Project 4: Find the Largest Digit - 551-001

Rashik Habib rashik.habib@mail.mcgill.ca
Daniel Lutes daniel.lutes@mail.mcgill.ca
Josh Liu josh.liu@mail.mcgill.ca

I. INTRODUCTION

For this project our aim was to train a model which could accurately classify an image based on the largest (by area) number that occurred on the image. To solve this problem we preprocessed data and then used three different methods to classify the images. The Linear SVC model yielded an accuracy of 20%, the feed-forward Neural Net had a maximum accuracy of 14.6%, and the Convolutional Neural Net which has a maximum accuracy of 92.6%.

II. FEATURE DESIGN

The initial approach to the preprocessing was to take the image and then apply a threshold to the grayscale values such that all pixels with a grayscale value less than 250 would be converted to a 0 and any value equal to or greater than 250 would be counted as a 1. This means that all of the noise of the background images was largely reduced. The binary representation of the image also allowed the image data to be stored as a sparse array. This reduced the size of the training data from 2GB to 14MB.

Our more involved approach worked as follows, given the dataset contained examples of multiple handwritten digits on top of grayscale images, the first step was to extract the digits from the images by keeping the values of pixels above a certain pixel intensity threshold value since the darkest pixels corresponded to the digits. The threshold value of 240 was chosen via trial and error on a few examples such that most pixels relevant to the digits were still present without the background image appearing.

Considering that the labels for each image corresponded to only the digit occupying the largest bounding box, the other digits were considered irrelevant for the learning algorithm and were deemed to be detrimental to its performance. To remove the irrelevant digits, a depth-first search was conducted on each image, so that the maximum length of each connected component (single digit in most cases) in the axes parallel to the image boundaries could be calculated. The largest of these values was used to find the maximum bounding square for each of the digits, and all pixel values apart from that of the largest digit were set to 0, so that the only non-zero pixels were that of the labeled digit.

These semi-preprocessed 50000 images of 64 X 64 pixels each were then cropped around the digit by considering the minimum bounding rectangle around it and rotated such that the longest axis was parallel to the y-axis. The images were also blurred and rescaled to 28 X 28 images (based on expected performance improvements and faster training). These images was expected to resemble the original MNIST dataset, and constituted our feature set used to train the learning algorithms [Fig. 1]. The 10000 test images were identically converted before predicting their labels to ensure the algorithms best performance.

III. ALGORITHMS

1. Linear SVC: For our baseline algorithm we chose to use a Linear SVC model. Despite reducing the training size of the data set by using sparse arrays, the training still took a very long time. It took so long that it wasn't feasible to adjust hyper-parameters. With the pre-processed data on the Linear SVC model we received a score of 20% as our baseline.

2. Neural Network: Our implementation of the neural network used the standard back-propagation algorithm with mini-batch gradient descent (batches of 5000 were used for 10 epochs), and allowed for any number of layers/neurons in each layer. The input was a vector of size 4096, representing each pixel of the 64x64 image. A learning rate was applied to each layer in the back-propagation. The sigmoid function was used as an activation function. The sparse input arrays were used as input, but not the data that we did more filtering on (this is explained in part 3 of this section).

The output was a vector of size 10, where the value at each index represented the probability that the largest digit in the image was equal to that index. For example, [0.1, 0.4, 0.5, 0, 0, 0, 0, 0, 0, 0] represented a 10% chance of the input being a 0, 40% of being a 1, and 50% of being a
2. The error was calculated using square sum of the output vector and a vector of size 10 representing the real answer to the input. For this vector, the value of the index of the real digit was set to 1, and all other values were set to 0. For example, if the real largest digit of an input was 5, the resulting answer vector was [0, 0, 0, 0, 0, 1.0, 0, 0, 0, 0].

The training data was separated into a training and validation set, where 40000 values were in the training set, and 10000 values were in the validation set. For each model attempted, 3 different cross-validation sets were made (the data was first shuffled and then split) and the model was trained on all these training sets. The validation data was

used as a sort of 'testing' set to determine the accuracy of the model, and the three trials were averaged to determine the accuracy of that particular model. The validation set was used as a 'testing' set due to the limited number of attempts we were allowed to make on the test data on Kaggle.

The hyperparameters of the models were the number of hidden layers, the number of neurons in each hidden layer, and the learning rate. For each model of variable neurons (in the table 1), learning rates of 0.01, 0.1, 0.3, 0.5, 0.7, 1, and 3 were attempted. The best result of the different learning rates are only shown in the table 1. An annealing of the learning rate was attempted for some of the earlier models, but this had no effect on the results so a constant value was used instead.

3. Convolutional Neural Network: CNNs are neural networks often applied to visual imagery and have proven to provide high rates of success. This served as the inspiration for applying such a network to our modified MNIST dataset.

During the feedforward stage, CNNs essentially use convolutional layers of pre-determined kernel sizes to attempt to preserve the 2-dimensional relationship between features in images as opposed to the 1-dimensional input of traditional neural networks. The outputs of these convolutional layers are then pooled to reduce the feature dimensions. These steps are often repeated before a fully-connected dense layer is used to compute the relationships between features making use of the weights (to be learnt) associated with the neurons. These logits are transformed using a softmax function and the corresponding class probabilities are calculated, the maximum of which leads to the predicted class label.

Training the CNN depends primarily on the backpropagation stage, where a loss function (such as mean-squared error) is minimized by calculating its derivative with respect to the weights in the fully-connected layers, which are to be learnt. By specifying a batch size, the training data is fed forward in batches, calculating the loss and back propagating through the network to readjust the weights so that this loss is minimized. Traditional optimizers (such as gradient descent) are used to minimize these weights.

IV. METHODOLOGY

The provided 50000 images were shuffled and split into 45000 images for the training set and 5000 for the validation set. By using 90% of the provided data for training, we ensured that we had a large enough data set for the learning algorithms to be trained, while enough data remained to validate our choices for the hyper parameters and architecture.

For the CNN, the architecture was tuned starting from simple single convolutional, pooling and dense layers and made more complex whenever the performance improved on the validation set. Similarly the kernel size was adjusted between 3, 5, 7 and 9 manually for the convolutional layers while the number of neurons in the dense layer was varied between 128, 256, 512 and 1028. Any changes leading to an improvement in validation performance were kept. The stride was kept constant at 2 for the pooling. Adam and Gradient Descent optimizers were compared with learning rates between 0.01, 0.001 and 0.0001 and chosen in a similar fashion. A full grid-search was considered too time-consuming and hence, online sources were consulted to choose the range of hyper parameters for fine-tuning.

The data were loaded as unsigned 8-bit integers or 32-bit integers wherever possible to reduce the size of the arrays loaded or computed, in order to save time.

V. RESULTS

1. Linear SVC: The accuracy of this model was relatively low, when trained on a the semi preprocessed data the model yielded an accuracy of 20%.

2. Neural Network: The hyperparameters and resulting accuracies are listed in the table below. The first column shows the number of layers used, and the number of neurons in each layer (layers are separated by a hyphen and the input and output layers are included). The learning rate displayed is the best rate chosen out of multiple different trials.

Table 1: Number Of Neurons and Layers, with Learning Rate and Accuracy

layers/neurons	learning-rate	accuracy
4096-100-10	0.5	0.1079
4096-500-10	0.7	0.1341
4096-1000-10	0.3	0.1029
4096-100-50-10	0.7	0.1466
4096-100-50-20-10	0.7	0.1187
4096-500-100-10	0.7	0.1385

3. Convolutional Neural Network: Our results from this competition showed the importance of pre-processing methods for feature extraction. When using the initial raw images, the performance of the classifiers was seen to be considerably low, displaying no significant updates in weights, and hence predicting labels as incorrectly as a random classifier (Table 2, Fig 5). The importance of proper pre-processing became more apparent when we decided to crop, rotate, blur and scale the images (Table 2, Fig 5). The considerable differences lead us to believe that a more rigorous pre-processing tactic could lead us to even better performances. Moreover, a deeper CNN seemed to perform better overall, but required careful attention to the architecture and hyper parameters used.

Table 2: Best performance of 2 layer-CNN and 7 layer-CNN with varying preprocessing

	2 Layer CNN	7 Layer CNN
no pre-processing	10.94%	11.60%
semi pre-processing	76.10%	86.71%
full pre-processing	85.58%	92.57%

For the optimizers, although Adam was found to be faster than Gradient Descent in most cases, the best performance was found for Gradient Descent using the combination of hyper parameters that were tried. Too high a value for the learning rate would cause the Gradient Descent optimizer to diverge. Converging values resulted in similar performances (around 90%)

The architecture of the CNN was found to generally improve when the filter and kernel sizes were gradually increased, while a relatively low number of neurons in the fully connected layers was found to be beneficial. The stride for pooling was kept at 2 and a step-size of 50-100 was found to be good for our network. The CNN used for our best performance (Fig 3) of 92.566% was partially inspired by the VGG network for image recognition (Fig 2)

VI. DISCUSSION

A. Areas Of Future Work

Possible issues with the feature design were discovered when the images were observed by eye. Due to certain bright background images, the threshold value lead to unwanted pixels remaining in the images and possibly leading to reductions in classifier performance. Furthermore, the assumption used to rotate the digits fails whenever a digit has greater width than height. Such wide digits were seen to occur rather rarely, but may have led to lower classifier performances.

Spatial Transformer Networks (STN) have proven to be effective methods of feature extraction [1]. Using an STN to learn the required transformations for our modified dataset would allow the images to have a similar appearance throughout, essentially preventing the CNN from having to learn the non-linearities of these affine transformations, which would possibly require a larger dataset. Theoretically, this pre-processing step would allow the learners performance to improve and would avoid some of the possible errors during pre-processing as mentioned earlier.

Further combinations of CNN architectures could be tried out to find ones with better validation performance. Similarly, more hyper parameter combinations could be used, with a full grid search to find the optimal ones if sufficient time and computational resources were available.

B. Conclusion

The CNN is a very powerful classifier for image recog-nition only when the features have been properly pre-processed. The architecture of the network is often dependent on trial and error, but commonly used CNN architectures can prove to be useful inspirations. Deeper networks making clever use of convolutional layers can be very efficient at understanding the 2-dimensional relationships between features, which is a very effective tool in handwritten digit classification.

VII. STATEMENT OF CONTRIBUTIONS

Rashik built the preprocessing which considered the size of the number that was drawn, and he also did all of the work related to the convolutional neural network. Josh build the neural network from scratch. Daniel built the preprocessing algorithm which considered the image as a threshold and stored the data as sparse arrays, he also trained the linear SVC model. The work on the write up was distributed based on the work done on the code.

We hereby state that all the work presented in this report is that of the authors.

VIII. REFERENCES

1. Spatial Transformation Networks: Jaderberg, Max, et al. ”Spatial Transformer Networks.” arXiv preprint arXiv:1506.02025 (2015)
2. Handwritten Digits Recognition using OpenCV: https://www.unilim.fr/pagesperso=vincent:neiger=publications=reportd
3. CS231n: Convolutional Neural Networks for Visual Recognition: http://cs231n.stanford.edu/

IX. APPENDIX

Fig. 1. Fully pre-processed image examples of digits 2, 7, 7, 0 (top-left going clockwise)

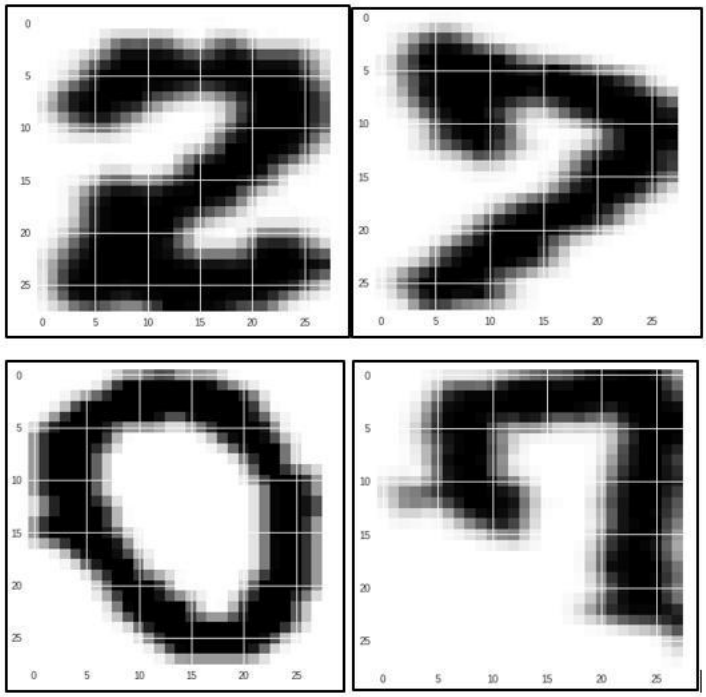

Fig. 2. VGG Neural Network

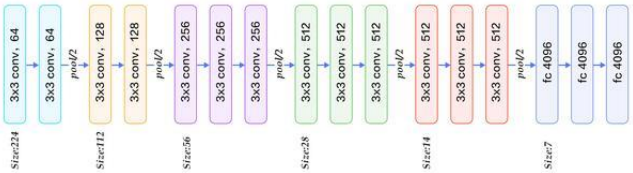

Fig. 3. Architecture of best-performing CNN

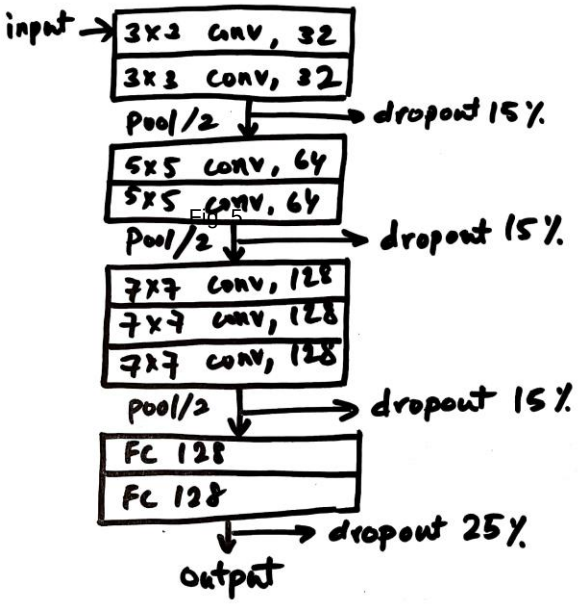

Fig. 5

Loss vs Step for 7-Layer CNN model with varying pre-processing

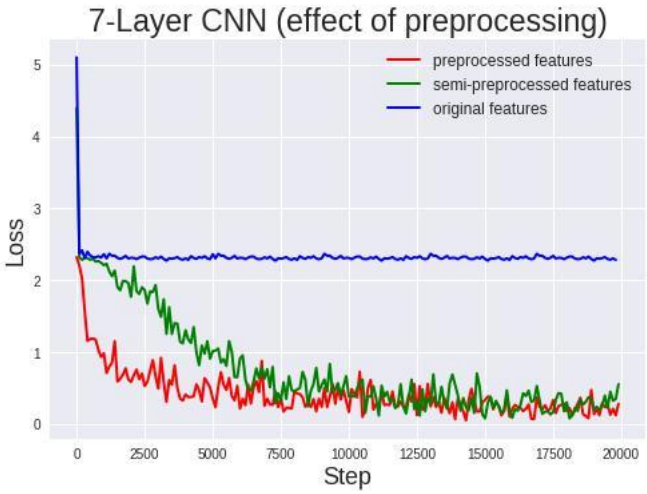

Fig. 4. Loss vs Step for best-performing 7-Layer CNN model