Report on Homework 3 – SVM vs. Neural Networks

CS420, Machine Learning, Shikui Tu, Summer 2018

1 Introduction

In machine learning, SVM (Support Vector Machine) is a commonly used classfication method due to its high efficiency and accuracy. Recent years, the neural network has been attracting more and more attention, and also used to solve classification problems. In this homework, I would investigate the performances of SVM and neural network (e.g. MLP) on some classification datasets under different experimental settings.

2 Approach

In this section, I would introduce the datasets and models in my experiments.

2.1 Datasets

In my experiments, I use two datasets that are from **LIBSVM Data** [1] to explore the performances of SVM and neural network. One is **splice** [2], a binary classification dataset with 60 features, 1000 training samples and 2175 testing samples. Another is called **satimage** [3], which is for multi-class classification and has 36 features, 6 classes, 3104 training samples and 2000 testing samples.

Additionally, I choose a dataset called **CIFAR-10** [4] from LISA [5] to conduct comparison between SVM and deep learning algorithm benchmarks. It is a multi-class classification dataset that contains 10 classes, $32 \times 32 \times 3 = 3072$ features, 50000 training samples (divided equally into 5 batches) and 10000 testing samples.

More details about these datasets can refer to Appendix A.1.

2.2 Models

For SVM, I would explore its performance as comprehensive as possible from different parameter settings (e.g. kernels, penalty parameters).

For neural network, considering the complexity of features and scale of samples, I choose MLP (Multi-layer Perceptron) instead of popular DNN or CNN.

In experiments, I would investigate the performances of MLP under different architectures or parameter settings (e.g. number of hidden layers or hidden neurons).

3 Experiments and Results

3.1 Preprocess

Most datasets are likely to have missing data, and those in LIBSVM Data are no exception. Therefore, I first make up for the omission in the datasets, replacing empty data with corresponding mean values. Afterwards, I convert the labels from numbers to one-hot vectors for the calculation of *loss function*.

3.2 SVM

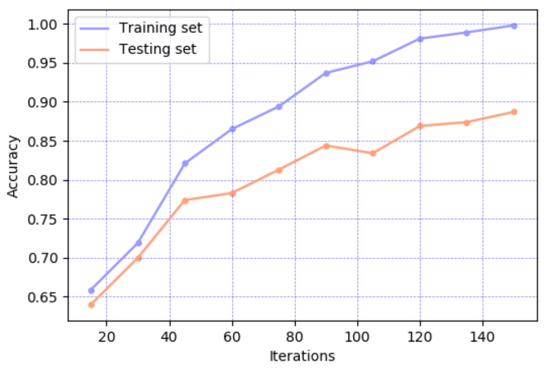
In this section, I would show the performances of SVM in both *splice* and *satimage* datasets under different configurations. Note that the two datasets have different problem settings, sample sizes and etc. I would thus conduct some pertinent discussion based on each experimental result.

Without special explanation, except for the target parameter, the experiments are based on default parameters of *sklearn* [6] (a Python package for machine learning), which can refer to Tab. 1.

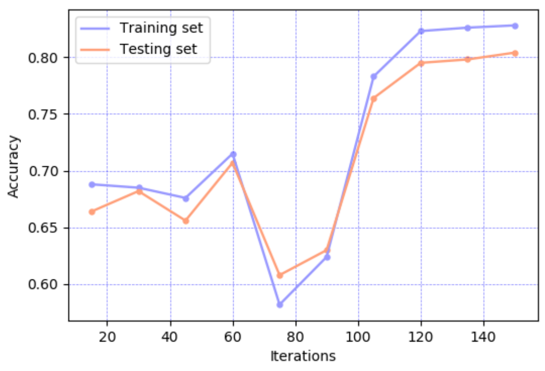
Table 1: Some important default parameters of SVM experiments

Name	Meaning	Value
C	penalty parameter of the error term	1.0
degree	degree of the polynomial kernel function	3
gamma	kernel coefficient for 'rbf', 'poly' and 'sigmoid'	1/n_features
coef0	independent term in kernel function	0.0
shrinking	whether to use the shrinking heuristic	True
tol	tolerance for stopping criterion	1e-3
decision_function_shape	one-vs-rest or one-vs-one	ovr

With these default parameters, I train a baseline model as a reference (Fig. 1). According to the common principle, I should measure the performance with f1-score, but I tend to make the results more intuitive and I consider the **accuracy** of classification as the criterion.



(a) Splice

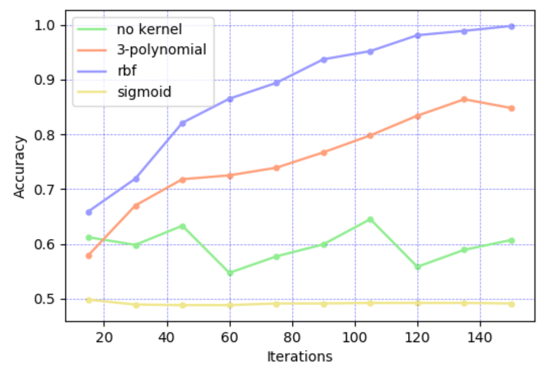


(b) Satimage

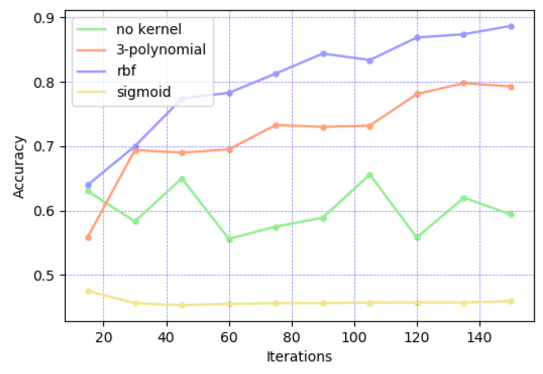
Figure 1: The baseline of SVM.

3.2.1 Kernel

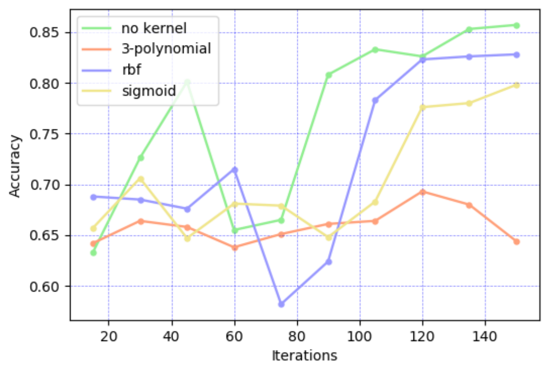
The ability of SVM might be constrainted when dealing with non-linear features, where kernel trick could resolve this issue. Nevertheless, a successful choice of kernel is still based on experience or luck. I apply several kernels to SVM and get the following results (Fig. 2).



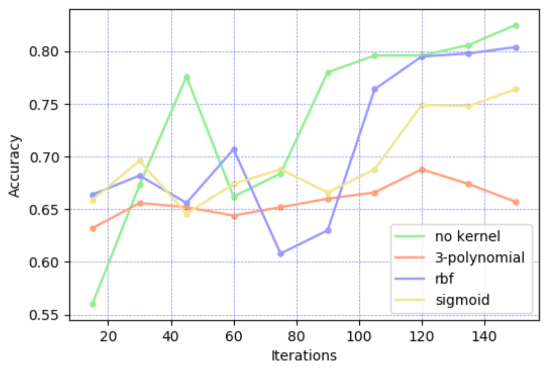
(a) Splice-Training Set



(b) Splice-Tetsing Set



(c) Satimage-Training Set



(d) Satimage-Testing Set

Figure 2: The performances of SVM under different kernels.

It is apparent that the effect of these kernels differ greatly. Rbf (Radial Basis Function) kernel is most suitable for *splice* while no kernel function is the best choice for *satimage*.

Additionally, the selection of specific parameters of a kernel would also affect the results. I test polynomial kernel with different degrees, the test error can refer to Tab. 2.

Table 2: The performances of polynomial kernel under different degrees

Degree	Test Error (Splice)	Test Error (Satimage)
2	0.786	0.635
3	0.857	0.666
4	0.865	0.505
5	0.871	0.554
6	0.880	0.488
7	0.874	0.512
8	0.864	0.477
9	0.867	0.528
10	0.851	0.403

The trend reflected in the above table is that higher degree can reduce training error, but lead to more test error in the meantime. Still, the performance would also drop a lot in both training set and testing set when the degree is too large. In short, this result proves the significance of choosing the complexity of model properly.

3.2.2 Penalty Parameter

Although SVM would encounter difficulties on non-linear data, it can still handle slightly non-separable data via soft-margin technique: transforming the optimization object from

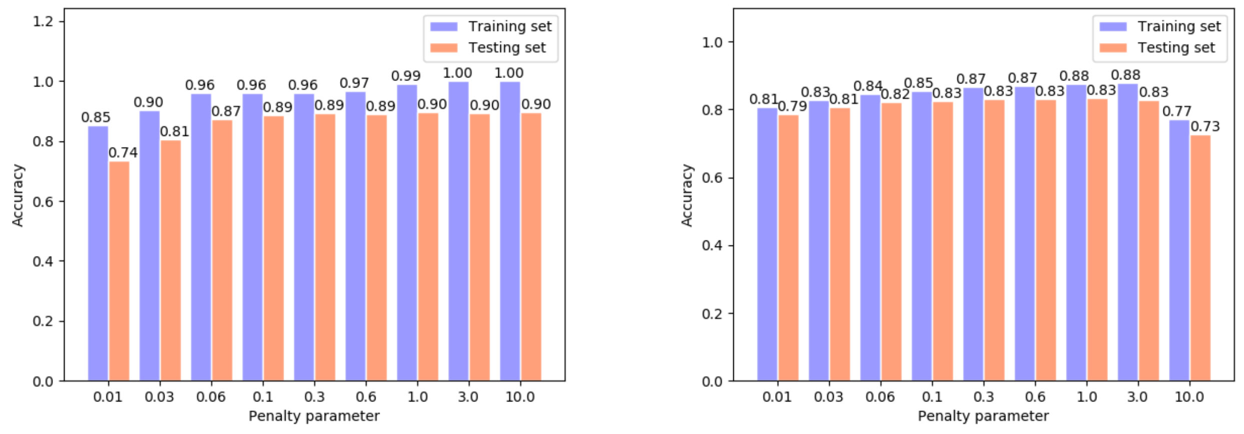
$$\frac{1}{2}w^T w$$

into

$$\frac{1}{2}w^T w + C\sum_{n=1}^{N}\xi_n,$$

where the penalty parameter C is critical for classification performance.

I have tried the C on different orders of magnitude, and the results can be found in Fig. 3.



(a) Splice (b) Satimage

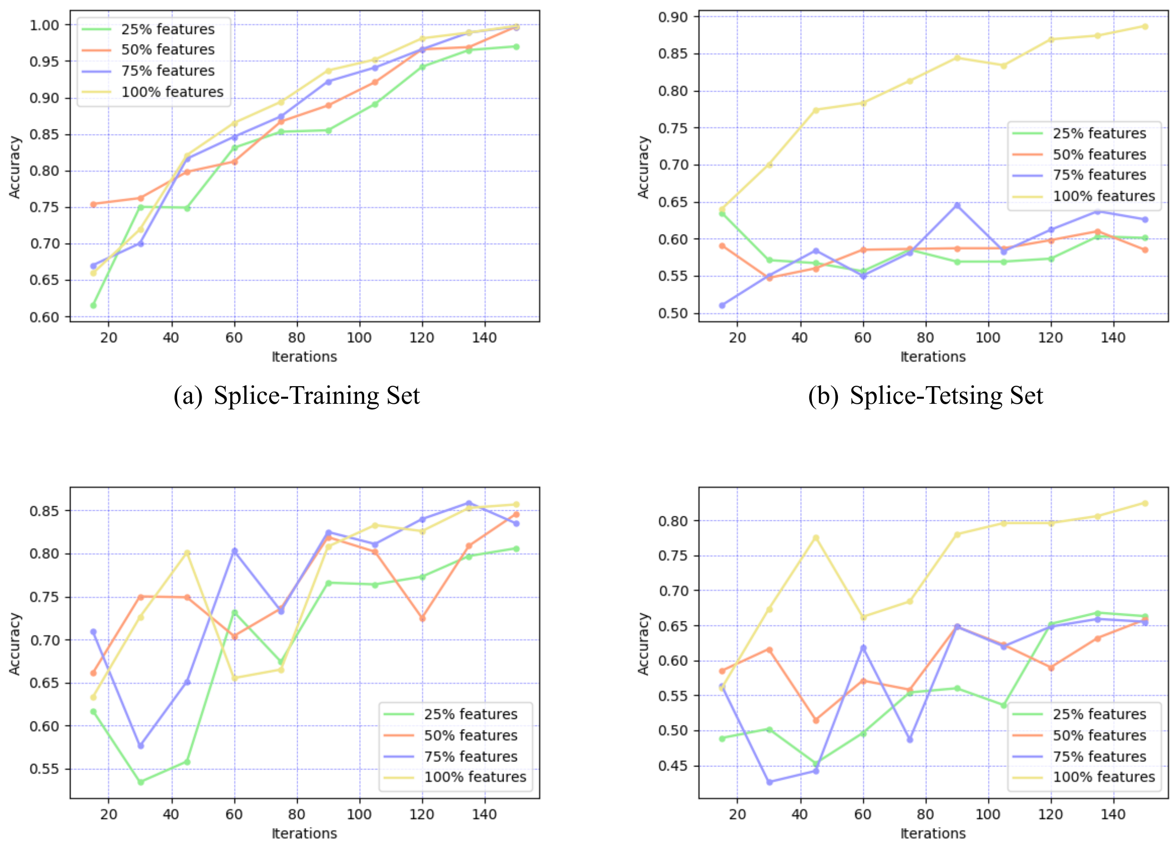
Figure 3: The performances of SVM under different penalty parameter C.

According to the results, the introduction of C is sometimes beneficial to the classification. More specifically, proper C could improve the performance of SVM while excessive C might produce additional errors.

In general, the circumstances might vary a little bit according to the data, where C should be determined by large scale of experiments.

3.2.3 Dimension of Features

In some cases, the performance of a classification model might be reduced due to the redundant features of the dataset. Therefore, I test the performances of SVM under different variance radios of features. I implement this dimension reduction via PCA, corresponding results can refer to Fig. 4.



(a) Splice-Training Set (b) Splice-Tetsing Set

(c) Satimage-Training Set (d) Satimage-Testing Set

Figure 4: The performances of SVM under different variance radios of features.

This result indicates both of *splice* and *satimage* have few redundant features, and less features lead to great performance reduction in testing set, namely affect the generalization ability.

3.2.4 Summarization

SVM is a powerful tool for classification, which can achieve satisfactory results in both binary and multi-class classification problems. Meanwhile, SVM could take advantages of many tricks (e.g. soft-margin, kernel) to extend its application scope and improve its performances. If SVM is applied to small scale problems with appropriate configurations, it would have a surprisingly performance.

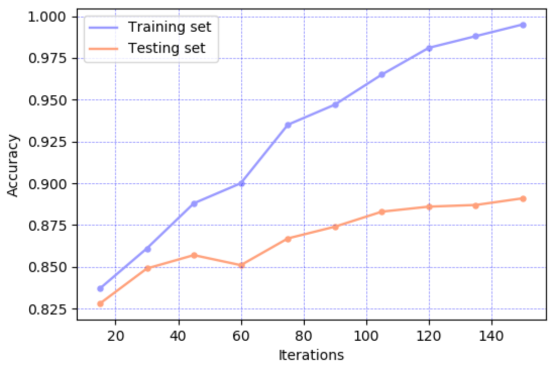
3.3 Neural Network

In this section, I would also show the performances of MLP in *splice* and *satimage* under different configurations (e.g. optimization algorithms, network architectures). In the meantime, I would also analyze each result concisely. Note that MLP and neural network here are the same meaning. CNN might achieve better performances, but my computing resources can hardly afford it.

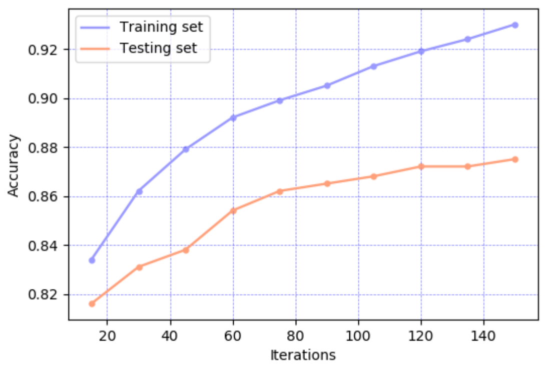
Same as Sec. 3.2, I conduct the following experiments based on default parameters in *sklearn*. Some important parameters are shown in Tab. 3 and the baseline model can refer to Fig. 5.

Table 3: Some important default parameters of MLP experiments

Name	Meaning	Value
hidden_layer_sizes	the size of hidden layers	(100,)
activation	activation function for the hidden layer	relu
solver	the solver for weight optimization	adam
alpha	L2 penalty (regularization term) parameter	0.0001
batch_size	size of minibatches for stochastic optimizers	min(200, n_samples)
learning_rate_init	the initial learning rate used	0.001
power_t	the exponent for inverse scaling learning rate	0.5
tol	tolerance for the optimization	1e-4



(a) Splice



(b) Satimage

Figure 5: The baseline of MLP.

3.3.1 Optimization Algorithm

Optimization algorithms refer to a series algorithms used to update parameters of neural network (e.g. sgd, adam [7]). The optimal directions and the effects of these algorithms are quite different, Fig. 6 is a visual illustration.

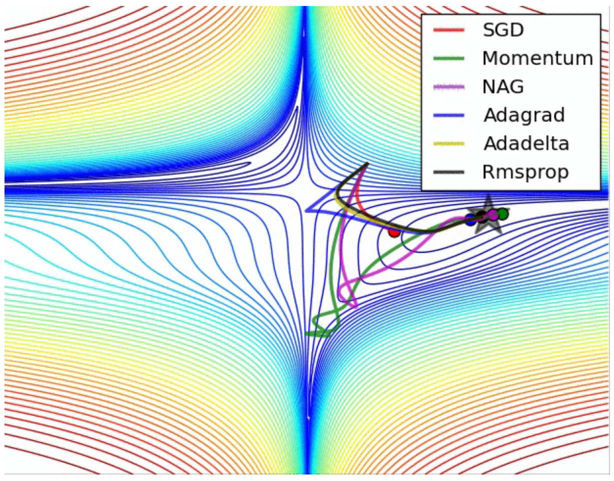
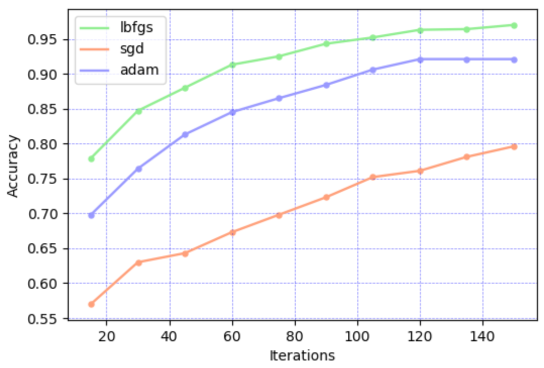
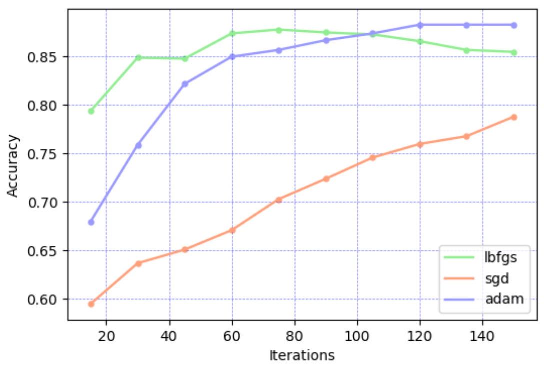


Figure 6: The visual illustration of optimization algorithms [8].

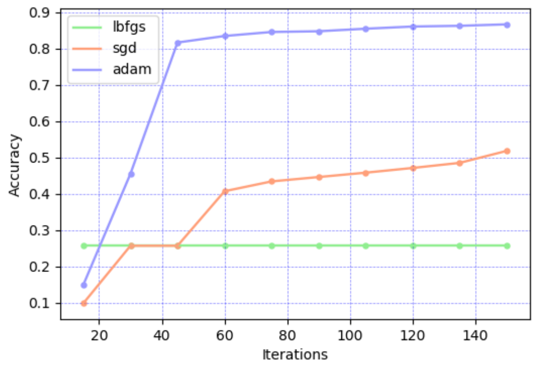
Since the performance of MLP is greatly influenced by the optimization algorithms, I tend to investigate it in detail. I test the effects of three typical algorithms: **lbfgs**, **sgd** and **adam**. The corresponding training processes are shown in Fig. 7.



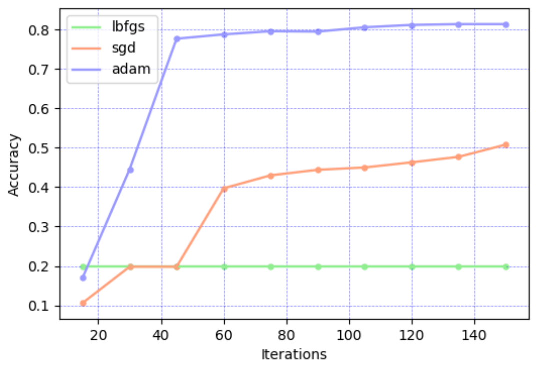
(a) Splice-Training Set



(b) Splice-Tetsing Set



(c) Satimage-Training Set



(d) Satimage-Testing Set

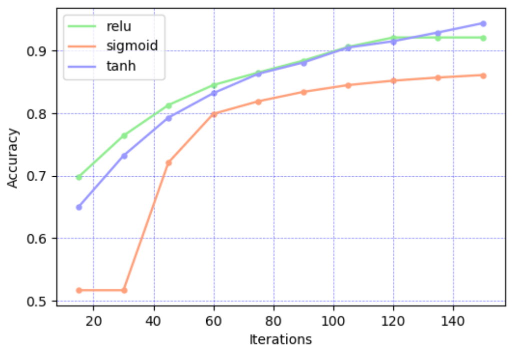
Figure 7: The performances of MLP under different optimization algorithms.

The results in two datasets are consistent with the theoretical analysis: the effect of optimization algorithms differ awfully. It seems that *adam* achieves the best performance in both datasets (though *lbfgs* has lower training error, it might encounter overfitting in *splice*). In *satimage*, the rest two algorithms have poor performance, *lbfgs* even falls into a local minimum at the beginning of training.

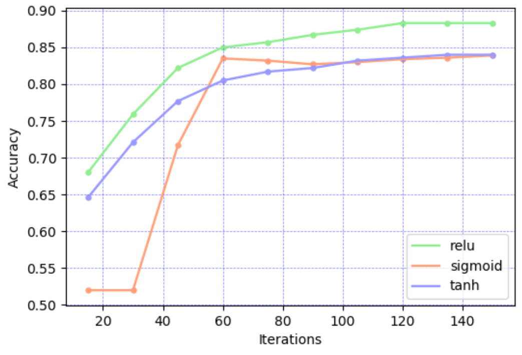
In brief, *adam* algorithm could achieve satisfactory results in most cases, the optimization effect of *sgd* is weaker and *lbfgs* might be instable.

3.3.2 Activation Function

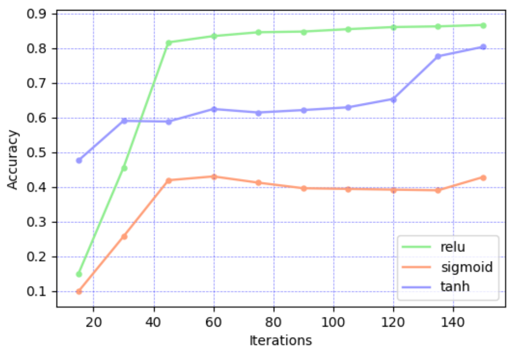
Activation function is an indispensable component of MLP. It extends the scope that MLP can represent by introducing non-linear factors into MLP. I compare the effects of 3 commonly used activation functions (**relu** [9], **sigmoid** and **tanh**) on the two datasets (Fig. 8).



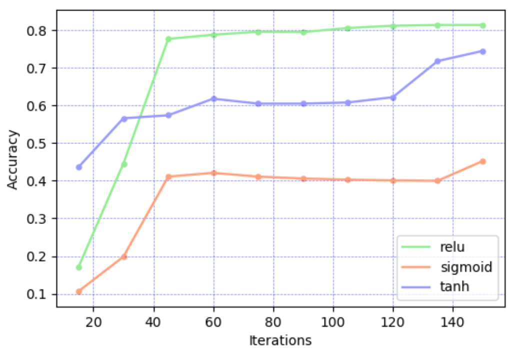
(a) Splice-Training Set



(b) Splice-Tetsing Set



(c) Satimage-Training Set



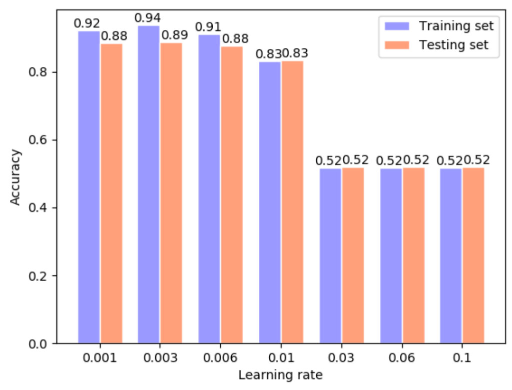
(d) Satimage-Testing Set

Figure 8: The performances of MLP under different activation functions.

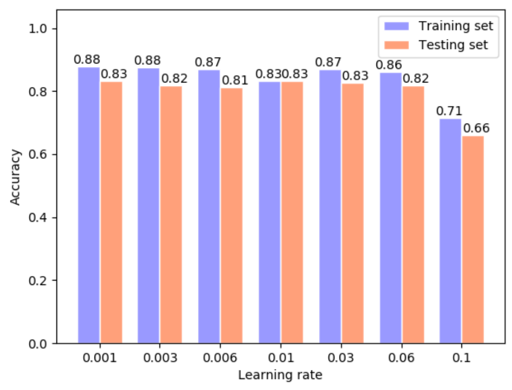
Similar to Sec. 3.3.1, the effects of different activation function are widely divergent, especially on *satimage*. In most cases, *relu* is undoubtedly the best choice. *sigmoid* and *tanh* could be applied to simple problems, their effects would be greatly reduced when confronted with complex problems.

3.3.3 Learning Rate

In training process, learning rate controls the speed of parameters updating. Researchers often pours quantities of time into adjusting learning rate due to its large effect to training results. I investigate the classification results on several orders of magnitude of learning rate (Fig. 9).



(a) Splice



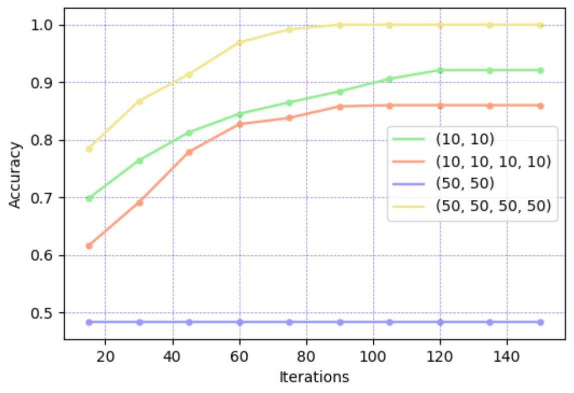
(b) Satimage

Figure 9: The performances of MLP under different learning rate.

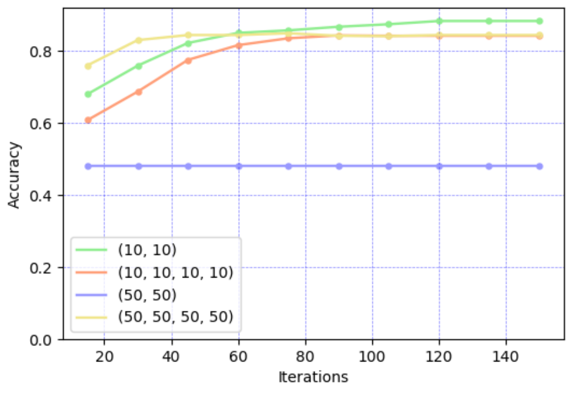
According to above results, learning rates of 0.003 and 0.001 could provide the best performances for MLP on *splice* and *satimage* respectively. Unfortunately, even the best results in Fig. 9 have no improvement compared with baseline (Fig. 5), this also illustrates the difficulty on choosing a appropriate learning rate.

3.3.4 Network Architecture

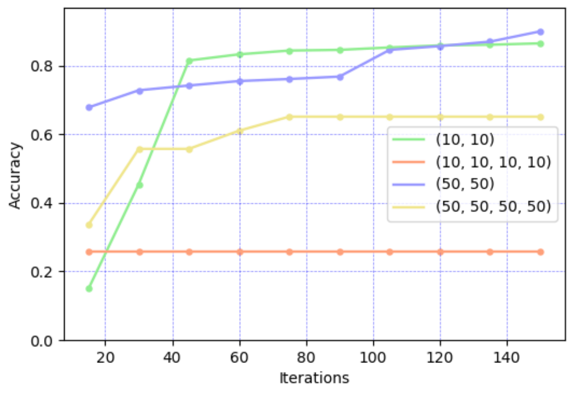
In addition to the above configuration factors, the network architecture of MLP also contributes a lot to final results. There are two simple changes to the network architecture: deepening the network and increasing the number of neurons of each layer. I have tried several architectures based on these two changes (Fig. 10).



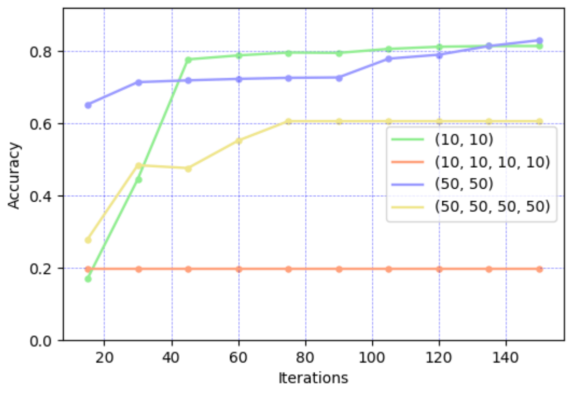
(a) Splice-Training Set



(b) Splice-Tetsing Set



(c) Satimage-Training Set



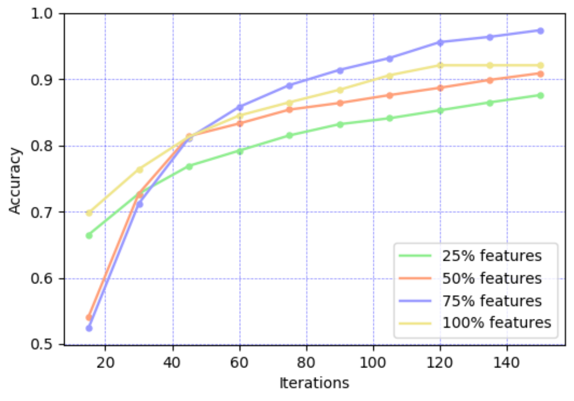
(d) Satimage-Testing Set

Figure 10: The performances of MLP under different network architectures (the numbers in brackets indicate the number of neurons in each layer).

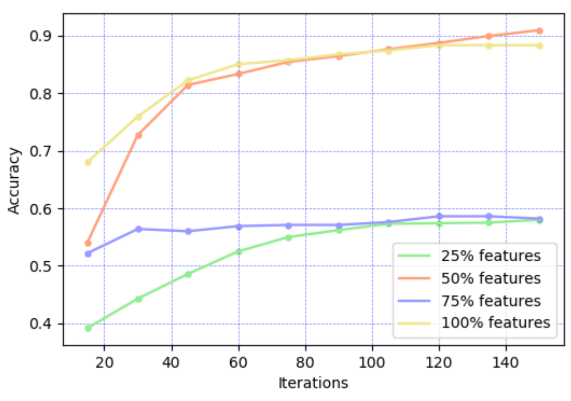
For a certain dataset, appropriate architectures would bring about good performances, like red and yellow lines in Fig. 10(b) and yellow line in Fig. 10(d). Otherwise, unsuitable architectures might crash the whole training process (blue line in Fig. 10(b) and red line in Fig. 10(d)).

3.3.5 Dimension of Features

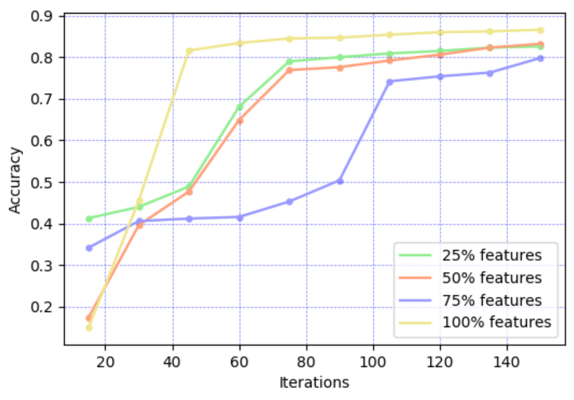
Similar to Sec. 3.2.3, I also fit my MLP model under different variance radios of features (Fig. 11).



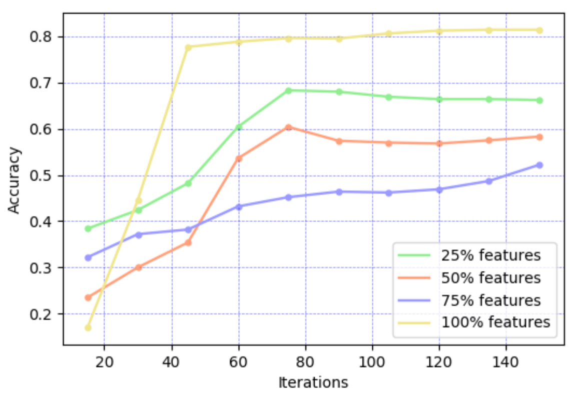
(a) Splice-Training Set



(b) Splice-Tetsing Set



(c) Satimage-Training Set



(d) Satimage-Testing Set

Figure 11: The performances of MLP under different variance radios of features.

The above result has the same trend as that in Fig. 4, which is reasonable due to the same datasets are used.

3.3.6 Summarization

MLP is a common way for a variety of problems. It owns a broader representation ability and stability compared with SVM and the baselines (Fig. 1 and Fig. 5) also prove this.

Still, an available MLP model often takes much time to finetune, even a small change on any one component might make a great difference.

4 Comparison Between SVM and Deep Learning Algorithm Benchmarks

In this section, I would apply SVM on CIFAR-10 dataset [4] and compare the results to corresponding deep learning algorithm benchmarks. These results were obtained with convolutional neural networks or some other fancy network architectures.

4.1 Baseline of SVM

I first fit SVM with default parameters of *sklearn* on CIFAR-10 and get a terrible result: 88% training error and 90% test error with 10,000 iterations, taking about 3 hours on a INTEL I5-5200U CPU. I infer that such result might be out of the high dimension of features and large scale of samples. Then I start to adjust the SVM model to improve its performance.

4.2 Finetune

In this section, I would show the process of finetuning my SVM model. Due to the limited computing resources, I can hardly conduct perfectly comprehensive experiments. I tend to focus on the points I think are pivotal. Note that I only need to observe the trend here, so I just use the original data to finetune parameters in 1000 iterations.

1. Kernel Since the performance of SVM on CIFAR-10 is pretty poor, I think it is difficult to improving this just by adjusting some numerical parameters. Therefore, I start with picking out the most appropriate kernel.

I have tried 3 kinds of kernels on each batch of CIFAR-10 with 1,000 iterations and default parameters of *sklearn*. The results can refer to Tab. 4.

Table 4: The performances of different kernels on CIFAR-10

	Batch 1	Batch 2	Batch 3	Batch 4	Batch 5	Whole data
Rbf (training error)	0.193	0.192	0.196	0.192	0.198	0.123
Rbf (test error)	0.105	0.103	0.105	0.105	0.105	0.105
Polynomial (training error)	0.223	0.236	0.245	0.248	0.244	0.170
Polynomial (test error)	0.209	0.218	0.213	0.229	0.228	0.172
Sigmoid (training error)	0.098	0.098	0.103	0.098	0.102	0.100
Sigmoid (test error)	0.100	0.100	0.100	0.100	0.100	0.100
Linear kernel (training error)	0.234	0.201	0.254	0.249	0.209	0.191
Linear kernel (test error)	0.218	0.184	0.240	0.223	0.186	0.180

As is shown in the above table, linear kernel achieves best performance, polynomial kernel is the next one and sigmoid kernel owns the worst performance. Nevertheless, the performances of other kernels are still possible to exceed that of linear kernel by modifying corresponding parameters.

2. Parameters of Rbf Kernel Basically, rbf kernel ($K(x_i, x_j) = exp(-\gamma\|x_i - x_j\|^2), \gamma > 0$) is the most commonly applied kernel function, linear kernel is just a special case of rbf kernel and sigmoid kernel tend to be similar with rbf kernel under some special parameters. Compared with poly kernel, rbf kernel has less parameters to train and would reduce the difficulty of calculating.

In consideration of the above factors, I decide to adjust the γ of rbf kernel to obtain a better performance (Tab. 5), the default value is the $1/numberOfFeatures$, namely $1/3072 = 0.0003$.[1]

[1]The parameters adjustments for other kernels can refer to Appendix A.2.

Table 5: The performances of different kernels on CIFAR-10 (Considering the limited computing resources, I treat the performance on batch 1 as a representation of that in whole data)

The value of γ	Training error on batch 1	Test error on batch 1
3×10^{-7}	0.411	0.318
3×10^{-7}	0.464	0.330
6×10^{-7}	0.470	0.304
1×10^{-6}	0.466	0.282
3×10^{-6}	0.423	0.229
6×10^{-6}	0.404	0.233
1×10^{-5}	0.344	0.242
3×10^{-5}	0.320	0.239
6×10^{-5}	0.319	0.238
1×10^{-4}	0.309	0.227
3×10^{-4}	0.193	0.105
1×10^{-3}	0.188	0.100
3×10^{-3}	0.188	0.100
6×10^{-3}	0.188	0.100
1×10^{-2}	0.188	0.100

3. Penalty Parameter Ultimately, I focus on finetuning penalty parameter C. C represents the tolerance for error, inappropriate C would reduce the generalization ability of SVM model. I adjust C based on the best result above (results can be found in Fig. 12).

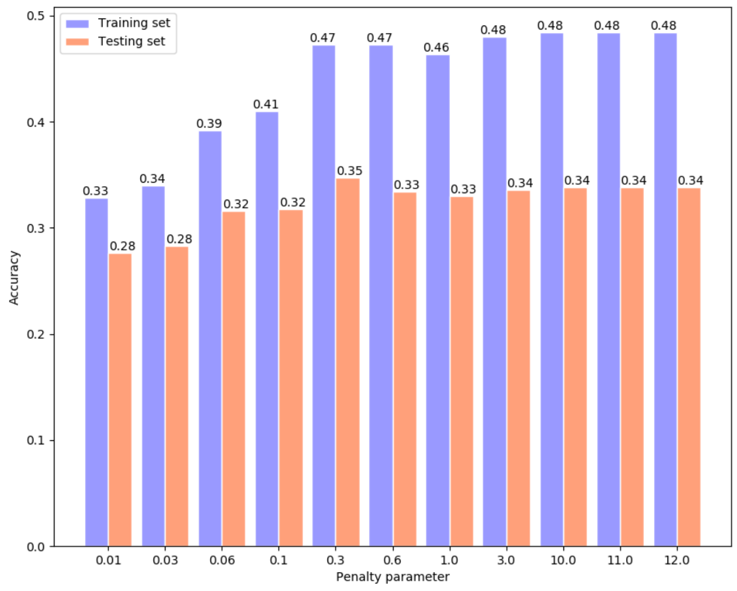


Figure 12: The performances of SVM model under different penalty parameters.

Based on the above results, I infer that the most appropriate C for SVM on CIFAR-10 might near 10.0. Further explorations narrow this scope (Tab 6).

Table 6: The performances of different penalty parameters C on CIFAR-10

The value of C	Training error on batch 1	Test error on batch 1
7.5	0.484	0.338
8.0	0.484	0.338
8.5	0.484	0.338
9.0	0.484	0.338
9.5	0.484	0.338
10.0	0.484	0.338
10.5	0.484	0.338

Since the values of C in the above table correspond to the same performance, I just select C as 10.0 for my SVM model.

4.3 Final Result

So far, I have got relatively appropriate parameters of SVM model, then I compare it with some deep learning algorithm benchmarks [10] (Tab. 7). To give full play to the performance of SVM, I also employ some preprocessing methods. Note that I remove the limitation of the number of training iterations and train SVM model until it converges or iterates into 100,000 steps.

Table 7: The comparisons between SVM (without limitation of the number of iterations) and deep learning algorithm benchmarks, some results might be missing due to the massive time consumption

Model	Test error on CIFAR-10 (& training time)
SVM	77.05% more than 10 hours
SVM (standardization)	71.69% more than 10 hours
SVM (max-min normalization)	73.25% more than 10 hours
SVM (pca 90% variance radio)	68.74% 42 minutes
SVM (pca 80% variance radio)	70.01% 18 minutes
SVM (standardization & pca 90% variance radio)	62.33% 38 minutes
Fractional Max-Pooling [11]	3.47%
Spatially-sparse CNN [12]	6.28%
5-layers MLP	6.37%
Batch-normalized maxout NIN [13]	6.75%
NIN [14]	8.20%
Maxout Networks [15]	8.80%
Cuda-convnet [16] (with data augmentation)	11.00%
Multi-Column DNN [17]	11.21%
ReNet [18]	12.35%
Cuda-convnet (without data augmentation)	13.00%
Convolutional Kernel Networks [19]	17.82%
PCANet [20]	21.33%

The results indicate that the performances of SVM on CIFAR-10 are greatly inferior to deep learning algorithms, even with the assistance of some common data preprocessing methods.

4.4 Strengths and Weaknesses of SVM on Big Datasets

According to the results in Sec. 4.2 and Sec. 4.3, SVM performs poorly on a big dataset CIFAR-10. Note that the so-called 'big dataset' here refer to those datasets with more than 10,000 samples or 1,000 features. In combination with the theoretical and experimental results, I think the weaknesses of SVM on big datasets consist of the following aspects:

1. Basically, the aim of SVM is to find hyper-planes. When the distribution of data is not comprehensive enough, most data might be beneficial to adjust the planes. Once the data has enough coverage, most data would have no influence to planes, and some noise data might even be considered as support vectors. Then the model tends to contain more support vectors and encounter over-fitting problem;

2. The support vectors are solved via quadratic programming, which would consume quantities of memories and time under large scale of samples. Suppose the number of samples and features are n and m respectively, the complexity of SVM is $O(mn^2)$ when using kernel tricks (e.g. rbf kernel);

3. The classical SVM mainly deals with binary classification problems, while most big datasets are constucted for multi-class classification. Under such circumstances, the most common solution is to combine multiple SVM models (e.g. one-vs-one, one-vs-rest). Still, the performance of SVM on multi-class classification is inferior to binary classification problems;

4. A common weakness of machine learning algorithms on big datasets is the tremendous cost for parameters adjustment, SVM is no exception, which has been discussed detailedly in Sec. 3.2. Additionally, SVM also suffers from unstable problems sometimes.

Although SVM has the above weaknesses on big datasets, it also owns some unique strengths:

1. SVM constructs its optimal hyper-plane based on structural risk minimization theory, which could be represented as a convex optimization problem. Since such problems have been thoroughly solved, SVM would be more likely to obtain global optimum solution, while most popular methods like MLP only achieve local minimum in most cases;

2. Since the optimization objective of SVM could be solved by classical algorithms, SVM owns faster convergence velocity than neural network and some other machine learning algorithms. That is, SVM would converge with less iterations on big datasets, despite of the enormous time consumption for each iteration.

In conclusion, SVM is a flexible and high-performanced model on small or medium datasets, where SVM can achieve similar performances to neural network (e.g. MLP). However, SVM is inappropriate for big datasets, where its weaknesses, especially the time cinsumption, might conceal its strengths in most cases. Still, methods to make up for the weaknesses are being constantly developed (e.g. parallelization to optimize time consumption).

A Appendix

A.1 Details of Datasets in Experiments

A.1.1 Splice [2]

Basically, according to the biological knowledge, splice junctions are points on a DNA sequence at which 'superfluous' DNA is removed during the process of protein creation in higher organisms.

Splice dataset aims to recognize two classes of splice junctions, given a DNA sequence, the boundaries between exons (the parts of the DNA sequence retained after splicing) and introns (the parts of the DNA sequence that are spliced out).

A.1.2 Satimage [3]

Satimage dataset is also called **satalog** dataset. It contains multi-spectral values of pixels in 3×3 neighbourhoods in satellite images, and the classification associated with the central pixel in each neighbourhood.

As a classification dataset, the aim of satimage is to predict the classification, given the multi-spectral values. In the sample database, the class of a pixel is coded as a number.

A.1.3 CIFAR-10 [4]

The CIFAR-10 dataset is labeled subsets of the 80 million tiny images dataset, collected by Alex Krizhevsky, Vinod Nair, and Geoffrey Hinton. CIFAR-10 consists of 60000 32×32 colour images (50000 for training (divided into 5 batches), 10000 for test) in 10 classes, with 6000 images per class.

Note that the classes of CIFAR-10 are completely mutually exclusive and there is no overlap between two classes (e.g. automobiles and trucks).

A.2 The Parameters Adjustments of Kernels.

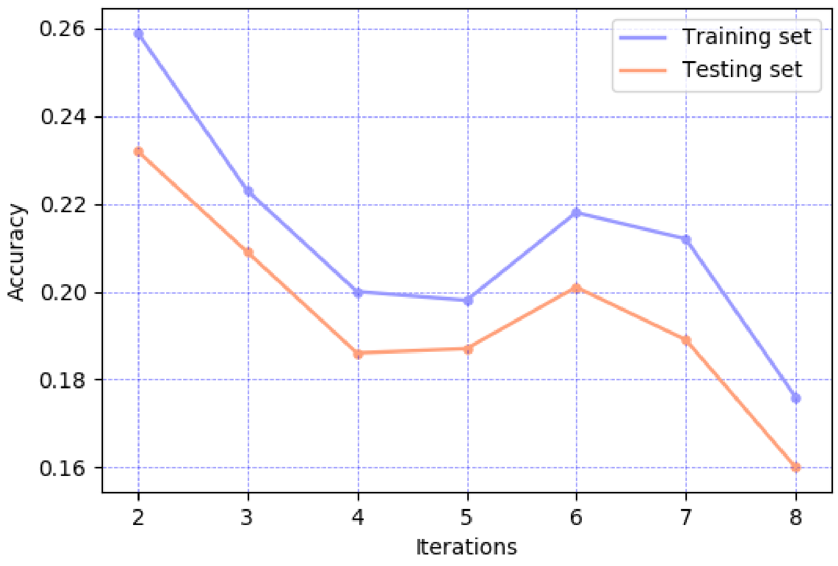


Figure 13: The performances of polynomial kernel under different degrees.

References

[1] "Libsvm data: Classification, regression, and multi-label." `https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/`.

[2] "Molecular biology (splice-junction gene sequences) data set." `http://archive.ics.uci.edu/ml/datasets/Molecular+Biology+%28Splice-junction+Gene+Sequences%29`.

[3] "Statlog (landsat satellite) data set." `https://archive.ics.uci.edu/ml/datasets/Statlog+(Landsat+Satellite)`.

[4] "The cifar-10 dataset." `http://www.cs.utoronto.ca/~kriz/cifar.html`.

[5] "Lisa." `http://www.iro.umontreal.ca/~lisa/twiki/bin/view.cgi/Public/WebHome)`.

[6] "scikit-learn." `http://scikit-learn.org/stable/`.

[7] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[8] "Cs231n: Convolutional neural networks for visual recognition." `http://cs231n.github.io/neural-networks-3/`.

[9] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.

[10] "Classification datasets results." `http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html`.

[11] B. Graham, "Fractional max-pooling," *arXiv preprint arXiv:1412.6071*, 2014.

[12] B. Graham, "Spatially-sparse convolutional neural networks," *arXiv preprint arXiv:1409.6070*, 2014.

[13] J.-R. Chang and Y.-S. Chen, "Batch-normalized maxout network in network," *arXiv preprint arXiv:1511.02583*, 2015.

[14] M. Lin, Q. Chen, and S. Yan, "Network in network," *arXiv preprint arXiv:1312.4400*, 2013.

[15] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio, "Maxout networks," *arXiv preprint arXiv:1302.4389*, 2013.

[16] "cuda-convnet." `https://code.google.com/archive/p/cuda-convnet/`.

[17] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Computer vision and pattern recognition (CVPR), 2012 IEEE conference on*, pp. 3642–3649, IEEE, 2012.

[18] F. Visin, K. Kastner, K. Cho, M. Matteucci, A. Courville, and Y. Bengio, "Renet: A recurrent neural network based alternative to convolutional networks," *arXiv preprint arXiv:1505.00393*, 2015.

[19] J. Mairal, P. Koniusz, Z. Harchaoui, and C. Schmid, "Convolutional kernel networks," in *Advances in neural information processing systems*, pp. 2627–2635, 2014.

[20] T.-H. Chan, K. Jia, S. Gao, J. Lu, Z. Zeng, and Y. Ma, "Pcanet: A simple deep learning baseline for image classification?," *IEEE Transactions on Image Processing*, vol. 24, no. 12, pp. 5017–5032, 2015.