Machine Learning Engineer Nanodegree

Deep Learning Capstone Project Report

Devinder Saini
January 6, 2017

Definition

Project Overview

Optical Character Recognition (OCR) is one of the most important motivations of computer vision and machine learning. It involves the identification of characters or sequences of characters from images. It has significant applications like check processing, document transcription, package routing in couriers and post offices, automatic navigation, etc.

Several datasets have been proposed for benchmarking OCR techniques in different contexts, like handwriting recognition (MNIST Dataset), printed and glyph characters (notMNIST Dataset) and identification in real world images (Street View House Numbers Dataset)[1]. While classifying a single character is an easier challenge, identifying a full sequence of characters is a more challenging problem. Street View House Numbers is one such dataset of real world images of house numbers taken from Google Street View images. It has two datasets - character level images and full sequence images in highly varying backgrounds, colors, fonts, relative sizes, resolutions etc.

The Street View House Numbers dataset has been used to develop models to automatically geotag house numbers using street view imagery. A lot of research has been done using the dataset and many algorithms have been proposed.[1]–[3]

Problem Statement

The goal of this project is to predict a full sequence of digits present in an image. The digits must also be in the same left to right reading order. We want to predict every digit correctly, and there will be no partial credit for some correct digits, because, a single digit error can result in a completely different house number. We also want our architecture to be computationally efficient so that it can give near real time results on handheld devices.

For solving this problem, we will train a multi-input multi-output deep learning network. We will divide the problem into two parts – identifying the number of digits in an image, and then identifying each digit in left to right reading order. A counter will count the number of images and generate a range of indices of digits to be identified. The indices

and an intermediate image convolution will be input to a label detector, which will then output the digit at each index.

Metrics

We use full sequence prediction accuracy as the primary performance metric. There's no partial credit for correctly predicting some digits. Since full sequence predictions can either be correct or wrong, accuracy of making correct predictions is an appropriate metric.

Analysis

Data Exploration

Street View House Numbers Dataset has two formats, full sequence house numbers and cropped individual digits. In our setup, we will only use the full sequence format, since we'll train our deep learning network to read directly from full sequence images.

Following are the number of samples in each dataset:

Training samples	33402
Testing samples	13068
Flattened training samples	73257

The datasets also contain bounding box information for each digit in every image. However, we will not be using the bounding box data since our algorithm doesn't depend on it. Also, the dataset represents digit 0 with a label value 10. We will change this and represent a digit 0 with a 0. Following are some samples from training dataset.

Figure 1 : Samples of training images

Exploratory Visualization

We can visualize the distribution of the number of digits in training and test datasets as shown below.

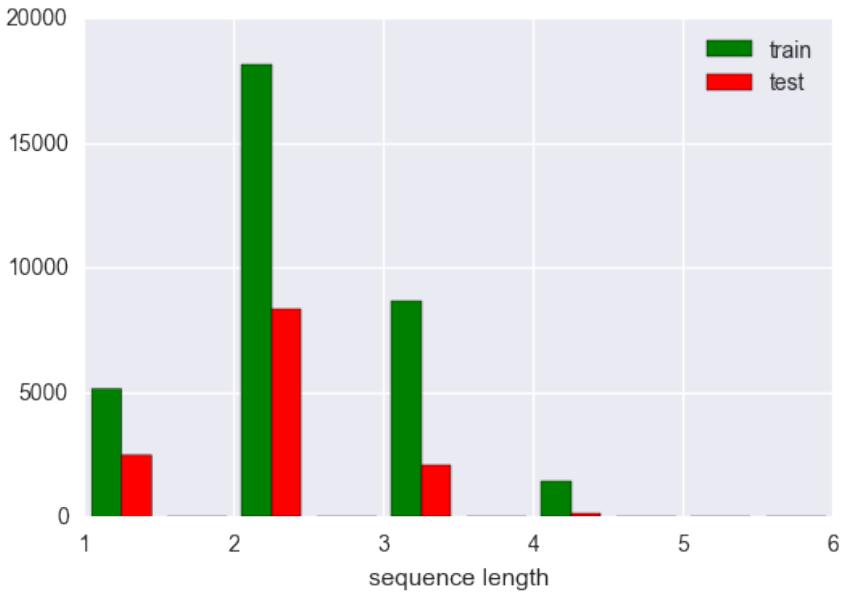

Figure 2 : Distribution of sequence length

Here we can see that the distribution of number of digits is quite asymmetric. Two-digit house numbers are much more common that the rest. Also, house numbers longer than four digits are negligible.

We can also visualize the distribution of digits that make up house numbers as shown below.

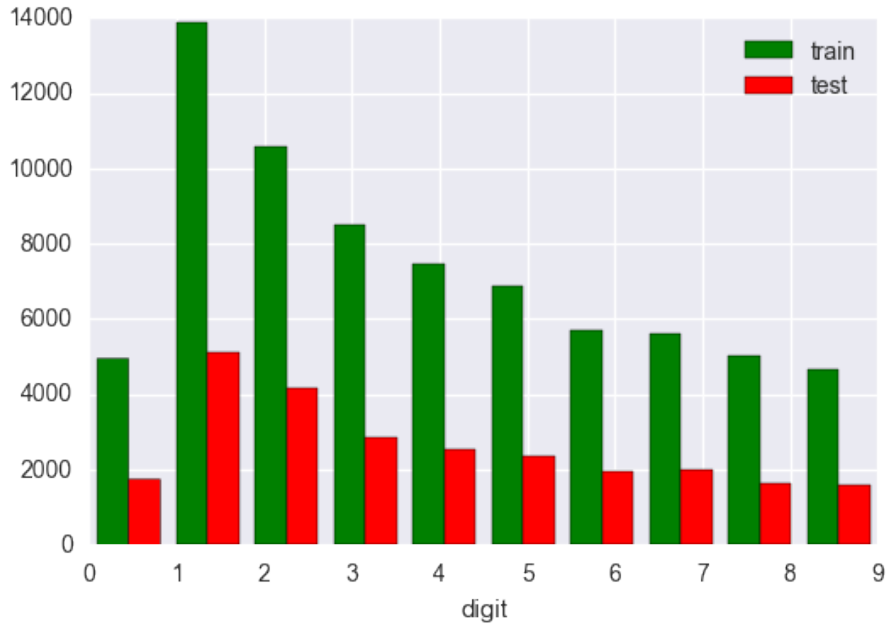

Figure 3 : Distribution of digits

Interestingly, the digit distribution is also quite asymmetric. Higher digits are less common than the lower digits, with digit 1 being the most common.

Algorithms and Techniques

Our goal is to train a model that predicts a sequence Y composed of individual digits $y_1 y_2 \dots y_n$ in natural left to right reading order, given an image X, by maximizing the probability $\log P(Y|X)$. We will divide our problem into two parts – predicting a length of sequence n' that maximizes the probability $P(n' = n|X)$, and predicting labels ${y_i}'$ that maximize $P({y_i}' = y_i|X)$ for $0 \leq i < n$. Hence, we can express our model as one that maximizes

$$P(Y' = Y|X) = P(n' = n|X) \prod_{i=0}^{n-1} P({y_i}' = y_i|X)$$

which is a combination of a digit counter and a digit detector for each index i such that $0 \leq i < n$.

We will use a multi-layer convolutional neural network followed by fully connected layers to form the counter and label detector models. Convolutional networks are well suited for inputs with spatial structures like images and videos.

Since the transformations between detecting digits in an image and making their counts should relatively be simple, we can hope to find a deterministic intermediate output H given input X, such that $P(Y|X) = P(Y|H)$. H will be a one-dimensional tensor which should contain information about all the digits in the image. Using H, the counter model should be able to count the number of digits, and label detector should be able to detect each digit, given its index i. With the shared convolutional model, we can hope to save a lot on computations.

After each activation layer, we use batch normalization[4] to center the mean and have unit variance. During training, batch normalization layers use per batch mean and variance to center node outputs. During prediction, they use the values learnt during training to center the node outputs. In the paper [4], it was suggested that batch normalization should be used before non-linear activation. However, later research and trends show that better results are achieved if it is used after activation.

Since deep learning networks are quite susceptible to overfitting, we use dropout layers[5] and image augmentation[6] to ensure that training and validation errors don't diverge.

We also use tensor concatenation to input index and transcoded image to the label detector. To train both the networks, we'll try to minimize categorical cross-entropy log loss using Adamax optimizer[7]. Through experimentation, we found that Adamax outperformed other optimizers for this problem in terms of convergence speed.

Benchmark

Our model is evaluated with a similar model proposed in [3], where they've used separate softmax layers for each digit. They also use image cropping to bring the digit sequence upfront. The benchmark model achieves an accuracy of 96.03% on SVHN test dataset. Another sequence prediction model proposed in [8] uses a combination of Convolutional Neural Networks and Hidden Markov Model with embedded Viterbi training. It achieves an accuracy of 81%.

Methodology

Data Preprocessing

For training, we flatten the dataset so that each sample point has a processed image, an index of the digit to be detected as inputs, and number of digits in image and the digit present at the given index as output. We do not do any flattening for test data since we have to predict all labels in an image together.

All images for training and testing are scaled to a fixed size of 54 x 128 pixels. An image augmentation system is also used that introduces slight horizontal, vertical, shear and zoom variations in the training images. It also transforms the images to have zero mean and unit variance. Validation and test images are also transformed for zero mean and unit variance. The image augmentation system generates a batch of 64 augmented training images and corresponding data per batch.

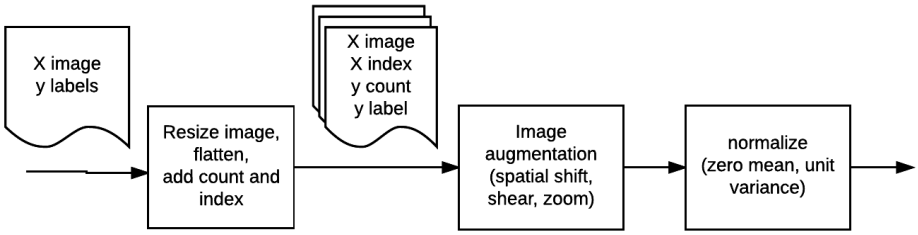

Figure 4 : Data preprocessing for training

Figure 5 : sample of image augmentations

During testing, the images are rescaled to 54 x 128 pixels, and normalized to have zero mean and unit variance.

Implementation

We implemented our deep learning network in Keras with Tensorflow backend. Our deep learning network is composed of following three macro models:

1. Vision

Vision model is a shared convolutional network that processes input images and transforms them into a dense tensor of length 1024. It has six convolution layers with feature sizes 32, 32, 64, 64, 128 and 128. All convolution layers have a window size of 3x3. Every convolution layer is followed by rectified linear unit activation and batch normalization. The input layer uses tanh activation as it provided better results. After every two convolution/activation/normalization pairs, we also use dropout and max pooling layers. After the convolution layers we have two fully connected layers with ReLU activations and batch normalizations. The output of last fully connected layer is the intermediate output H of width 1024 from the vision model, which is input to counter and detector models.

2. Counter

The counter model takes the output from vision model and predicts the number of digits present in the input image. It contains a fully connected layer of 256 nodes with ReLU activations and an output layer of 7 nodes with softmax activations.

3. Detector

The detector model has two inputs, H and the one hot encoded index of the digit to be detected. The first layer of detector merges these two inputs by concatenating. It then has two fully connected layers of width 512 nodes and ReLU activations, and an output layer of 10 nodes with softmax activations.

Graph Creation

The organization of these models in training and testing graphs is slightly different. During training phase, the index input to detector is provided from flattened training data, whereas during testing, this input is generated from the range of count that counter predicts. To achieve this reorganization, the individual models were first created using Keras' functional api with names clearly defined. For example,

```
#  define counter model
h_in_counter = Input(shape = (1024, ))
yc = Dense(256, activation = 'relu')(h_in_counter)
yc = BatchNormalization()(yc)
yc = Dropout(0.2)(yc)
yc = Dense(max_digits, activation = 'softmax')(yc)
counter_model = Model(input = h_in_counter, output = yc, name = 'counter')
```

We can then define the training graph by calling the models like functions

```
Ximg_in = Input(shape = (image_size[0], image_size[1], 3), name = 'train_input_
img')
Xidx_in = Input(shape = (max_digits, ), name = 'train_input_idx')
h = vision_model(Ximg_in)
yc = counter_model(h)
yl = detector_model([h, Xidx_in])
train_graph = Model(input = [Ximg_in, Xidx_in], output = [yc, yl])
```

The training graph is then saved using Keras CheckPoint callback. The individual models are retrieved later during testing by enumerating graph layers and referencing appropriately.

```
#  extract the individual models from training graph
vision = model.layers[1]
counter = model.layers[3]
detector = model.layers[4]
```

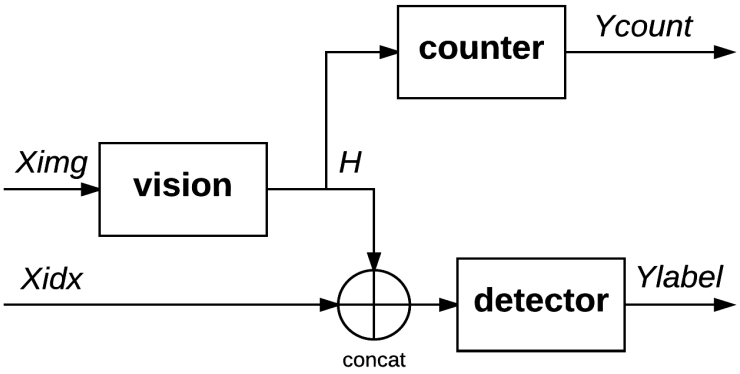

Figure 6 : architecture for training

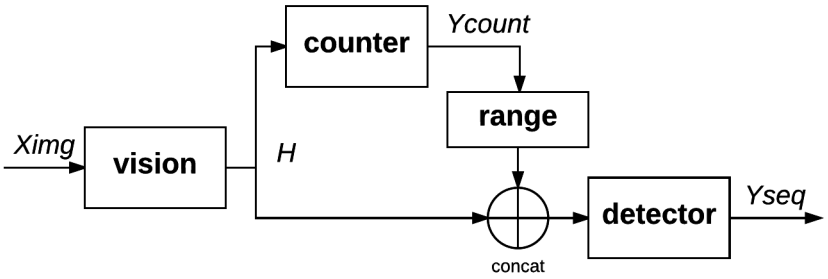

Figure 7 : architecture for prediction

The model was trained for 50 epochs on an Amazon AWS p2.xlarge instance using a custom AMI that had Keras, Tensorflow, CUDA and CuDNN installed among other libraries. 5% of training samples were used for validation.

Refinement

In initial models, we did not use image augmentation and batch normalization, and only used dropouts for regularization. These models had significant overfitting and very poor accuracy on flattened validation data. Increasing dropouts didn't help as the network would then stop converging. Following is one such training session with large overfitting.

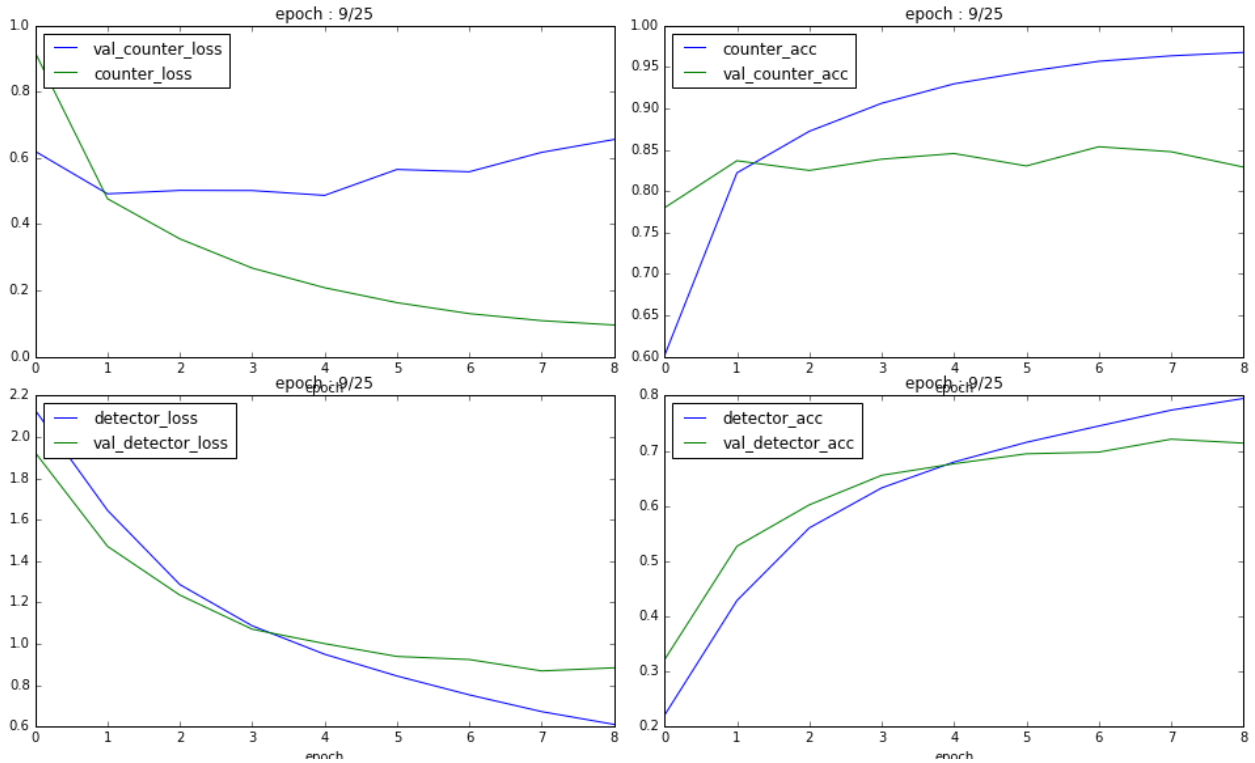

Figure 8 : training session without image augmentation

However, after an image augmentation generator was included, we saw almost no overfitting as evident from following training and validation graphs. Image augmentation makes sure that the network being trained doesn't necessarily see the same training image twice.

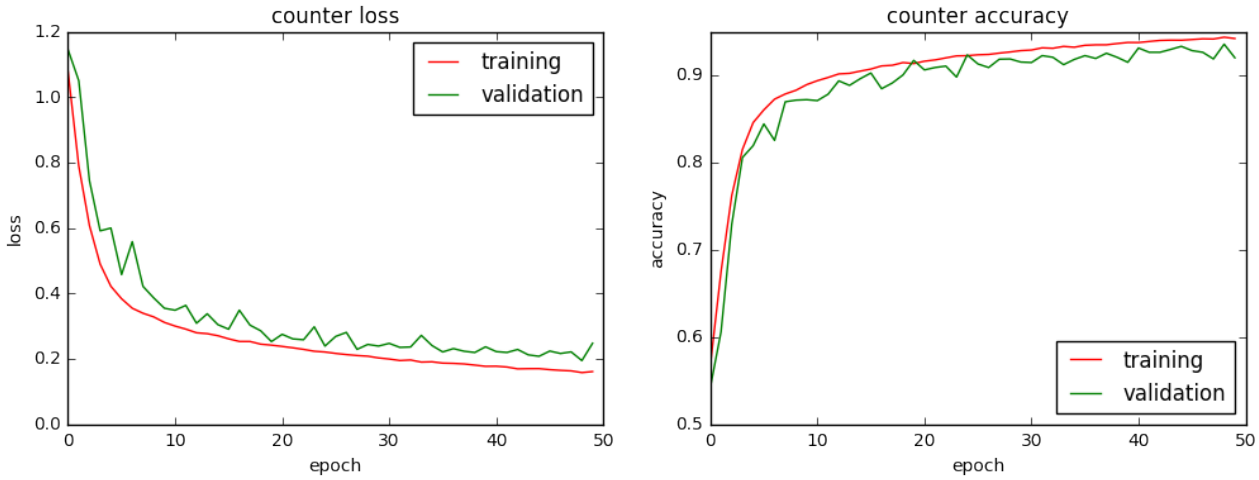

Figure 9 : counter training with image augmentation and batch normalization

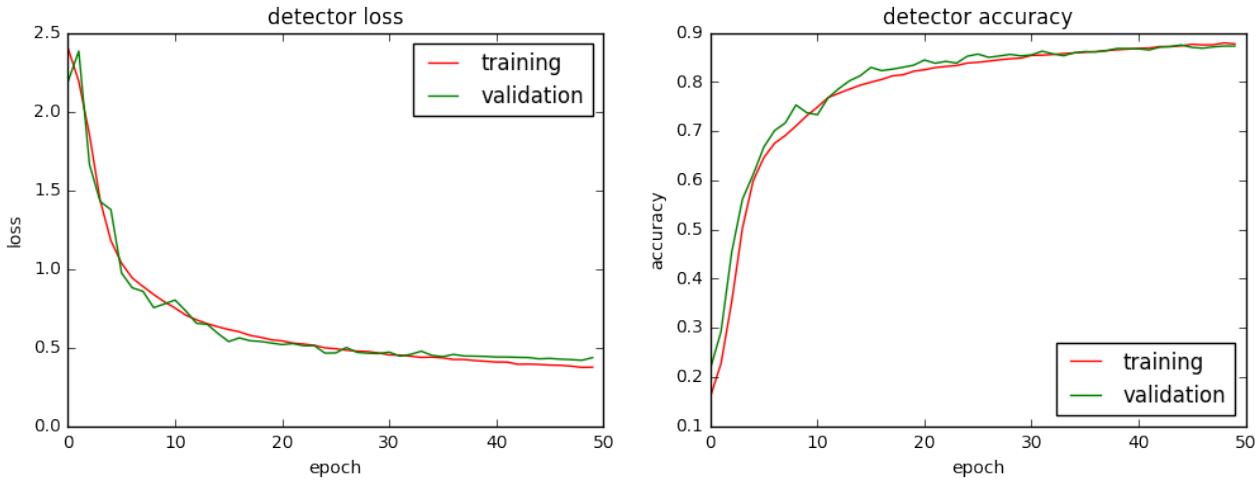

Figure 10 : detector training with image augmentation and batch normalization

On adding batch normalization, the individual accuracies of counter and detector further shot up by 4% on validation data.

Results

Model Evaluation and Validation

The final model which was described previously was chosen on the basis of validation and testing accuracies. It was observed and is understandable that higher accuracies of both counter and label detector did in fact translate to a higher sequence transcription accuracy on the test set. The final model achieved the following accuracies after training for 50 epochs:

Model	Training	Validation	Test
Counter	94.21%	92%	86%
Detector	87.73%	87.28%	83%
Sequence	-	-	68.28%

Following are some image samples with the predictions made by the system. Values in brackets are true values, and red indicates a wrong prediction.

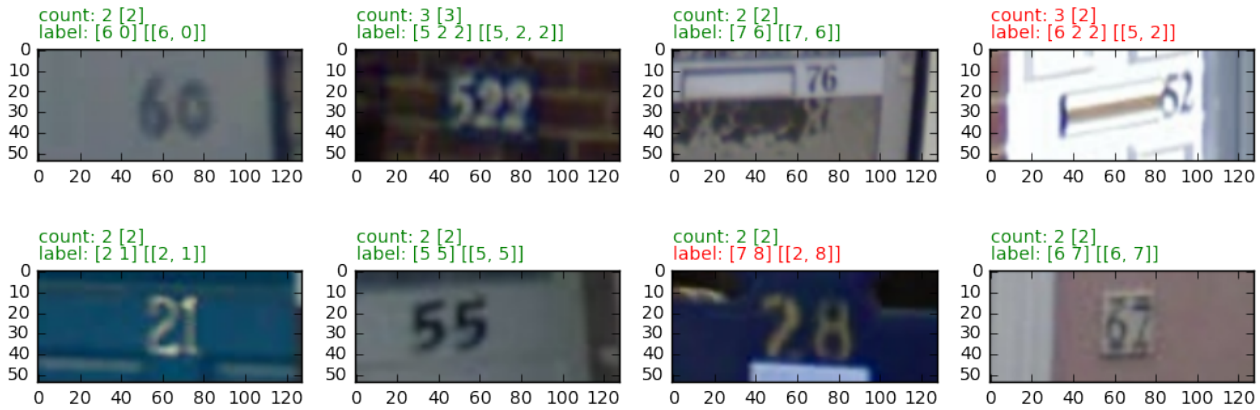

Figure 11 : samples with prediction from test set (values in brackets are true and red shows a mismatch)

As can be seen from accuracy metrics and samples above from test dataset, we can say that the model has done fairly well on unseen data.

Justification

The sequence transcription accuracy achieved by the model is 68.28% which is lower than the benchmark models. However, the benchmark model [3] was trained on a DistBelief cluster for 6 days, while our model was trained on a single AWS p2.xlarge for about 4 hours. From the training plots of our model, we see that there is a scope to increase the accuracy of the models further if trained for substantially longer periods. Also, the benchmark model uses a kind of localizer to bring the digit sequence upfront, making it easier for deep learning network to identify the sequence correctly.

The most powerful aspect of our design is its speed. Due to the fact that we process the input image with convolutional model only once, and intermediate output is shared by counter and detector models, the overall network is quite efficient. The network took only around 20 seconds to make predictions on the entire test dataset on Amazon AWS p2.xlarge instance.

Moreover, the label detector in our system is more generic in the sense that a single label detector is able to detect digits at all places with only the index input varying. In the benchmark model, there is a softmax layer for every digit place. So our model can be

extended to detect more digits without much increase in complexity, only the counter output and detector input width will increase.

Conclusion

Free-Form Visualization

Using Keras functional api, we can actually see the convolution features at different layers in the vision model for a given input. The convolutions of the first two convolution layers are shown for a sample input.

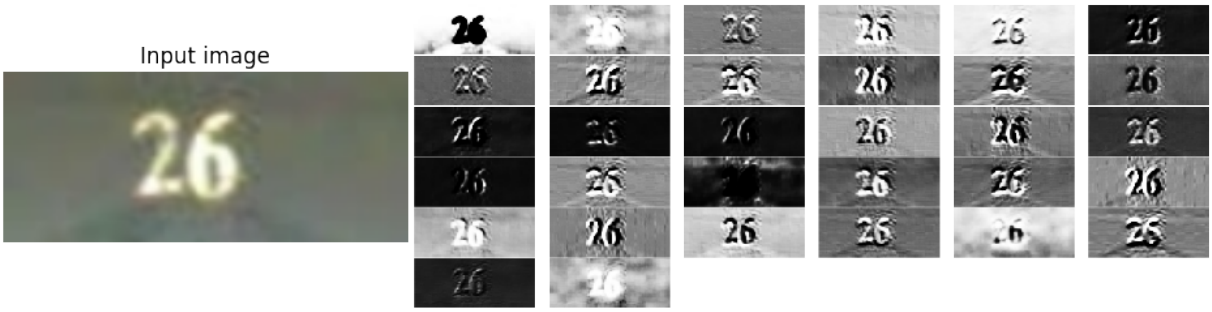

Figure 12 : convolution layer 1 features for a sample input

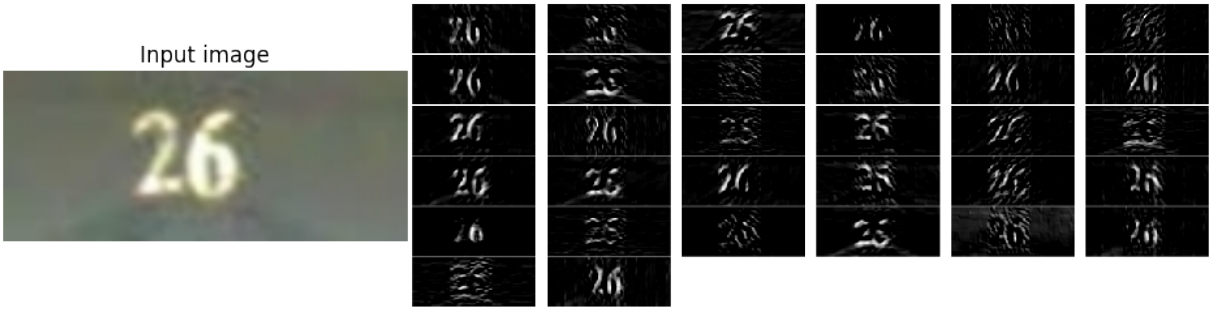

Figure 13 : convolution layer 2 features for a sample input

Interestingly, the convolution features appear to be results of edge detection filters from different directions. This tallies with literature explaining internals of CNNs[9]. We can also see that in layer two, the CNN was able to reject most of the noise and bias and had prominent digit edges, which would in the end be combined together to affirm presence of a particular digit.

Reflection

To summarize, a convolutional neural network based solution was designed to solve the publicly available SVHN dataset. The solution we designed comprised of three macro models, a shared vision model, a counter and a label detector. During training, the counter and label detector were trained with independently true data, which is a flattened version of SVHN training dataset. But during prediction, the label detector depends on counter output.

The architecture of this solution was a challenge to implement because of slightly different training and testing graphs. It was tried in both Tensorflow and Keras. Saving and loading the model weights and graphs was especially challenging in Tensorflow so Keras was used in the end because of its high level functional api and ability to directly use numpy arrays between models.

The biggest challenge we encountered was overfitting. Dropout layers were of very little help. It was eventually figured out that image augmentation does exceptionally well against overfitting. In our final model, we encountered little to no overfitting. It is perhaps the biggest lesson learnt from this project.

We also learnt that batch normalization significantly boosts the performance of deep learning networks. Even though training with batch normalization is much slower per epoch, we were able to achieve better results in much fewer epochs than without it.

The final solution that we developed did meet our expectations. There can definitely be improvements, but the solution developed can in fact be used in the same general setting. For example, we can also develop an English word reader using the same methodology.

Improvement

The rescaling step that is performed as part of preprocessing does result in loss of detail. If the digits are quite small in size compared to the image dimensions, then rescaling the image can render the digits completely unrecognizable. If we develop a localizer that detects and crops the digit sequence in an image, then we can pass sequences to our models in much more detail. For example, see the results of an original image from test set and a cropped section of it passed to our solution below.

Figure 14 : original vs cropped image passed to prediction network

We can also try increasing the number of layers in our models, as suggested in the reference [3]. If the network is trained for more time, we can expect the losses to further go down.

References

[1] Y. Netzer and T. Wang, “Reading digits in natural images with unsupervised feature learning,” *Nips*, pp. 1–9, 2011.

[2] T. Wang, D. J. Wu, A. Coates, and A. Y. Ng, “End-to-end text recognition with convolutional neural networks,” *ICPR, Int. Conf. Pattern Recognit.*, pp. 3304–3308, 2012.

[3] I. J. Goodfellow, Y. Bulatov, J. Ibarz, S. Arnoud, and V. Shet, “Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks,” *arXiv Prepr. arXiv …*, pp. 1–13, 2013.

[4] S. Ioffe and C. Szegedy, “Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift,” *Arxiv*, pp. 1–11, 2015.

[5] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A Simple Way to Prevent Neural Networks from Overfitting,” *J. Mach. Learn. Res.*, vol. 15, pp. 1929–1958, 2014.

[6] S. C. Wong, A. Gatt, V. Stamatescu, and M. D. McDonnell, “Understanding data augmentation for classification: when to warp?,” *DICTA*, Sep. 2016.

[7] D. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization,” *Int. Conf. Learn. Represent.*, pp. 1–13, 2014.

[8] Q. Guo, D. Tu, J. Lei, and G. Li, “Hybrid CNN-HMM Model for Street View House Number Recognition.”

[9] D. Stutz, “Understanding Convolutional Neural Networks,” *Nips 2016*, no. 3, pp. 1–23, 2014.