COGS 260, Assignment 2

Premanand Kumar p8kumar@ucsd.edu

Abstract

For problem 1, Convolutional Neural Network, I intend to create a neural network that could work faster than LeNet but with comparable accuracy. I call the network as RapidNet. The number of parameters in the LeNet provided with assignment was 1,256,080 and the architecture that I built has 316,266 parameters. Tuning is done for optimiser selection and learning rate. So, with comparable accuracy, I expect a significant speed up in the training time. For problem 2, I used Principal component analysis for prototype selection. Number of components for PCA and the distance function (L1, L2 and so-forth) are the hyper-parameters. For problem 3, I used PCA dimensionality reduction again in a bid to decrease the time with comparable accuracy to 1-Nearest Neighbour. For problem 4, I chose random sampling to reduce the number of training examples. The computational cost of this experiment was high. However, I was still able to tune the hyper-parameters like number of pyramids and dense SIFT descriptor size. The accuracy was the lowest of all the methods I tried. For problem 5, I was able to run a DBF model with good accuracy on the MNIST dataset.

1 Problem 1 - Convolutional Neural Network

1.1 Method

The CNN was implemented in keras[1]. The architecture is as in Figure 1. The first layer is average pool layer (pool_size = (2,2)) which kinds of acts like preprocessing layer with dimensionality reduction. This is followed by two CNN layers (Conv2D(32,kernel_size = (5,5), activation= 'relu')) that are separated by a maxpool layer (MaxPooling2D(pool_size = (2,2))). The output is then fed into a dense layer with 'relu' activation (Dense(1024, activation='relu')) and the final output with 'sigmoid' activation function (Dense(num_classes, activation='softmax')). Thus, the key here is the average pool layer right at the input that does a good job in blurring the images. This is especially useful in the MNIST dataset since the detection does involve only foreground and background separation.

1.2 Experiment

First a grid search was conducted with default settings to choose one of the optimisers. The number of epochs was kept at a constant of 12. All the available optimisers namely 'SGD', 'RMSprop', 'Adagrad', 'Adadelta', 'Adam', 'Adamax' and 'Nadam' were tried and the results were as in table 1.

Preprint. Work in progress.

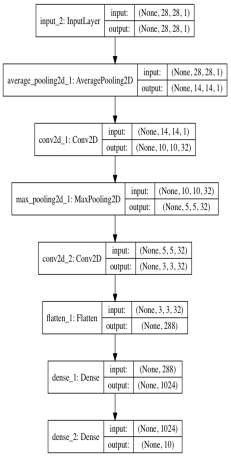

Figure 1: RapidNet Architecture

Table 1: Grid search for Optimiser - RapidNet

Optimizer	Accuracy on test set(%)
SGD	96.69
RMSprop	99.01
Adagrad	99.15
Adadelta	99.17
Adam	99.09
Adamax	99.16
Nadam	98.88

As from the table, it is clear that 'Adamax' gave the one of the highest accuracy with default settings after 12 epoch. Further experiment was conducted with 'Adamax' optimiser. Grid search was again conducted for learning rate increased exponentially. The results are as in table 2.

So, I chose Learning rate 0.002 and continued the experiment with 36 epochs. I was able to achieve the accuracy of 99.96 % on train set, 99.04 % on test set. The plot of loss on accuracy vs iteration is as in Figure 2. The comparison of speed with LeNet is in Figure 3.

Table 2: Grid search for Learning rate - RapidNet network

Learning Rate	Accuracy on test set(%)
0.0002	97.39
0.002	98.95
0.02	98.95
0.2	9.8

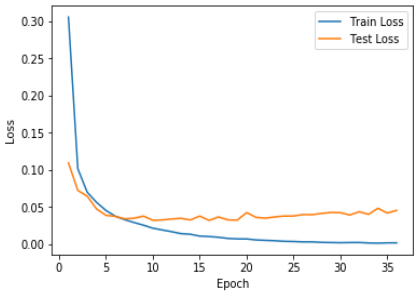

Figure 2: Loss plot for RapidNet Architecture

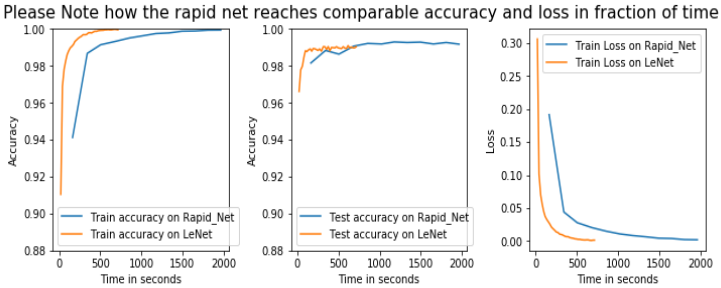

Figure 3: Speed Comparison - RapidNet vs LeNet

1.3 discussion

The reduction in dimension because of using average pooling reduced the number of parameters by 75 %. It is to be noted that since the architecture is faster, it took only 22.42 minutes to run the entire loop of optimiser selection in CPU of Macbook air 2013. Furthermore, the hyperparameter tuning took 12.36 minutes. As we can see from figure 3, the architecture approached the accuracy of LeNet training set by running 36 epochs with the time taken for running a single epoch in LeNet.

2 Problem 2 - 1-Nearest Neighbor

2.1 Method

The 1 Nearest neighbour was done in sklearn neighbours [2]. The prototype selection was done with Principal component analysis [3]. Before feeding it into PCA transformation, the data needs to be standardised by sklearn preprocessing [4]. The standardisation is done on the basis of statistics of the train data and the same statistics is used for test data. The same procedure is followed for PCA reduction.

2.2 Experiment

Experiments were conducted to grid search over both number of components in PCA reduction and distance function for 1-Neighbour classifier. The results were as in table 3. The best result was when we used 2 norm and 130 components with the accuracy of 95.74 %. The computational time for computing higher dimension did not justify the potential small increase in accuracy and so tuning above norm 4 was not done. The confusion matrix[8] for the best detector is as in Figure 4.

Table 3: Grid search for PCA reduction and distance norm - 1- Nearest Neighbour.

Number of components	norm	Accuracy on test set(%)
49	1	95.31
65	1	95.39
98	1	95.50
130	1	95.31
196	1	94.80
392	1	93.03
784	1	91.59
49	2	95.45
65	2	95.57
98	2	95.73
130	2	95.74
196	2	95.49
392	2	94.65
784	2	94.34
130	3	95.49
130	4	95.37

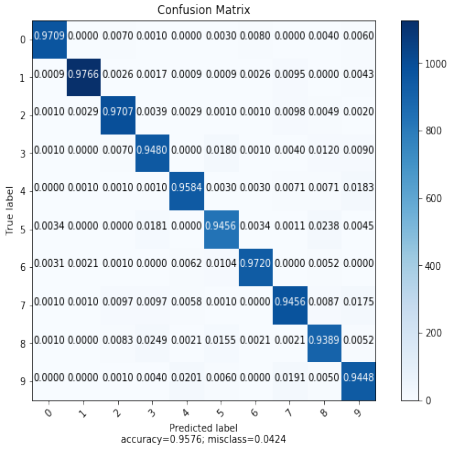

Figure 4: Confusion Matrix for 1NN - 2 norm and 130 component by PCA reduction

2.3 discussion

The PCA retains the dimensions which have maximum information. This is particularly useful in this dataset as the pixel density is more or less useless over most area of the image landscape. The level of detail required is very less as compared to a standard object recognition. Thus we can see that the algorithm performed very well with reduced dimension (which is 17% of the original data).

3 Problem 3 - Support Vector Machines

3.1 Method

Support Vector Machine was done sklearn svm[5]. The dimensions are again reduce by PCA[4]. The hyper parameters here are the type of kernel and after choosing the best kernel, gamma and the parameter corresponding to the best kernel was tuned.

3.2 Experiment

After reducing the dimension using PCA, the grid search was done for the following kernels: rbf, 'linear', 'poly', 'sigmoid' using the default settings. The results are as tabulated in table 4. As we can see, the 'poly' kernel performed the best. Now, grid search was performed for 'poly' kernel with parameter gamma as exponential function of 1/number of samples and the number of degrees vaied from 2 to 4. The results were as from table 5. The confusion matrix for 'poly' kernel with degree 3 is in Figure 5.

Table 4: Grid search for kernel selection - Support Vector Machines

Kernel	Accuracy on test set(%)
linear	94.40
poly	97.78
rbf	95.19
sigmoid	61.92

Table 5: Grid search for gamma and number of degrees - Support Vector Machines

Number of degree	gamma	Accuracy on test set(%)
3	7.6e-1	97.66
3	7.6e-2	97.66
3	7.6e-3	97.78
3	7.6e-4	89.92
2	7.6e-1	97.21
2	7.6e-2	97.21
2	7.6e-3	97.76
2	7.6e-4	95.92
4	7.6e-1	96.89
4	7.6e-2	96.89
4	7.6e-3	96.91
4	7.6e-4	51.17

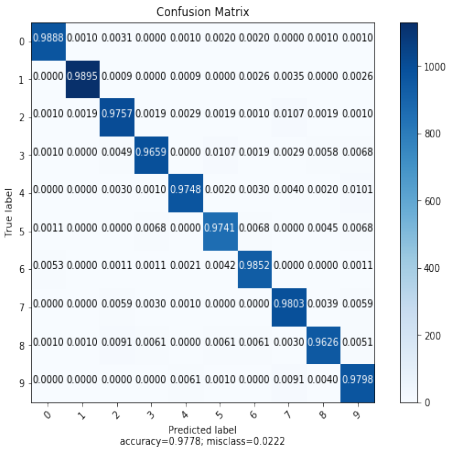

Figure 5: Confusion Matrix -SVM Poly kernel, degree = 3

3.3 discussion

The support vector machine is usually known for the high computational cost. Hence, PCA reduction is a natural choice to increase the computational speed. It was clear from KNN experiment that loss

in information by performing PCA was not as significant because of the high accuracy attained. So, with PCA reduction, we were still able to achieve an impressive accuracy of 97.78 %.

4 Problem 4 - Spatial Pyramid Matching

4.1 Method

The Spatial Pyramid Matching was done by adapting Cyrus Chiu adaptation[6]. The code was modified to able to be run ipython notebook with MNIST input. Here, the SIFT was done in openCV2[9], kmeans[10] for cookbook was done by sklearn and the SVM after feature extraction is done in sklearn. The hypertuning is done on the Cross Validated train data for SVM.

4.2 experiment

The computational time was very high for this algorithm. Since, MNIST data did not have label imbalance, I decided to take large enough subset random sampling (10000 training and 10000 test examples) that would be a good representation of the problem. The hyperparameter here are number of pyramid and dense SIFT descriptor size. The experiment was conducted varying the number of samples and the number of pyramids. The results are as in the table 6.

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	0.95	0.97	0.96	1004
1	0.98	0.99	0.99	1110
2	0.93	0.95	0.94	1023
3	0.86	0.89	0.87	994
4	0.94	0.91	0.93	982
5	0.93	0.87	0.90	908
6	0.95	0.96	0.95	947
7	0.90	0.91	0.91	1007
8	0.87	0.86	0.86	1017
9	0.88	0.87	0.87	1008
avg / total	0.92	0.92	0.92	10000

Figure 6: Spatial Pyramid Summary - Pyramid - 2 , SIFT size - 4

Table 6: Grid search for number of pyramid and SIFT size - Spatial Pyramid Matching

Pyramid	dense SIFT descriptor size	Accuracy on test set(%)
1	4	82.18
1	7	77.57
2	4	91.20
2	7	76.55

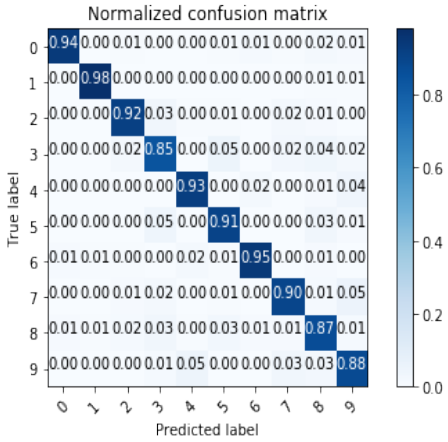

Figure 7: Spatial Pyramid Confusion Matrix - Pyramid - 2 , SIFT size - 4

4.3 Discussion

Though the computational time will was high, I did anticipate the lower accuracy because of the nature of the dataset. However, I believe that the algorithm will perform better on tasks which involve a lot more detail and spacial context. Maybe preprocessing to reduce the dimensionality just like I did for neural network would have helped as I could feed in more training examples. The highest accuracy achieved was 91.2 % with pyramid =2. This is understandable because, with pyramid = 2, we are looking at higher dimensional projection of training data as compared to pyramid = 1.

5 Deep Belief Network

5.1 Method

The deep belief network was executed from the tutorial from Deep Belief Networks[7] and the code[7] was provided in python.

5.2 Experiment

The hyperparameters here are number of hidden layers size. I ran the experiment for 5 epoch pretraining and 10 training epoch. The result of the experiment are in table 7.

Table 7: Grid search for number of hidden layers - Deep Belief Network

DBN Hidden Layer Size	Error on test set(%)
1000	2.11
100	3.24
1500	2.22

6 Conclusion

The important learning from this assignment was the amount of uncertainty even in the deterministic models like knn, svm etc. I did grid search for almost all the deterministic models. They were computationally expensive though some clever preprocessing helped. The significant learning was the fast RapidNet model which outdid the computational speed of LeNet which was lightweight in itself. Though, I can still appreciate the vast uncertainty in the architecture. The constant resource exhaustion while doing Spatial Pyramid Matching was an example of wonderful idea which was hampered by the computational cost. From this exercise, the highest accuracy was from neural

network followed by SVM and the simple nearest neighbour is powerful enough to attain excellent accuracy.

References

[1] Keras Tutorial https://keras.io/getting-started/functional-api-guide/

[2] sklearn.neighbors.KNeighborsClassifier http://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html

[3] sklearn.decomposition.PCA http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html

[4] sklearn.preprocessing.StandardScaler http://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

[5] Support Vector Machines http://scikit-learn.org/stable/modules/svm.html

[6] CyrusChiu https://github.com/CyrusChiu/Image-recognition

[7] Deep Belief Networks http://deeplearning.net/tutorial/DBN.html code - http://deeplearning.net/tutorial/code/DBN.py

[8] Confusion Matrix http://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html

[9] cv::xfeatures2d::SIFT Class https://docs.opencv.org/3.2.0/d5/d3c/classcv_1_1xfeatures2d_1_1SIFT.html

[10] sklearn.cluster.KMeans http://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html