ENEE633 Project 2 Report

Yang Jiao

December 6, 2015

Abstract

Project 2 is composed of 2 parts: Support Vector Machine(SVM) and Convolutional Neural Network(CNN). In the first part, linear SVM, PCA linear SVM and LDA linear SVM are trained and tested. Cross validation is used to facilitate model selection. Then RBF and polynomial kernel SVM are experimented with. In the second part, LeNet is implemented with Caffe. The training progress is monitored with periodic testing. Finally a forward pass is carried out on the trained model and convolutional layers are visualized.

1 Data Preparation

Experiments are conducted over MNIST dataset of handwritten digit images. The MNIST set is composed of 60,000 training patterns and 10,000 testing patterns. Each pattern is a 28×28 image. Experiment data are scaled to the range [0, 1].

In the experiments with support vector machine classifier, 2,000 training patterns and 1,000 testing patterns are selected from the MNIST dataset. Digits are approximately uniformly distributed in both training and testing sets.

In the experiments with LeNet, all training and testing patterns are used.

2 Software Packages Used

In the experiments with support vector machine classifier, LIBSVM library and their Matlab interface is used.

In the experiments with LeNet, Caffe library and their Python interface is used.

3 Linear Support Vector Machine

Linear kernel is used in this section.

3.1 Plain SVM

SVM is applied directly to data vectors in this subsection. A one-vs-one strategy is used to solve the multi-class SVM. A 5-fold cross-validation is conducted over the training samples. The optimal model derived from this process is then tested over the testing set. Experiment results are presented in Table 1, where 2^C is the penalty parameter of the error term.

Table 1: Plain SVM

Kernel	CV Fold	C = -5	C = -4	C = -3	C = -2	C = -1
Linear	5	91.2	90.85	90.35	89.8	89.65
Linear(Test)	-	87.3	-	-	-	-

3.2 PCA SVM

In this subsection, PCA is used to reduce the dimension of the data. In the `do_pca` routing, 95% components is preserved. Then the data is fed to a linear SVM. Experiment results are presented in Table 2. The performance happens to be better than without PCA by 0.3%.

Table 2: PCA SVM

Kernel	CV Fold	C = -5	C = -4	C = -3	C = -2	C = -1
Linear	5	91	90.3	89.45	89.4	89.4
Linear(Test)	-	87.6	-	-	-	-

4 LDA SVM

In this subsection, LDA is used to reduce the dimension of data to 9. Then a linear SVM is used. In the `do_lda` routine, an identity matrix is added to the within-class scatter matrix to make it invertible. Note that LDA achieves performance 0.7% worse than without LDA.

Table 3: LDA SVM

Kernel	CV Fold	C = -10	C = -9	C = -8	C = -7	C = -6
Linear	5	92.2	91.9	92.15	91	90.75
Linear(Test)	-	86.6	-	-	-	-

5 Kernel Support Vector Machine

Kernel SVMs are implemented in this section.

5.1 Gaussian Kernel

A Gaussian kernel is experimented with in this subsection. Similar to Sec 3, cross-validation is used to select model. The experiment results are presented in Table 4. The kernel function is given by $K(x_i, x_j) = \exp(-2^\gamma ||x_i - x_j||^2)$. Note that Gaussian kernel achieved better performance than a linear one.

Table 4: Plain SVM with RBF Kernel

Kernel	CV Fold	C	γ = -9	γ = -8	γ = -7	γ = -6	γ = -5
Gaussian	5	1	89.35	90.8	92.35	93.45	93.75
Gaussian	5	2	90.6	91.7	92.85	93.9	94.05
Gaussian	5	4	91.85	91.85	92.85	93.75	94.05
Gaussian(Test)	-	4	-	-	-	-	92

5.2 Polynomial Kernel

A polynomial kernel is used in this subsection. The kernel function is given by $K(x_i, x_j) = (2^\gamma x_i^T x_j + r)^d$. Note that a polynomial kernel achieved better performance than a linear one.

Table 5: Plain SVM with Polynomial Kernel

Training Set	Testing Set	Kernel	d	C	γ	r	Accuracy
2000	1000	Poly	3	-2	-5	1	90.8

6 LeNet

LeNet is implemented using the Caffe framework in this section.

6.1 Network Structure

The framework follows the example given by Caffe and is summerized in Table 6.

6.2 Performance

The testing is carried out every 500 training iterations to monitor the learning progress. The test results are presented in Table 7. The optimal test accuracy and lost is achieved at 6,000th iteration.

Table 6: Network Structure

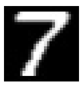
Layer	Channels	Kernel Size	Description
Data	-	-	{train, test} batch {64, 100}
Convolution 1	20	5	{weight, bias} learning rate {0.01, 0.02}
Pooling 1	-	2	max pooling
Convolution 2	50	5	{weight, bias} learning rate {0.01, 0.02}
Pooling 2	-	2	max pooling
Inner Product 1	500	-	{weight, bias} learning rate {0.01, 0.02}
ReLu	500	-	-
Inner Product 2	10	-	{weight, bias} learning rate {0.01, 0.02}
Loss	-	-	softmax loss

Table 7: Test Accuracy and Loss

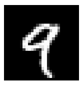
Iter#	Accuracy	Loss	Iter#	Accuracy	Loss
500	0.9734	0.0839642	5500	0.9895	0.0336801
1000	0.9844	0.0508039	**6000**	**0.9915**	**0.0286821**
1500	0.9858	0.0455569	6500	0.9907	0.0315601
2000	0.9858	0.0424465	7000	0.9908	0.0294037
2500	0.9846	0.0479665	7500	0.99	0.0314783
3000	0.9886	0.038409	8000	0.9907	0.0289854
3500	0.9875	0.0391475	8500	0.9903	0.0288363
4000	0.9906	0.0314391	9000	0.991	0.0290497
4500	0.9899	0.0333752	9500	0.9895	0.0351744
5000	0.9898	0.0303246	10000	0.9909	0.028745

6.3 Output of Convolution Layer

Two input images are fed into the trained model obtained at the 10,000th iteration. Digit 7 is incorrectly classified to 1. Digit 9 is correctly classified.



In Fig 6.3 and Fig 6.3, the output of convolution layers are plotted. Each of the output image is a result of filtering the input image with a 5×5 kernel.



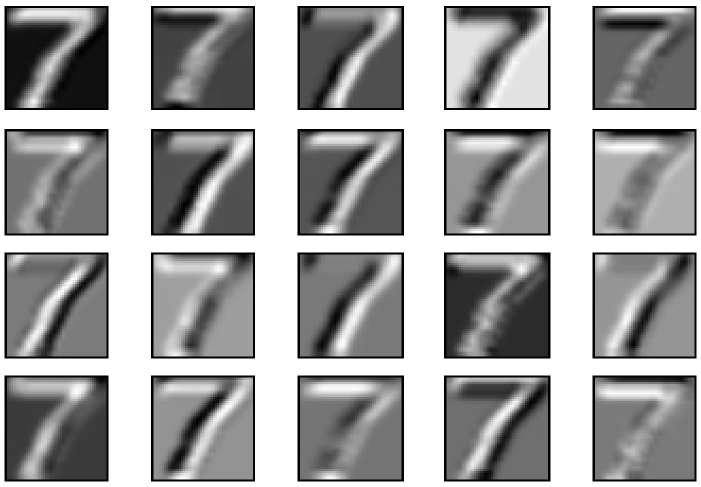


Figure 1: Seven

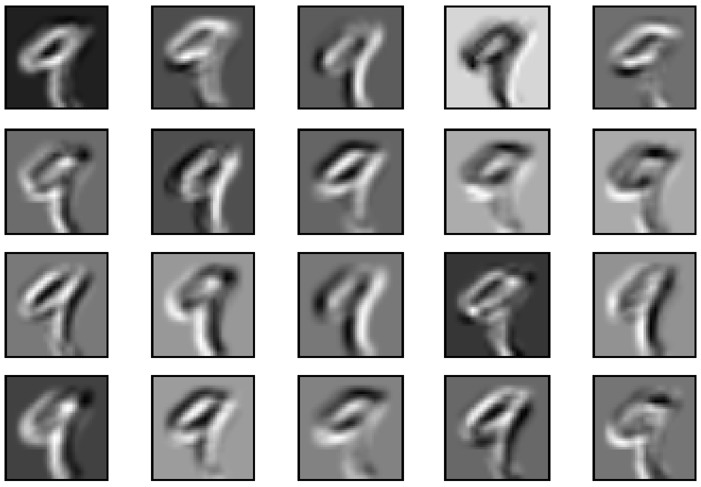


Figure 2: Nine

Figure 3: Filtered images at first layer

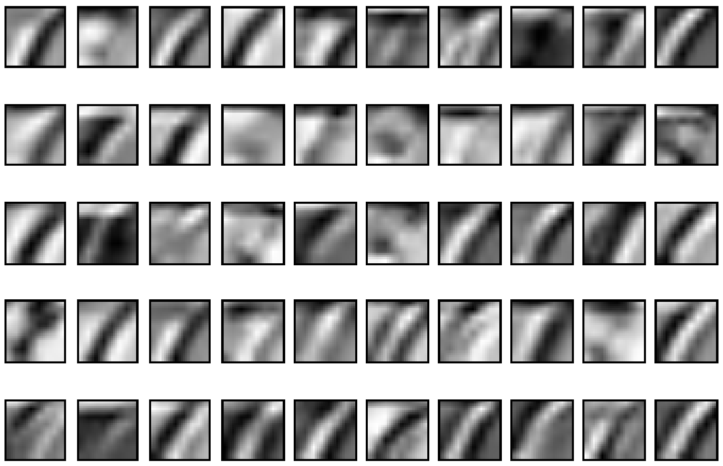


Figure 4: Seven

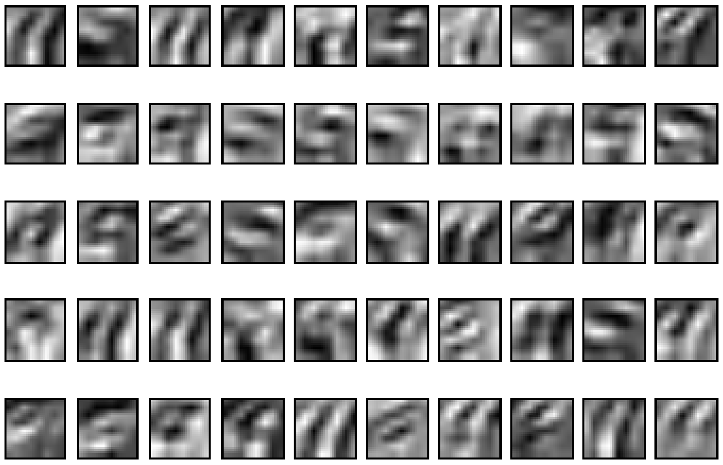


Figure 5: Nine

Figure 6: Filtered images at second layer