Bachelor practical: Report

Marco Unternährer

December 31, 2015

1 Introduction

Training a spiking neural network is a challenging task. Neuromorphic vision researchers have to create spiking datasets since there is a lack of such datasets publicly available . One method for converting static image datasets into neuromorphic vision datasets was proposed by moving a sensor in front of an image [14].

Another approach for training a spiking neural network was developed by researchers at the Institute of Neuroinformatics [7]. A non-spiking neural network is trained by using a regular static image dataset like MNIST or Caltech-101. The resulting weights of the trained neural network can then be used to create a spiking neural network. One constraint for this to work is that units must have zero bias.

The task for this bachelor practical was to train a convolutional neural network on the Caltech-101 dataset.

2 Image Classification on Caltech-101

This section will give a short introduction to the Caltech-101 dataset, the used methods and techniques.

2.1 Caltech-101

Caltech-101 is a dataset of pictures of objects, which was collected in September 2003 by Fei-Fei et al. [9] at the California Institute of Technology. Most categories have about 50 images, but the number ranges from 40 to 800 per category [9]. Caltech-101 contains a total of 9'146 images, split between 101 distinct object categories (faces, watches, ants, pianos, etc.) and a background category [1]. The size of each image is roughly 300 x 200 pixels [9].

Unlike other datasets, Caltech-101 doesn't have a predefined split between train and test images. Fei-Fei et al. [9] suggest to use a fixed number of images per category for training and testing. Due to the fact that categories have vastly different number of images, the accuracy per category can also vary a lot. On an additional note, [9] pointed out that if a fixed number of images per category is chosen, then the overall error rate should be reported. However, if all the available images are being used for testing, then the average error rate across categories should be reported.

The state-of-the-art classification on the Caltech-101 dataset has an accuracy of 91.44% [10].

2.2 Convolutional Neural Network

A convolutional neural network (CNN) is a type of feed-forward artificial neural network. CNNs are biologically-inspired variants of multilayer perceptrons [2]. From Hubel and Wiesel's early work on the cat's visual cortex [11], it is known that the visual cortex contains a complex arrangement of cells [2]. These cells are sensitive to small sub-regions of the visual field, called a receptive field [2]. The sub-regions are tiled to cover the entire visual field. These cells act as local filters over the input space and are well-suited to exploit the strong spatially local correlation present in natural images. Additionally, two basic cell types have been identified: Simple cells respond maximally to specific edge-like patterns within their receptive field. Complex cells have larger receptive fields and are locally invariant to the exact position of the pattern [2].

Like their biological relatives, units in a CNN take advantage of the 2D structure of the input data (images or other 2D input such as a speech signal). This is achieved with local connections and tied weights followed by some form of pooling which results in translation invariant features [4].

2.3 Batch Normalization

Stochastic gradient descent (SGD) and its variants are proven to be an effective way of training deep networks [12]. However, they require careful tuning of the model-hyperparameters, specifically the learning rate used in optimization, as well as the initial values for the model parameters [12]. Inputs to each layer are affected by parameters of all preceding layers, which makes training highly complicated [12]. Thus, the smallest changes to the network amplify as the network becomes deeper [12]. The problem seems to be the change in the distribution of layer's inputs, because the layers need to continuously adapt to the new distribution [12]. When the input distribution to a learning system changes, it is said to experience covariate shift [15].

Batch normalization is a new mechanism that takes a step towards reducing internal covariate shift, and in doing so dramatically accelerates the training of deep neural nets [12]. It accomplishes this via a normalization step that fixes the means and variances of layer inputs [12]. Simplifications of the idea have to be made in order to accommodate mini-batches in stochastic gradient training: each mini-batch produces estimates of the mean and variance of each activation. For completeness, the batch normalization transform algorithm by Ioffe and Szegedy [12] can be seen in algorithm 1.

For convolutional layers, the normalization also has to obey the convolutional property – so that different elements of the same feature map, at different locations, are normalized in the same way [12]. To achieve this, we jointly normalize all the activations in a mini-batch, over all locations [12].

Input : Values of x over a mini-batch: $B = \{x_{1..m}\}$;
Parameters to be learned: γ, β
Output: $\{y_i = BN_{\gamma,\beta}(x_i)\}$

$$\mu_B \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_B^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2 \qquad \text{// mini-batch variance}$$

$$\hat{x_i} \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\hat{x_i} + \beta \equiv BN_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Algorithm 1: Batch Normalizing Transform, applied to activation x over a mini-batch

2.4 Cyclical Learning Rates

Bengio [5] says that the learning rate is the most important hyper-parameter to optimize and if "there is only time to optimize one hyper-parameter and one uses stochastic gradient descent, then this is the hyper-parameter that is worth tuning". It is well known that too small a learning rate will make a training algorithm converge slowly while too large a learning rate will make the training algorithm diverge [20]. Hence one must experiment with a variety of learning rates and schedules (i.e., the timing of learning rate changes) [17].

The conventional wisdom is that the learning rate should be a single value that monotonically decreases during the training [17]. However, Smith demonstrates the surprising phenomenon that increasing the learning rate is overall beneficial and thus proposes to let the global learning rate vary cyclically within a band of values rather than setting it to a fixed value [17].

3 Experimental Setup

3.1 Data Pre-processing & Data Augmentation

Since the heights and widths of the Caltech-101 images vary, all images were scaled to a common size. Like Orchard et al. [14], each image was resized to be as large as possible while maintaining the original aspect ratio and ensuring that width does not exceed 240 pixels and height does not exceed 180 pixels.

After running a few experiments with a fairly deep architecture, it became clear, that the roughly 9k images of the Caltech-101 dataset were not enough to accomplish a decent test accuracy. Starting from the original dataset, data augmentation was pursued by randomly transform each picture ten times, and thus leading to approximately 100k images (including the original dataset).

Using an image data generator [6], we ended up with the following augmentation parameters:

- rotation: random with angles in the range 0° to 20°
- translation: random with shift between 0% and 20% of total width/height
- flipping: horizontal/vertical, yes or no (Bernoulli)
- zooming: random zoom axis by +/- 20%

The created images were then resized again, but with the difference that the images were padded with the edge pixels to fill the needed aspect ratio. The pixel values were normalized to the range of (0, 1) after loading the images. Data standardization, subtracting the mean and dividing by the standard deviation, was also applied.

The generated images are available online and can be downloaded under [19].

3.2 Architecture

Our architectures were inspired by Simonyan and Zisserman [16] and Karpathy and Li [13]. However, due to the fact that the neural network will be converted into a spiking neural network, we had two constraints. The first constraint was that units must not have any bias. The second constraint had to do with performance: Since the network should be able to classify pictures near real-time, we limited the depth of our architecture to three convolutional and two fully connected layers. Obviously, when introducing batch normalization, there are more layers. Those additional layers should not have a high impact on the speed of classification though, since batch normalization is only a linear transformation.

Tables 1 to 3 show the used network architectures.

Layer type	**Parameters**
convolution	128x5x5, stride 2x2
relu	
maxpool	2x2
convolution	256x3x3
relu	
maxpool	2x2
convolution	512x3x3
relu	
maxpool	2x2
full	1024
relu	
full	102
softmax	

Table 1: Base Network Architecture

Layer type	**Parameters**
convolution	128x5x5, stride 2x2
batch normalization	
relu	
maxpool	2x2
convolution	256x3x3
batch normalization	
relu	
maxpool	2x2
convolution	512x3x3
batch normalization	
relu	
maxpool	2x2
full	1024
relu	
full	102
softmax	

Table 2: Network Architecture with Batch Normalization

Layer type	Parameters
convolution	128x5x5, stride 2x2
relu	
maxpool	2x2
dropout	0.35
convolution	256x3x3
relu	
maxpool	2x2
dropout	0.35
convolution	512x3x3
relu	
maxpool	2x2
dropout	0.35
full	1024
relu	
dropout	0.5
full	102
softmax	

Table 3: Network Architecture without Batch Normalization

3.3 Training

The splitting of the generated images into training and test samples was stratified, meaning that each class had 90% in the training set and 10% in the test set. That training set was then further split into 10 folds, such that cross-validation could be applied later on.

Stochastic gradient descent was used as the optimizer, with Nesterov momentum of 0.9 and a learning rate decay of 5e-4. Batch size was 64 most of the time, but was decreased for some architecture configurations due to GPU memory overallocation.

We applied a triangular learning rate schedule in order to speed up training as well as reduce the need for hyper-parameter optimization [17]. In combination with a batch normalized network, the minimum learning rate was 0.001, the maximum learning rate was 0.02 and the step size was set to 8 times the number of iterations in an epoch.

3.4 Implementation Details

The neural network framework of choice was Keras [6]. Keras is a minimalist, highly modular neural networks library, written in Python and running on top of Theano. Being Python-based and especially developed with a focus on enabling fast experimentation, it was a good candidate for this project. Most importantly, Keras seemed to provide almost all needed functionality out of the box. Another relevant aspect, that lead to the decision to use Keras, was that model architectures are created with code. That made rapid prototyping and experimentation feasible.

The networks were trained on a system equipped with a single NVIDIA GTX 970 GPU.

The source code for the entire bachelor practical can be found at [18].

4 Results

After running many preliminary experiments, we have settled for the architectures shown in table 2 (with batch normalization) and table 3 (without batch normalization).

Unfortunately, the high imbalance of the dataset was not considered in the preliminary experiments and appendix A explains the mistakes made.

Apart from creating a good classifier for the Caltech-101 dataset, we had specific experiments to run in mind, which we describe in sections 4.1 to 4.4. Section 4.5 shows the cross-validation of the best performing network, which also respects the needed constraints.

Experiment	**Validation Acc**	**Test Acc**
base line - batch normalization, with zero bias constraint, fig. 1	82.66%	82.41%
batch normalization, without zero bias constraint, fig. 2	82.92%	82.54%
no batch normalization, with zero bias constraint, fig. 3	73.04%	73.98%
no batch normalization, without zero bias constraint, fig. 4	78.09%	78.26%
no data standardization, fig. 5	82.93%	81.86%
increased kernel to 9x9 in first convolutional layer, fig. 6	82.99%	82.23%
triangular learning rate schedule, start with minimum lr, fig. 7	77.91%	77.40%

Table 4: Experiment Results

4.1 Batch Normalization and Zero Bias Constraint

As we can see in figs. 1 and 2, there is basically no difference in loss or accuracy when introducing the zero bias constraint in the case we train with the batch normalization architecture (table 2). However, comparing figs. 3 and 4 indicates that the zero bias constraint has quite an impact on training speed as well as the resulting validation and test accuracies (see table 4).

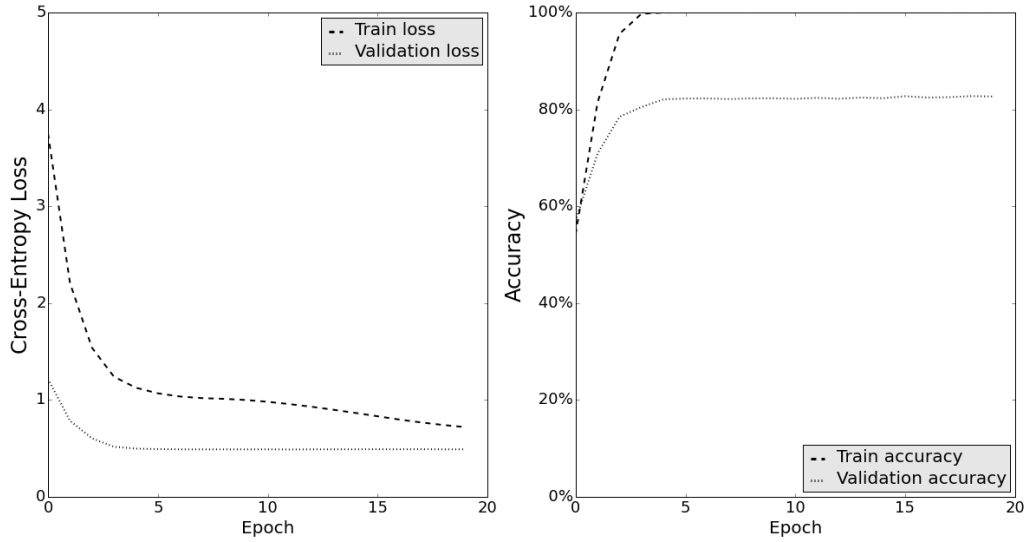

Figure 1: Base line - batch normalization, with zero bias constraint

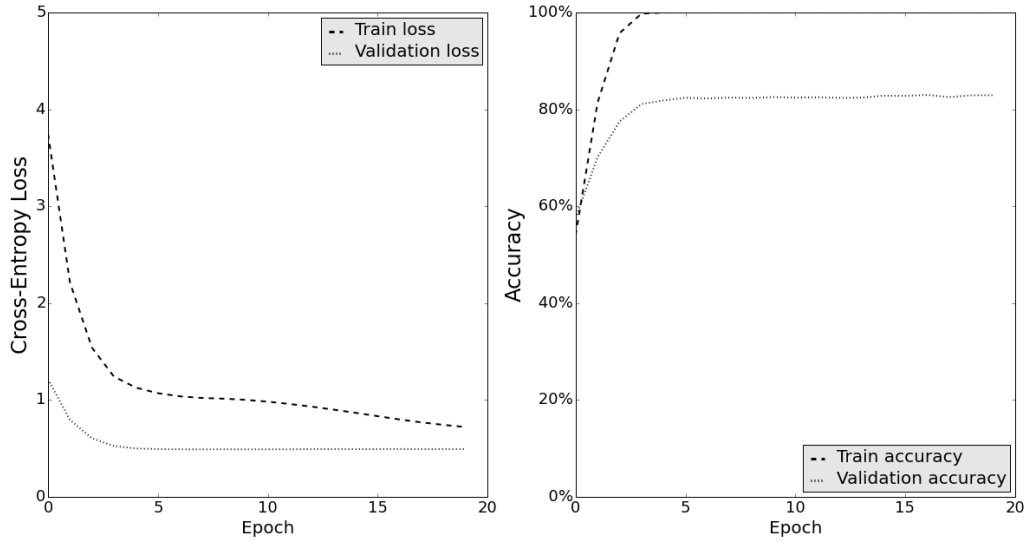

Figure 2: Batch normalization, without zero bias constraint

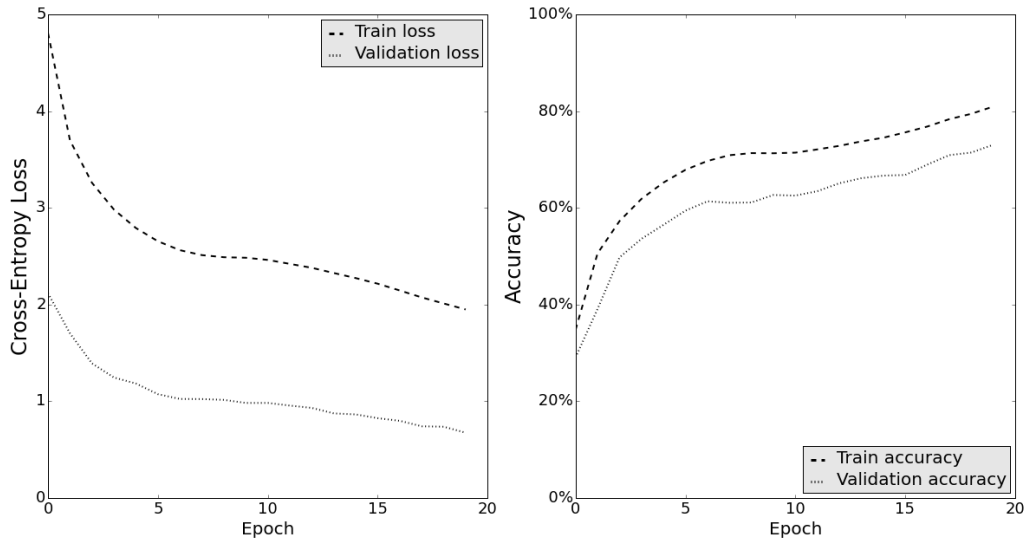

Figure 3: No batch normalization, with zero bias constraint

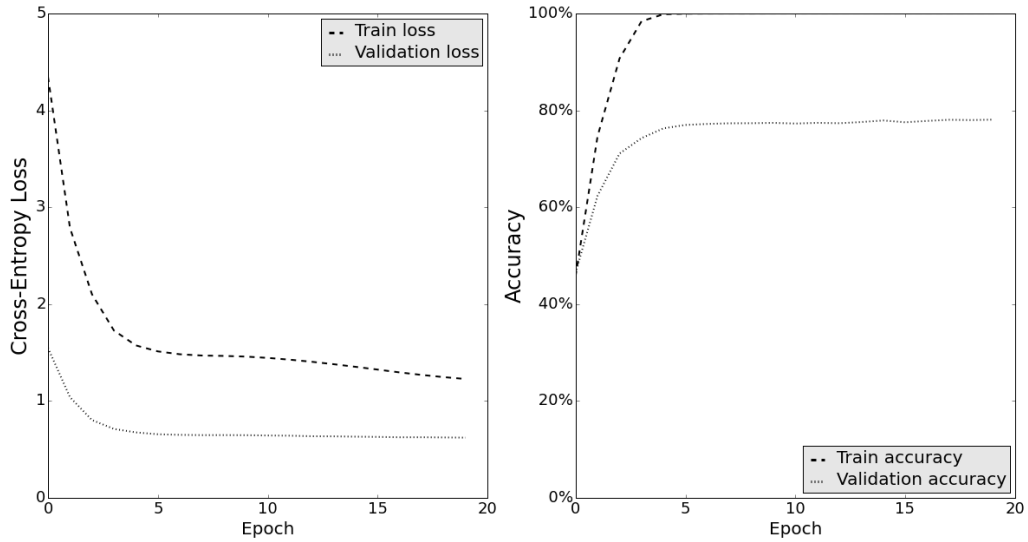

Figure 4: No batch normalization, without zero bias constraint

4.2 Data Standardization

We have trained the same network as the base line, but with the exception that no data standardization was applied. Data normalization, bringing the pixel values into the range of (0, 1), however, was applied. Comparing experiments figs. 1 and 5 shows that there is virtually no difference in the behavior of the networks. Also, from table 4 it can be seen that the validation and test accuracies are within a very small range. This suggests that using batch normalization also eliminates the need to previously standardize the data to have zero mean and unit standard deviation.

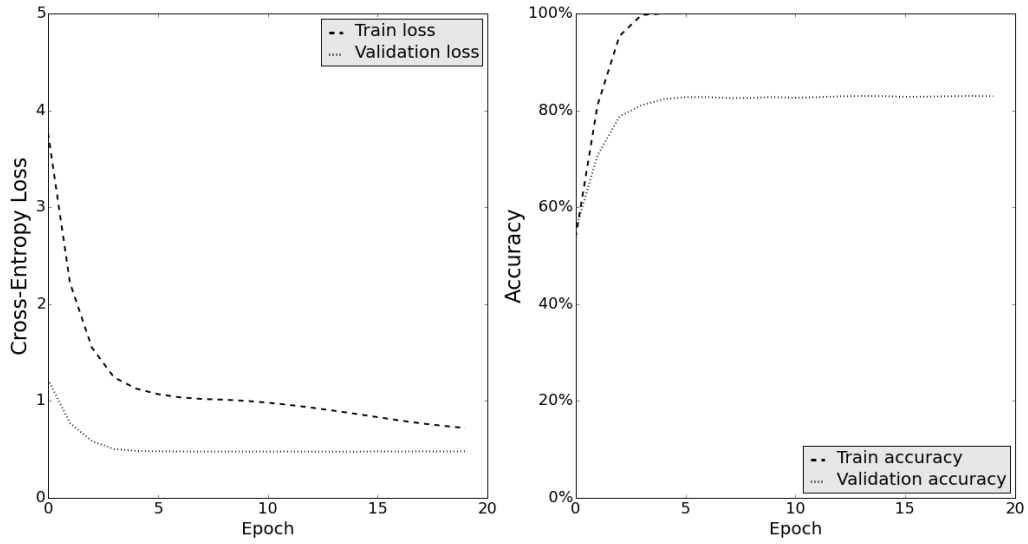

Figure 5: No data standardization

4.3 Convolutional Kernel Size

In this experiment, we simply increased the kernel size in the first convolutional layer from 5x5 to 9x9. Comparing figs. 1 and 6 as well as the accomplished results in table 4 indicates that the kernel size has no major impact on the performance.

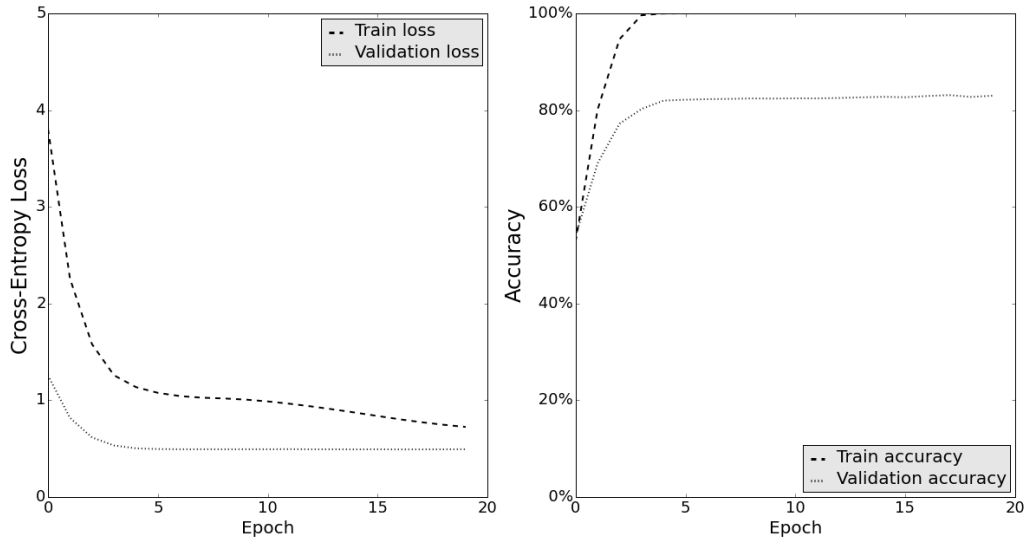

Figure 6: Increased kernel to 9x9 in first convolutional layer

4.4 Triangular Learning Rate

Smith [17] suggests to start the triangular learning rate schedule with the minimum, then increasing to the maximum, and decreasing again. However, one suggestion to accelerate batch normalized networks is to increase the learning rate decay [12]. Comparing figs. 1 and 7 and their according validation and test accuracies (table 4), this suggests that starting out with the maximum learning rate, then decreasing to the minimum is the preferred way when using a triangular learning rate schedule for a batch normalized network.

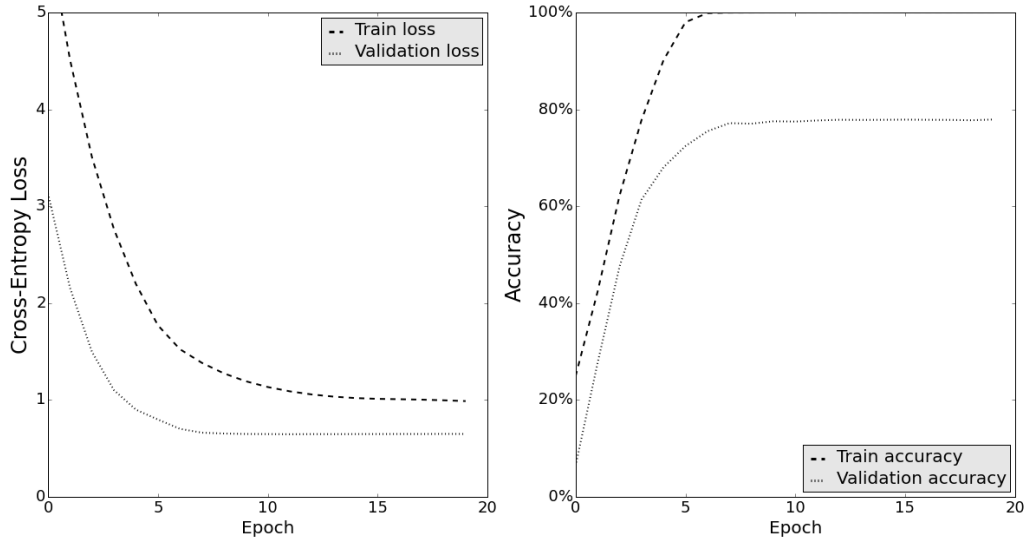

Figure 7: Triangular Learning Rate Schedule, start with minimum lr

4.5 Cross-Validation

We ran the batch normalized network architecture (table 2) on a 10-fold cross-validation. The resulting mean validation accuracy was 82.21%, while the mean test accuracy was 82.10%.

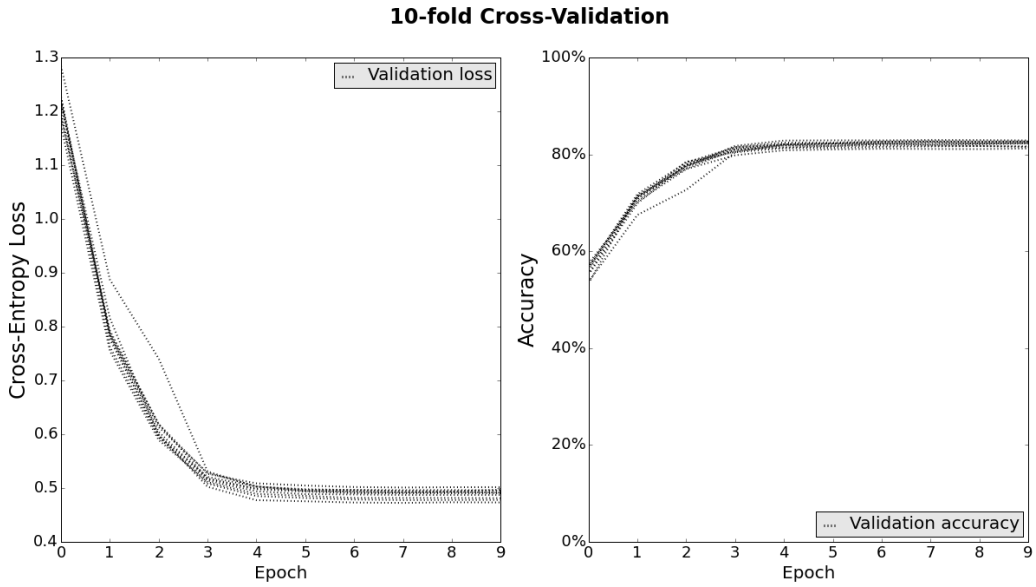

Figure 8: Cross-Validation

5 Conclusion

In order to achieve a decent accuracy on a fairly deep neural network, having enough data is important [8]. If not enough data is available, as in our case, coming up with a way to generate more data is recommended [3].

Creating sensible network architectures is not easy. We relied on the expertise of several authors [13, 16], which lead us to the base architecture described in table 1.

Batch normalization can speed up training significantly [12]. In this work, we have shown that batch normalization is outperforming non-batch-normalized models, especially if we have a zero bias constraint. Hence, we recommend to use batch normalization whenever the zero bias constraint is needed.

Future work should focus on the performance impact of an additional convolutional layer as well as a running more hyper-parameter optimization experiments. These methods will hopefully lead to getting closer to the state-of-the-art accuracy for the Caltech-101 dataset of 91.44% [10].

References

[1] Caltech 101 Wikipedia. URL https://en.wikipedia.org/wiki/Caltech_101. Last visited 2015-12-19.

[2] Convolutional Neural Networks (LeNet). URL http://deeplearning.net/tutorial/lenet.html. Last visited 2015-12-13.

[3] Classifying plankton with deep neural networks. URL http://benanne.github.io/2015/03/17/plankton.html. Last visited 2015-12-31.

[4] Convolutional Neural Network. URL http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/. Last visited 2015-12-13.

[5] Y. Bengio. Practical recommendations for gradient-based training of deep architectures. *arXiv*, jun 2012. URL http://arxiv.org/abs/1206.5533.

[6] F. Chollet. Keras. URL http://keras.io. Version 0.2.0.

[7] P. U. Diehl, D. Neil, J. Binas, M. Cook, S.-c. Liu, and M. Pfeiffer. Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8. IEEE, jul 2015. ISBN 978-1-4799-1960-4. doi: 10.1109/IJCNN.2015.7280696. URL http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=7280696.

[8] P. Domingos. A Few Useful Things to Know about Machine Learning. pages 1–9, 2014. ISSN 00010782. doi: 10.1145/2347736.2347755. URL http://homes.cs.washington.edu/$\sim$pedrod/papers/cacm12.pdfpapers3://publication/uuid/D9E0EB7E-2AE2-4B35-AB6F-4E2CFD3CA6A0.

[9] L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: An incremental Bayesian approach tested on 101 object categories. *Computer Vision and Image Understanding*, 106(1):59–70, apr 2007. ISSN 10773142. doi: 10.1016/j.cviu.2005.09.012. URL http://linkinghub.elsevier.com/retrieve/pii/S1077314206001688.

[10] K. He, X. Zhang, S. Ren, and J. Sun. Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition. *Eccv*, pages 346–361, 2014. ISSN 0162-8828. doi: 10.1109/TPAMI.2015.2389824. URL http://arxiv.org/abs/1406.4729v1.

[11] D. H. Hubel and T. N. Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of Physiology*, 195(1):215–243, mar 1968. ISSN 0022-3751. URL http://www.ncbi.nlm.nih.gov/pmc/articles/PMC1557912/.

[12] S. Ioffe and C. Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *Arxiv*, 2015. URL http://arxiv.org/abs/1502.03167.

[13] A. Karpathy and F.-F. Li. Convolutional Neural Networks. URL http://cs231n.github.io/convolutional-networks/. Last visited 2015-12-13.

[14] G. Orchard, A. Jayawant, G. Cohen, and N. Thakor. Converting Static Image Datasets to Spiking Neuromorphic Datasets Using Saccades. pages 1–15, jul 2015. ISSN 1662-453X. doi: 10.3389/fnins.2015.00437. URL http://arxiv.org/abs/1507.07629.

[15] H. Shimodaira. Improving predictive inference under covariate shift by weighting the log-likelihood function. *Journal of Statistical Planning and Inference*, 90(2):227–244, oct 2000. ISSN 03783758. doi: 10.1016/S0378-3758(00)00115-4. URL http://linkinghub.elsevier.com/retrieve/pii/S0378375800001154.

[16] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Image Recognition. *arXiv*, pages 1–14, sep 2014. URL `http://arxiv.org/abs/1409.1556`.

[17] L. N. Smith. No More Pesky Learning Rate Guessing Games. *Arxiv*, jun 2015. URL `http://arxiv.org/abs/1506.01186`.

[18] M. Unternährer. INI Caltech101 Code, 2015. URL `https://github.com/marcuniq/ini_caltech101`.

[19] M. Unternährer. INI Caltech101 Generated Images, 2015. URL `https://drive.google.com/folderview?id=0B6t56IB_eb6hVFRGOFp3QVpaR2M&usp=sharing`.

[20] M. D. Zeiler. ADADELTA: An Adaptive Learning Rate Method. *arXiv*, page 6, 2012. URL `http://arxiv.org/abs/1212.5701`.

A Imbalanced Data

As we have already mentioned in section 2.1, the dataset is highly imbalanced. Unfortunately, we have discovered only at a very late stage of this bachelor practical, that this imbalance was not considered in the numerous experiments we have already run. Thus, all previous experiments were rendered invalid by this mistake.

We have developed a function, which calculates the accuracy on a per-category basis. The mean of those accuracies were then used as the overall accuracy, as Fei-Fei et al. [9] suggest in order to normalize the performance across categories. Additionally, category weights were calculated and applied while training, such that the effects of the imbalance were less severe.

A.1 Results

Below we show some experiments, which could be of interest, even though the validation and test accuracies are not considering the imbalance of the categories and thus overconfident.

A.1.1 Dropout

Comparing the training and the validation accuracy in fig. 9, it seemed that the network is overfitting (architecture see table 2). To circumvent this, we ran an experiment, where we introduced dropout (with probability 0.5) between the fully connected layer (see fig. 10). No significant increase in the validation and test accuracies was the result of this experiment (see table 5).

Experiment	**Validation Acc***	**Test Acc***
base line, fig. 9	88.08%	88.20%
dropout (0.5) between fully connected layers, fig. 10	88.44%	88.30%

Table 5: Dropout experiment results - Not considering imbalance of categories

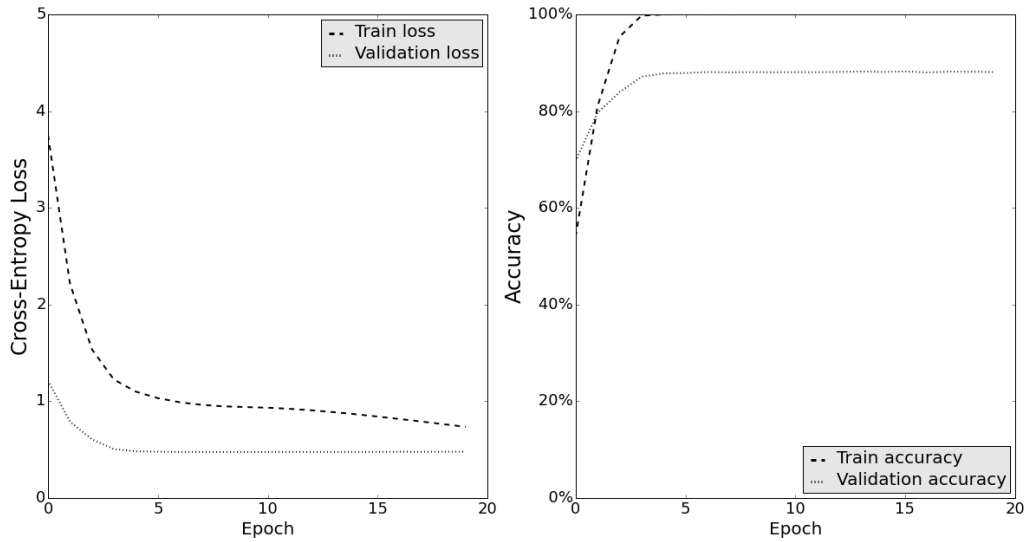

Figure 9: Base line for dropout

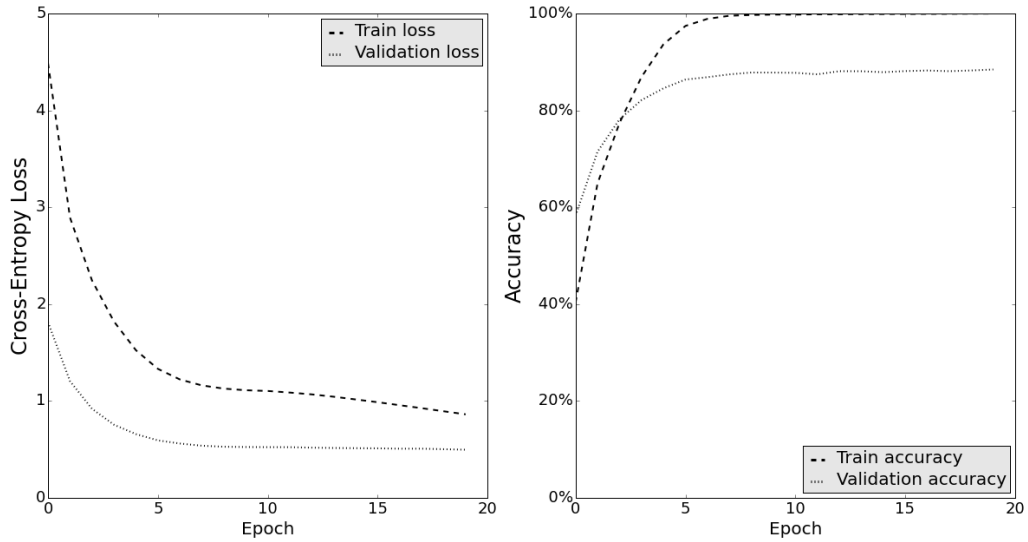

Figure 10: Dropout (0.5) between fully connected layers

A.1.2 Maxpooling

Running a quick experiment (fig. 12), where the maxpooling after the first convolutional layer was removed, has shown that the accuracies turn out much worse (see table 6).

Experiment	**Validation Acc***	**Test Acc***
base line, fig. 11	81.36%	84.30%
no maxpooling after conv1, fig. 12	73.30%	74.27%

Table 6: Maxpooling experiment results - Not considering imbalance of categories

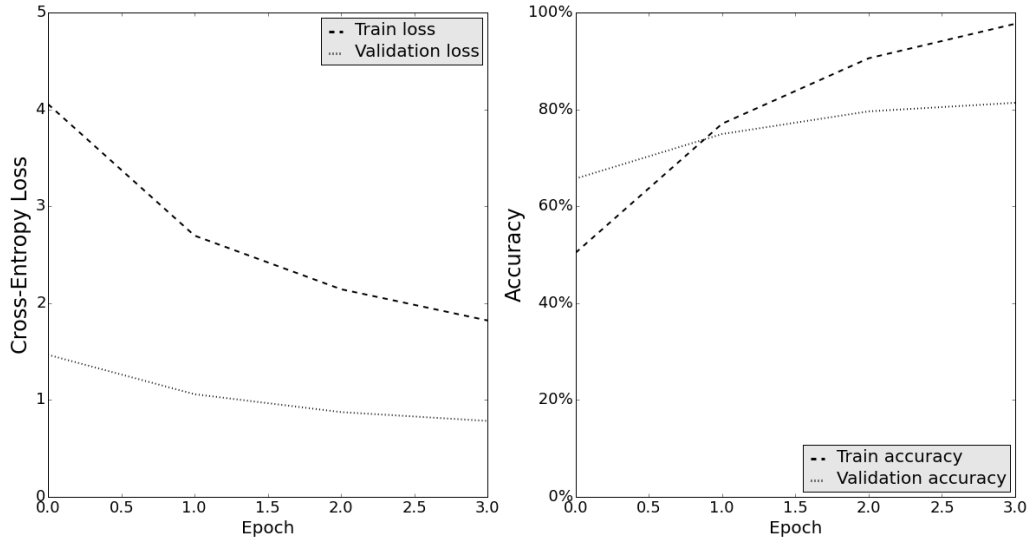

Figure 11: Base line for maxpooling experiment

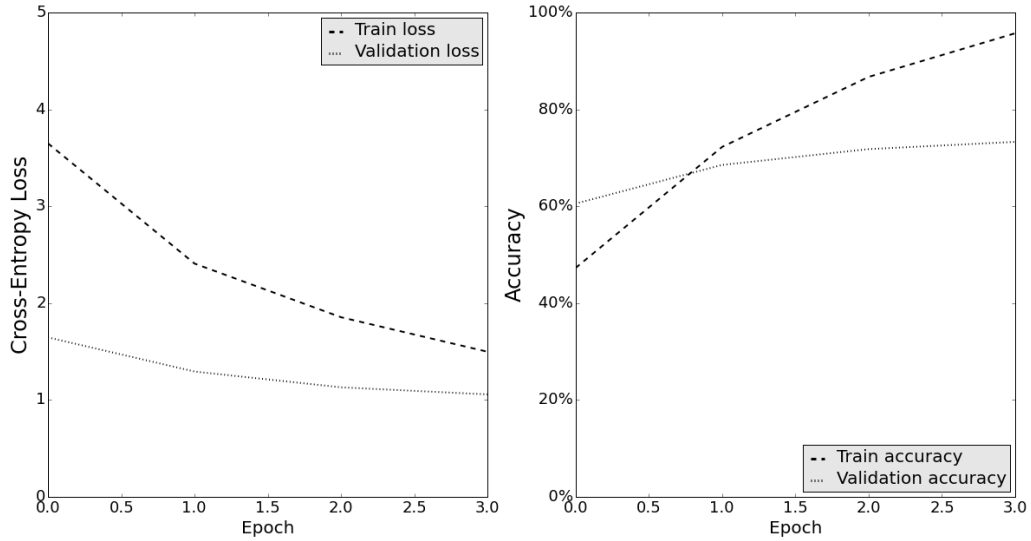

Figure 12: No maxpooling after conv1

A.1.3 Batch Normalization

For this experiment, we have used the architectures described in tables 2 and 3 and training was done with a fixed learning rate of 0.005. Comparing figs. 13 and 14 shows that the network converges extremely fast when using batch normalization but settles around the 83% accuracy mark, while the non-batch-normalized model keeps improving. We assume that the learning rate decay was not increasing fast enough, as Ioffe and Szegedy [12] recommends.

Experiment	**Validation Acc***	**Test Acc***
base line, lr 0.005, fig. 13	83.36%	83.62%
no batch normalization, fig. 14	84.03%	84.00%

Table 7: Batch normalization experiment results - Not considering imbalance of categories

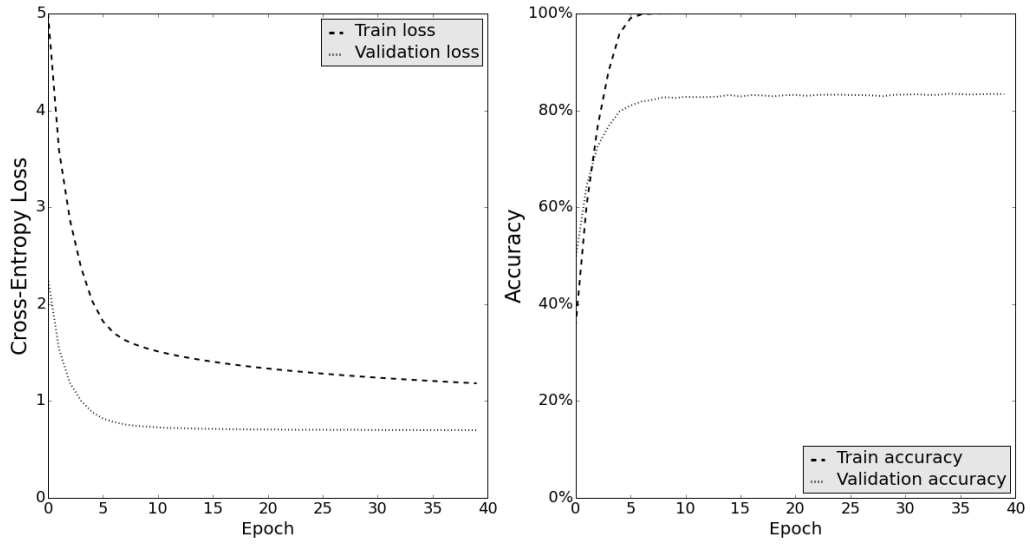

Figure 13: Base line for batch normalization experiment

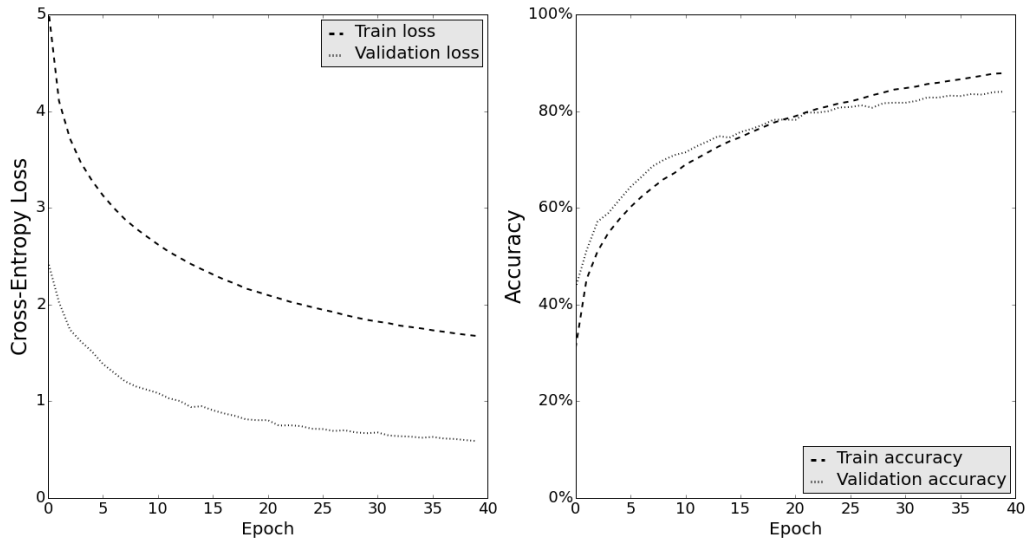

Figure 14: No batch normalization