Machine Learning Engineer Nanodegree

Capstone Project

Mohd Firdause Bin Zainuddin

November 9th, 2016

I. Definition

Project Overview

Figure 1: MNIST datasets

First and formost, I have chosen the "Digit Recognizer" competition(which is hosted by Kaggle), https://www.kaggle.com/c/digit-recognizer, as my capstone project. Figure 1 refers to the MNIST dataset. Originally, MNIST dataset was developed and compiled from a huge library of scanned document dataset available from the National Institute of Standards and Technology which is also known as "NIST". Together, they merged and form the datasets which is also known as the Modified National Institute of Standards and Technology (MNIST).

Figure 2: SVHN datasets

However, due to its simplicity, I have decided to substitute MNIST dataset with Street View House Number(SVHN) datasets which is represented in figure 2, http://ufldl.stanford.edu/housenumbers/. In general, SVHN is a real-world image dataset for developing machine learning and object recognition algorithms with minimal requirement on data preprocessing and formatting. It can be seen as similar in flavor to MNIST (e.g., the images are of small cropped digits), but incorporates an order of magnitude more labeled data (over 600,000 digit images) and comes from a significantly harder, unsolved, real world problem (recognizing digits and numbers in natural scene images). SVHN is obtained from house numbers in Google Street View images.

According to my analysis, problem such as digit recognition should be solved for betterment of uses in domains such as automatic bank-check recognition [1]. This project will focus on classifying digit into one of 0 until 9. Digit recognition is almost a solved problem as deep learning algorithms have achieved a 99% + accuracy on the MNIST database [2]. Neural networks (even the shallow ones) work really well in the digit recognition problems. In my opinion, these research papers are relevant because they contain results that will provide evidence in how well each machine learning algorithms performs.

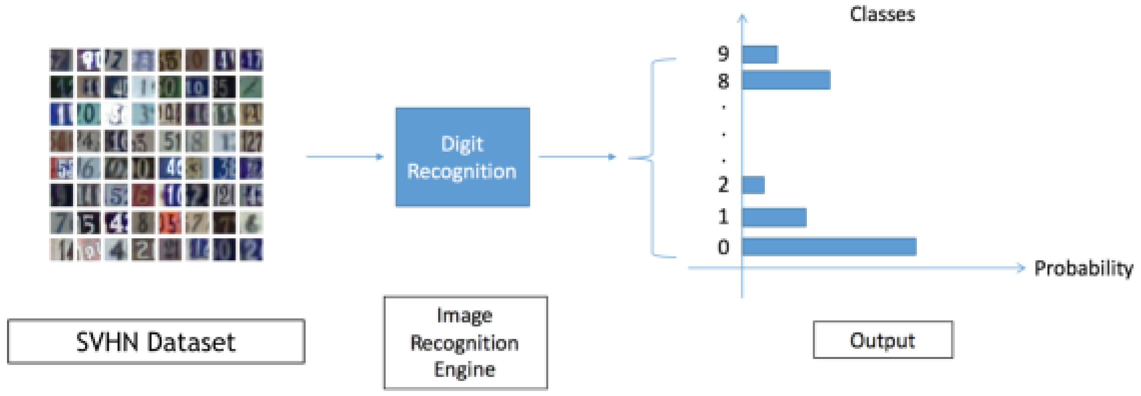


Figure 3: Convolutional neural network model

Figure 3 refers to the high level overview of CNN model that I have designed. First of all, SVHN dataset will be trained via the image recognition engine. Once the engine has been successfully trained, the engine will be able to classify the digits into 0-9 classes. The goal of this project is to explore convolutional neural network(CNN) based image recognition algorithm and their usage in identifying street house number. I have developed a simple image recognition algorithm via CNN and SVHN dataset. The architecture of the machine learning system will be discussed further in this report.

Problem Statement

The main goal of this project is to develop a CNN model that is able to decipher house numbers via SVHN datasets. I have simplified the problem by not taking multi-digits into account. It will be able to detect numbers from just image pixels. This project will focus on classifying a house numbers into one of 0, 1, 2 ... 9 via Convolutional Neural Network (CNN), a concept of Deep Learning.

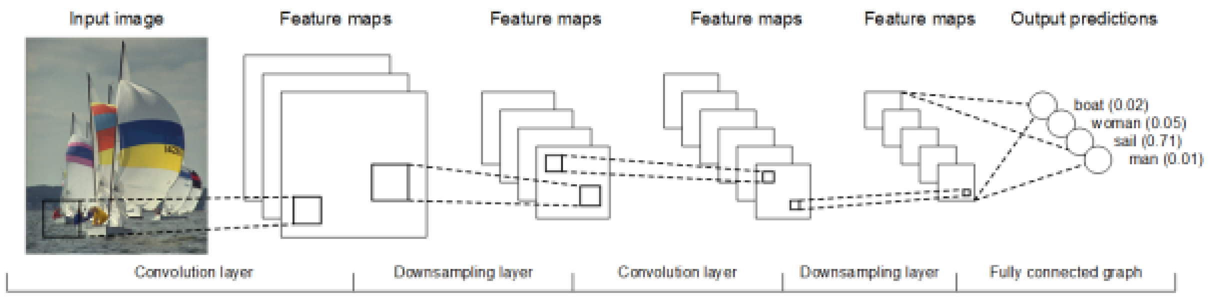


Figure 4: Architecture of CNN model

Solution:

- Separate SVHN dataset into training and testing set.
- Pre-process SVHN dataset via one hot encoding.
- Class labels are defined from 0 - 9.
- Develop CNN model and train the model via training set.
- Perform optimizations in order to increase the accuracy of the CNN model.

Metrics

$$Accuracy = \frac{true\ positives + true\ negatives}{dataset\ size}$$

Figure 5: Accuracy formula

The purpose of solving digit recognition problem is to deploy real world application such as automatic bank-check recognition via machine learning. Both true positive and true negatives matter as we need to be accurate when it comes to predicting the correct digits. As a result, accuracy will be the prime candidate to be used as a metric in this scenario.

The important thing to note is that I will not be doing multi-digit predictions via SVHN datasets. Instead, the model that I have designed assume that each images corresponds to one digit. The SVHN dataset format that I have chosen is MNIST-like 32-by-32 images centered around a single character (many of the images do contain some distractors at the sides).

In general, there are lots of different performance measures that answer different aspects of performance for classifiers, one of them is known as F1 score. Although F1 score is a better metric for this problem (it deals with class imbalance), however the report that I have written is not just meant for machine learning audience, it covers non-machine learning audience as well. When it comes to accuracy, the accuracy metric is sufficient enough (which can be easily understood by non-machine learning audience). It means a fraction of misclassifications.

II. Analysis

Data Exploration

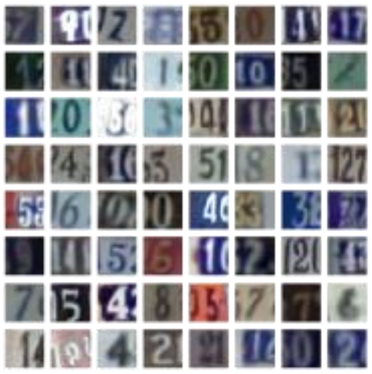


Figure 6: SVHN datasets

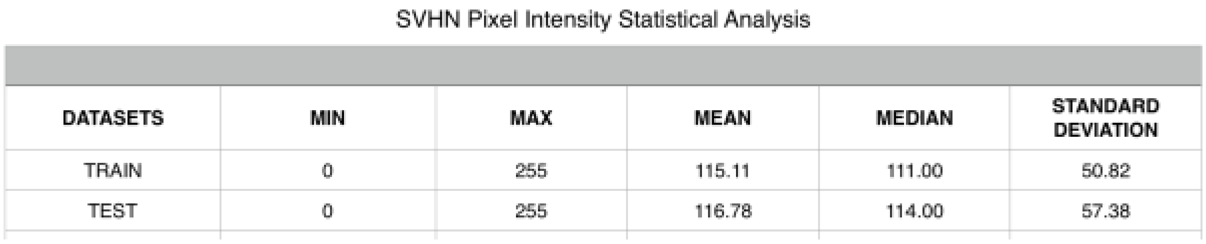
SVHN Datasets Overview:

1. 10 classes, 1 for each digit. Digit '1' has label 1, '9' has label 9 and '0' has label 10.
2. 73,257 digits for training set, 26,032 digits for testing set.
3. MNIST-like 32-by-32 images centered around a single character.
4. These images are RGB format(3 channels, one dimension for each color component R, G, B).
5. Apart from that, these images come with different fonts as well.
6. The images were taken from different angles.

7. Some images are blurry to the point where human beings are not able to identify them.

Figure 7: Blur images of SVHN datasets

SVHN datasets is a more challenging datasets as they are not neatly-lined up and has various skews, fonts and colors.



SVHN Pixel Intensity Statistical Analysis

DATASETS	MIN	MAX	MEAN	MEDIAN	STANDARD DEVIATION
TRAIN	0	255	115.11	111.00	50.82
TEST	0	255	116.78	114.00	57.38

Figure 8: Pixel intensity for train and test images

Referring to the table above, the mean, median, maximum and minimum pixel intensity for both training and testing datasets are quite similar. The only slight difference is their standard deviation, which is 50.82 for training set and 57.38 for testing set.

Exploratory Visualization

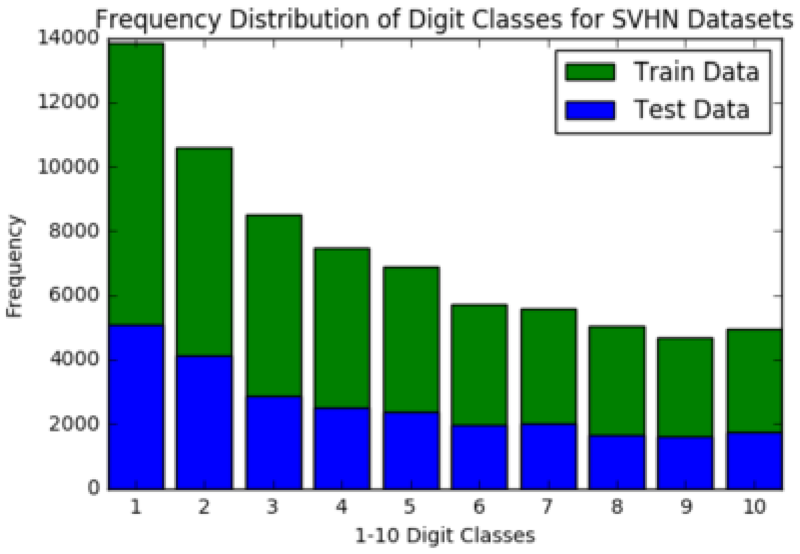


Figure 9: Frequency distribution of digit classes for SVHN datasets

I have counted the number of occurrence of each labels in the training and test sets. It is obvious from the figure that the distribution is not uniform for both train and test sets. The plot shows that the datasets are skewed towards class '1'.

Algorithms and Techniques

According to my analysis, digit recognition problem is a classification problem. This problem can be solved via convolutional neural network. Convolutional neural network can be used to predict a sequence of digits. This is mainly because the it has the ability to identify images at different scales and work on raw pixels. Convolutional neural networks consist of multiple layers of receptive fields. These are small neuron collections which process portions of the input image. The outputs of these collections are then tiled so that their input regions overlap, to obtain a better representation of the original image; this is repeated for every such layer. One major advantage of convolutional networks is the use of shared weight in convolutional layers, which means that the same filter (weights bank) is used for each pixel in the layer; this both reduces memory footprint and improves performance [3].

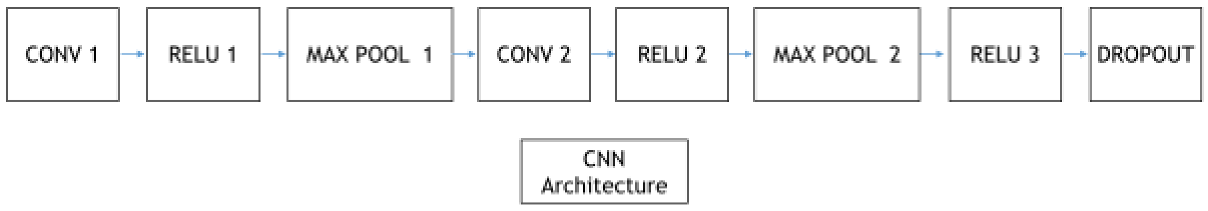


Figure 10: 4 layer CNN model

Convolutional neural network is split into five modules:

1. Input Layer:

    This is where all the pixels of all images are stored.

2. Convolutional Layer:

    It takes advantage of the fact that the input consists of images and they constrain the architecture in a more sensible way. In particular the layers of a convolutional neural networks have neurones arranged in 3 dimensions: width, height, depth.

3. ReLu Layer:

    Rectified Linear Units(ReLu) are used to implement element wise activation function. This does not alter the dimension of the output as compared to the input.

4. Pooling Layer:

    Pooling is equivalent to downsampling which can be used to reduce the spatial extent of convolutional neural network. It combines all the information from the previous convolution and pass it down to the next convolution.

5. Dropout Layer:

    Dropout is a technique that is used to reduce overfitting in neural networks. It works by rejecting a small amount of random neural network units in order to prevent overfitting. While neural networks and other pattern detection methods have been around for the past 50 years, there has been significant development in the area of convolutional neural networks in the recent past. This section covers the advantages of using CNN for image recognition.

Advantages of CNN:

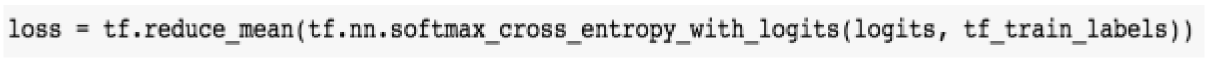
- Ruggedness to shifts and distortion in the image

    Detection using CNN is rugged to distortions such as change in shape due to camera lens, different lighting conditions, different poses, presence of partial occlusions, horizontal and vertical shifts, etc. However, CNNs are shift invariant since the same weight configuration is used across space. In theory, we also can achieve shift invariants using fully connected layers. But the outcome of training in this case is multiple units with identical weight patterns at different locations of the input. To learn these weight configurations, a large number of training instances would be required to cover the space of possible variations.

- Fewer memory requirements

    In this same hypothetical case where we use a fully connected layer to extract the features, the input image of size 32x32 and a hidden layer having 1000 features will require an order of 106 coefficients, a huge memory requirement. In the convolutional layer, the same coefficients are used across different locations in the space, so the memory requirement is drastically reduced.

The formula for cross entropy is:



```
loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits, tf_train_labels))
```

Figure 11: Cross entropy formula

AdamOptimizer:

AdamOptimizer uses Kingma and Ba's Adam algorithm to control the learning rate. Adam offers several advantages over the simple gradient descent optimizer. It uses moving averages of the parameters (momentum); Simply put, this enables Adam to use a larger effective step size, and the algorithm will converge to this step size without fine tuning [4].

Benchmark

SVHN datasets were created by human operators manually, with an accuracy of at least 98%. In order to beat that score, I will have to resort to convolutional neural networks. The convolutional neural network models were developed via neural networks and the distortions used tend to be either affine distortions or elastic distortions [5]. Sometimes, these models can be very successful; one such model achieved an error rate on the database of 0.39 percent [6].

Unfortunately, in order to beat the score of 98% I will need a very powerful computer to train the CNN model, which are currently not at my disposal. Therefore, I have decided to achieve accuracy approximately 90% via my laptop.

III. Methodology

Data Preprocessing

Before the CNN model was trained, the SVHN datasets need to be pre-processed.
Data preprocessing steps:

1. Reading the data:

   Firstly, the datasets were downloaded via http://ufldl.stanford.edu/housenumbers/. Both training and test sets come in Matlab's .mat format. Reading of the data is accomplished via scipi.io.loadmat() function.

2. Segregate data and classes:

   Once data has been read, they will be split into training data(73527 images, train_X, by reading the X values), training labels(10 classes, train_y, by reading the y values), test data(26032 images, test_X, by reading the X values) and test labels(10 classes, test_y, by reading the y values). I did not made any changes to prevent data imbalance.

3. Normalization:

   The images were normalized via the following formula:

      (Pixel/128.0 - 128.0) where "Pixel" is equivalent to the pixel(range from 0-255) of the images. The purpose of subtracting the dataset with 128.0 is to "center" the data. The reason we do both of those things is because in the process of training our network, we're going to be multiplying (weights) and adding to (biases) these initial inputs in order to cause activations that we then backpropogate with the gradients to train the model. We'd like in this process for each feature to have a similar range so that our gradients don't go out of control. The resulting range for all the features is (-1, 1) which should be enough for the CNN.

We are going to want the values involved in the calculations of this big loss function to never get too big or too small. We always want our variables to have zero mean and equal variance whenever possible. This is useful when we are doing optimization. A badly conditioned problem means the optimizer has to do a lot of searching in order to find the best solution. In contrast, a well-conditioned problem makes it easier for optimizer to search for the best solution. Normalization does not change the content of the SVHN images, but it makes it much easier for the optimization to proceed numerically.

4. One-hot encoding:

The final step is to transpose four-dimension data matrix and apply one-hot encoding to the training and test labels. For most classification problems "one-hot vectors" are used. A one-hot vector is a vector that contains a single element equal to 1 and the rest of the elements equal to 0. In this case, the nth digit is represented as a zero vector with 1 in the nth position.

5. Algorithm used:

I have chosen to use convolutional neural network(deep learning framework via Tensorflow) for this project.

Implementation

A convolutional neural network is a type of feed-forward artificial neural network in which the connectivity pattern between its neurons is inspired by the organization of the animal visual cortex. Convolutional networks were inspired by biological processes [7] and are variations of multilayer perceptron designed to use minimal amounts of preprocessing [8].

Convolutional neural networks are made out of three parts, namely local receptive fields, shared weights/biases, and pooling.

General Overview of CNN:
Part 1: Local Receptive Fields

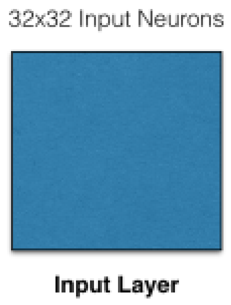


Figure 12: SVHN 32x32 input images

In MLPs, the inputs were depicted as a vertical line of neurons. In a convolutional net, it'll help to think instead of the inputs as a 32×32 square of neurons, whose values correspond to the 32×32 pixel intensities that will be treated as inputs. Input pixels will be connected to a layer of hidden neurons(but not every input pixel will be connected to every hidden neuron). Instead, each neurons in the first hidden layer will be connected to a small (localized region) of the input neurons for example, a 5×5 region.

That region in the input image is known as local receptive field for the hidden neuron. Each connection learns a weight and the hidden neuron learns an overall bias.

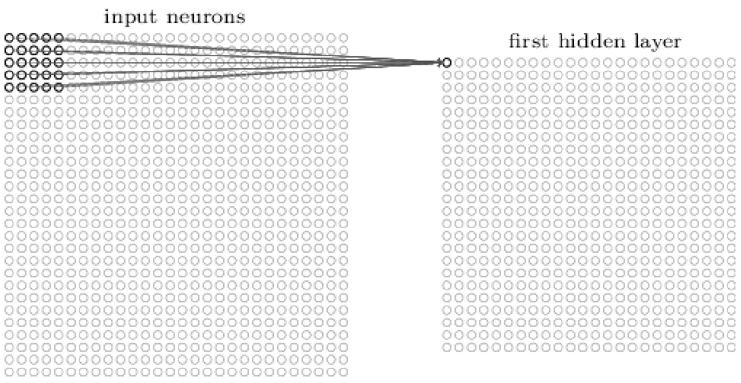
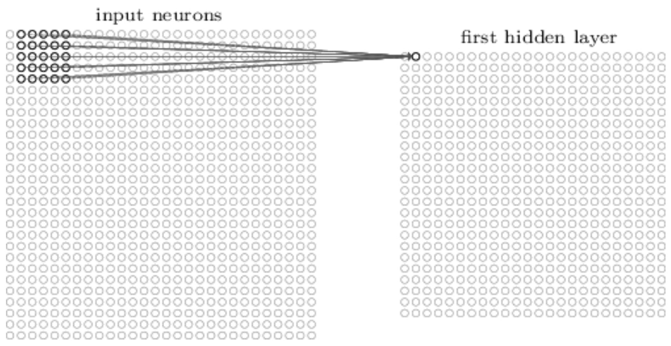


Figure 13: 5x5 patch size filter

Once the neuron connections have been made, the local receptive field will be slide across the entire input image. For each local receptive field, there is a different hidden neuron in the first hidden layer.



Filter 14: 5X5 patch size filter with stride of 1

Next, the local receptive field will be slide over by one pixel to the right in order to connect to a second hidden neuron and so on(building up the first hidden layer). Since SVHN datasets consist of 32x32 input images, and 5x5 local receptive fields are used, then there will be a total 28x28 neurons in the hidden layer. This is due to the fact that local receptive field can only be moved 27 neurons across and 27 neurons down, before colliding with the right-hand side (or bottom) of the input image.

Part 2: Shared Weights/Biases

$$\sigma\left(b + \sum_{l=0}^{4}\sum_{m=0}^{4} w_{l,m} a_{j+l,k+m}\right)$$

Figure 15: Shared weight and biases formula

Each hidden neuron has a bias and 5×5 weights connected to its local receptive field. The same weights and bias will be implemented on each of the 27×27 hidden neurons. In other words, for the j,kth hidden neuron, the output is shown in the equation above. (σ) is known as the neural activation function, (b) is the shared value for the bias, (wl,m) is a 5x5 array of shared weights (ax,y) will be used to denote the input activation at position (x,y).

Shared weights and biases act as filter.As a result neurons in the first hidden layer will be able to identify exactly the same feature(for instance edge of an image or a certain shape) at different locations in the input SVHN images. It is important to apply the same feature detection everywhere in the image. For an example, if the image of a "1" is moved by a little, convolutional neural networks will be able to identify it as an image of a "1".

<img src="diagram_1.png" style="height: 200px;" style="width: 200px;">
In order to perform image recognition, convolutional neural network requires more than one feature map. Based on the example shown, there are 3 feature maps. Each feature map is defined by a set of 5×5 shared weights, and a single shared bias. The result is that the network can detect 3 different kinds of features, with each feature being detectable across the entire image.

Part 3: Pooling
In general, pooling layers are utilized immediately after convolutional layers. The purpose of the pooling layers is to convert the output from the convolutional layer into a simplified form(down sampling). The process of pooling usually involves summarizing a region of neurones in the previous layer. It takes each feature map output from the convolutional layer and prepare a condensed feature map. Pooling usually consumes small disjoint portions of the image (typically 2x2) and aggregates them into a single value. To date, there are several

possible schemes for the aggregation- the most popular being max-pooling, where the maximum pixel value within each chunk is taken.
Max-pooling to will be applied to each feature map separately.

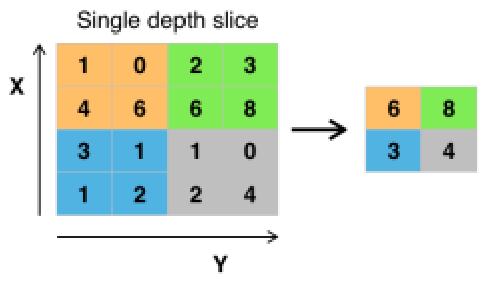


Figure 16: Max-pool diagram

To put it in layman's terms, max-pooling is equivalent to a scanner that checks for certain features in a region of the image and then gives out the exact positional information. Once a feature has been found, its exact location isn't as important as its rough location relative to other features. The advantage is that there are many fewer pooled features, and so this helps reduce the number of parameters needed in later layers.

Final CNN Design:
This project makes use of the CNN module of Tensorflow. The final CNN design will be explained below:

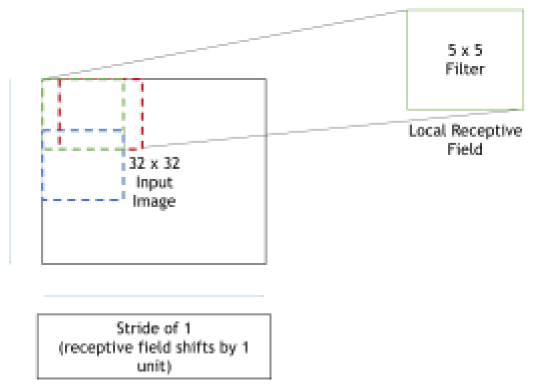
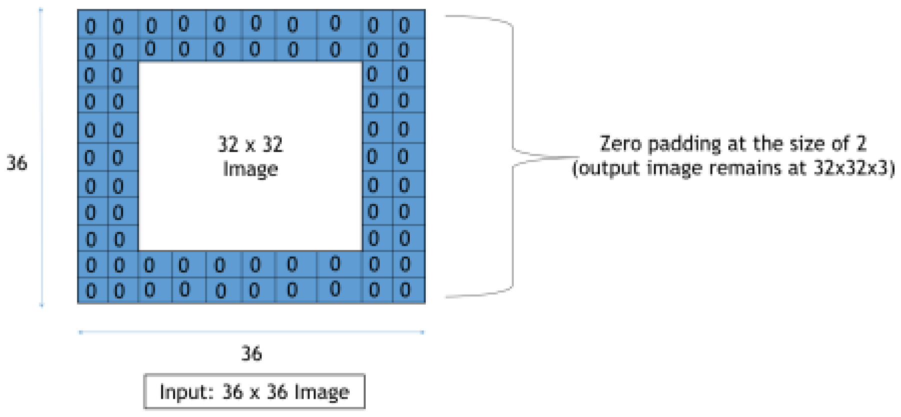


Figure 17: Mechanism of filter striding

Referring to the diagram above, the input 32x32 images will be injected into the first convolutional layer (CONV 1). Convolution layer is used to get the features of the data. In the case of digit recognition - a shape of each digit. It uses learnable kernels/filters each of which corresponds to one particular shape pattern. The convolution computes 32 features for each 5x5 patch. Its weight tensor has a shape of [5, 5, 3, 32]. The first two dimensions are the patch size, the next is the number of input channels (3 means that images are in RGB format), and the last is the number of output channels. There is also a bias vector with a component for each output channel. To apply the layer, we reshape the input data to a four-dimensional tensor, with the first dimension corresponding to the number of images, second and third - to image width and height, and the final dimension - to the number of colour channels. We also implemented stride of 1 where the receptive field is shifted by 1 units.


36
32 x 32
Image
Zero padding at the size of 2
(output image remains at 32x32x3)
36
Input: 36 x 36 Image
Figure 18: Zero padding of size 2

As we keep applying convolutional layers, the size of the volume decrease faster than we would like. In the early layers of our network, we want to preserve as much information as possible(about the original input volume). In order to do so, we apply zero padding of size 2 to the layer. Zero padding pads the input volume with zeros around the border. This will result in a 36x36x3 input volume.

Next comes a rectified linear unit(ReLu 1) which combines the first convolution output and the first layer biases. The output dimension remains the same 32x32x32.

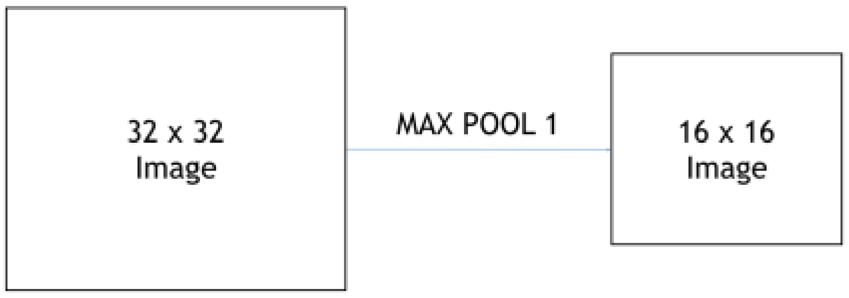


Figure 19: Max pool 1 down sample to 16x16 images

After that, the output of ReLu 1 will be passed to the first max pooling layer(MAX POOL 1). Max pooling reduces the dimensionality of the images. As a result, the output becomes 16x16x32.

The second layer has 64 features for each 5x5 patch. Its weight tensor has a shape of [5, 5, 32, 64]. The first two dimensions are the patch size, the next is the number of input channels (32 channels correspond to 32 featured that we got from previous convolutional layer), and the last is the number of output channels. There is also a bias vector with a component for each output channel.

Because the image is down-sampled by pooling to 16x16 size second convolutional layer picks up more general characteristics of the images. Filters cover more space of the picture. Therefore, it is adjusted for more generic features while the first layer finds smaller details.

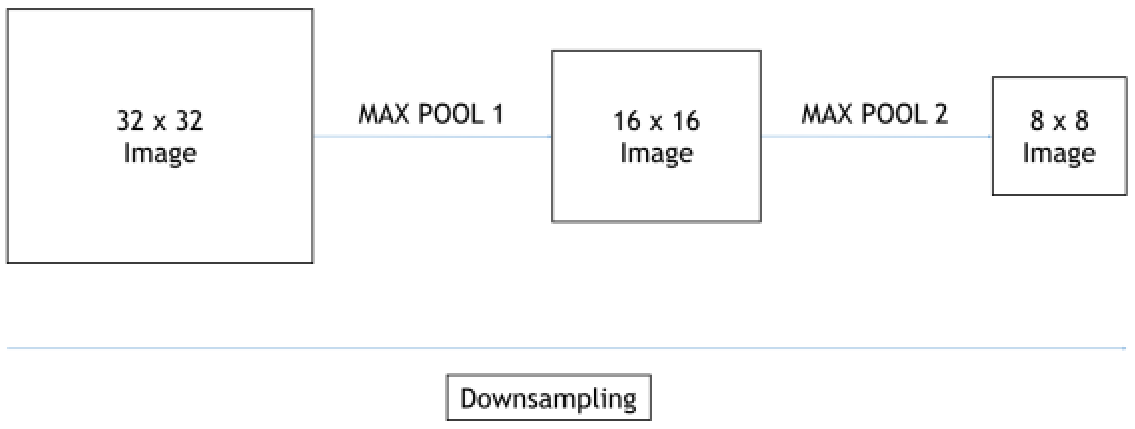


Figure 20: Max pool 2 down sample to 8x8 images

Now that the image size is reduced to 8x8, the next layer converts the input into shape (128, 10). We add a fully-connected layer to allow processing on the entire image(each of the neurones of the fully connected layer is connected to all the activations/outputs of the previous layer).

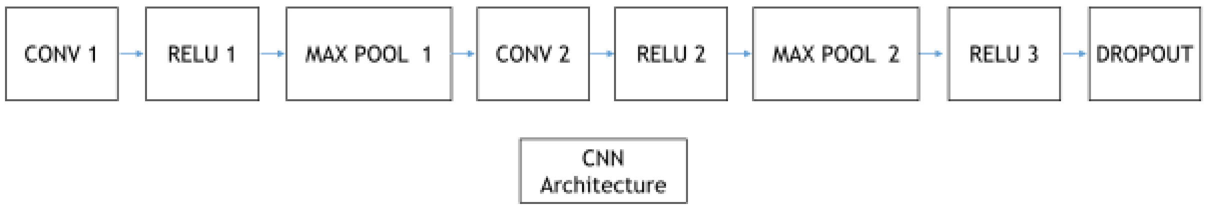
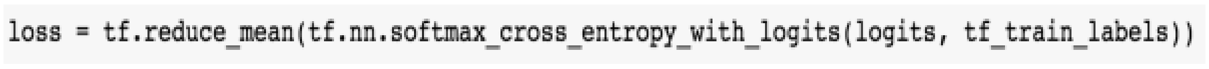


Figure 21: 4 layer CNN model

To prevent overfitting, dropout was applied. Dropout removes some nodes from the network at each training stage. Lastly, the output will be passed to the classifier.

Softmax cross entropies are calculated and reduced for the model against the labels. using the following function:



```
loss = tf.reduce_mean(tf.nn.softmax_cross_entropy_with_logits(logits, tf_train_labels))
```

Figure 22: Cross entropy formula

Original implementation:

1. The CNN model starts with the first convolutional layer(CONV 1) with the shape of 5x5x3x32. The layer utilized 5x5 filters(zero padding at the size of 2) and the stride is set to 1.

2. Next, the first relative linear unit(RELU 1) that combines the CONV 1 output and the first layer biases. This will result in output dimension of 32x32x32.

3. Later, the output of RELU 1 will be down sampled via MAX POOL 1. As a result, the output dimension of MAX POOL 1 becomes 16x16x32.

4. The next convolutional layer(CONV 2) receives an input dimension of 16x16x32. The filters used on CONV 2 IS 5x5x32x64.

5. The output of CONV 2 will be combined with the second layer biases in the RELU 2 layer.

6. MAX POOL 2 layer is then applied to get an output dimension of 8x8x64.

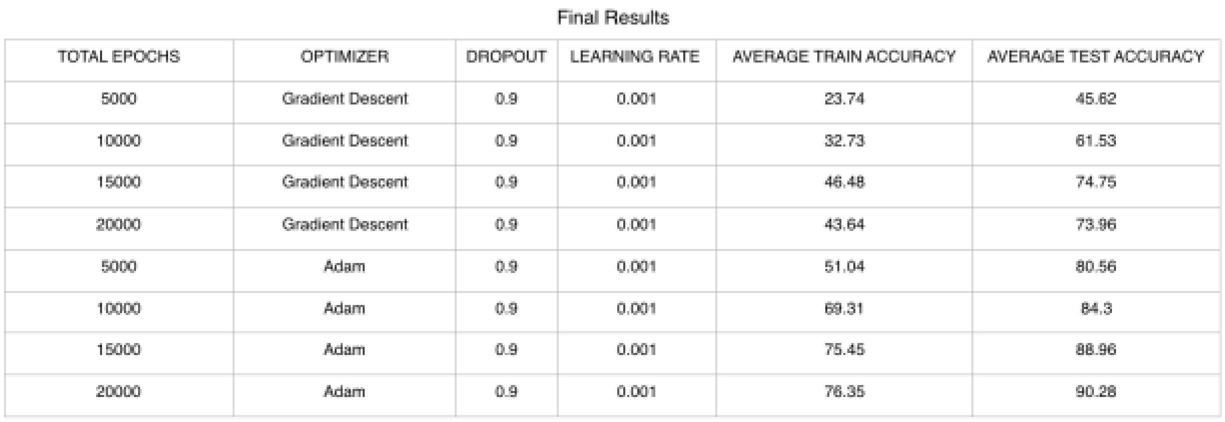
7. Furthermore, the next layer converts the input shape to (128, 10) which acts as the fully connected layer which produces the matrix to be fed to the model.

8. Lastly, the dropout layer was added in order to prevent overfitting from occuring.

9. CNN model consists of 4 layers in total.

10. Initially, it was trained with 5000 epochs and gradient descent optimizer was used to minimise loss.

Refinement



Final Results

TOTAL EPOCHS	OPTIMIZER	DROPOUT	LEARNING RATE	AVERAGE TRAIN ACCURACY	AVERAGE TEST ACCURACY
5000	Gradient Descent	0.9	0.001	23.74	45.62
10000	Gradient Descent	0.9	0.001	32.73	61.53
15000	Gradient Descent	0.9	0.001	46.48	74.75
20000	Gradient Descent	0.9	0.001	43.64	73.96
5000	Adam	0.9	0.001	51.04	80.56
10000	Adam	0.9	0.001	69.31	84.3
15000	Adam	0.9	0.001	75.45	88.96
20000	Adam	0.9	0.001	76.35	90.28

Figure 23: Final results for CNN model

According to my research and analysis, I have made the following adjustment in order to improve the model:

1. I have increased the total number epochs to 10,000, 15,000 and 20,000. It was shown that as the total number of epochs increases, both train and test accuracy increases as well.
2. Adam optimizer have been used for stochastic optimization. Adam offers several advantages over the simple gradient descent optimizer. Foremost is that it uses moving averages of the parameters (momentum); Bengio discusses the reasons for why this is beneficial in Section 3.1.1 of this paper [9]. Simply put, this enables Adam to use a larger effective step size, and the algorithm will converge to this step size without fine tuning. The main down side of the algorithm is that Adam requires more computation to be performed for each parameter in each training step (to maintain the moving averages and variance, and calculate the scaled gradient); and more state to be retained for each parameter (approximately tripling the size of the model to store the average and variance for each parameter). Adam optimizer is able to achieve an accuracy of 90.28%.

IV. Results

Model Evaluation and Validation

In general, the final model is able to achieve accuracy of 90.284% on the test set, which is above the benchmark set for this project. The model is tested over a wide range of epochs(ranging from 5,000 to 20,000). Note that when the total number of epochs increases, both train and test accuracy increases as well (this applies to both gradient descent and Adam optimizer).

When training a CNN, each unit is dropout with certain probability (0.9 for data layers) at each epochs [10]. Dropout is set to 0.9 in order to ensure that the model is robust and able to generalize well.

When the optimizer was switched from simple gradient descent to Adam, the test accuracy had a huge boost. The main difference between Adam optimizer and gradient descent optimer is momentum. Momentum can give faster convergence. For an example, let us imagine an objective function that is shaped like a long canyon that is shaped like a bowl. If we want to minimize the function via gradient descent, we need to start from certain point on the canyon wall. In general, negative gradient will point in the direction of steepest descent. This is mainly because the canyon walls are steeper than the gradual slope towards the minimum.

I have set the learning rate to 0.001(which is small this project and I do not want to overshoot the direction), we could descend to the canyon floor and toward the minimum. Unfortunately, the progress would be extremely slow. The learning rate can be increase in order to solve this problem but this would not change the direction of the steps. As a result, we will overshoot the canyon floor and end up on the opposite wall. The process of jumping from wall to wall causes slow progress toward the minimum. This is where Adam optimizer's momentum comes in handy.

Momentum utilizes some fraction of the previous update and add them to the current update. Repeat updates in a particular direction compound. As a result, momentum is build up, proceeding faster in that direction. By using back the canyon example, we will be able to build up momentum in the direction of the minimum since all updates have a component in that direct. In contrast, jumping from wall to wall involves constantly reversing direction hence momentum would help to decrease the oscillations in those directions.

Justification

The model performs well above the set benchmark of 90%. According to my analysis and observation, it is expected that the accuracy will increase further with longer training duration. The objective of the project is to deploy automatic bank-check digit recognition. Studies show that the accuracy of human operators who are involved in tagging the SVHN dataset was around 98%. In my opinion, this CNN model still have a long way from being used to replace human operators.

I strongly believe that with the appropriate hardware(powerful PCs), I will be able to tweak the model further and enhance its accuracy. By increasing the total number of epochs, it can be a robust model that can be deployed in the real world.

V. Conclusion

Free-Form Visualization

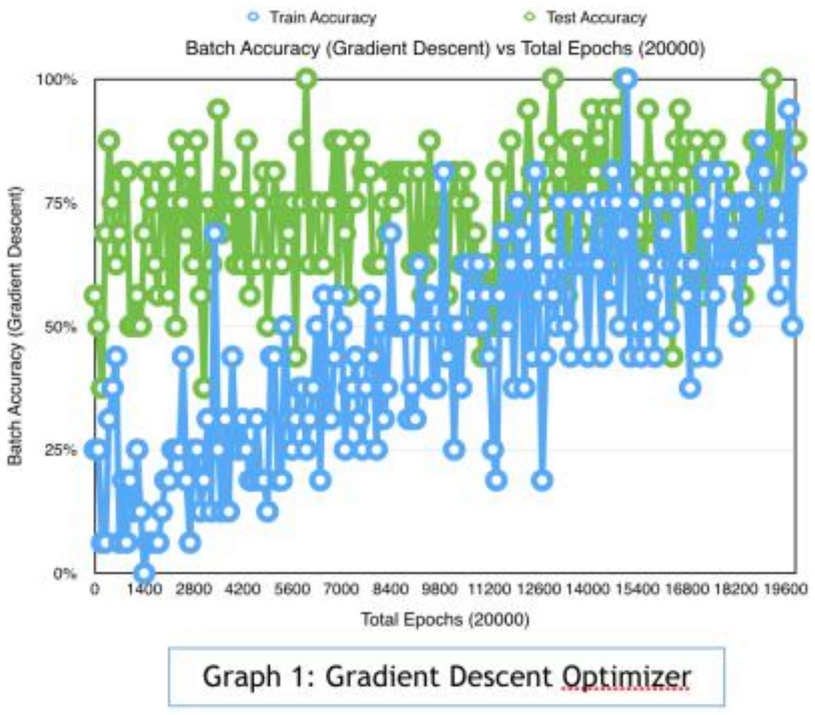


Figure 24: Graph of batch accuracy (gradient descent) vs total epochs (20000)

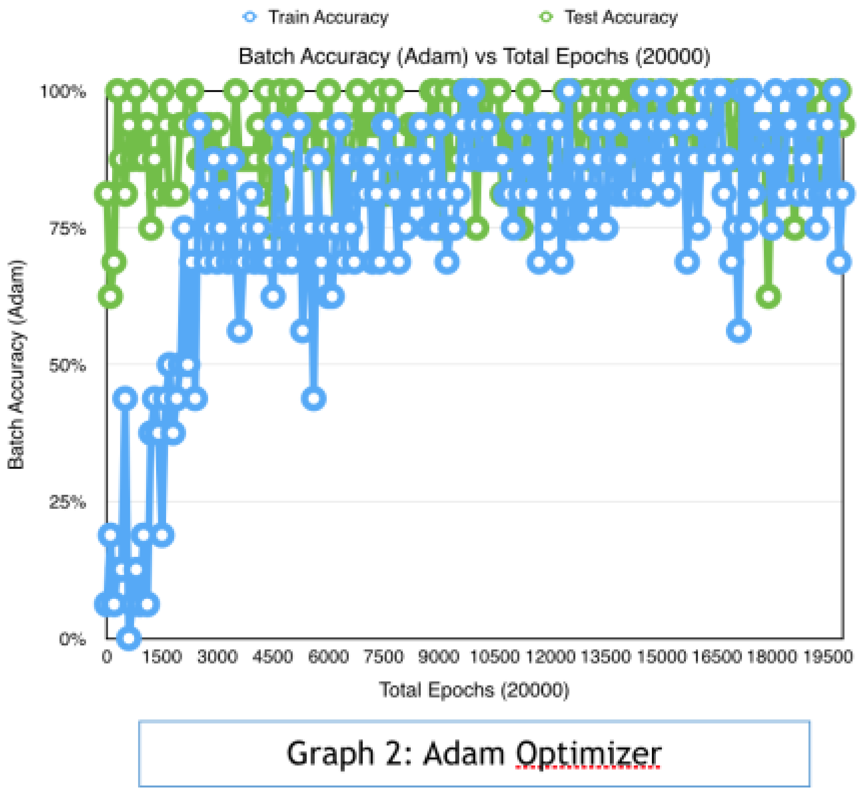


Figure 25: Graph of batch accuracy (Adam) vs total epochs (20000)

Referring to the graphs above, when the total number of epochs increases, the batch accuracy for both train and test sets increases as well. Figure 24 uses gradient descent optimizer whereas figure 25 uses Adam optimizer. Adam optimizer is more suited for this project as it yields a much higher accuracy when compared to gradient descent optimizer. Referring to graph 2, batch accuracy for most train and test sets stays within 75% - 100% range. In contrast, batch accuracy for most train and test sets in graph 1 fluctuates a lot. This shows that gradient descent optimizer is not well suited for CNN model as it requires more epochs in order to match the accuracy that was achieved by Adam optimizer.

Adam optimizer utilizes momentum. Momentum simply means that some fraction of the previous update is added to the current update, so that repeat updates in a particular direction compound. As a result, Adam optimizer is able to converge faster to global minimum than gradient descent. Therefore, resulting in a higher accuracy via 20000 epochs.

Reflection

In general, identifying low resolution Street View House Numbers images is not an easy task. There are several images(extremely blur) that cannot be identified even by our eyes. This is where Convolutional Neural Network(Deep Learning) comes in handy. I have to admit that designing a CNN model is not an easy task as it involves tuning various hyper parameters. In theory, there are infinite combinations of hyper parameters. Apart from that, I have to balance between accuracy and computation time. CNN is an expensive model to begin with as it requires a lot of processing power to be deployed in real life.

One difficult aspect of this project was to choose the appropriate CNN design for the problem. Since there are various ways to design CNN model, it is very hard to understand why a particular CNN design will work best for particular type of data. The CNN model that was is simple and robust. However, more tuning has to be done in order to make the model perform as good as the human operators.

Deep learning is an exciting field of machine learning and can be applied to many exciting problems. While my laptop can handle the total number of epochs up to 20,000, anything beyond that is really slow. Graphics cards are becoming more affordable these days and I strongly believe that if I can get access to one of these GPUs, I can enhance the model further.

Last but not least, please note that I have simplified the problem by not considering multi-digit images. In order to deploy this model for the real world, multi-digit images must be taken into account since real world street numbers contain multi digits.

In the nutshell, this project thought me to familiarize with Tensorflow's deep learning framework as well. Tensorflow is relatively much 'lower level' as compared to Scikit-learn. The Kaggle forum and Slack Machine Learning Nanodegree channel are really helpful as many of this ideas comes from the project comes from them. In the nutshell, this project is fun!

Improvement

Improvement to the current CNN model:

1. Finding the best filter depth and hyper parameter tuning:

   - Correct combinations of filter depth and hyper parameter are crucial to developing the optimum CNN. I strongly believe that if I had tried more combinations of filter depth and hyper parameter, I would be able to enhance the accuracy.

2. Computational power:

   - Due to lack of powerful machine, the total number of epochs that I have tried is limited to 20,000 epochs (anything that is more than 20,000 will take extremely long to process). If I have access to powerful computer, the accuracy of the CNN model can be increased. Based on the results that we have obtained, when the total number of epochs increases, the accuracy increases.

Reference

[1] E. Omidiora, I. Adeyanju and O. Fenwa (2013), "Comparison of Machine Learning Classifiers for Recognition of Online and Offline Handwritten Digits", Department of Computer Science and Engineering, Ladoke Akintola University of Technology, Vol.4, No.13.

[2] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. "Gradient-based learning applied to document recognition." Proceedings of the IEEE, 86(11):2278-2324, November 1998.

[3] LeCun, Yann. "LeNet-5, convolutional neural networks". Retrieved 16 November 2013.

[4] Kingma, Lei Ba. "Adam: A method for stochastic optimization". Retrieved 23 July 2015

[5] LeCun, Yann; Corinna Cortes; Christopher J.C. Burges. "MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges". Retrieved 17 August 2013.

[6] Ranzato, Marc’Aurelio; Christopher Poultney; Sumit Chopra; Yann LeCun (2006). "Efficient Learning of Sparse Representations with an Energy-Based Model". Advances in Neural Information Processing Systems. 19: 1137–1144. Retrieved 20 September 2013.

[7] Matusugu, Masakazu; Katsuhiko Mori; Yusuke Mitari; Yuji Kaneda (2003). "vSubject independent facial expression recognition with robust face detection using a convolutional neural network". Neural Networks. 16 (5): 555–559. Retrieved 17 November 2013.

[8] LeCun, Yann. "LeNet-5, convolutional neural networks". Retrieved 16 November 2013.

[9] Bengio, Yoshua. "Practical recommendations for gradient-based training of deep architectures". Retrieved 16 September 2012

[10] Tieniu, Xuelong, Xilin, Jie, Jian, Hong. "Pattern Recognition: 7th Chinese Conference, CCPR 2016, Chengdu". Retrieved 5 November 2016