MiniProject 3 Report

Sym Piracha, Mano Salvador, Max Zhang

Multi-Label Classification of Image Data Using Neural Networks

COMP 551
McGill University
Canada
November 27, 2021

1 Abstract

In this project, multi-label image classification of image data was performed using a neural network. The primary task consisted of correctly identifying the digit and letter from the English language present on each input image. This objective was achieved by experimenting with different layers of various size, tuning many hyper-parameters, and observing the effects of various deep learning techniques to achieve a satisfactory model. After much tweaking via trial and error, we were able to achieve a test accuracy of **86.933%** on Kaggle by using several hidden layers and various data augmentation techniques.

2 Introduction

This project's objective was to create and experiment with various neural network model architectures to achieve multi-label image classification of image data with great accuracy. More precisely, our models primarily varied in their number of convolution layers and the techniques used within them such as pooling, kernel size, and the number of in and out channels. The models also differed in their choice of hyper-parameters such as batch size, learning rate for gradient descent, and number of epochs over which the model was trained.

The task of training neural networks to achieve an established goal is a well-defined and well-explored topic in the machine learning community. This article provided a general overview for training our models and gave insight on the topics of regularization and tuning. Since our models used the PyTorch framework, we consulted this piece to gain knowledge on how to leverage the library improve our results. Similarly, this document explained how feature engineering through image augmentation for the purpose of regularization can be used with the PyTorch framework.

3 Datasets

The data provided for this project consists of n = 30,000 labeled samples, n = 30,000 unlabeled samples, and n = 15,000 samples for testing. Initially, only the labeled data was used to train our models as this was the most straightforward approach, so this dataset was split into training (n = 24,000) and validation (n = 6,000) subsets. We later augmented the labeled data by applying simple rotations by 10 degrees to effectively double the size of the training subset to n = 48,000 to achieve a small increase in test accuracy.

We also attempted several other processing techniques on the image data, but these proved to have either no effect or a negative effect on our test accuracy. For example, we followed this guide in an attempt to threshold the data to yield simpler binary images. Unfortunately, this technique had no effect on the results of the model, so we discarded it for the sake of simplicity. We also attempted to apply Gaussian blurring to the images with the goal of reducing noise. Ideally, blurring is applied to remove the noise present in the image so the letter and digit become more identifiable. This did not prove to be useful as the Gaussian blurring ultimately led to a decreased test accuracy.

4 Results

Choosing the optimum parameters and architecture, we achieved an unbiased test accuracy of **86.933%**.

By first visualizing the dataset, we were able to gain more intuition on how to train the model and later augment the data (Figure1). After training the model, visualizing the predictions allowed us to grasp the confusion that the network was facing, e.g. 'i' and 'j', '0' and 'O' (Figure 4). By building a baseline model and pipeline we could then observe any substantial improvements from tuning the neural network. over-fitting the model allowed us to verify that it was actually deep enough to learn and predict the labeled dataset accurately. Then, setting aside a validation dataset, we could apply regularization methods such as dropout, batch normalization, and adjusting depth to increasing validation accuracy and thus better generalization for test accuracy improvement.

Further, applying rotation to the labeled data allowed us to increase the size of our training dataset which subsequently gave an 2% increase in test accuracy. Being given the unlabeled data to work with, we attempted semi-supervised learning but it did not yield favourable results. We elaborate on this topic in the Discussion section. Finally, we chose to use Adam, as though stochastic gradient descent with momentum is used for imageNets for many state of the art neural networks, comparing the two optimizers side to side yielded results in Adam's favour.

All in all, the final performance of our model can be attributed to a number of factors. Adjusting the bias and variance in our model by tuning hyper-parameters and regularization allowed us to avoid over and under-fitting. Normalizing, adding noise, augmenting and faking data gave us a

bigger sample size on which to train our model on, seeing small but significant improvement in accuracy. Other than that, by increasing the training epochs we were able to improve overall performance, as we suspect that the epoch limit of over-fitting is high enough that we do not possess the resources to compute it anyway.

5 Discussion and Conclusion

Throughout this project, we attempted to incorporate many different machine learning techniques to improve our test accuracy. For example, we implemented residual networks (ResNets) to solve the issue of vanishing gradients as explained in this article. We did not choose to use ResNets in our final model however for one main reason. First, since ResNets are pre-trained in PyTorch, their restricted nature inhibited our ability to tinker and fine tune our model.

Furthermore, we also implemented two regularization methods. The first was batch normalization, as this effectively normalized the activations of layers as explained in this article. Unlike ResNets, batch normalization proved to be a useful in regularizing our model to increase the final test accuracy. The second regularization method was dropout. Dropout randomly removes a subset of the neurons during training to reduce over-fitting. This paper explains that dropout allows each unit to detect features without being dependent on other units. In theory, this technique should improve the model's accuracy significantly, and we indeed observed such results.

Moreover, we attempted several different data augmentation techniques with the goal of both increasing the quality and quantity of the labeled input data. The three main approaches we tested were applying Gaussian blurring, thresholding, and transforming the labeled dataset. Initially, we believed that applying Gaussian blurring would remove the Salt and Pepper noise present in the input data while preserving the characters. The nature of the dataset required such a large amount of blurring to remove the noise that the characters became too difficult to identify. As such, we decided to not use Gaussian blurring in our optimal model, although we believe that this idea could be useful in other contexts. Likewise, thresholding can be used to improve the quality of the dataset by setting all pixels with an intensity below a certain value to 0 to obtain simpler and superior binary images. Given the supplied input dataset, this strategy did not improve our results so we once again opted to discard this strategy to maintain the simplicity of our model. Nonetheless, we do believe that thresholding could improve the accuracy of a model in a different task with different data. We also attempted to transform the dataset to increase the number of input samples. Transformations usually consist of translations, rotations, or adding noise to the input. We concluded that rotating the input images proved to be the most useful transformation in this case, so we applied one rotation to effectively double the size of the training data set. We found that random rotation of +/-10 yielded the best results as anything more would cause to great of transformation of a data point (i.e. a 180 degree rotation changes a 6 to a 9).

Unfortunately, we struggled to leverage the unlabeled data to improve our test accuracy. More precisely, our strategy consisted of first training a model on the labeled data, then using this model to predict a label for each data point in the unlabeled set. We then filtered these results to create a predicted label dataset by discarding all predictions where several predictions had a high positive activation for numbers and letters respectively. Next, we retrained the model using the newfound predicted label dataset. This strategy did not prove to be ultimately useful as the test accuracy decreased from **89.933%** to **86.723** in one testing round. We believe this negative effect can be attributed to our initial model, which was trained on just the labeled data, not being sufficiently accurate when making predictions on the unlabeled data. This led to the model being trained on falsely-labeled data. We theorize that a superior initial model could reverse this decrease in performance and would thus take advantage of the unlabeled dataset to increase test accuracy.

Overall, we discovered that a combination of eight convolution layers and three linear layers along with several regularization strategies (batch normalization and dropout) and data augmentation via rotation yielded the best result throughout our trials. This model achieved a test accuracy of **86.933%**.

6 Statement of Contributions

Equal contributions were made to this project overall. For the coding aspect, Sym handled most of the implementation of the model, whereas we all participated in the research and fine tuning to increase the test accuracy. In terms of writing, Mano wrote the Absract, Introduction, Datasets, and Discussion and Conclusion sections, while Max wrote the Results section and also produced the graphs featured in this report.

7 References

Brownlee, Jason. (2019, January 16). A Gentle Introduction to Batch Normalization for Deep Neural Networks. Link.

Data Carpentry. (2021, November 26). Image Processing with Python – Thresholding. Link.

Godoy, Daniel. (2019, May 7). Understanding PyTorch with an example: a step-by-step tutorial . Towards Data Science. Link.

Hinton, Geoffrey E et al. (2012, July 3). Improving neural networks by preventing co-adaptation of feature detectors. Link.

Karpathy, Andrej. (2019, April 25). A Recipe for Training Neural Networks. Andrej Karpathy Blog. Link.

Ruiz, Pablo. (2018, October 8). Understanding and visualizing ResNets. Link.

Sharma, Pulkit. (2019, December 5). Image Augmentation for Deep Learning using PyTorch – Feature Engineering for Images. Link.

8 Appendix

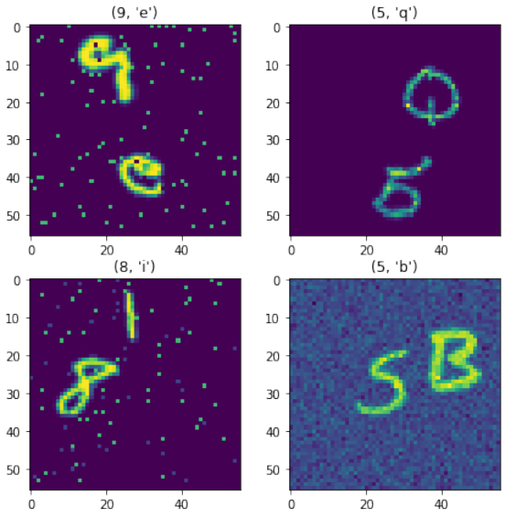

Figure 1: Visualization of the input data. Each image contains one letter from the English alphabet and one digit from 0-9. Some images contain Salt and Pepper noise while others do not.

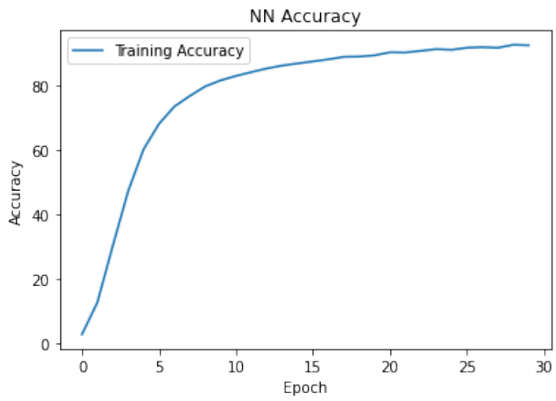

Figure 2: Neural Network training accuracy for different numbers of epochs. The graph demonstrates that the model begins to overfit the data as the number of epochs increases.

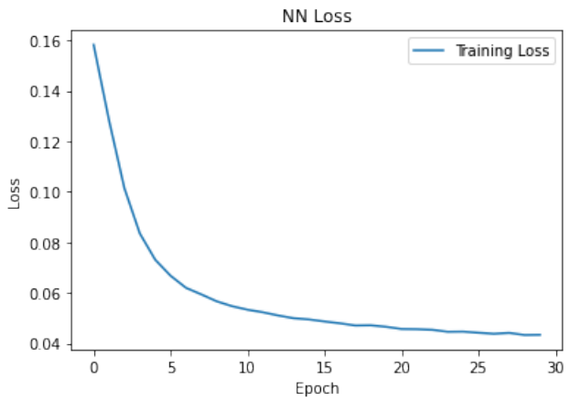

Figure 3: Neural Network training loss for different number of epochs. The graphs shows that the rate of decrease slows dramatically after approximately 20 epochs.

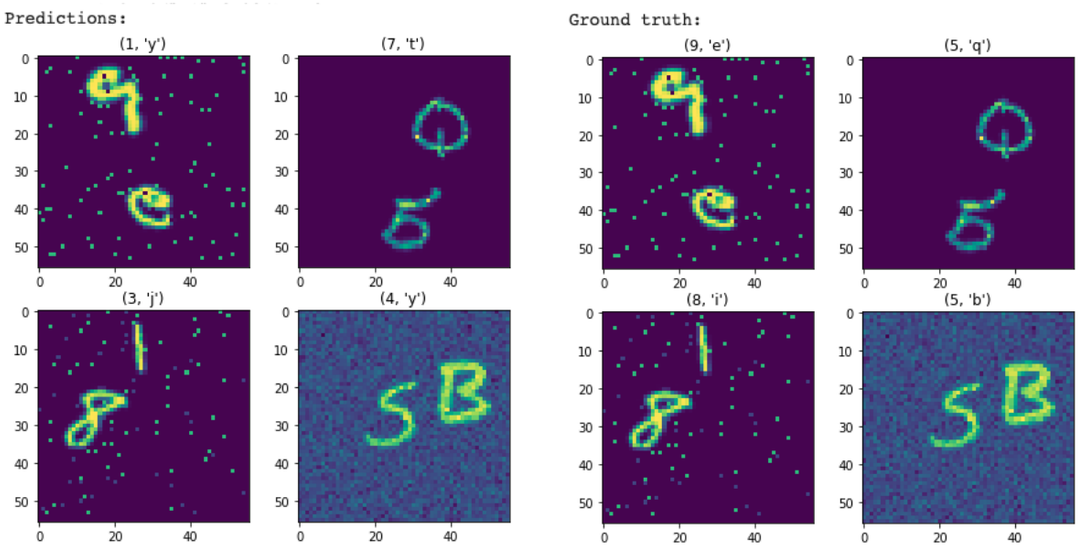

Figure 4: Prediction made by our model versus ground truth of labeled data.