Project Report On

Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks

Submitted by

Aditya Jayasimha 15IT103
Vrishabh Sharma 15IT242
Shivani Shrivastava 15IT243

VI SEM B.Tech (IT)

Under the Guidance of

Dr.Biju R Mohan
Dept of IT,NITK Surathkal

in partial fulfillment for the award of the degree

of

Bachelor of Technology
In
Information Technology
At

Department of Information Technology
National Institute of Technology Karnataka, Surathkal.
March 2018

Abstract

Recognizing arbitrary multi-character text in unconstrained natural photographs is a hard problem. In this paper, we address an equally hard sub-problem in this domain viz. recognizing arbitrary multi-digit numbers from Street View imagery. Traditional approaches to solve this problem typically separate out the localization, segmentation, and recognition steps. In this propose we propose a unified approach that integrates these three steps via the use of a deep convolutional neural network that operates directly on the image pixels. This project explores how convolutional neural networks (ConvNets) can be used to identify series of digits in natural images taken from The Street View House Numbers (SVHN) dataset. The dataset contains real-world images of street numbers annotated with bounding boxes and labels. Learning to recognise sequences of digits or characters in real-world images is a fairly general problem with many real-world use-case such as reading the number and text from photographs such as e.g. reading the number plates of cars. The goal of the project is to train a ConvNet on the (i) 32x32 images SVHN dataset (ii) original SVHN dataset containing multi digit house numbers and measure how it performs on previously unseen images. The model must be able to identify sequences of multiple digits with a high accuracy in natural images containing various distortions including varying viewing angles, rotations, background noise etc. We use the TensorFlow framework for the implementation of our project.

Contents

List of Figures

List of Tables

1 Introduction

This project explores how convolutional neural networks (ConvNets) can be used to identify series of digits in natural images taken from The Street View House Numbers (SVHN) dataset [1]. The dataset contains real-world images of street numbers annotated with bounding boxes and labels. Learning to recognise sequences of digits or characters in real-world images is a fairly general problem with many real-world use-case such as reading the number and text from photographs such as e.g. reading the number plates of cars. A previous paper written by Goodfellow et al. [2] which has influenced this work is used a ConvNet to automatically geo-tag some addresses currently in Google Maps.

2 Literature review

2.1 Literature Survey

Link	Approach	Methodology
http://ufldl.stanford.edu/housenumbers/nips2011_housenumbers.pdf	Determine how features generated by feature learning systems compare to hand-constructed feature representations that are commonly used in other computer vision systems.	It explores the use of several existing unsupervised feature learning algorithms that learn these features from the data itself. Each of these methods involves an unsupervised training stage where the algorithm learns a parametrized feature representation from the training data. The result of this training stage is a learned feature mapping that takes in an input image and outputs a fixed-length feature vector to be used for supervised training in the same supervised training and testing pipeline as used for hand-crafted features.
	Evaluation of various feature representations and classification models on the SVHN datasets.	We have experimented with two different feature learning algorithms : (i) stacked sparse auto-encoders and (ii) the K-means-based system of .
	Observation : The K-means-based system appears to perform slightly better. This appears to come from the spatial-pooling stage, which leads to increased robustness to translations and local distortions.	Accuracy : HOG 85.0% , BINARY FEATURES 63.3% , K-MEANS 90.6% , STACKED SPARSE AUTO-ENCODERS 89.7%
https://arxiv.org/pdf/1409.2752.pdf	It introduces fully-connected winner-take-all autoencoders that learn to do sparse coding by directly enforcing a winner-take-all lifetime sparsity constraint.	The FC-WTA autoencoder is a nonsymmetric autoencoder where the encoding stage is typically a stack of several ReLUlayers and the decoder is just a linear layer.In the forward feed, we employ spatial sparsity and in backward propagation: we apply lifetime sparsity
	Accuracy : Stacked CONV-WTA Autoencoder, 256 and 1024 maps (N=600K) 93.1%	Problems : FC-WTA autoencoder if the receptive field is small, this method will not capture relevant features (imagine the extreme of 1 1 patches). Increasing the receptive field size is problematic, because then a very large number of features are needed to account for all the position-specific variations within the receptive field.

Table 1: Literature Review 1

Link	Approach	Methodology
https://arxiv.org/pdf/1301.3557.pdf	It replaces the conventional deterministic pooling operations with a stochastic procedure, randomly picking the activation within each pooling region according to a multinomial distribution, given by the activities within the pooling region.	In stochastic pooling, we select the pooled map response by sampling from a multinomial distribution formed from the activations of each pooling region. More precisely, we first compute the probabilities p for each region j by normalizing the activations within the region: pi = P ai kRj. We then sample from the multinomial distribution based on p to pick a location l within the region. The pooled activation is then simply al : sj = al where l P(p1, . . . , p—Rj —)
	The existing state-of-theart on this dataset is a multi-stage convolutional network, but stochastic pooling beats this by 2.10% (relative gain of 43%).	The approach is hyper-parameter free and can be combined with other regularization approaches, such as dropout.Our stochastic pooling helps to prevent overfitting even in this large model.
https://arxiv.org/pdf/1302.4389.pdf	It defines a simple new model called maxout (so named because its output is the max of a set of inputs, and because it is a natural companion to dropout) designed to both facilitate optimization by dropout and improve the accuracy of dropouts fast approximate model averaging technique.	Dropout can be thought of as a form of model averaging in which a random subnetwork is trained at every iteration and in the end the weights of the different such random networks are averaged. Since one cannot average the weights explicitly, an approximation is used. This approximation is exact for for a linear network.
	Maxout works better than max pooled rectified linear units Small model on large dataset 2 convolutional layers Training with big SVHN dataset (600,000 samples) Maxout error rate : 5.1% Rectifier error : 7.3%	In a convolutional network, a maxout feature map can be constructed by taking the maximum across k affine feature maps (i.e., pool across channels, in addition spatial locations). When training with dropout, we perform the elementwise multiplication with the dropout mask immediately prior to the multiplication by the weights in all caseswe do not drop inputs to the max operator. A single maxout unit can be interpreted as making a piecewise linear approximation to an arbitrary convex function. Maxout networks learn not just the relationship between hidden units, but also the activation function of each hidden unit.

Table 2: Literature Review 2

Link	Approach	Methodology
http://yann.lecun.com/exdb/publis/pdf/wan-icml-13.pdf	It introduces DropConnect, a generalization of Dropout, for regularizing large fully-connected layers within neural networks.When training with DropConnect we set a randomly selected subset of weights within the network to zero. A key component to successfully training with DropConnect is the selection of a different mask for each training example. Selecting a single mask for a subset of training examples, such as a mini-batch of 128 examples, does not regularize the model enough in practice. Since the memory requirement for the Ms now grows with the size of each mini-batch, the implementation needs to be carefully designed as described in Section 5. Once a mask is chosen, it is applied to the weights and biases in order to compute the input to the activation function. This results in r, the input to the softmax layer which outputs class predictions from which cross entropy between the ground truth labels is computed. The parameters throughout the model then can be updated via stochastic gradient descent (SGD) by backpropagating gradients of the loss function with respect to the parameters.To update the weight matrix W in a DropConnect layer, the mask is applied to the gradient to update only those elements that were active in the forward pass.	Each element of the mask M is drawn independently for each example during training, essentially instantiating a different connectivity for each example seen. Additionally, the biases are also masked out during training. The approach is hyper-parameter free and can be combined with other regularization approaches, such as dropout.Our stochastic pooling helps to prevent overfitting even in this large model. Error rate : No-Drop 1.94 , Dropout 1.96 , DropConnect 1.94

Table 3: Literature Review 3

2.2 Outcome of Literature Review

The first paper helped us to wisely choose the feature extraction technique. We hence, chose machine feature extraction strategy(Stacked encoders, Relu , pooling layers) over hand feature extraction techniques(Histogram of oriented gradients. binarisation etc.). Even though it involves hyperparameters tuning, we prefer that approach as the acuuracy is higher as discussed in the paper. The second paper helped us to understand the forward feeding and also the role of sparsing in the neural network.The author also mentioned the use of back propagation for further accurate results. The third paper introduces the concept of stochastic pooling, that considers the activation of the neutrons before randoly deactivating it. The fourth paper introduced us to Maxout pooling that helps to approximate non-linear or convex functions as linear functions. In our project, we have implemented the maxpooling technique. The fifth paper has proposed the idea of DropConnect, similar to the concept of Dropout. Both show better performances over no-drop models. We have used the dropout model for our project

2.3 Problem Statement

The goal of the project is to train a ConvNet on the original SVHN dataset containing multi digit house numbers and measure how it performs on previously unseen images. The model must be able to identify sequences of multiple digits with a high accuracy in natural images containing various distortions including varying viewing angles, rotations,background noise etc.

2.4 Research Objectives

1. Tweaking the model/architecture in the hope of improving the architecture.
2. Analysing our results in terms of the architecture.
3. Trying to apply the model to similar datasets and analysing performance

3 Methodology

3.1 System Architecture

Our basic approach is to train a probabilistic model of sequences given images. Let S represent the output sequence and X represent the input image. Our goal is then to learn a model of $P(S|X)$ by maximizing $logP(S|X)$ on the training set.

To model S, we define S as a collection of N random variables S1, . . . , SN representing the elements of the sequence and an additional random variable L representing the length of the sequence. We assume that the identities of the separate digits are independent from each other, so that the probability of a specific sequence s = s1, . . . , sn is given by

$$P(S = s|X) = P(L = n|X) * \Pi(i = 1ton)P(Si = si|X).$$

This model can be extended to detect when our assumption that the sequence has length at most N is violated. To allow for detecting this case, we simply add an additional value of L that represents this outcome.

Each of the variables above is discrete, and when applied to the street number transcription problem, each has a small number of possible values: L has only 7 values (0,1,2,3,4,5 and more than 5), and each of the digit variables has 10 possible values. This means it is feasible to represent each of them with a softmax classifier that receives as input features extracted from X by a convolutional neural network. We can represent these features as a random variable H whose value is deterministic given X. In this model, $P(S|X) = P(S|H)$.

To train the model, one can maximize $logP(S|X)$ on the training set using a generic method like stochastic gradient descent. Each of the softmax models (the model for L and each Si) can use exactly the same backpropagation learning rule as when training an isolated softmax layer, except that a digit classifier softmax model backprops nothing on examples for which that digit is not present.

At test time, we predict s = (l, s1, . . . , sl) = argmax L,S1,...,SL $logP(S|X)$.
This argmax can be computed in linear time. The argmax for each character can be computed independently. We then incrementally add up the log probabilities for each character. For each length l, the complete log probability is given by this running sum of character log probabilities, plus log P(l — x). The total runtime is thus O(N).

3.2 Detailed Explanation of Dataset

The SVHN dataset is a real-world images of street addresses. It can be seen as similar to the MNIST dataset but it incorporates over 250,000 images and comes from a significantly harder real-world problem of recognising sequences of digits in natural images of different sizes containing various distortions, viewing angles, rotations, background noise etc. The dataset has 10 classes, 1 for each digit. Digit '1' has label 1, '9' has label 9 and '0' has label 10. It has 3 different files : Training set, test set and extra set. The training set consists of fewer, more difficult images while the

extra set consists of a large number of simpler images to be used as extra training data for the model.

The SVHN dataset comes in two formats:

1. MNIST-like 32-by-32 images centered around a single character (many of the images do contain some distractors at the sides) :
   This dataset has character level ground truth in an MNIST-like format. All digits have been resized to a fixed resolution of 32-by-32 pixels. The original character bounding boxes are extended in the appropriate dimension to become square windows, so that resizing them to 32-by-32 pixels does not introduce aspect ratio distortions. Nevertheless this preprocessing introduces some distracting digits to the sides of the digit of interest. Loading the .mat files creates 2 variables: X which is a 4-D matrix containing the images, and y which is a vector of class labels. To access the images, X(:,:,:,i) gives the i-th 32-by-32 RGB image, with class label y(i).

2. Original images with character level bounding boxes :
   These are the original, variable-resolution, color house-number images with character level bounding boxes. The bounding box information are stored in digitStruct.mat instead of drawn directly on the images in the dataset.) Each tar.gz file (one each for train, test and extra) contains the original images in png format, together with a digitStruct.mat file, which can be loaded using Matlab. The digitStruct.mat file contains a struct called digitStruct with the same length as the number of original images. Each element in digitStruct has the following fields: name which is a string containing the filename of the corresponding image. bbox which is a struct array that contains the position, size and label of each digit bounding box in the image. Eg: digitStruct(300).bbox(2).height gives height of the 2nd digit bounding box in the 300th image.

3.3 Algorithms/Techniques

ConvNets are very similar to ordinary Neural Networks. However, ConvNet architectures make the explicit assumption that the inputs are images, which allows us to encode certain properties into the architecture. There are three main types of layers used to build ConvNet architectures: Convolutional Layer, RELU layer, Pooling Layer and a Fully-Connected Layer. We stack these together in an architecture.

The Input Layer (containing the image) typically has three dimensions. The width, height and the number of color channels in the image. A 32 x 32 color image will have a shape or (32 x 32 x 3) while grayscale images only have one color channel (32 x 32 x 1). The width and height should preferably be divisible by 2 many times. Common numbers include 32 (e.g. CIFAR-10), 64, 96 (e.g. STL-10), or 224 (e.g. common ImageNet ConvNets), 384, and 512.

The Convolutional Layer (CONV) layer is the core building block of a ConvNet. The CONV layer's parameters consists of a set of learnable filters. Each filter is small spatially (along width and height), but extends through the full depth of the input volume. A typical filter of a ConvNet might have size 5x5x3 (i.e. 5 pixels width and height, and 3 because the image have depth 3, the color channels). During the pass we slide each filter across the width and height of the input volume

and compute dot products between the entries of the filter and the input at any position. This means that the network will learn filters that activate when they see a particular visual feature in the image such as e.g. an edge. If you stack multiple CONV layers with 3x3x3 filters on top of each other (with non-linearities in between) each neural on the first CONV layer will have e.g. a 3x3 view of the input volume, the neuron in the second CONV layer will have a 5x5 view of the the input layer and so on. This filters are typically memory intensive and computationally expensive. It is generally preferred to stack small filters than having a single layer with big filters as this allows us to express more powerful features. However this is computationally expensive and might require more memory. The CONV layers should be using small filters (e.g. 3x3 or at most 5x5), a stride of 1 and zero padding.

The RELU layer Applies an element-wise activation function, such as max(0, x) thresholding at zero. This leaves the size of the volume unchanged.
The Pooling layer perform downsampling of the spatial dimensions in the input. The most common setting is to use max-pooling with 2x2 receptive fields and with stride 2. This will discard 75[32, 32, 12] the resulting volume will be [16, 16, 12].
In the Fully-Connected Layer the neutrons in a fully connected layer will have connections to all activations in the previous layer as seen in regular neural networks. The last fully-connected layer is called the output layer or the softmax layer and in classification settings it represents the class scores.

The most common form of ConvNet architecture stacks a few CONV-RELU layers followed by POOL layers, and repeats this pattern has been merged spatially to a small size. At some point it is common to transition to fully-connected layers and the last fully connected layer will contain the class scores. ConvNet architectures generally follow the following pattern:
INPUT $\rightarrow [[CONV \rightarrow RELU] * N \rightarrow POOL] * M \rightarrow [FC \rightarrow RELU] * K \rightarrow FC$

The **softmax** function is often used in the final layer of a neural network-based classifier. Such networks are commonly trained under a log loss (or cross-entropy) regime, giving a non-linear variant of multinomial logistic regression.
Dropout is a regularization technique for reducing overfitting in neural networks by preventing complex co-adaptations on training data. It is a very efficient way of performing model averaging with neural networks. **Batch Normalization** is a method to reduce internal covariate shift in neural networks, leading to the possible usage of higher learning rates. In principle, the method adds an additional step between the layers, in which the output of the layer before is normalized. It basically subtracts the mean of the data and divides by standard deviation. It introduces two learnable parameters which can be adjusted by the optimizer function. **Cross Entropy** is a method of estimating the loss during classification task. Loss functions are minimized by using **Optimization** techniques.Common Optimization techniques are Adam optimizer, Adagrad optimizer, Gradient Descent Optimizer.

3.4 Detailed design methodology

The paper preprocesses the dataset in the following way first step is to find the small rectangular bounding box that will contain individual character bounding boxes. We then expand this bounding box by 30% in both the x and the y direction, crop the image to that bounding box and resize

the crop to 64x64 pixels.

The best architecture of the paper consists of eight convolutional hidden layers, one locally connected hidden layer, and two densely connected hidden layers. All connections are feedforward and go from one layer to the next (no skip connections). The first hidden layer contains maxout units (with three filters per unit) while the others contain rectifier units. The number of units at each spatial location in each layer is [48, 64, 128, 256] for the first four layers and 256 for all other locally connected layers. The fully connected layers contain 256 units each. Each convolutional layer includes max pooling and subtractive normalization. The max pooling window size is 2x2. The stride alternates between 2 and 1 at each layer, so that half of the layers dont reduce the spatial size of the representation. All convolutions use zero padding on the input to preserve representation size. The subtractive normalization operates on 3x3 windows and preserves representation size. All convolution kernels were of size 5x5. Training is done with dropout applied to all hidden layers but not the input.

4 Work Done

4.1 Experimental Framework

1.Framework - TensorFlow
2.Server Details - Tesla Server

We first work on the SVHN 32x32 dataset and develop a ConvNet model using TensorFlow for identifying the single digits with maximum possible accuracy. After doing this, we then shift to the Original SVHN dataset and repeat the same procedure.

4.2 Preprocessing of SVHN 32x32 dataset

Before we train a model for the SVHN 32x32 dataset we follow the following procedure.

1. We first download the dataset (train,test and extra set) and store it in a folder (say data).
2. The dataset comes in the form of 3 MATLAB files: train_32x32.mat, test_32x32.mat, extra_32x32.mat. We load the dataset using loadmat function of scipy to get the images and the labels(number from 1 to 10 for each cropped 32x32 image).
3. We check a few images and the dimensions of the training, testing and extra set. Training set has 73257 images, test set has 26032 images and extra set has 531131 images. Each image is of dimension 32x32x3 (width - height - color channels).
4. We change the labels 10 (for digit 0) to 0.
5. We create a balanced validation set consisting of 4000 images from train set and 2000 images from extra set in which there are 400 images of each label from train set and 200 images of each label from validation set. After this we delete all the images in the validation set from train/extra set.
6. The remaining train plus extra set becomes the new training data to our CNN model.
7. We convert RGB images to grayscale images. Now the training set has 598388 images, test set has 26032 images and validation set has 6000 images.
8. One hot encoding is done to the labels.
9. The RGB images as well as Grayscale images are serialized and stored using h5py package.

4.3 Training a Convolutional Neural Network to identify the numbers

We trained a Convolutional Neural Network for identifying the numbers in the images. We tried out a few architectures and the architecture which got us best results were as follows:

$$[CONV-RELU-POOL]->[CONV-RELU-POOL]->DROPOUT->FC->FC->SOFTMAX$$

The architecture consists of 2 Convolutional Layers. Each Convolutional Layer is followed by a RELU activation function which basically rectifies the input (output = max(0,x) where x is input).

This is followed by pooling layer. First Convolutional Layer has filter size 5 and uses 32 filters. Second Convolutional Layer has filter size 5 and uses 64 filters. The strides in the Convolutional Layers are kept 1x1 and strides in the pooling layer is kept to 2x2. After the 2 convolutional layers, before passing the output to Fully Connected Layer, dropout operation is performed which randomly drops neurons in order to prevent overfitting of the training data. Dropout rate is kept to 0.5. The number of output neurons in the first fully connected layer is 256 while the number of output neurons in the second convolutional layer is the number of classes i.e 10. Output of second Fully Connected Layer is fed to Softmax layer which converts the 10 outputs into a set of probabilities for each of the 10 labels. The loss function used is Cross Entropy. The optimizer used for minimizing the loss function is AdaGrad Optimizer. Learning rate is kept to 0.05 and exponentially decays by a factor of 0.96 after every 10000 iterations.

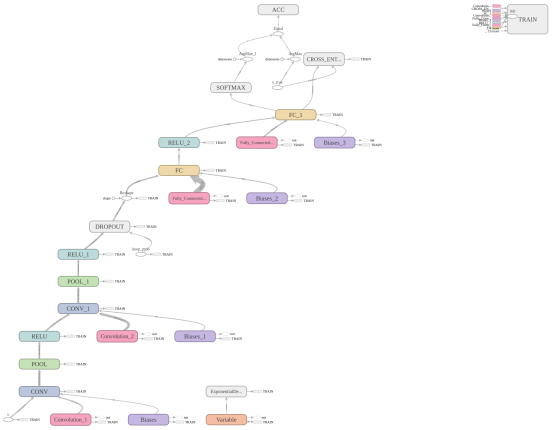

Figure 1: Architecture 32x32

4.4 Preprocessing the Original SVHN Dataset

Before we train a model for the Original SVHN dataset we follow the following procedure.

1. We first download the dataset (train,test and extra set) and store it in a folder (say data).
2. We extract the tar.gz files and get the bounding box information from digitStruct.mat file for each of the train, test and extra sets.
3. We create a dataframe to store the filename (for each image) (in the format /data/train/_ or data/test/_ or data/extra/_) , label (the number), the width, height and position from top and left of each bounding box with respect to the image.
4. We find the digit sequence bounding box by taking the minimum (x0, y0) and maximum (x1, y1) of each image. Also we append all the labels for a single image to list in proper order. We also keep a note of length of the number. These information are appended to the dataframe.

5. We expand the bounding box by 30% in both width and height.
6. We also obtain the image sizes of each of the images.
7. We merge this data to our dataframe.
8. Next we apply the correction to the bounding box such that it is contained in the image.
9. The dataset has only 1 image of a number of length 6 so we omit it
10. We change the label 10 for a number to label 0.
11. We crop the bounding boxes.
12. We resize the bounding box to 64x64 pixels and segregate the images to train, test and extra set based on the filename.
13. The list of numbers for an image are concatenated to form the label of the number.
14. We examine the number of examples in each set for a given length of the number.
15. We create a balanced validation set consisting of 3000 images from train set and 2000 images from extra set in which there are 600 images of each label from train set and 400 images of each possible length of a number (1 to 5) from validation set. After this we delete all the images in the validation set from train/extra set.
16. The remaining train plus extra set becomes the new training data to our CNN model.
17. We convert RGB images to grayscale images. Now the training set has 598388 images, test set has 230754 images, validation set has 5000 images, test set has 26032 images.
18. The RGB images as well as Grayscale images are serialized and stored using h5py package.

4.5 Training for Original SVHN dataset

1. The original SVHN dataset consists of images with numbers varying from 1 digit to 5 digit.
2. The CNN architecture is as follows:

$$[CONV->BN->RELU->POOL->DROPOUT]x7->CONV->BN->RELU->$$

$$POOL->FC->FC->SOFTMAX$$

3. The first layer is convolutional layer. This is followed by Batch Normalization. Batch Normalization is followed by RELU activation layer. After this is the pooling layer. This is followed by dropout. This entire unit is repeated 8 times.
4. After 8 layers, there are 2 fully connected layers followed by 5 softmax layers one for each digit of the number.
5. Both the fully connected layers have 256 neurons.

6. Each softmax layer will map the output of previous layers to a number between 1 and 10 where 10 specifies 0.

7. The filter size is 5 for each Convolutional layer. The number of filters is 32 for the first 2 Convolutional layers, 64 for the 3rd and 4th convolutional layer, 128 for the 5th and 6th convolutional layer and 256 for the last 2 Convolutional layers.

8. Batch normalization is done at each layer to ensure that the data is of zero mean and constant variance

9. Pooling is done at each layer. At the first, third, fifth and seventh layers the stride is kept to 2x2 while in the second, fourth, sixth and eighth layers, stride is kept to 1x1. This means that in every alternate layer only the images get downsized by a factor of 2 while in the other layers, dimension of image remains same.

10. The loss function used is Cross entropy measure.

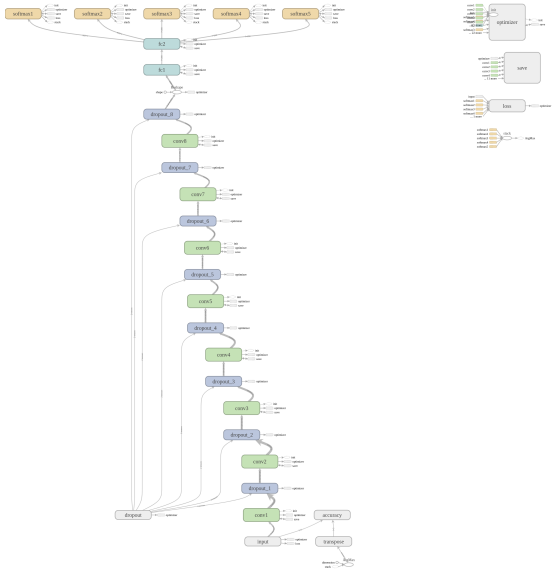

Figure 2: Architecture 64x64

4.6 Real time prediction

Once the model has been built this model is applied on real time images to see whether this model can detect numbers from real time street view house images. So a total of 35 images were taken from Google images and our model was run on them. We found that the numbers were correctly/ almost correctly detected by our model. Some of the examples are as follows:

Figure 3: Accuracy real time

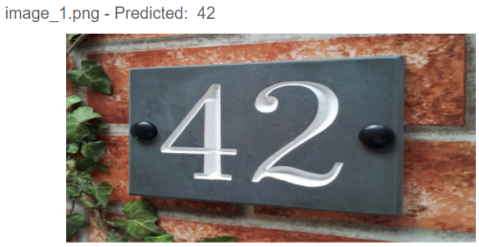

Figure 4: Image 1

4.7 Results

We obtained an accuracy of 96.26% on the test set for the SVHN 32x32 dataset for a batch size of 64 with the architecture specified above, with the number of iterations in each epoch being number of images in train set divided by 64. We ran for a total of 20 epochs which took about 4 and a half hours to run.

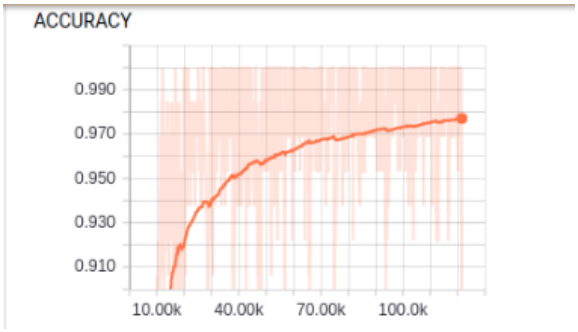

Figure 5: Accuracy 32x32

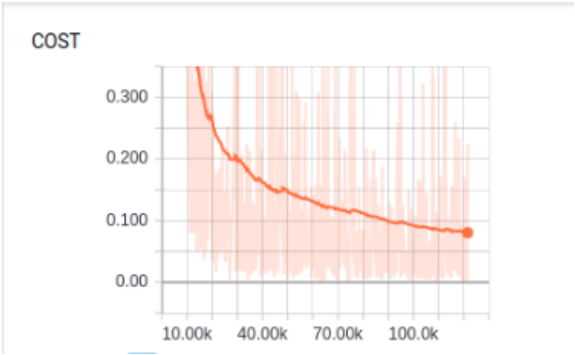

Figure 6: Cost 32x32

Our best architecture for SVHN 64x64 original dataset has an accuracy of 93.23%.

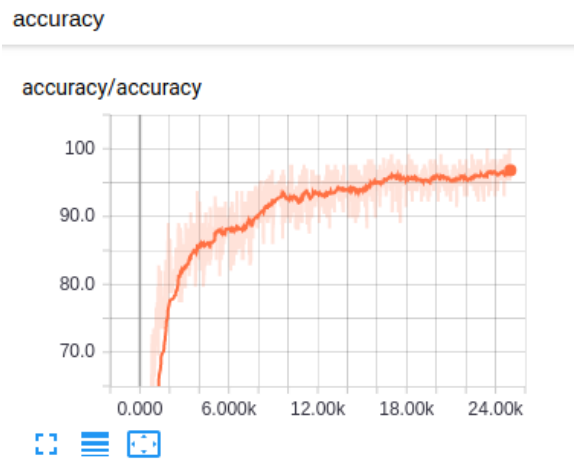

Figure 7: Accuracy 64x64

The parameters set for this architecture are as follows:

1. Batch Size : 128
2. Dropout Keep Probability : 0.9
3. Loss Function : Cross Entropy
4. Exponentially Decaying Learning Rate
5. Optimizer : AdamOptimizer
6. Number of iterations : 25000

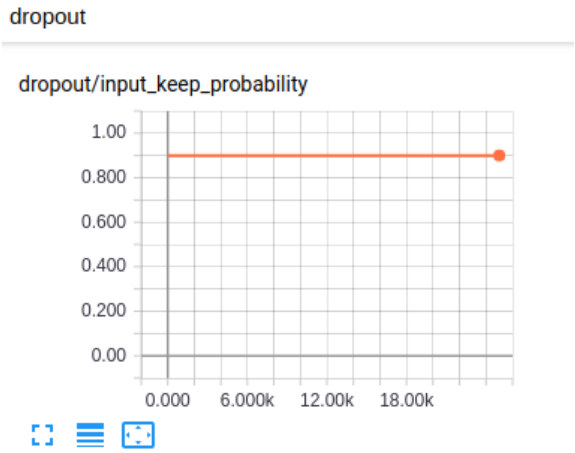

Figure 8: dropout 64x64

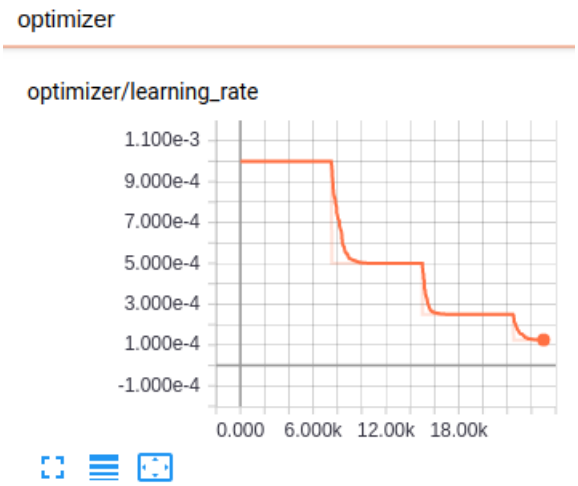

Figure 9: optimizer 64x64

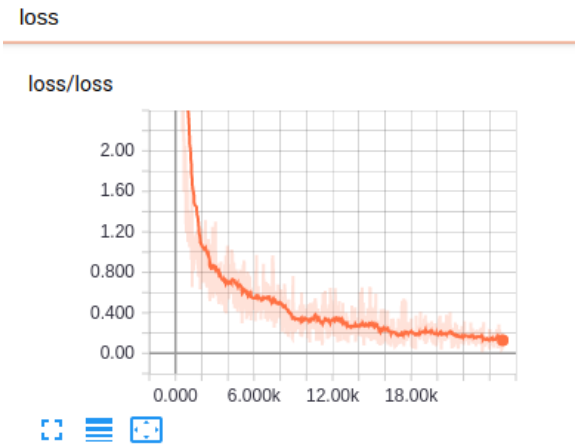

Figure 10: loss 64x64

4.8 Discussion

We also tried to train using the architecture for the original SVHN dataset which trains on 64x64 input images for the 32x32 dataset. That architecture has 8 Convolutional Hidden Layers each having Conv Layer followed by Batch Normalization, followed by RELU activation function, followed by Pooling Layer, followed by Dropout layer. This is followed by 2 dense layers followed by Softmax layer for length of number and for each digit (in original SVHN dataset). Zero padding applied to preserve the width and height of the image in each layer. Strides in convolutional layers are 1 while in pooling layer it alternates between 2 and 1 which results in every alternate layer reducing the width and height by half. Since we are dealing with 32x32 images here we chose to use 6 layers (instead of 8 layers - our intuition being that the 64X64 image also is a 32x32 image after 2 Convolutional Layers). So we chose to have 6 Convolutional Layers followed by 2 Dense layers. We trained with a dropout probability of 0.2 keeping in mind that dropout happens after every layer. We used the Gradient Descent Optimizer to train our model. Findings: By the end of 5th epoch, the train set accuracy is almost 100 percent for all the batches but between the 5th and 6th epoch the accuracy reduced from 96.04% to 95.97%. Reason: The architecture is too deep for the simple dataset and running many epochs has resulted in overfitting the training set. Hence causing the test set accuracy to go down.
The best architecture for SVHN 64x64 was obtained after fine tuning parameters such as changing the optimizer, changing the dropout probability, changing the loss function etc. A number of variations of fine tuning were carried out before using the above architecture.The variations included

- Modifying the dropout keeping probability makes the network becomes less sensitive to the specific weights of neurons. This in turn results in a network that is capable of better generalization and is less likely to overfit the training data.
- Using tf.reduce_sum over tf.reduce_mean leads to the increase in accuracy by 0.5 percent as the error being propagated in the later case is normalised which equally affects all the weights in the architecture.

- Before the choice of Adam Optimizer was finalized upon, RMSProp optimizer was tried for optimization function.We introduce Adam, an algorithm for first-order gradient-based optimization of stochastic objective functions, based on adaptive estimates of lower-order moments.
- Before choosing the number of iterations as 25000, we first carried out the process for 10000 iterations. The accuracy was 91.04%. The intuition behind opting for increased number of iterations was that train accuracy had still not reached 100 percent so there was scope for better training in subsequent iterations.
- Again we further increased number of iterations to 35000 but found the test accuracy to again go down to 93.08 percent. The reason for this is overfitting train data. So we decided to stick with 25000 iterations

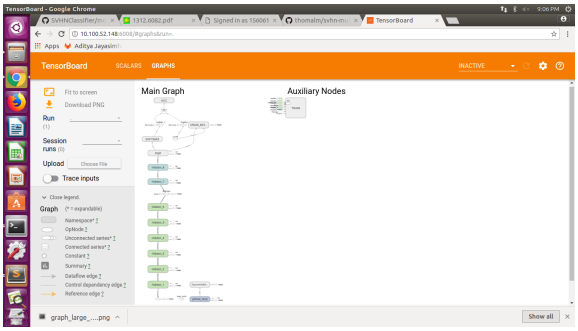

Figure 11: 6 Layer Architecture 32x32

4.9 Individual contribution

1. Literature Survey and Gap Analysis- Shivani
2. Preprocessing the 32x32 SVHN dataset : Aditya
3. Fitting CNN for 32x32 SVHN dataset: Vrishabh
4. Variations for 32x32 SVHN dataset : Aditya
5. Preprocessing original SVHN dataset: Shivani
6. Tuning of parameters for CNN architecture - Shivani and Aditya
7. Fitting the model for original CNN architecture - Vrishabh
8. Applying the model to real time images - Aditya
9. Report and presentation of the project - Vrishabh, Shivani and Aditya

5 Conclusion and future work

5.1 Conclusion

In this project, we carried out multidigit number recognition from street view images using the SVHN dataset. We first put our focus on the SVHN 32x32 dataset. This is a problem of single digit recognition but where images have background noise. We trained a 2 Convolutional followed by 2 fully connected layer followed by a Softmax layer architecture. We got a best accuracy of 96.26 percent. We then shifted our focus on to the original SVHN dataset which consists of 1 to 5 digit numbers. Our best architecture consisted of 8 Convolutional layers followed by 2 fully connected layers followed by 5 Softmax layers one for each digit, with Batch normalization and dropout after each convolutional layer. We achieved a best accuracy of 93.23 percent for this architecture after fine tuning some of the parameters and running for 25000 iterations keeping batch size 128. The model was then applied on real world images picked from Google images randomly and the results were satisfactory.

5.2 Future work

Beyond the scope of this project, we intend to carry out the following as a part of future work:

1. Data Augmentation can improve the accuracy by a small margin. Data Augmentation is to augment similar images to the images in the dataset to increase the dataset size. The intuition is that more the data, better is the accuracy. So with the hope of seeking higher accuracy, data is augmented with similar images (e.g consider an image classification problem to identify cats. A left facing cat can undergo reflection to a right facing cat which is also a cat itself). In our dataset we could do data augmentation by rotating the house number images by a few degrees, randomly cropping 54x54 images from the 64x64 images etc.

2. Try out other models such as RCNN models. RCNN stands for Recurrent Convolutional Neural Network. It consists of recurrent Convolutional layers ,that is output of the network is fed back to itself.

3. Test on more real time images and study if there exists a pattern of images which are predicted wrongly so that we could know if some digit or some style of writing a digit has not been trained properly.

4. Extend this model to other datasets such as character datasets, word datasets etc .

References

[1] "The Street View House Numbers (SVHN) Dataset",http://ufldl.stanford.edu/housenumbers

[2] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, Andrew Y. Ng "Reading Digits in Natural Images with Unsupervised Feature Learning",NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011

[3] "Numpy Reference", https://docs.scipy.org/doc/numpy-1.13.0/reference/

Appendix

- Multi-digit Number Recognition from Street View Imagery using Deep Convolutional Neural Networks