Comparison of Multiclass Kernel Support Vector Machines and Neural Networks for Image Classification

Kernel-based Machine Learning and Multivariate Modeling - project

Lucas Robin and Matthias Hertel

December 30, 2016

Contents

1 Introduction

1.1 Task

In this project we compare Kernel Support Vector Machines (KSVMs) and state of the art techniques like Convolutional Neural Networks (CNNs) for image classification.
Since image classification is a multi-class problem, we implement our own strategies of how to use KSVMs in order to decide which class will be assigned to a given image, and we compare our strategies to the strategies which are implemented in the R-library *kernlab*.
Both CNNs and KSVMs have a lot of parameters, so a main part of our work was to search for parameter settings that lead to good classification results.

1.2 Datasets

We have three different datasets with varying kinds of images, image sizes and dataset sizes.

1.2.1 ZIP

The ZIP dataset was provided in the second part of the course for some homeworks. It contains 16x16-pixel grayscale images of handdrawn digits from 0 to 9. The dataset is rather small with a training set of 7291 and a test set of 2007 images.

1.2.2 MNIST

The MNIST dataset is frequently used in the image classification literature and can be considered to be a standard dataset. We downloaded the dataset from [Yann LeCun's homepage]. It contains 28x28-pixel grayscale images of handdrawn digits from 0 to 9. The dataset is bigger than the ZIP dataset and comes with a training set of 60000 and a test set of 10000 images.

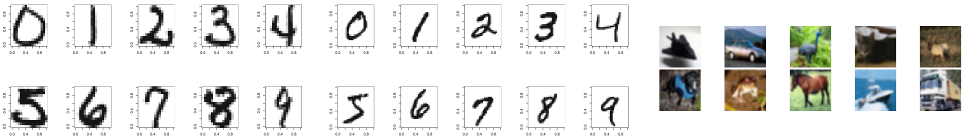


Figure 1: Example images of the classes of the ZIP, MNIST and CIFAR10-dataset.

1.2.3 CIFAR-10

The CIFAR-10 dataset contains 32x32-pixel RGB-coloured images of the following ten classes: airplane, automobile, bird, cat, deer, dog, frog, horse, ship, truck.

The dataset is divided into a training set of 50000 and a test set of 10000 images.
All the data can be found on [Alex Krizhevsky's homepage].

2 Neural Networks

2.1 Architectures

We compare five different network architectures, which are all inspired by [LeCun et al., 1998].

2.1.1 Layer types

First note that a 32x32-pixel image with 3 colour-channels from the CIFAR-10 dataset is represented by a matrix with dimensions (3, 32, 32). We call the first dimension "depth" (i.e. the number of channels) and the other two dimensions "height" and "width".
Analogue, grayscaled images of the ZIP dataset have shape (1, 16, 16) and images of the MNIST dataset have shape (1, 28, 28).

Each layer of a neural network takes an image representation matrix as input and transforms it to another representation with a potentially different dimensions. In the following we explain how that works for the three layer types that we use, namely dense layers, convolution layers and subsampling layers.

Dense layer

Each neuron of a dense layer is connected to all the values of the layer's input. A dense layer with k neurons produces an output of dimensions (k, 1, 1), independent from the input shape.
In this work we always use the RelU activation function for dense layers.
Each of our networks has a dense layer with 10 neurons as last layer, where the activation of a neuron stands for the score for one of the 10 classes.

Convolution layer

A convolution layer has f filters with filter shape (s, s). Each neuron is connected fully in depth to the input, but is only connected to a field of size (s, s) in width and height of the input.
A convolution layer with input shape (d, h, w) produces an output with shape (f, $h - s + 1$, $w - s + 1$).

Subsampling layer

A subsampling layer reduces the width and height of the image representation, but conserves its depth. We use maximum subsampling with a stride of size (2, 2), which means each neuron of the subsampling layer creates the maximum value of 4 input values as output. By doing so, a subsampling layer with input shape (d, h, w) produces an output with shape (d, $h/2$, $w/2$).

2.1.2 Dense networks

We compare three networks which only have dense layers.

Dense300

The first of these networks has only one single dense layer with 300 neurons. It is meant to be a rather simple baseline, that should be beaten by the other networks.

Dense1000

This network has one dense layer with 1000 neurons. By comparing its performance with the Dense300-network, we can see whether an increase of the number of neurons leads to better results.

Dense300-100

This network has two sequential dense layers with 300 and 100 neurons. By comparing its performance with the Dense300-network, we can see whether an increase of the depth[1] of the network leads to better results.

2.1.3 Lenet1

This network architecture is similar to the one named “lenet1” in [LeCun et al.].

[1] Note that when talking about networks the term “depth” refers to the number of layers, but when talking about convolution layers it means the number of filters.

It has the following layers:

layer	output shape
input image shape	(3, 32, 32)
convolution layer with 4 filters of size (5, 5)	(4, 28, 28)
subsampling layer	(4, 14, 14)
convolution layer with 12 filters of size (5, 5)	(12, 10, 10)
subsampling layer	(12, 5, 5)
convolution layer with 10 filters of size (5, 5)	(10, 1, 1)
output layer (dense)	(10, 1, 1)

When using this network on the ZIP data, we had to remove the two subsampling layers, in order to match the smaller width and height of the input images.
For the same reason, the second subsampling layer was removed when dealing with the MNIST dataset.

2.1.4 Lenet5

This network is similar to the one named "lenet5" from [LeCun et al., 1998], which was especially designed for the MNIST dataset.

The structure is the following:

layer	output shape
input image shape	(3, 32, 32)
convolution layer with 6 filters of size (5, 5)	(6, 28, 28)
subsampling layer	(6, 14, 14)
convolution layer with 16 filters of size (5, 5)	(16, 10, 10)
subsampling layer	(16, 5, 5)
convolution layer with 120 filters of size (5, 5)	(120, 1, 1)
dense layer with 84 neurons	(84, 1, 1)
output layer (dense)	(10, 1, 1)

Lenet5 is similar to lenet1, but has convolution layers with more filters and an additional dense layer in the end.

For the CIFAR-10 dataset we used the architecture as described above.
We had to adapt the structure for the MNIST dataset, because [LeCun et al., 1998] had 32x32-pixel versions of images whereas our images are of size 28x28. It was enough to reduce the filter size of the last convolutional layer from (5, 5) to (4, 4) in order to make the network applicable to the smaller input images.
When using it for the ZIP dataset, the filters stayed unchanged but we again had to remove the subsampling layers.

2.2 Results

We kept 1/3 of the training set of each dataset as a validation set and trained the networks on a smaller training set of only 2/3 of its original size. We then evaluated the performance of the networks on the validation sets in order to decide which network architecture is the best one for each of the datasets. Only in the end the chosen network was trained on the full training set and evaluated on the test set.

For the first training phase with the smaller training set we provide a plot that shows how the accuracy on the training set and the accuracy on the validation set evolve over 100 training iterations (i.e. the network sees each training image 100 times).
The plot shows the training accuracies as dots and the validation accuracies as lines.
This evaluation shows us which network performs best and until which iteration the training and validation accuracies increase.

We provide another plot that shows the validation accuracy over the training accuracy for each iteration.
When the points in this plot are close to the diagonal from (0,0) to (1,1), we consider this as a good learning process, because the training accuracy and the validation accuracy increase with the same pace.
When the points range from left to right, the training accuracy increases but the network does not generalize to new data and therefore the validation accuracy stays the same.
When the points move to the right and downwards, this is a sign that overfitting happens, i.e. the training accuracy increases but the validation accuracy becomes worse.

2.2.1 ZIP dataset

See the plots of the training process in Figure 2.
All the five networks perform nearly equally well, only lenet5 is better than the others by a small margin.

network	best valid. acc.	at iteration
dense300	96.95 %	83
dense1000	96.87 %	94
dense300-100	97.28 %	95
lenet1	97.28 %	76
lenet5	**98.39** %	84

2.2.2 MNIST dataset

See the plots of the training process in Figure 3.
We observe that lenet5 behaves best on the MNIST dataset, followed by lenet1.
The dense networks are a little bit worse. The best dense network is dense1000, whereas the others perform nearly equally well.

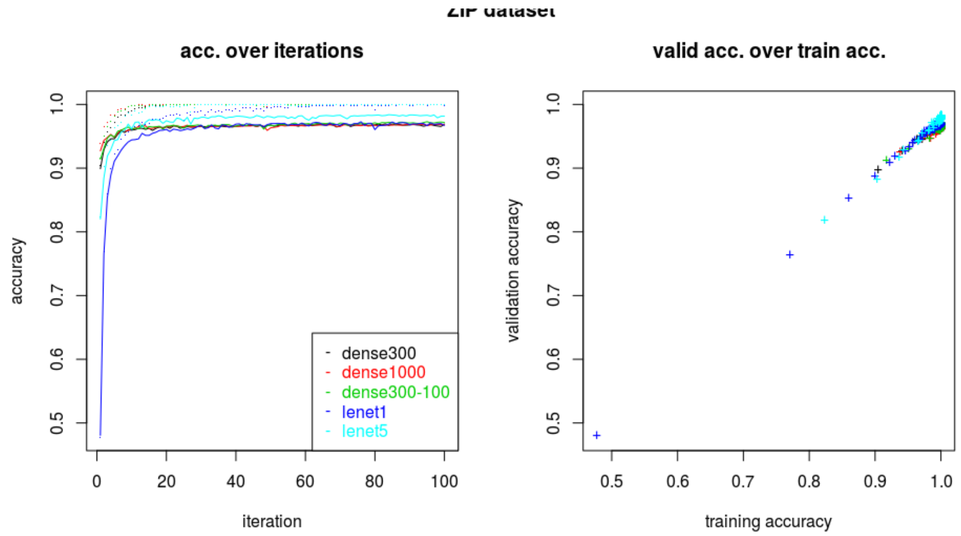


Figure 2: Evaluation of the neural networks on the ZIP dataset.

network	best valid. acc.	at iteration
dense300	98.04 %	35
dense1000	98.23 %	96
dense300-100	98.06 %	45
lenet1	98.65 %	88
lenet5	**99.12** %	87

2.2.3 CIFAR-10 dataset

See the plots of the training process in Figure 4.
We observe that lenet5 beats the other networks by a huge margin, but starts to overfit after 30 iterations.
Lenet1 has a constant learning curve, whereas the dense networks do no longer improve after several iterations.

network	best valid. acc.	at iteration
dense300	50.46 %	61
dense1000	51.16 %	28
dense300-100	50.08 %	46
lenet1	56.15 %	99
lenet5	**61.15** %	37

More filters

Since the validation accuracies on the CIFAR-10 dataset are not very good, we try to

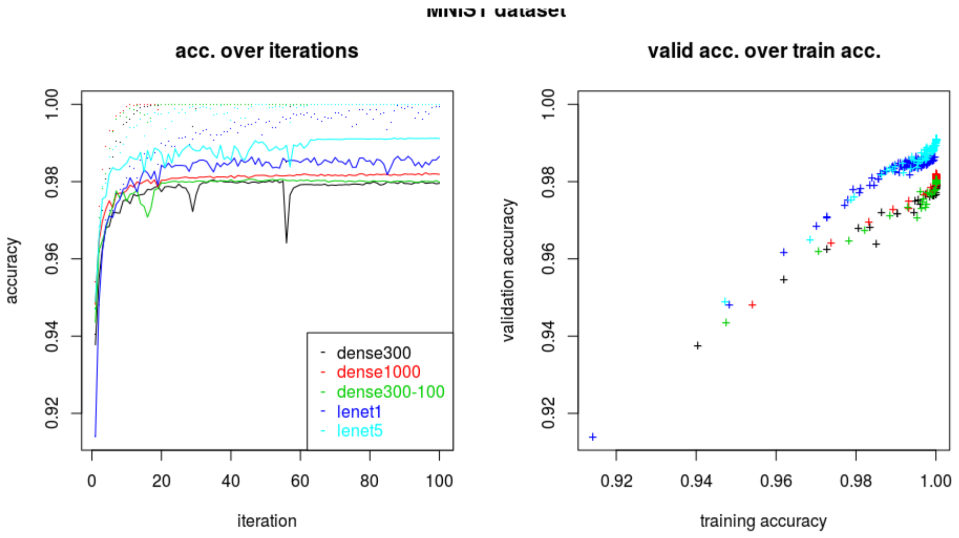


Figure 3: Evaluation of the neural networks on the MNIST dataset.

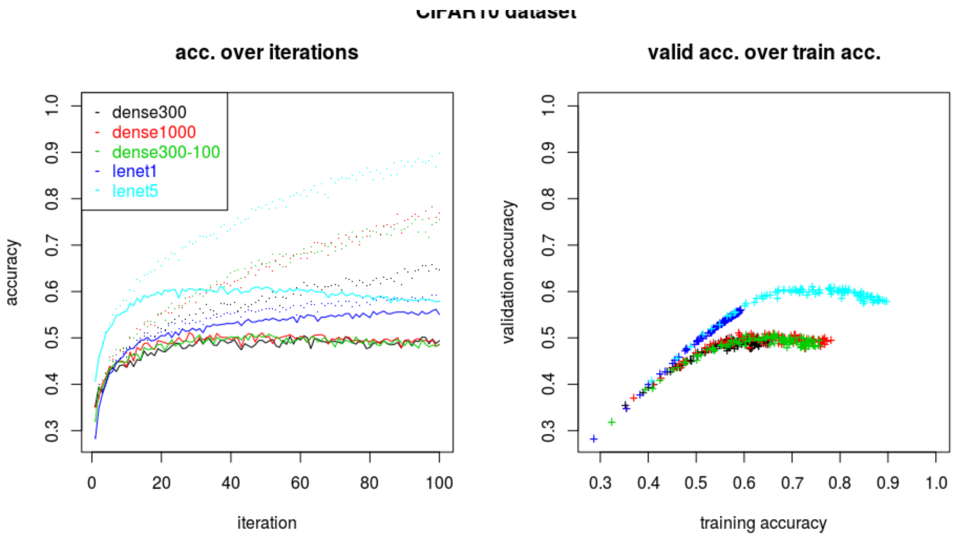


Figure 4: Evaluation of the neural networks on the CIFAR-10 dataset.

improve the lenet5 network by adding more filters to each convolution layer.

After 30 iterations we get the following results:

network(filters)	training accuracy	validation accuracy
lenet5(6,16,120)	71.02 %	60.29 %
lenet5(16,32,120)	84.31 %	65.85 %
lenet5(32,64,256)	97.42 %	**67.15** %

We observe that adding more filters increases the validation accuracy by some points, but has a higher effect on the training accuracy.

2.3 Final results and state of the art

Finally, we trained the models selected above on the whole training set of each dataset and compare the test accuracy with the state of the art results listed on [Rodrigo Benenson's homepage].

network(filters):iterations @ dataset	test accuracy	state of the art
lenet5(6,16,120):40 @ ZIP	94.71 %	-
lenet5(6,16,120):70 @ MNIST	98.96 %	99.79 %
lenet5(32,64,256):30 @ CIFAR-10	68.12 %	96.53 %

3 Multiclass Kernel SVMs

3.1 Multiclass SVM strategies

3.1.1 spoc-svc

This method for multi-class classification is based on a new definition of the margin. This generalized notion of margin gives to the method the ability to learn a multi-class classifier simply by solving a constrained optimization problem with a quadratic objective function.
See [Crammer and Singer, 2001] for more details.

3.1.2 kbb-svc

In this case, we extend the binary SVM optimisation problem by adding new decision variables and new constraints. This method implies that the size of the optimisation problem is proportional to the number categories, which can be a problem.
See [Weston and Watkins] for more details.

3.1.3 one-vs-all approach

The idea is to train one model for each class, that predicts whether a sample belongs to the class or not. We use KSVM for regression in order to avoid conflicts, which would arise when we would only use binary classification (i.e. multiple models or no model could predict “yes” for the same sample). Then we can finally predict the class where the model was the most confident (i.e. the regression output is the highest).

3.1.4 tree-based approach

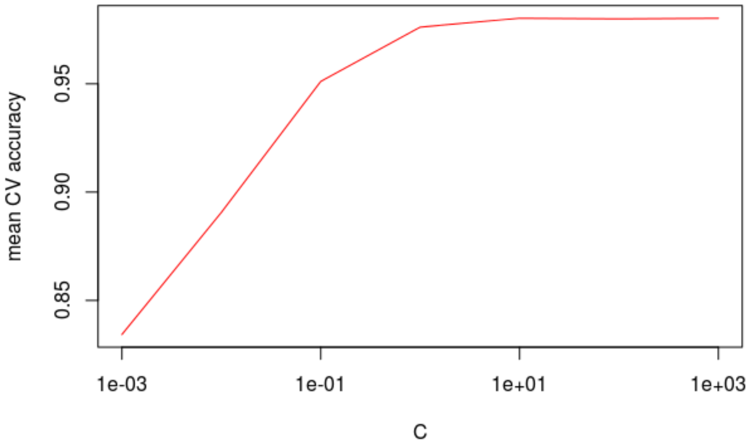
3.2 Parameter optimization and results

3.2.1 spoc-svc

3.2.2 kbb-svc

3.2.3 one-vs-all approach

We decided to use the RBF-kernel and the same C-parameter for all of the 10 models, in order to reduce the parameter searchspace.
The following plot shows the mean validation accuracy on the ZIP dataset for different C-parameters, when doing a 10-fold crossvalidation.



On the ZIP dataset, the best crossvalidation result was achieved with $C = 1000$, when comparing 7 different values which were varying from 0.001 to 1000 in a logarithmic scale.

Due to a lack of memory and time constraints, we decided to reduce the training set of the MNIST dataset to 30% of its original size, which means that 18,000 training images remained.
On this new training set the best crossvalidation result was achieved with $C = 100$.

With the parameters chosen as described above, we got the following accuracies on the test sets:

dataset[training set size]	test accuracy
ZIP[7291]	95.31 %
MNIST[18000]	97.97 %
CIFAR-10[5000]	XX.XX %

3.2.4 tree-based approach

4 Conclusions

See the following table for our final accuracy rates on each dataset. For the model selection that was done for each dataset, refer to the sections above.

dataset	CNN	one-vs-all SVMs	state of the art
ZIP	94.71 %	95.31 %	-
MNIST	98.96 %	97.97 %	99.79 %
CIFAR-10	68.12 %	xx.xx %	96.53 %

4.1 Difficulties of the datasets

Although the type of images are the same for the ZIP and MNIST datasets, they differ in the image size and training set size. The smaller images and the smaller training set makes the ZIP dataset harder to predict.
Using data augmentation and therefore increasing the size of the training set would possibly improve the results.
Although the CIFAR-10 dataset has coloured images and therefore each sample contains more information, the variety of images makes it a hard task.

4.2 Comparison of Neural Networks and SVMs

We conclude that our multiclass SVM strategies can compete with the CNNs.
On ZIP and MNIST we had close to equal results when using SVMs and CNNs.
On both datasets our results are not far away from the state of the art techniques.
For CNNs it was hard to find a good network architecture, especially for the CIFAR-10 dataset.
Fine-tuning SVMs also takes a lot of time, but a simple grid-search can already lead to good results.

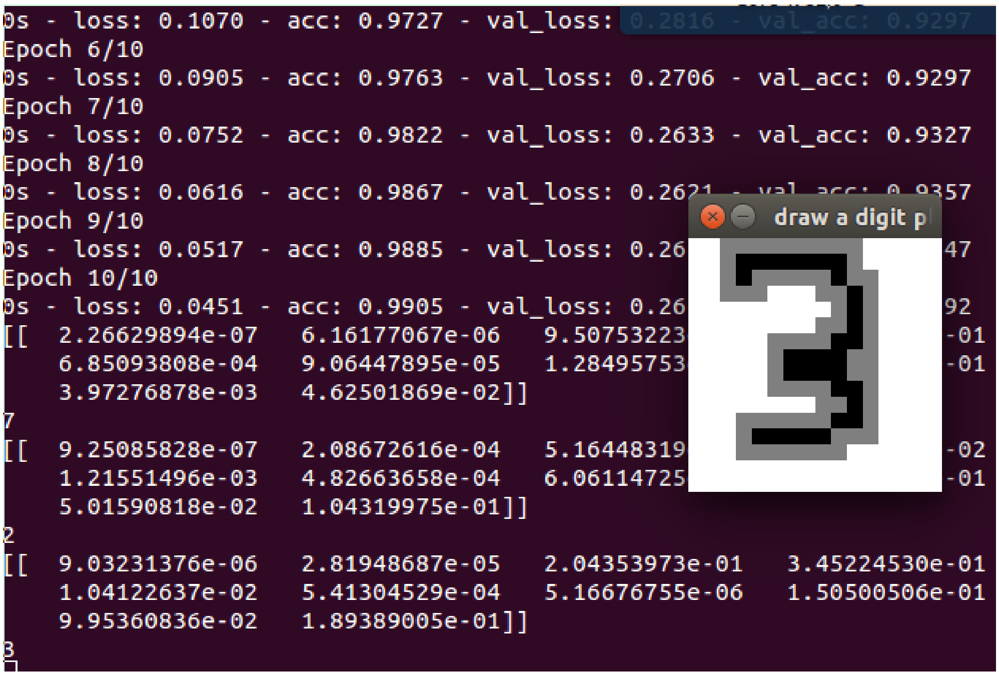
4.3 Problems we encountered

We had to reduce the sizes of the MNIST and CIFAR-10 datasets when training SVMs, because we realized that the R-implementations need a lot of RAM and modelling the SVMs sometimes took several hours.

The training time of the bigger CNNs (e.g. lenet5) did also run for several hours, at least for the MNIST and CIFAR-10 datasets which are quite big.

5 Application

5.1 Description



```
0s - loss: 0.1070 - acc: 0.9727 - val_loss: 0.2816 - val_acc: 0.9297
Epoch 6/10
0s - loss: 0.0905 - acc: 0.9763 - val_loss: 0.2706 - val_acc: 0.9297
Epoch 7/10
0s - loss: 0.0752 - acc: 0.9822 - val_loss: 0.2633 - val_acc: 0.9327
Epoch 8/10
0s - loss: 0.0616 - acc: 0.9867 - val_loss: 0.2621 - val_acc: 0.9357
Epoch 9/10
0s - loss: 0.0517 - acc: 0.9885 - val_loss: 0.26                   47
Epoch 10/10
0s - loss: 0.0451 - acc: 0.9905 - val_loss: 0.26                   92
[[  2.26629894e-07   6.16177067e-06   9.50753223                  -01
    6.85093808e-04   9.06447895e-05   1.28495753                  -01
    3.97276878e-03   4.62501869e-02]]
7
[[  9.25085828e-07   2.08672616e-04   5.16448319                  -02
    1.21551496e-03   4.82663658e-04   6.06114725                  -01
    5.01590818e-02   1.04319975e-01]]
2
[[  9.03231376e-06   2.81948687e-05   2.04353973e-01   3.45224530e-01
    1.04122637e-02   5.41304529e-04   5.16676755e-06   1.50500506e-01
    9.95360836e-02   1.89389005e-01]]
3
```

The application allows the user to draw a digit and get the prediction which is done by the neural network that was trained on the MNIST training set.

5.2 Manual

You can run the program *interactive.py* using python3. It requires the libraries pygame and keras.

After starting the program, wait until the neural network model is loaded. You can start drawing when the background became white.
Draw a number with the mouse and press “p” to get a prediction or “n” to get a blank board again.

References

[LeCun et al., 1998] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, *Gradient-Based Learning Applied to Document Recognition*, Proc. of the IEEE, November 1998.

[LeCun et al.] Y. LeCun et al., *Learning algorithms for classification: a comparison on handwritten digit recognition*

[Alex Krizhevsky's homepage] https://www.cs.toronto.edu/˜kriz/cifar.html

[Yann LeCun's homepage] http://yann.lecun.com/exdb/mnist/

[Crammer and Singer, 2001] K.Crammer, Y. Singer, *On the Algorithmic Implementation of Multiclass Kernel-based Vector Machines*, pp. 265-292, Journal of Machine Learning Research 2, 2001.

[Weston and Watkins] J. Weston, C.Watkins, *Support Vector Machines for Multi-Class Pattern Recognition*

[Rodrigo Benenson's homepage] http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results