TECHNIQUES FOR AVOIDING MODEL OVERFITTING ON SMALL DATASET

Lukas Kratochvila
Doctoral Degree Programme (2), FEEC BUT
E-mail: kratochvila@feec.vutbr.cz

Supervised by: Karel Horak
E-mail: horak@feec.vutbr.cz

Abstract: Building a deep learning model based on small dataset is difficult, even impossible. To avoiding overfitting, we must constrain model, which we train. Techniques as data augmentation, regularization or data normalization could be crucial. We have created a benchmark with a simple CNN image classifier in order to find the best techniques. As a result, we compare different types of data augmentation and weights regularization and data normalization on a small dataset.

Keywords: Deep Learning, Dataset size, Overfitting, Data Augmentation, Regularization, Image Classification, Batch Normalization, Data Normalization

1 INTRODUCTION

Recent deep learning algorithms deals with several problems. One of them is overfitting on the training dataset. Overfitting is a situation, when the model loss on the validation set (which represents the test set during training) starts increasing, but the loss on the training set still decreases. This situation is called memorising of the training dataset because we lose the generalization property of the model and we are only remembering the training set. The example of the overfitting is shown in Fig. 1.

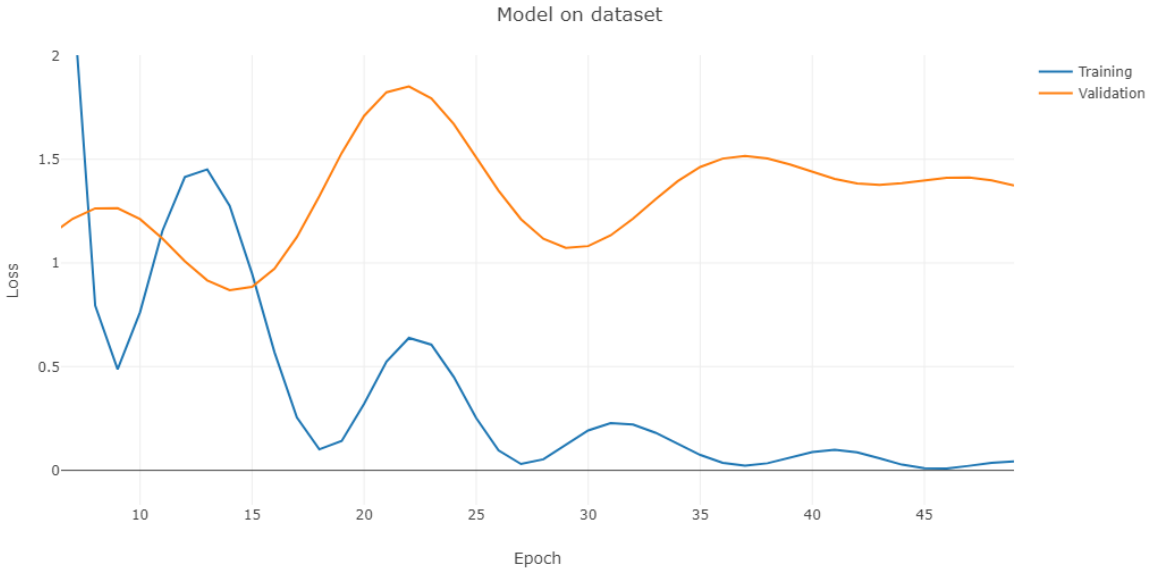

Figure 1: Example of model overfitting on training dataset.

2 METHODS

There are several methods preventing overfitting. The first one considers dataset and model size. When the model overfits, we probably have too intricate model, too small dataset, or both. This approach refers to Occam's Razor principle. This principle states "Entia non sunt multiplicanda praeter necessitatem"[1]. In deep learning models this principle says that we need the simplest model, which can represent the data. We can use statistics to find the right size of the model. The Eq. 1 represents an a posteriori probability for model M_i. Rasmussen and all.[2] suggest that a posteriori probability can show needed model complexity.

$$P(M_i|Y) = \frac{P(Y|M_i)P(M_i)}{P(Y)} \tag{1}$$

When we do not have enough data, we can try to create more with data augmentation. Augmentation is a process, which creates new data samples with transformation which preserves annotation. We can use two types of data augmentation. The first type is geometrical transformation, e.g. rotation, cropping, or flipping. The second kind is photometric transformation, e.g. colour shifting, brightness, or contrast adjustment.

The second approach to create more data is regularization. The idea of regularization came from maximum a posteriori (MAP) estimation and therefore minimization of the objective function Eq. 2 as we want to eliminate noise.[3]

$$J(f) = \sum_{i=1}^{N} [y_i - f(x_i)]^2 + \lambda\phi[f] \tag{2}$$

where λ refers to regularization parameter, y_i is the target and $f(x_i)$ is our model. This parameter controls the trade-off between the noise level and strength of priory assumption.[3] Term $\phi[f]$ refers to energy function and can be expressed as Eq. 3.[4]

$$\phi[f] \Leftrightarrow \sum_{j=1}^{M} |w_j|^q \leq \eta \tag{3}$$

The η refers to the constraint and with q we can choose, which regularization we want, i.e. $q = 1$ means Lasso regularization.

The third important approach is normalization. We consider three methods. The first one is data normalization to the interval $[0,1]$. The second is global contrast normalization, i.e. subtract mean across pixels and normalize by scale, which is either the standard deviation or L1-norm across pixels. And the third is batch normalization. This is a technique normalizes data in mini-batch. For d-dimensional input $x = (x^{(1)}...x^{(d)})$, the normalization is performed over dimension as show Eq. 4.[5]

$$x_{new}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{Var[x^{(k)}]}} \tag{4}$$

3 TOOLS AND LIBRARIES

For the purpose of increasing dataset size, when refering to the first approach to deal with model overfitting, we have collected tools and libraries for this task.

The authors in [6] created a CNN and compared some photometric and geometric data augmentations. Namely, authors compare flipping, rotating, cropping, color jittering, edge enhancement, and fancy PCA. Colour jittering and image cropping are the two most successful processes.

In the [7] authors create a rigorous comparison of shallow and deep learning methods. For the boost, they use geometric data augmentation.

Useful Python-base implementation for augmentation is *Imgaug*.[8]

3.1 AUTOMATED AUGMENTATION

AutoAugment [9] is an approach for dealing with small dataset. It uses reinforcement learning for finding the best augmentation policy. The algorithm can work in two ways: direct or transfer. The direct way finds a new policy for the current dataset. On the other side transfer way transfers learned policy to new dataset. Direct AutoAugment could be very slow, therefore Sungbin et al. developed faster AutoAugment named FastAutoAugment.[10]

An alternative could be the algorithm from Ratner et al., which uses General adversarial network for finding ideal augmentation policy.[11]

3.2 REGULARIZATION

Some of the regularization techniques are implemented in deep learning frameworks, e.g. in Tensorflow are implemented layer regularizers, which are written for this purpose. In PyTorch is implemented L2-regularization. Different regularizations must be hardcoded.[12]

4 EXPERIMENTAL SETUP

We have constructed a benchmark to consider dataset size, regularizations and preprocessing w.r.t model performance. This benchmark consist of 24 experiments, i.e. we have 6 diferrent approach for 4 datasets. The approches are *normalization*(N), *global constrain normalization*(GC), *L2-regularization*(R2), *batch normalization*(BN), *application of ImageNet Policy*(AA) and *application of CIFAR10 Policy*(AC).

4.1 DATASET

As dataset we have 1225 images with resolution 640*x*480px divided into 4 classes. The classes are named *in order* with 474 images, *broken* with 465 images, *bad colour* with 158 images and *foreign object* with 128 images. One example of each class is shown on Fig. 2. From this base of the dataset we construct 4 datasets with 100, 200, 300 and 400 samples. In each dataset, the label occurance is balanced, i.e. in the dataset with 400 samples are 100 samples from each class, rest of the images from base dataset are in test dataset. For the validation dataset the first 100 samples from each class of the test dataset were chosen if possible.

Figure 2: From left to right classes examples: *in order*, *broken*, *bad colour* and *foreign object*.

4.2 ARCHITECTURE

We choose a simple classification network for the evaluation. The network is created in the PyTorch library. The summary is shown in List. 1. Layer *BatchNorm* (3,6,9) is used only when specified. For each convolution and fully connected layer (Linear in the listings) we manually initialize weights.

```
INFO:MY_LeNetBN:Trainable parameters: 266592
INFO:MY_LeNeBNt:MY_LeNetBN(
(1) (conv1):Conv2d(3,32,kernel_size=(5,5),stride=(1,1),padding=(2,2),bias=True)
(2) (pool1):MaxPool2d(kernel_size=2,stride=2,padding=0,dilation=1,ceil_mode=False)
(3) (bn1):  BatchNorm2d(32,eps=0.0001,affine=False,track_running_stats=True)
(4) (conv2):Conv2d(32,64,kernel_size=(5,5),stride=(1,1),padding=(2,2),bias=True)
(5) (pool2):MaxPool2d(kernel_size=2,stride=2,padding=0,dilation=1,ceil_mode=False)
(6) (bn2):  BatchNorm2d(64,eps=0.0001,affine=False,track_running_stats=True)
(7) (conv3):Conv2d(64,128,kernel_size=(5,5),stride=(1,1),padding=(2,2),bias=True)
(8) (pool3):MaxPool2d(kernel_size=2,stride=2,padding=0,dilation=1,ceil_mode=False)
(9) (bn3):  BatchNorm2d(128,eps=0.0001,affine=False,track_running_stats=True)
(10)(fc1):  Linear(in_features=2048, out_features=4, bias=True)
```

Listing 1: Model summary

In the input, images are rescaled to 32*x*32px. The output is the same as the number of classes. Every experiment was trained for 100 epochs after 50 epochs was learning rate multiplied by 0.1. The rest of the hyperparameters are listed in Tab. 1.

Optimizer	Init learning rate	batch size	Activation function	Weight Initialisation
Adam	0.001	128	Leaky ReLu	Xavier

Table 1: Network hyper-parameters

4.3 RESULTS

For evaluation, we chose the average Top-1 accuracy metric. This metric consider only the best result and average throughout all classes. Results on the test dataset are shown in Tab. 2. The best results though experiments are bold.

	Test average Top-1 accuracy [%]			
Experiment	100	200	300	400
baseline	74.22	**88.02**	87.35	**89.94**
N	**74.49**	77.46	78.46	81.45
GC	70.04	83.71	**89.08**	89.09
BN	72.89	78.44	79.46	82.42
R2	74.09	83.90	87.24	89.33
AA	64.53	68.98	72.86	78.30
AC	63.29	70.54	77.08	76.00

Table 2: Experiment results: columns refer to the dataset size and rows refer to experiments results.

The Fig. 3 shows training and validation loss for all datasets at baseline. We can see strong overfitting on the training dataset, i.g. the training loss decreasing, but the validation loss stagnates or starts to raise. The Fig. 4 shows training and validation losses for all experiments on dataset with 400 samples.

4.4 DISCUSSION

Data augmentation methods does not reach the best results, but from comparison of the training and validation loss on the Fig. 4, we can see that the margin between the losses is reduced compare to

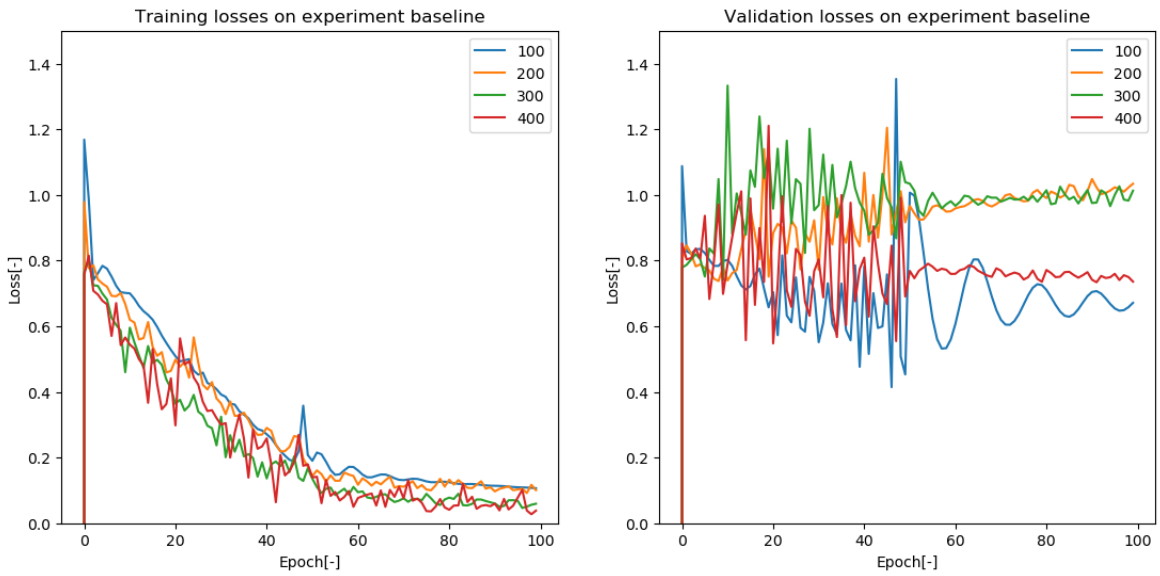

Figure 3: Training performance on baseline for all four datasets

the baseline. From this point of view the data augmentation methods improves model generalization ability. For the *batch normalization (BN)*, we can see the same reduction as for the data augmentation.

On the Fig. 4 the validation loss for experiment *normalization (N)* and *global constrain normalization (GC)* achieves the best results on validation dataset. This is not seen from Tab. 2 and it could be caused by small size of testing dataset.

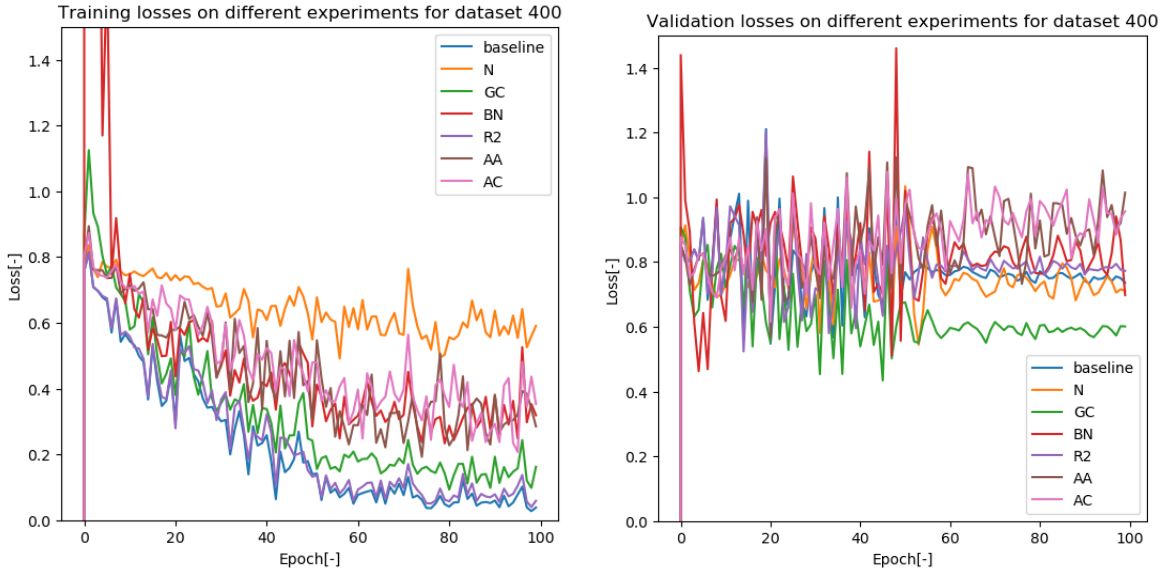

Figure 4: Training performance on all experiments for dataset with 400 samples.

5 CONCLUSION

This paper discuss some of the methods for avoiding overfitting on small training dataset. Focus is on three differrent approaches: data augmentation, regularization and data normalization. Necessary

theory is reminded with examples of existing implementations. The main acquisition of this work is benchmark situated for very small dataset. Based on the results, we conclude that data normalization methods are crucial for preventing overfitting on small dataset. Discussion in subsection 4.4, leads us to conclusion that the all methods mentioned above can help with overfitting on the training dataset.

In future work, we would like to extend developed benchmark and develop metric for ranking datasets and their augmentation versions. This metric could amplify evolving models. Next step will be creation of automated library like [9], [10] or [11] for easy use.

6 ACKNOWLEDGEMENT

The completion of this paper was made possible by the grant No. FEKT-S-20-6205 - "Research in Automation, Cybernetics and Artificial Intelligence within Industry 4.0" financially supported by the Internal science fund of Brno University of Technology.

REFERENCES

[1] W. M. Thorburn, "The myth of occam's razor," *Mind*, vol. 27, no. 107, pp. 345–353, 1918.

[2] C. E. Rasmussen and Z. Ghahramani, "Occam's razor," *Advances in neural information processing systems*, pp. 294–300, 2001.

[3] F. Girosi, M. Jones, and T. Poggio, "Regularization theory and neural networks architectures," *Neural computation*, vol. 7, no. 2, pp. 219–269, 1995.

[4] M. Mishra, "Regularization: An important concept in machine learning." `https://towardsdatascience.com/regularization-an-important-concept-in-machine-learning-5891628907ea`, 2018. [Online; accessed 10-Mar-2021].

[5] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," in *International conference on machine learning*, pp. 448–456, PMLR, 2015.

[6] L. Taylor and G. Nitschke, "Improving deep learning using generic data augmentation," *CoRR*, vol. abs/1708.06020, 2017.

[7] K. Chatfield, K. Simonyan, A. Vedaldi, and A. Zisserman, "Return of the devil in the details: Delving deep into convolutional nets," *CoRR*, vol. abs/1405.3531, 2014.

[8] A. B. Jung, "imgaug." `https://github.com/aleju/imgaug`, 2021. [Online; accessed 1-Jan-2021].

[9] E. D. Cubuk, B. Zoph, D. Mane, V. Vasudevan, and Q. V. Le, "Autoaugment: Learning augmentation strategies from data," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[10] S. Lim, I. Kim, T. Kim, C. Kim, and S. Kim, "Fast autoaugment," *CoRR*, vol. abs/1905.00397, 2019.

[11] A. J. Ratner, H. R. Ehrenberg, Z. Hussain, J. Dunnmon, and C. Ré, "Learning to compose domain-specific transformations for data augmentation," *Advances in neural information processing systems*, vol. 30, p. 3239, 2017.

[12] E. Stevens, L. Antiga, and T. Viehmann, *Deep Learning with PyTorch*. Manning Publications, 2020.