COGS 260 Project 2 Report

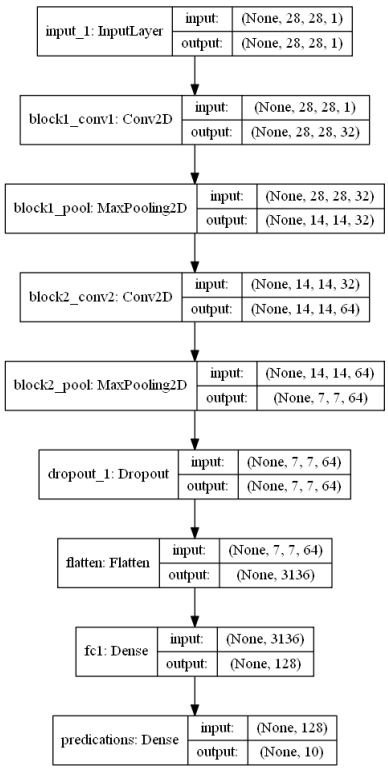
Mengjie Li
mel074@eng.ucsd.edu

Abstract

In this report, I am going to show the results I have tested on MNIST dataset by using Convolutional Neural Networks, 1-Nearest Neighbor, Support Vector Machine and Spatial Pyramid Matching. I will test the same model on different parameters and comopare the resulting running time and accuracy.

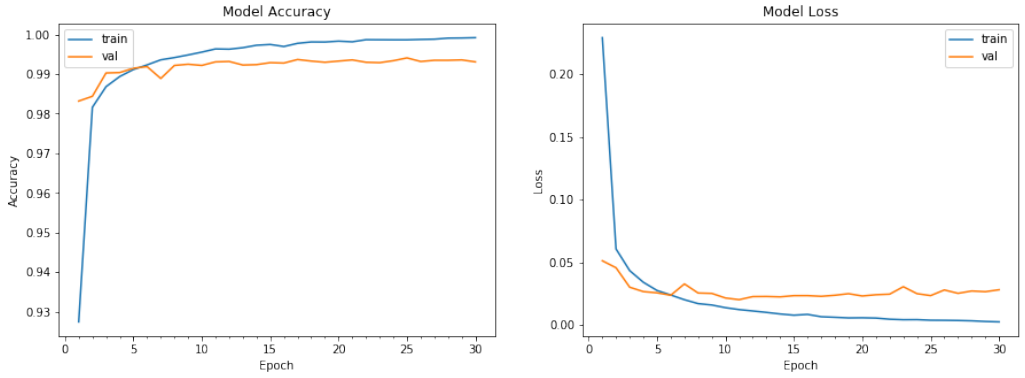
1 Convolutional Neural Network

In this section, I built a convolutional neural network based on LeNet. The overall architecture of LeNet is simiple:
INPUT => CONV => RELU => POOL => CONV => RELU => POOL => FC => RELU => FC
Here in my model, I have set the kernel size to be $3 * 3$ and the stride of maxpooling to be default 1.
Then right before the fully connected layer, I added one layer of dropout to prevent overfitting.
The architecture is shown below:



Submitted to 29th Conference on Neural Information Processing Systems (NIPS 2016). Do not distribute.

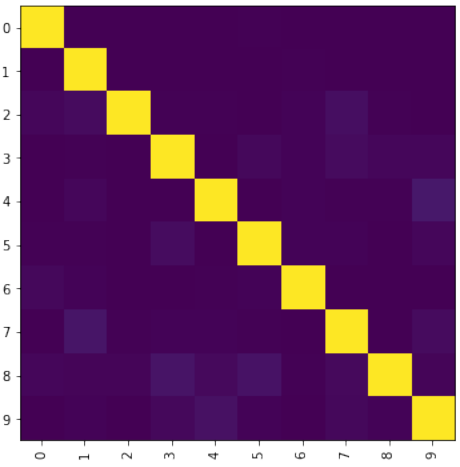
By using GPU, we can easily and efficiently train a neural network.
We know that Neural Network is easily get overfitting, so for such a simple dataset as MNIST, we need to pay attention and be very careful with this problem. To do this, we can stop the algorithm early or add randomness to the dataset (flipping, rotating, scaling, etc.).



As we can see from the plotting, the validation error tends to be stable starting from 5th epoch(10th iteration).
The final accuracy is 0.9931 and the final test loss is 0.0280.

2 1-Nearest Neighbor

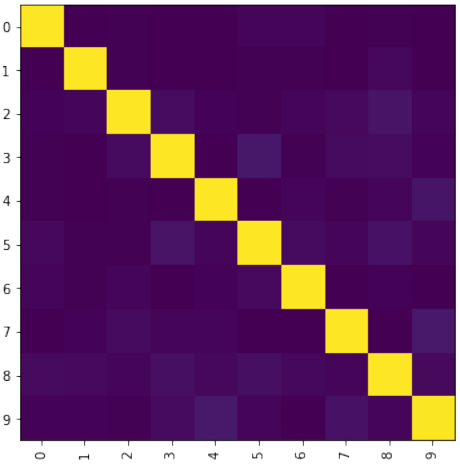
In this section, I have experimented 1-NN model on MNIST dataset. As we all know, one huge con of 1-NN is its efficiency. The running time grows exponentially as the size of dataset grows. Thus in order to save time, I have carefully chosen a balanced subset of MNIST as my experiment training set of size 6000.
Additionaly, to fit the given MNIST data into 1NN and other models following, I have converted the one-hot labels into decimal labels$(0, 1, 2, \ldots)$.
The reason why this model takes long time to train is because each time when we want to find the nearest neighbor of an image, we need to compare it to every single image in our training set and then compute the metric distance. Thus for a training set of size 6000 and a testing set of size 10000 for example, we need to compute 60000000 times.
To speed up this training process, we sometimes apply algorithms such as KD-tree and Ball-tree to prune the unnecessary part of the training data to shorten our searching time.
However, when I was doing the experiment, I found out that when we are using Manhattan distance, ball-tree and KD-tree are for sure shortening the running time by a lot. However, when I am using Minkowski(by setting p to 2, we get Euclidean) as my metric, it is surprisingly that brute force search only took 455 ms for the whole training and predicting process while the other two algorithms took about 1 minute. Ball tree ran slightly (20 seconds) faster than KD tree.
Overall, Minkowski(Euclidean when p = 2) gives higher accuracy than Manhattan(0.926). Ball-tree and KD-tree runs faster than brute force when applying Manhattan as metric. However and amazingly, brute force is a lot faster than the other tree methods when applying Minkowski. For the last observation, further explanation is needed.
The final confusion matrix using Minkowski and brute force search is:



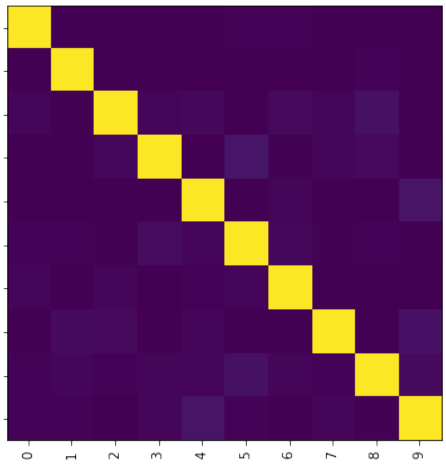
The lighter the diagonal is, the better the accuracy of this model performs. The final accuracy of this setting (on the 6000 subset of MNIST dataset) gives 0.938.

3 Support Vector Machine(SVM)

In this section, I have built SVM to make predictions on MNIST. I applied SVM from sklearn library and noticed that LinearSVC using Stochastic Gradient Descent runs much faster than the other SVC. For this test, still I am training the model on the 6000 subset of MNIST.
Using LinearSVC, it gives accuracy 0.8765 and runs 3 seconds.



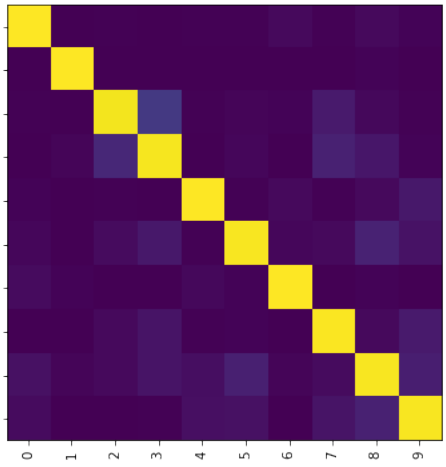
Using SVC, it gives accuracy 0.9113 and runs about 1 minute.



The confusion matrixes both algorithms give are quite similar and quite hard to tell from the plotting.

4 Spatial Pyramid Matching

In this section, I have trained model based on Spatial Pyramid Matching using the model provided in the reference.
This method preprocess the training data and extract sift features. Then it applies K means and SVM to train the model.
Generally speaking, the training process is very long compared to previous methods. I believe that if allowed to use the whole MNIST data, it can out perform pure SVM and K Means. However due to the lack of time here, I still chose to build the model on 6000 subset.
By setting number of clusters to 100, pyramid level to 1, sift step size to 4, the accuracy I have acquired is 0.831 and the confustion matrix is as follows(on the subset of 6000 MNIST dataset):



The detailed report is as follows:

```
Detailed classification report:

The model is trained on the full development set.
The scores are computed on the full evaluation set.

             precision    recall  f1-score   support

        0.0       0.89      0.92      0.90       980
        1.0       0.96      0.98      0.97      1000
        2.0       0.83      0.76      0.79      1000
        3.0       0.74      0.76      0.75      1000
        4.0       0.90      0.88      0.89       982
        5.0       0.81      0.76      0.79       892
        6.0       0.91      0.92      0.92       958
        7.0       0.79      0.84      0.81      1000
        8.0       0.71      0.72      0.71       974
        9.0       0.77      0.78      0.78      1000

avg / total       0.83      0.83      0.83      9786
```

By setting VOC SIZE to 200 and pyramid level to 4 did not contribute to the final accuracy by testing. The resulting accuracy is 0.830 and the confusion matrix looks extremely similar to the one before.

References

[1] The MNIST Database of handwritten digits. [Online]. Available: http://yann.lecun.com/exdb/mnist/

[2] K-Nearest Neighbor. [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

[3] Stochastic Gradient Descent SVM. [Online]. Available: http://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html

[4] Spatial Pyramid Matching. [Online]. Available: https://github.com/CyrusChiu/Image-recognition