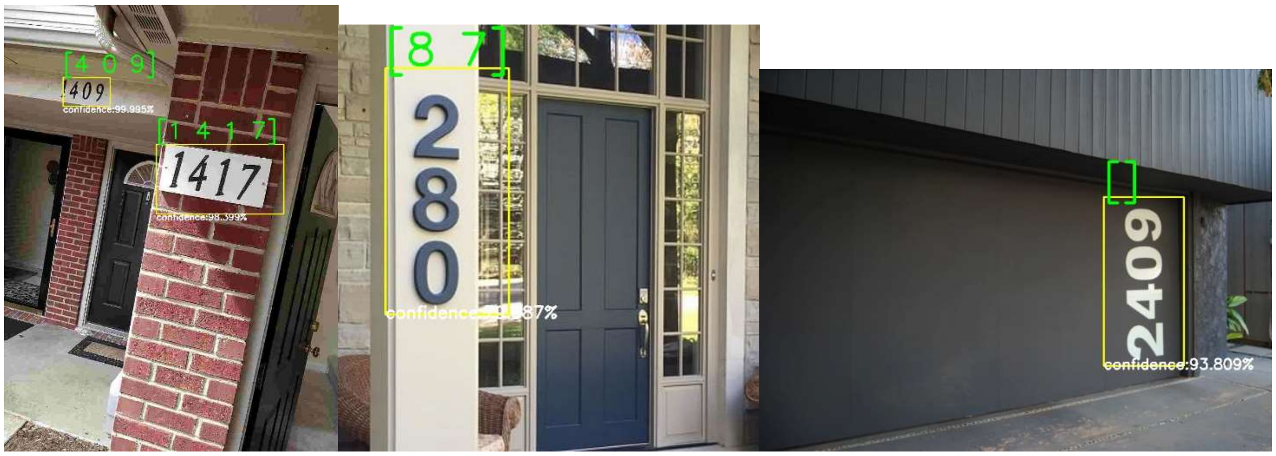
Figure 2 Localization followed by sliding windows creates higher probability candidates with multiple hits

A probability mask is created using the saved bounding boxes and their predictions. Each box is converted to a mask with its score being the overall prediction value. Locations with multiple hits will show a significantly higher value than others. This mask is used as non-maxima suppression to create bounding boxes from these locations. Resized patches of theses boxes which are the final candidates are passed again to the model. If the overall confidence is still over the threshold of 80%, a bounding box and digit are drawn on the final image

Results

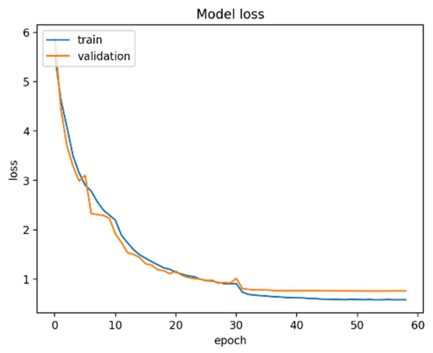
Examples where detection and classification works and fails. Images taken under varying lighting, rotation, noise, shadow, scale and font types. The model does not detect vertical and 90-degree rotated sequences. The model was never trained for these sequences.

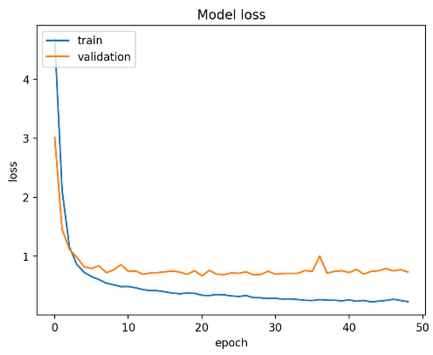


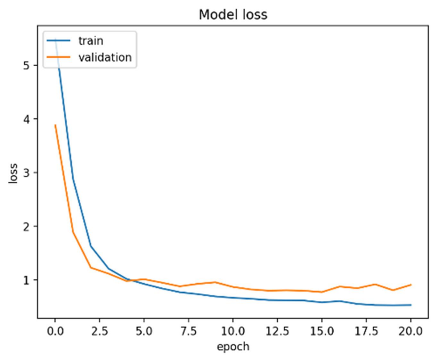


Performance Analysis

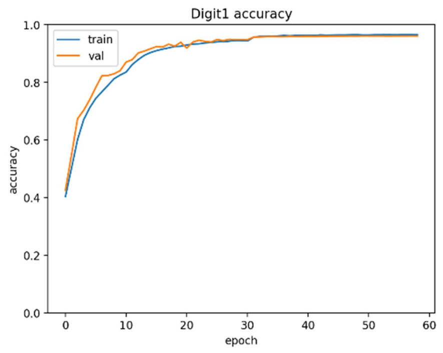
Below are the model loss and individual accuracy plots(digits 1 and 2) vs. number of epochs for all three models.

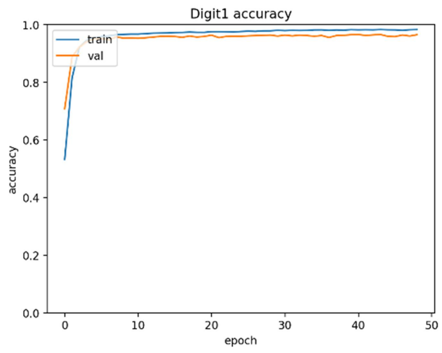


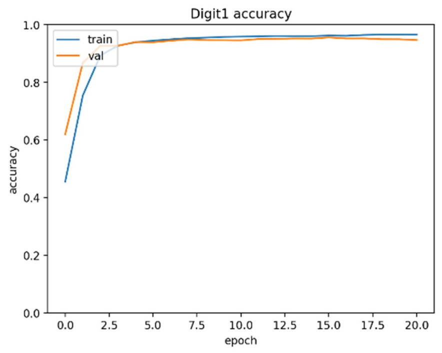


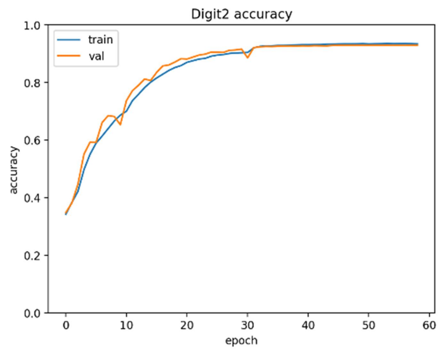


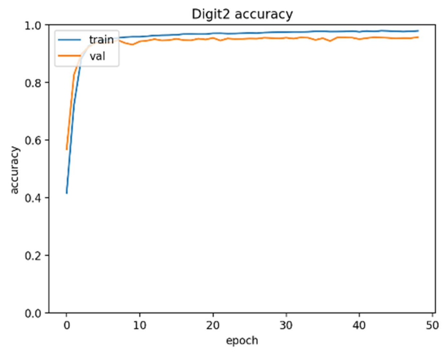
Designed Architecture *VGG16- Pre Trained* *VGG16 from scratch*

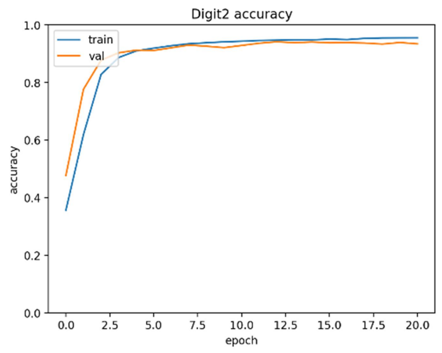












Comparing Models

Model	Train Loss	Test Loss	Val Loss	Train Seq. Accuracy	Val Seq. Accuracy	Test Seq. Accuracy
VGG Pre Train	0.17	0.76	0.88	96.62	87.91	91.24
VGG16	0.49	1.10	0.9	90.8	86.87	82.16
Designed	0.41	0.87	0.75	91.53	87.99	85.40

Discussion

Training the models was not straight forward where slight changes in normalization, filters, activation, and pooling showed varying differences in convergence. Hyper parameter tuning seemed like the main obstacle. I had to try multiple methods of image normalization, loss types and optimizers before using the current model. To save load and run time. models using color vs. grayscale, were tried and training color showed superior results(accuracy>5%). For the detection, the sliding window with scaling turned out be a very slow process even after a decent amount of image localization. Tackling blobs of potential candidates using segmentation may give faster results. Although I had invested a lot of time into tuning my own architecture, I was surprised to see the pre-trained model outperforming the other two by a slightly superior training and validation accuracy.

The CNN models using individual SoftMax models for each digit of the sequence were proposed by (Goodfellow et. al, 2013) [1]. They had reported a test sequence accuracy of ~96%. This method and results were set as the benchmark for this project. Work by (Netzer et. al, 2011)[2] had done a supervised approach using the same data set. I am happy to see results of 91% test sequence accuracy for a pre-trained model and 85% for my designed model.

Future work

Approximately 100K of the training samples were not used to train this set. The model is variant to rotation over a certain angle and does not read vertical sequences. Using a more representative set of true negatives (samples of alphabets and words) along with tuning the detection algorithm will help cutting down false positive detections and improve this model.

Reference Links

CNN video demonstration**: https://youtu.be/ZDRt21VEOpA**

Presentation (more details + charts): **https://youtu.be/CGf0heoR5nE**

References

1. Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks Ian J. Goodfellow, Yaroslav Bulatov, Julian Ibarz, Sacha A 2011
2. Reading Digits in Natural Images with Unsupervised Feature Learning Yuval Netzer1 , Tao Wang2 , Adam Coates2 , Alessandro Bissacco1 , Bo Wu1 , Andrew Y. Ng1,2
3. Andrej Karapathy Lectures https://cs231n.github.io/convolutional-networks/
4. Andrew NG Lectures https://www.coursera.org/specializations/deep-learning
5. Frome, G. Cheung, A. Abdulkader, M. Zennaro, B. Wu, A. Bissacco, H. Adam, H. Neven, and L. Vincent. Large-scale privacy protection in google street view. In ICCV, pages 2373– 2380. IEEE, 2009.
6. Keras Blogs https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html
7. Malisiewicz NMS implementation http://www.computervisionblog.com/2011/08/blazing-fast-nmsm-from-exemplar-svm.html