COMP 551 - Machine Learning

Mini Project 3: Image Recognition

Group #13
Emmanuelle Coutu-Nadeau - 260681550
Christian Martel - 260867191
Ragheed Qasmieh - 260780556

McGill University
November 27, 2021

Abstract

The objective of this project was to implement a model to classify different handwritten letter and digit combinations from the MNIST dataset. 30,000 images were labelled and used to train our model. 30,000 unlabelled images were used to perform semi-supervised learning. 15,000 unlabelled images were used to test our implementation and participate in the Kaggle competition. To achieve a decent classification, we decided to use a convolutional neural network based on the pytorch built-in implementation of an AlexNet. In addition, we preprocessed the data by normalizing the input, augmenting the number of images through rotation and translation, and adding noise for robustness. Lastly, we used bagging and cross validation to average the predictions of 10 trained models to help reduce the bias of our model.

The main takeaway from this project is that a combination of multiple preprocessing and training methods resulted in the best accuracy. Every preprocessing method alon did not seem to widely impact the validation accuracy scores. However, combining the preprocessing techniques together helped us go from a 60% to a 88% accuracy in the competition. Moreover, using the unlabelled data in semi-supervised learning and bagging helped us get even better accuracy. As a result, we achieved a 95% accuracy score on the Kaggle competition.

Introduction

The objective of this project was to develop a model to classify handwritten combinations of digits and letters from the MNIST dataset. Out of the 60,000 training images in the dataset, half were labelled. Moreover, 15,000 unlabelled images were kept for testing. Lastly, these 15,000 unlabelled test samples were used to participate in the Kaggle competition where the true labels of these samples were compared to our test predictions.

Methods and Results

1. Different models

Our first attempt to develop a model to classify handwritten combos of digits and letters was to try different neural networks models. To begin with, we started by testing built-in models provided by the pytorch framework. Comparing different models helped us understand how to play with neural network parameters such as learning rate and the effect these parameters have on the accuracy of the model. After testing both Resnet and Alexnet, we realized that the difference in validation accuracy that both models provided was not significantly different. See section "Graphs/Alexnet vs Resnet" of the notebook. Arbitrarily, we decided to improve our Alexnet model customizing it to the project requirements. First, to optimize time and space complexity of the model, we changed the number of input channels from three to one. Also, we modified the output of the model so that the last layer consists of two fully connected layers outputting respectively the digit prediction and the letter prediction.

2. Preprocessing

a. Normalization

To improve the performance of our model, we tried many different preprocessing techniques, starting with normalization. To normalize our data, we simply used a min-max scaler on the images. As can be seen in the "Graphs/Effect of normalizing" section of the notebook, the performance at this point was already decent with an accuracy score of 96% on training data and 82% on validation data after 20 epochs. However, compared to raw data, the difference was insignificant. Since it didn't decrease performance and was not more complex to implement, we continued using normalization.

b. Image segmentation

Another technique that we tried was to segment the digits from the images to resize the letters and digits to make our dataset more uniform. However, this method was quite complex and did not lead to better results than baseline. Segmenting and resizing reduced the bias of our model but increased considerably variance, so it did not perform well on new data. Hence, we discarded this technique.

c. Augmenting the dataset

To augment our training dataset, we applied some transformations on the provided images and added them to the original training set. We used rotation and translation to modify the images. The section "Graph/Augmentation" of the notebook shows the difference in validation accuracy from training a model by doubling the dataset with modified images vs the unprocessed dataset. Furthermore, the "Experiments/Different Levels of Data Augmentation" section of the notebook shows the impact of adding more levels of data augmentation on the validation accuracy for the custom Alexnet. It is observed that the more data we add, the higher is the maximum validation accuracy that we can reach. However, adding more training data reduces space and time complexity efficiency. Out of all the techniques, this one alone showed the biggest improvement with the model trained on raw data with a training accuracy of 97% and a validation accuracy of 91% after only doubling the data. This finding made sense since data augmentation helps to reduce overfitting to a specific type of data.

d. Adding/Removing noise

We noticed that some of the images in the dataset were noisy with high intensity pixels in different random locations. After initial experiments, we realized that this SnP noise would greatly affect our model's ability to correctly predict test images since test data was also partially noisy. Our first approach to tackle this problem was to apply a Gaussian blur kernel of size 3 on the images in all datasets followed by an image thresholder. The blur will first reduce the intensity of noisy pixels since it will take the Gaussian distribution of all the pixels in the 3*3 kernel. Subsequently, the thresholding will remove those pixels. However, we noticed that this denoising method is distorting many images which had their digits and letters overlapped or which had thin edges. We therefore decided to keep the original datasets as is and tackle this issue of noise in data augmentation. In augmentation, we added random SnP noise to the augmented data. This way, all images had noise in their distribution and since it is normally distributed, it would have minimal effect on predictions. This is a way of introducing noise robustness.

3. Pseudo-labelling

Since we had 30,000 images that were unlabelled, we decided to use them to implement a semi supervised algorithm. Indeed, once we were confident that one of our models provided decent accuracy scores on validation and test data, we used it to predict the labels of the 30,000 unlabelled images. For each prediction, we looked at the probability of this prediction. Using softmax confidence thresholds, we selected the labels that were over these thresholds and added them to the training dataset. Lastly, the model was re-trained with the new augmented dataset. By predicting with a 92%-test-accuracy model, the pseudo-labelling technique added around 26000 samples. The pseudo-labelling was then run two more times by predicting the unlabeled data using the previous model. At each run, the number of added samples increased considerably. The section "Experiments/Bagging Cross-Validation with Semi-Supervised Learning" of the notebook shows the average training and validation accuracy of our custom Alexnet model after three runs of semi-supervised learning on the unlabeled data.

4. Bagging

Using different trained models, we used bagging to average the predictions. Bagging is helpful to reduce our model variance. We trained ten models using 10-fold cross-validation and used the average prediction of the 10 models to predict the test set. The section "Experiments/Bagging Cross-Validation" of the notebook shows the average validation accuracy of our custom alexnet model by bagging the models resulting from a 5-fold cross-validation. Bagging increased considerably the performance on the test set since it is reducing the variance of our model.

Discussion and conclusion

By implementing and evaluating different models and preprocessing methods, this research established that there are useful techniques that can help increase accuracy scores in neural networks. We noticed that preprocessing the data had a considerable impact on our accuracy results. More precisely, combinations of normalizing, augmenting the dataset with rotated and translated images, and adding noise to the images increased our accuracy score on test data from 60% to 88% on Kaggle. Moreover, semi-supervised training and bagging greatly improved our model performance in the competition, going from 88% to as much as 95% accuracy. Hence, our best results were achieved using a combination of all our preprocessing techniques and using pseudo-labelling and bagging.

Statement of Contributions: Emmanuelle: Bagging, Pseudolabelling, Report; Christian: Model implementation, Bagging, Report; Ragheed: Normalization, Augmentation, adding noise, Report.