**PROJECT REPORT**

**Image Processing with Machine Learning(DA 526)**

**Street View Housing Number Recognition**

**Submitted By**

Diyora Ujas Vinodkumar (224161002)

Manoj Kumar (224161005)

Abhishek Khajuria (224161011)

Swateya Gupta(224357008)

Sarwan Dubey (224161012)

****

**IIT Guwahati**

**CHAPTER 1**

**PROBLEM STATEMENT**

The Street View House Numbers (SVHN) dataset is widely recognized as one of the leading benchmarks for object recognition tasks. It consists of images extracted from Google Street View, specifically capturing house numbers. The dataset shares a similar philosophy to the MNIST dataset, focusing on digit recognition. However, the SVHN dataset tackles a more challenging problem, as it involves recognizing digits and numbers within natural scene images.

This particular task is a supervised classification problem. Given an image containing a house number, the objective is to predict a sequence of digits. The difficulty of this task surpasses that of classifying digits in the MNIST dataset due to two main reasons. Firstly, the SVHN dataset comprises images from real-world scenes, introducing various factors such as perspective, lighting conditions, and the presence of other objects that can potentially cause distractions. Secondly, for a house number prediction to be deemed accurate, all the digits within the predicted sequence must match the target label.

Typically, house numbers range from 1 to 99999, with the majority falling within the 2-4 digit range within the SVHN dataset. Given the dataset's size and the distribution of house numbers' lengths, it would be impractical to define 99999 classes for this problem. Instead, we will assume that house numbers range from 1 to 5 digits in length and define 11 classes for each digit. The classes [0-9] represent the digit values [0-9], while class 10 represents "N/A" (not applicable). Thus, the final model will take an image containing a house number with 1-5 digits and output a sequence of 1-5 digits as the prediction. We will develop this model using both a Multi-layer Perceptron and a Deep Convolutional Neural Network architecture.

By addressing this challenging task and leveraging the power of neural networks, we aim to accurately predict house numbers in real-world scene images.

**CHAPTER 2**

**DATASET INFORMATION**

**2.1 ABOUT SVHN DATASET**

The SVHN dataset is a dataset of about 600k street numbers. It has 73257 digits for training, 26032 digits for testing and 531131 additional and somewhat less difficult samples, to use as a extra training data. It has 11 classes, for 0 to 9 digits , labels are from 0 to 9 and label 10 for no digit. This dataset comes into 2 formats.

1. The original, variable-resolution coloured house-number images with character level bounding boxes. It has 3 files, train.tar.gz, test.tar.gz and extra.tar.gz. The bounding box information is stored in digitStruct.mat file. Each tar.gz file contains the original images in png format, together with a digitStruct.mat file. The digitStruct.mat file contains a struct called digitStruct with the same length as the number of original images. We have used this format for out project.

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 of 300th image.

1. The cropped digits (32x32 pixels) which follow the philosophy of the MNIST dataset more closely, but also contain some distracting digits to the sides of the digit of interest.



*Fig: Images as per format 1*

**CHAPTER 3**

**LITERATURE REVIEW**

**3.1 RELATED WORKS**

Multi-digit-recognition is generally approached in two different ways. Traditionally, the localization, segmentation and recognition steps are separated. This strategy is well- researched and is essentially identical to the classification task completed on the MNIST dataset. This machine learning task has been accomplished via various techniques such as multinomial logistic regression, multi-layer perceptron or standard CNN. Results obtained from MLP were not so accurate in most papers. So they tried different approches than Multi-layer perceptron.

Marking of individual digits would require manual labelling , since automating the digit border identification is itself a complex ML problem. Therefore, the motivation for a model that can succesfully identify multiple digits is high. Some studies have used a single digits classifier and transfer learning to build the final multi digit classification model. This approach has proven less accurate than some of the others that were explored.

Convolutional neural networks [(Fukushima, 1980;](#_bookmark15) [LeCun *et al.*](#_bookmark23), [1998)](#_bookmark23) are neural networks with sets of neurons having tied parameters. Like most neural networks, they contain several filtering layers with each layer applying an affine transformation to the vector input followed by an elementwise non-linearity. In the case of convolutional networks, the affine transformation can be implemented as a discrete convolution rather than a fully general matrix multiplication. This makes convolutional networks computationally efficient, allowing them to scale to large images.

**CHAPTER 4**

**METHODOLOGY**

**4.1 DATA PREPROCESSING**

Firstly, we have extracted tarball files containing data and other resources related to the SVHN dataset. The digitstruct file holds crucial information about labels and bounding boxes, which are necessary for extracting image information and performing predictions with the model. To retrieve this information for each digit in all the images, we utilize the DigitStructWrapper class.

The unpack\_all function is responsible for returning a list of dictionaries. Each dictionary represents an image and includes the filename, along with a list of bounding box information for each digit present in that image. This bounding box information comprises the height and width of the box, as well as two coordinates: top and width, along with their corresponding labels. For example, if "Image 1" in the training dataset contains the number '19', the first dictionary in the list would contain information about the two individual digits, '1' and '9', separately.

Next, we proceed to create bounding boxes around each digit using the top and left coordinates. By determining the minimum and maximum coordinates, we construct a bounding box that encapsulates the entire image of multiple digits.

Subsequently, we create a dataframe that encompasses the following values:

* Label, width, and height of the image
* Top and left coordinates of the image, represented as columns in the dataframe.

In this manner, we construct the dataframe by concatenating the training and testing images.

A picture containing text

Description automatically generated 

*Fig: Data Preprocessing using bounding boxes*

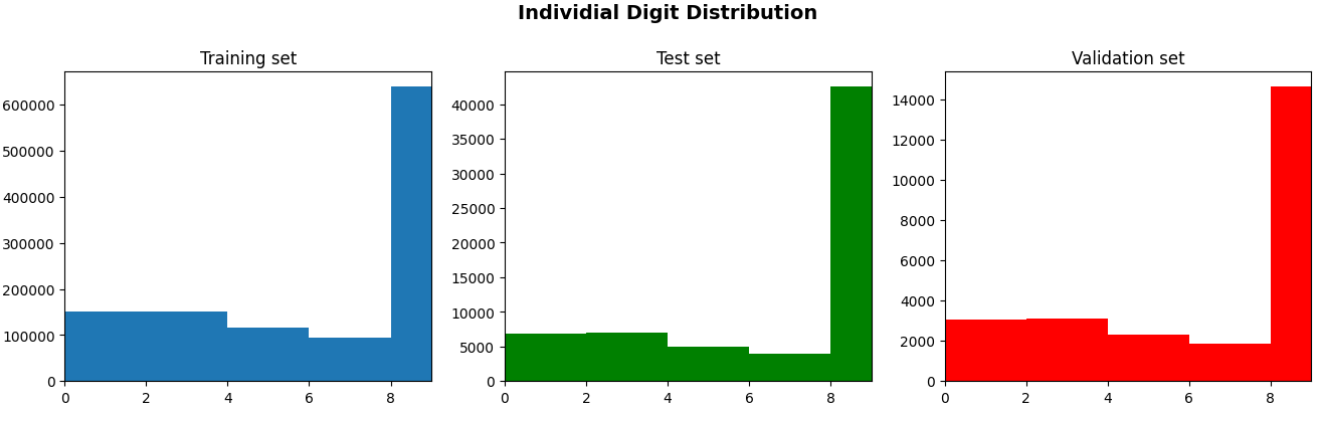
**4.2 DATA TRANSFORMATION**

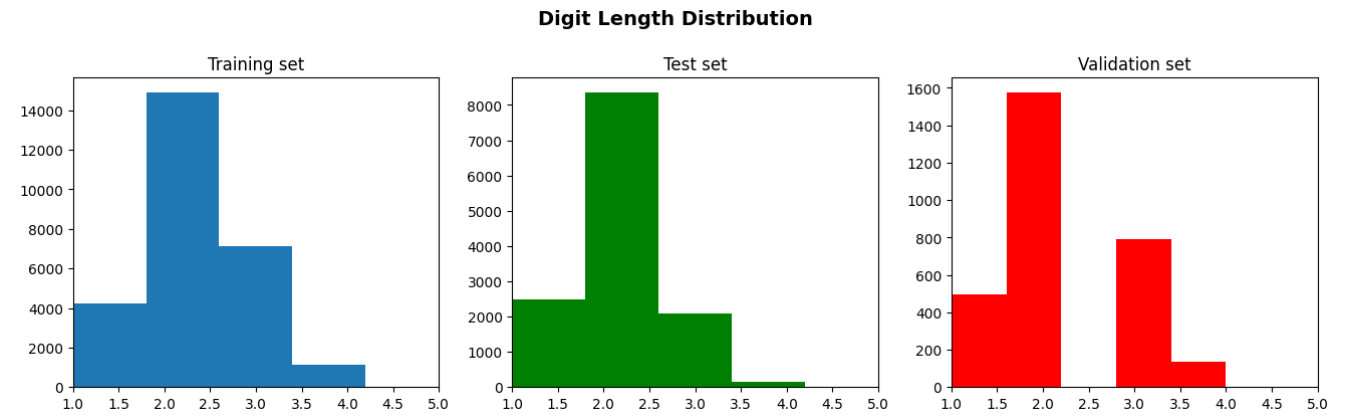
We have preprocessed the data in the following manner:

* We have enlarged the bounding box by 30% and crop the image around it.
* The resulting image is then resized to 64x64 pixels.
* To create a desired format, we have randomly crop a 64x64 image. An example of a cropped image is shown here.



* The data-frame is then divided into a training set and a testing set.
* From the training set, we have created a validation set by randomly selecting samples.
* Finally, we have analysed the distribution of digit lengths and individual digits in the training, testing, and validation sets.(as shown below)





*Fig: Digit distribution*

* 1. **DATA STORING**

The code utilizes an HDF5 file format to store and organize the datasets for the SVHN dataset, including the training, test, and validation data. The use of HDF5 is particularly beneficial in this context due to its efficiency in storing and retrieving large numerical dataset

Given that the SVHN dataset likely consists of a significant number of images and their corresponding labels, utilizing an HDF5 file offers advantages in terms of efficient storage and retrieval. The datasets can be organized and structured within the HDF5 file, facilitating compression and enabling efficient access to the data.

Furthermore, as part of the data preprocessing, the code converts RGB images into grayscale images. This conversion involves taking a weighted sum of the R, G, and B components of each pixel. The resulting grayscale images are then stored within the training, test, and validation datasets within the HDF5 file.

By employing the HDF5 file format, the code ensures efficient management and access to the large numerical datasets of the SVHN dataset. This facilitates subsequent data manipulation, training, and evaluation processes.

**4.4 LOADING DATASET**

After reading the data from the HDF5 file, we have stored it in a variable called "data". From this data, we obtained the final preprocessed dataset. To organize the data for further analysis, we have stored the training dataset, training labels, test dataset, test labels, validation dataset, and validation labels in NumPy arrays.

|  |  |  |
| --- | --- | --- |
| DATASET | SHAPE OF INPUT | SHAPE OF OUTPUT(LABELS) |
| Training set | (27401,64,64,1) | (27401,5) |
| Validation set | (3000,64,64,1) | (3000,5) |
| Test set | (13068,64,64,1) | (13068,5) |

**4.5 ONE-HOT ENCODING DIGIT LABELS AS 2D ARRAYS**

To encode digit labels, each label containing five digits is converted into a 2D NumPy array with a shape of (5, 11). The first dimension (5) represents the number of digits in one image label, while the second dimension (11) represents the number of classes.

For instance, an image with the label "12345" will have a resulting array shape of (5, 11). Each row in the array corresponds to a digit, and the value in each row indicates the presence of the digit in the image. Using one-hot encoding, the digit's column contains a value of 1, while the remaining columns are filled with 0s.

If an image has the label "1234", the resulting array shape will still be (5, 11), but the fifth row will only have a value of 1 in the 10th column index. This represents class 10, indicating the absence of a digit. Classes 0 to 9 represent the digits 0 to 9, respectively.

**4.6 MODEL FITTING**

During model fitting, the models learn from the training data and improve their ability to make accurate predictions. The process involves iteratively presenting batches of training data to the model, computing the output, comparing it with the known labels, and updating the model's parameters based on the calculated error. This iterative optimization aims to find the best set of parameter values that minimize the overall loss or error of the model's predictions.

**4.6.1 MULTI LAYER PERCEPTRON**

Now, for multi layer perceptron input size is required of 2D dimension, so for each dataset, we converted 3d image size into 2d. where each row represents number of images in that particular dataset and column size is number of pixels, that is equal to 64\*64.

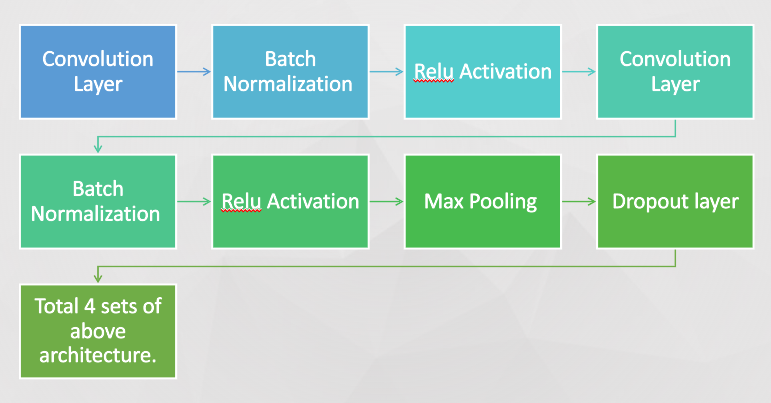
The model architecture is given as below-

* Loss function: Cross entropy
* Output activation function: Softmax
* Optimizer: Adam

We have used Early stopping to monitor loss on validation dataset. So If validation loss is not changing for number of continuous epochs equal to patience, then training will be stopped.

**4.6.2 CONVOLUTIONAL NEURAL NETWORK**

First, we have used 3 different architectures of CNN which is having 8, 12 and 20 layers. So, accuracy is compared for these 3 models. Then we compared model performance under different optimizers like Adam, Stochastic gradient descent with momentum, Adagrad and RMSprop for 20 layers of CNN.



Above architecture is for 20 layers. Convolution, Batch Normalization and Relu forms 1 layer, Pooling and dropout form 2 layers, and then flattening layer, dense layer(2056 neurons), dropout layer and prediction layer, hence total layers will be 20 here. For 8 and 12 layers, there are 1 and 2 sets of above combined layers shown in diagram.

Prediction layer has 5 different sub-layers for predicting 5 digits separately, which helps this network to learn these digits independently. So, we can analyse the accuracy or loss for each output layer to understand how well the model predicts each individual digit.

* Loss function: Cross entropy
* Output activation function: Softmax

**CHAPTER 5**

**EXPERIMENTS AND RESULTS**

* 1. **METRICS**

In this project, there are two important metrics used to evaluate the accuracy of the digit transcriber.

The first metric is the single character recognition accuracy. This metric focuses on how well each individual digit is recognized by the model. Since the model recognizes the digits separately and then combines them into a sequence, it is essential to determine the accuracy of recognizing each digit independently. This metric measures the percentage of correctly recognized single digits in the input.

***Character accuracy = (sum(single digit recognize correctly) / total digits in the dataset) \*100 %***

The second metric is the accuracy of the entire sequence. In some applications, a single mistaken recognition can significantly impact the interpretation of the sequence. Therefore, it is crucial to assess the proportion of input images where both the length of the sequence and every element of the sequence are predicted correctly. This metric calculates the percentage of input images for which the length of the sequence and all its elements are accurately predicted.

***Sequence accuracy = (total image recognized correctly / total images) \*100 %***

These two metrics provide a comprehensive evaluation of the digit transcriber's performance, both at the individual digit level and in the context of the complete sequence.

* 1. **OPTIMIZATION TECHNIQUES USED**

There is a vast variety of optimization techniques that can be used but in our project we have focused on some optimization techniques which are giving more powerful analysis on our problem statement.

1. **STOCHASTIC GRADIENT DESCENT WITH MOMENTUM** : Stochastic Gradient Descent (SGD) with momentum is an optimization algorithm that improves the convergence speed and stability of the training process. It introduces a momentum term that accumulates a fraction of the previous gradients to determine the direction and magnitude of the update at each iteration. This helps accelerate the gradient descent process and enables the model to move more smoothly through parameter space, avoiding potential local minima

**Update rule**: Δw(t) = (μ\*Δw(t−1)) − α\*∇J(w(t−1))

w(t) = w(t−1) + Δw(t)

Where:

* **Δw(t)**​ is the update to the weights at time step t.
* **μ** is the momentum coefficient (typically between 0 and 1).
* **α** is the learning rate.
* **∇J(w(t−1))** is the gradient of the loss function with respect to the weights at time step t-1.

1. **RMS PROP -** Root Mean Square Propagation is an optimization algorithm that adapts the learning rate for each parameter based on the magnitude of recent gradients. It maintains a moving average of squared gradients and divides the learning rate by the square root of this average. By normalizing the learning rate, RMSprop allows for faster convergence and better handling of sparse gradients.

**Update rule**: s(t) = (β\*s(t−1)) + (1−β)\*(∇J(w(t−1)))^2

w(t) = w(t−1) - (α/sqrt(s(t)+ ϵ)\*∇J(w(t−1))

Where:

* **s(t​)** is the moving average of the squared gradients at time step t.
* **β** is the decay rate for the moving average (typically close to 1).
* **ϵ** is a small constant (usually added for numerical stability).

1. **ADAGRAD**: ADAGRAD (Adaptive Gradient) is an optimization algorithm that adapts the learning rate of each parameter based on the historical gradient information. It accumulates the squares of gradients over time, giving larger updates to parameters associated with infrequent features and smaller updates to frequently occurring ones. ADAGRAD performs well in sparse domains but may experience a decaying learning rate over time, leading to premature convergence.

**Update rule**: G(t) = G(t−1) + (∇J(w(t−1)))^2

w(t) = w(t−1) - (α/sqrt(G(t)+ϵ))\*∇J(w(t)−1)

​ Where:

* **G(t)​** is the sum of squared gradients at time step t.
* **ϵ** is a small constant (added for numerical stability).

1. **ADAM**: ADAM (Adaptive Moment Estimation) is an optimization algorithm that combines the benefits of both momentum and adaptive learning rate methods. It maintains exponential moving averages of both gradients and their squares, which are then used to update the parameters. ADAM dynamically adjusts the learning rate for each parameter and incorporates bias correction to account for the initialization bias in the first few iterations. This algorithm performs well in a wide range of problems and is widely used in practice

**Update rule**: m(t)=β1\*m(t−1) + (1−β1)\*∇J(w(t−1))

V(t)=β2\*v(t−1) + (1−β2)\*(∇J(w(t−1)))^2

m’(t)=m(t)/(1−(β1)^t)

​​ v’(t)=v(t)/(1−(β2)^t)

​​ w(t)=w(t−1)−(α/sqrt(v’(t)+ϵ))\*m’(t)

Where:

* **m(t)​** and **v(t)​** are the first and second moment estimates of the gradients at time step t.
* **β1**​ and **β2**​ are the decay rates for the moment estimates (typically close to 1).
* **m’(t)​** and **v’(t)​** are bias-corrected moment estimates.
* **ϵ** is a small constant (added for numerical stability).

These optimization algorithms help improve the training efficiency, convergence speed, and performance of CNN models by adjusting the learning rate, utilizing historical gradients, and incorporating momentum for smoother updates.

* 1. **RESULTS**

**1.MLP –**

Using MLP (Multilayer perceptron) we have done model analysis for single digit and multi digit using number of epochs as ten, optimizer as ‘ADAM’ and three hidden layers. For Single digit image accuracy is 67.99% while 4.55% for multi digit image.

**2.CNN –**

We have used different models to do an analysis on how the accuracy for single digit recognition and sequence digit recognition is improving or decreasing by changing the hyper-parameters using CNN

* Performance with different number of layers in CNN

|  |  |  |  |
| --- | --- | --- | --- |
| Model | No of Layers | Single digit accuracy | Multi digit accuracy |
| Model | 8 | 54.23 | 31.97 |
| Model -2 | 12 | 81.51 | 53.90 |
| Model-3 | 20 | 95.78 | 83.70 |

* Optimizer used in these all above models (simple, 2 and 3) is ‘ADAM’.
* Performance with different optimizers in CNN

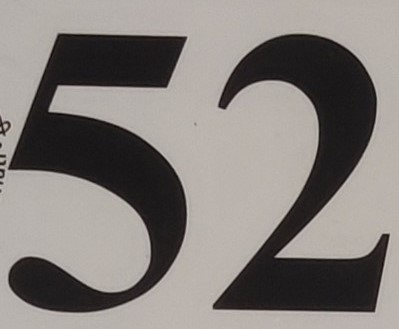
|  |  |  |  |
| --- | --- | --- | --- |
| Model | Optimizers | Single digit accuracy | Multi digit accuracy |
| Model-4 | SGD | 87.13 | 54.19 |
| Model -5 | ADAGRAD | 83.62 | 61.23 |
| Model-6 | RMS PROP | 94.34 | 81.77 |

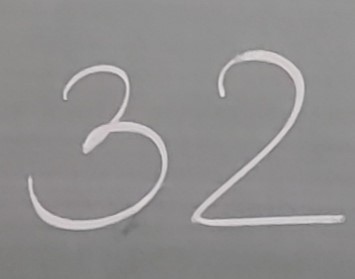
* No of layers used in above models (4, 5 and 6) are 20.

1. **New images**

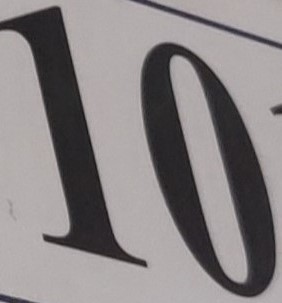
This images are taken by us. We have taken images at all different possible angles. Some of the results are shown here.

Actual image and predicted images are shown below.

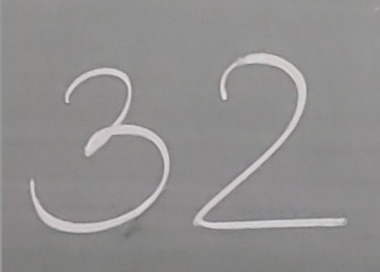
****



** **

** **

****

**CHAPTER 6**

**CONCLUSIONS**

1. CNN outperforms MLP:

Due to large parameter involved and with all dense layers increases the time complexity for MLP while in CNN the specialized architecture, local receptive fields, weight sharing, and translation invariance make it superior to MLPs in this task

2. Performance with different number of layers in CNN:

* Increasing the number of layers improves accuracy for both single digit and multi-digit recognition, which aligns with the conclusion drawn.
* The highest accuracy achieved in results (95.78% for single digit and 83.70% for multi-digit) is consistent with the conclusion that Model-3 with 20 layers performed the best.

3. Performance with different optimizers in CNN:

* According to the results Adam yielded higher accuracy compared to SGD and Adagrad, and RMSprop.