DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Robotics, Cognition, Intelligence

Evaluation and Generalization of Capsule Networks in Neurorobotics

Jean Amadeus Elsner

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Robotics, Cognition, Intelligence

Evaluation and Generalization of Capsule Networks in Neurorobotics

Evaluation und Generalisierung von Kapsel Netzwerken in der Neurorobotik

Author:	Jean Amadeus Elsner
Supervisor:	Prof. Dr. Alois C. Knoll
Advisor:	Alexander Kuhn
Submission Date:	TBD

I confirm that this master's thesis in robotics, cognition, intelligence is my own work and I have documented all sources and material used.

Munich, TBD Jean Amadeus Elsner

Acknowledgments

Abstract

Contents

1 Introduction

2 State of the Art

This chapter presents an overview of state of the art approaches to object recognition, while focusing on two families of architectures, which are motivated quite differently. Object recognition techniques based on convolutional neural networks (*CNN*s) currently dominate the field, achieving state of the art performance on many datasets [1, 2]. CNNs however, are only loosely based on biological neurons. Spiking neural networks (*SNN*s) on the other hand, try to mimic the physical properties of neurons more closely and therefore constitute biologically more plausible models [3]. Generally speaking, CNNs may be regarded as a more engineering-based approach (or top-down), while SNNs are motivated by results from neuroscience and biology (bottom-up approach).

2.1 Deep Learning for Object Recognition

Recent years have seen a surge of interest in deep learning methods, especially in the field of computer vision. While the theory behind many deep learning methods has been around for many years, their recent success is mainly due to the availability of large labelled data sets and highly parallel computing powered by GPUs. One of the specific tasks, deep learning based methods excel at, is object recognition: the identification of objects in images or videos (cf. figure 2.1). The significantly better performance of deep neural networks over traditional machine learning methods can be explained by: (i) their hierarchical topology of parameterized non-linear processing units is a fundamentally better probabilistic model and prior for real world data leading to better generalization and (ii) they automatically find good features to extract based on the training data. The potential applications for a robust image classification system are myriad and range from automated driving and image-based diagnosis to robot vision and many more. As deep learning is currently the best candidate for such a system, it is well worth exploring.

2.1.1 Convolutional Neural Networks

CNN architectures are generally distinguished by their use of specific types of neuron-layers, namely, convolutional, pooling and fully connected layers. While wildly different network topologies may be found in literature, characterized by their use of skip connections, number of layers, number of paths etc., CNNs can always be reduced to these three basic layer types.

Figure 2.1: Sample images from the CIFAR-10 [4] dataset and their corresponding classes. CIFAR-10 consists of 6000 images at 32 by 32 pixels for each of the 10 classes. Datasets such as this are often used as a benchmark to evaluate the performance of novel deep learning architectures for image recognition. This is done by using a subset of the dataset to train the neural network. The remainder of the images, which the network has not seen before, are used to evaluate the accuracy.

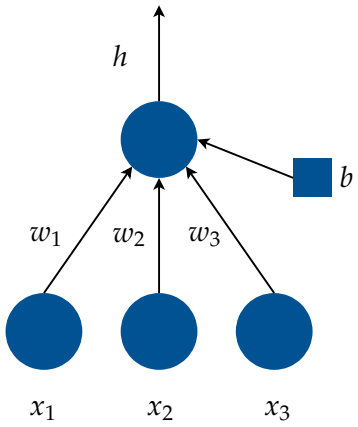

Figure 2.2: Illustration of an artificial neuron with three input connections. Artificial neurons constitute the basic non-linear computational units of neural networks. The neuron receives the activations of three lower level neurons weighted by the learnable w_i as well as a learnable bias b. After applying the nonlinearity (also known as activation function), the neuron outputs its activation h as in equation 2.1

Fully Connected Layer

Each neuron in a fully connected layer is connected to all the activations in the previous layer. The activation of a single neuron (cf. figure 2.2) is calculated by applying a nonlinearity to the weighted sum of its inputs and a bias.

$$h = g\left(\sum_i w_i x_i + b\right) \tag{2.1}$$

With the nonlinear function g, the learnable weights w_i, the input activations x_i and the learnable bias b. In the case of a fully connected layer, the activations can be computed using matrix multiplication. In tensor notation this may be written as:

$$\mathbf{h_l} = g_l\left(\mathbf{W_l^T h_{l-1}} + \mathbf{b_l}\right). \tag{2.2}$$

With N_l denoting the number of neurons in layer l, $\mathbf{W_l}$ is an $N_{l-1} \times N_l$ dimensional weight matrix, b_l an N_l dimensional vector and g_l the N_l dimensional vectorized activation function of layer l.

$$g_l(\mathbf{x}) = \left(g_l(x_1), ..., g_l(x_{N_l})\right)^T \tag{2.3}$$

The computational power of neural networks is shown by the universal approximation theorem. The theorem states, that networks with a single hidden layer (cf. figure 2.3) containing a finite number of neurons can approximate arbitrary continuous functions on compact subsets of $\mathbb{R}^n$ [5, 6].

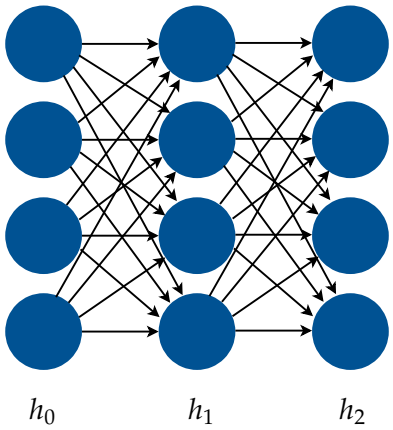

Figure 2.3: Illustration of a fully connected neural network as a directed graph with input layer h_0 and output layer h_2. Layers between the input and output layer are usually referred to as *hidden* layers. Note that the biases are not explicitly shown.

Convolutional Layer

In a convolutional layer, the activities of the input layer are convolved with a number of trainable kernels so as to create the same number of feature maps. In computer vision it is common to view the layer's neurons as two-dimensional grids of neurons arranged in channels (cf. figure 2.5). These grids correspond to pixels and color channels in the case of the input layer or activities (feature maps) resulting from convolution with different kernels in the case of intermediate convolutional layers. For a feature kernel $F^l_{m,n}$ of size $M_l \times M_l$ and an input layer $l-1$ with $N_{l-1} \times N_{l-1}$ neurons, the corresponding feature map activities of layer l are computed as

$$h^l_{m,n} = g_l\left(\sum_{m'}^{M_l}\sum_{n'}^{M_l} F^l_{m',n'} h^l_{m+m',n+n'} + b_l\right). \tag{2.4}$$

This is the same as a two-dimensional discrete cross correlation.

$$(f \star g)[n] \overset{\text{def}}{=} \sum_{m=-\infty}^{\infty} f^*[m]\; g[m+n] \tag{2.5}$$

Where f^* is the complex conjugate of the discrete function f. Strictly speaking, the use of the term *convolution* in neural network literature is therefore a misnomer, as either the filter or the image (or feature map) would have to be flipped before the operation. However, as the weights of the kernel are actually learned by the network, the result will be the same.

For a 2D single channel input layer of size $N_0 \times N_0$, the number of trainable parameters for a convolutional layer with a single $M \times M$ filter is $M^2 + 1$ for the kernel weights and bias respectively, compared to $N_0^2 N_1^2 + N_1^2$ for a fully connected layer of size $N_1 \times N_1$.

The sparsity in learnable parameters in convolutional layers compared to fully connected layers (cf. figure 2.4) is often referred to as *weight sharing* (an entire channel *shares* the weights of a single kernel).

In the case of multiple input channels, the kernels actually extend through the whole depth of the input layer's volume. Equation 2.4 can be extended to accommodate multiple input channels and feature kernels. The activities of feature map k in layer l are then computed as

$$h^l_{k,m,n} = g_l\left(\sum_{k'}^{K_{l-1}}\sum_{m'}^{M_l}\sum_{n'}^{M_l} F^l_{k,k',m',n'} h^l_{k',m+m',n+n'} + b_{l,k}\right). \tag{2.6}$$

With K_l the number of channels in layer l. Most deep learning frameworks offer additional hyperparameters for convolutional layers, like stride and zero padding. Zero padding adds additional rows and columns of zero-activities around the input layer channels, effectively allowing the output feature maps to have the same size (i.e. height and width, depth is determined by the number of kernels) as the unpadded input layer. Stride on the other hand defines the integer value by which the kernels are moved during convolution. This can be used to keep the receptive fields from overlapping too much. Equation 2.6 is easily extended to take the stride S_l of layer l into consideration.

$$h^l_{k,m,n} = g_l\left(\sum_{k'}^{K_{l-1}}\sum_{m'}^{M_l}\sum_{n'}^{M_l} F^l_{k,k',m',n'} h^l_{k',S_l m+m',S_l n+n'} + b_{l,k}\right) \tag{2.7}$$

The spatial dimension of this feature map can be computed as a function of the input layer size $N_{l-1} \times N_{l-1}$ and the amount of zero padding P_l.

$$N_l = \frac{N_{l-1} - M_l + 2P_l}{S_l} + 1 \tag{2.8}$$

Pooling Layer

Pooling is a form of sub- or downsampling using a sliding window similar to convolution. Most often the stride is set equal to the window size - that way the windows don't overlap. An operator is applied to each window, which selects a single neuron. Common operators are maximum, average or L_2-Norm. The pooling is applied to each channel and leads to a reduction in the spatial dimensions. This reduction corresponds to a loss of information or reduction in degrees of freedom and therefore reduces the amount of computation required as well as the possibility of overfitting. The choice of pooling operator has a profound effect on the generalization and speed of convergence of CNNs and recently maximum pooling has proven to be the most successful method [7, 8].

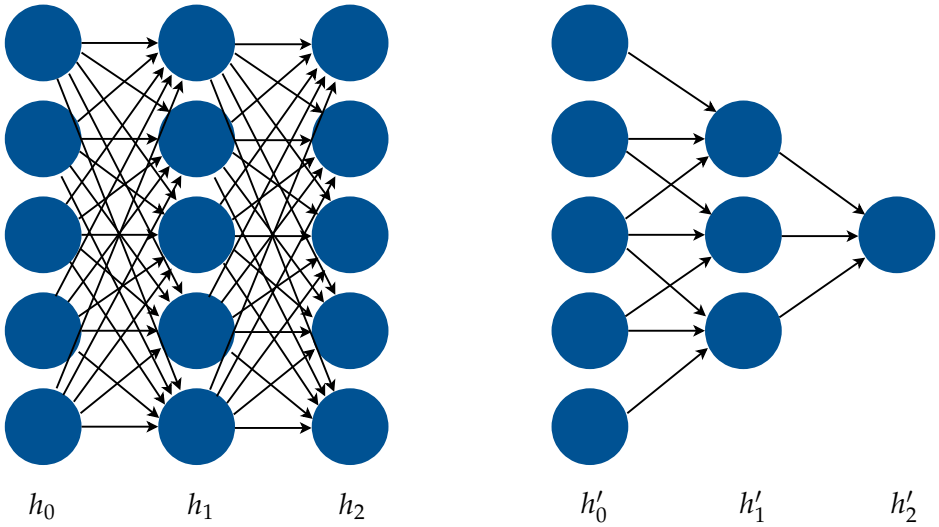

Figure 2.4: Illustration of fully connected layers compared to convolutional layers, making apparent the sparsity of the latter relative to the former. The convolutional layers h'_1 and h'_2 perform a 1D convolution with a filter of size 3. Note that due to weight sharing the number of weight parameters between h'_0 and h'_1 is actually only 3. Also note, that convolution reduces the number of neurons (or pixel resolution in the case of an image). For an input layer with N neurons and a filter kernel of size M the number of neurons in the convolutional layer computes as $N - M + 1$.

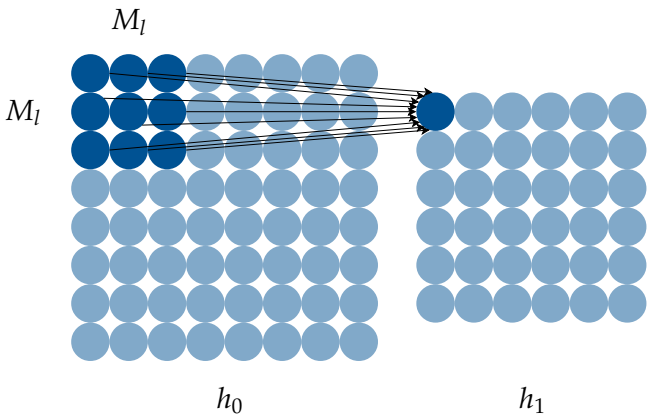

Figure 2.5: Illustration of a 2D convolution with a kernel of size $M_l \times M_l$. As the resulting feature map h_1 has a lower resolution than h_0, applying a convolutional layer is sometimes also referred to as downsampling. For an input layer with $N \times N$ neurons, the feature map's size is computed as $(N - M_l + 1) \times (N - M_l + 1)$. The area comprised of the input neurons that are used to calculate the activity of a feature map neuron (highlighted neurons in layer h_0) are known as that neuron's *receptive field*.

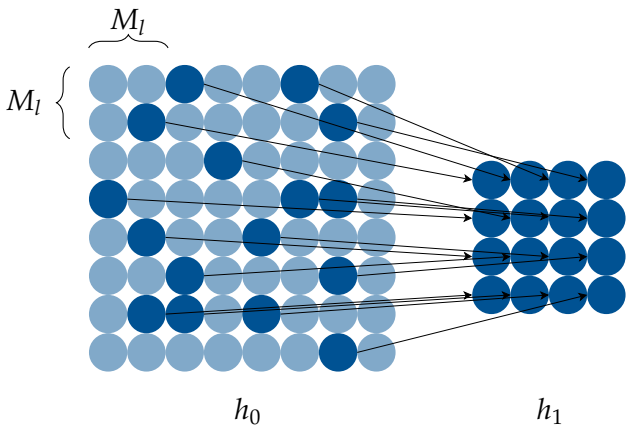

Figure 2.6: Illustration of a pooling layer. The pooling uses a 2×2 window with stride 2 to segment the input layer into non-overlapping tiles. An operator (e.g. maximum) is applied to each 2×2 segment to select a single neuron (the highlighted neurons in layer h_0). The activities of the selected neurons are aranged in a new layer h_1 of size $\frac{N_{l-1}}{M_l} \times \frac{N_{l-1}}{M_l}$ with $M_l \times M_l$ and $N_{l-1} \times N_{l-1}$ the size of the window and the input layer respectively.

CNN Architecture

The key idea behind the combination of these three types of layers is, that essential visual features such as edges and corners within a convolutional layer's receptive field are combined to form higher level features such as shapes by subsequent convolutional layers. In between these convolutions, pooling operations select the most salient features and reduce the computational size of the network. The convolutional kernels are effectively trainable feature detectors and due to weight sharing and pooling naturally incorporate a measure of translational invariance. This hierarchical organization of receptive fields is similar in structure to the mammalian visual cortex [9] and sometimes CNNs are seen to be directly derived from it [10, 11]. More often than not however, the use of convolutional layers with weight sharing is motivated as a means to achieve translational equivariance and faster computation compared to fully connected layers [12, 13, 14]. Finally, fully connected layers are placed on top of the network in order to perform high level inference (cf. figure 2.7). A CNN's architecture is therefore largely defined by its *hyperparameters*: the number and types of layers and their connections. The number of stacked layers is referred to as the network's depth. Current networks often employ multiple paths and skip connections (i.e. a layer's output not only flows to its direct descendent, but skips several layers) allowing for topologies several hundred layers deep, hence *deep learning* [15, 16, 17, 18].

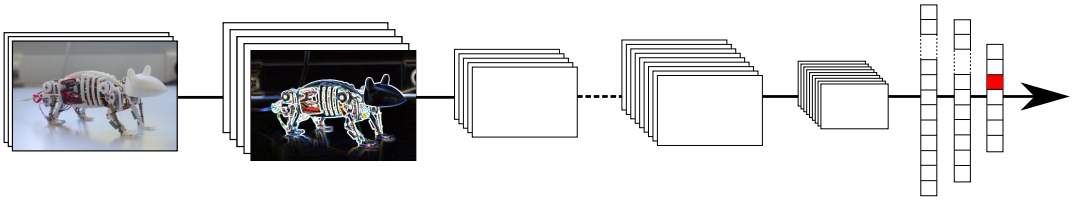

Figure 2.7: Illustration of a typical CNN architecture for object recognition. Note that the feature maps of the first convolutional layer extracted elementary features like edges. The last layer's neurons correspond to the trained classes, while the output neuron with the highest probability is picked as the network's prediction during inference (highlighted in red).

Training

For object recognition, the output of the final layers are the class probabilities. This is achieved by applying a normalizing activation function to the the output of the last layer, such as the *softmax* function. In case of the softmax function, the network's output $y_k(\mathbf{x})$ for the class k given input sample $\mathbf{x}$ thus becomes

$$y_k(\mathbf{x}) = \frac{\exp(a_k)}{\sum_{k'=1}^{K} \exp(a_{k'})}. \tag{2.9}$$

With $\mathbf{a} = (a_1, ..., a_K)^T$ the linear activities of the last layer's neurons (in machine learning literature often referred to as *logits*) and K the number of classes. Using one-hot coding for the target class

$$\mathbf{t} = (t_1, ..., t_k, .., t_K)^T = (0, ..., 1, ..., 0)^T, \tag{2.10}$$

the probabilistic model can be defined as a function of the neural network.

$$P(\mathbf{t} \mid \mathbf{x}, \boldsymbol{\theta}) = \prod_{k=1}^{K} y_k(\mathbf{x})^{t_k} \tag{2.11}$$

With $\boldsymbol{\theta} = (\mathbf{W}, \mathbf{b})$ the set of the network's trainable parameters. The model's likelihood is then given by

$$P(\mathbf{T} \mid \mathbf{X}, \boldsymbol{\theta}) = \prod_{n} P(\mathbf{t_n} \mid \mathbf{x_n}, \boldsymbol{\theta}). \tag{2.12}$$

Where the index n denotes nth training sample and target class (also known as label). The negative logarithm of the likelihood, called the negative *log-likelihod* defines the classifier's error function E_{ML}.

$$E_{\mathrm{ML}}(\boldsymbol{\theta}) = -\log(P(\mathbf{T} \mid \mathbf{X}, \boldsymbol{\theta})) = -\sum_{n} \log P(\mathbf{t_n} \mid \mathbf{x_n}, \boldsymbol{\theta}) \tag{2.13}$$

$$= -\sum_{n} \sum_{k=1}^{K} t_{n,k} \log(y_k(\mathbf{x_n})) \tag{2.14}$$

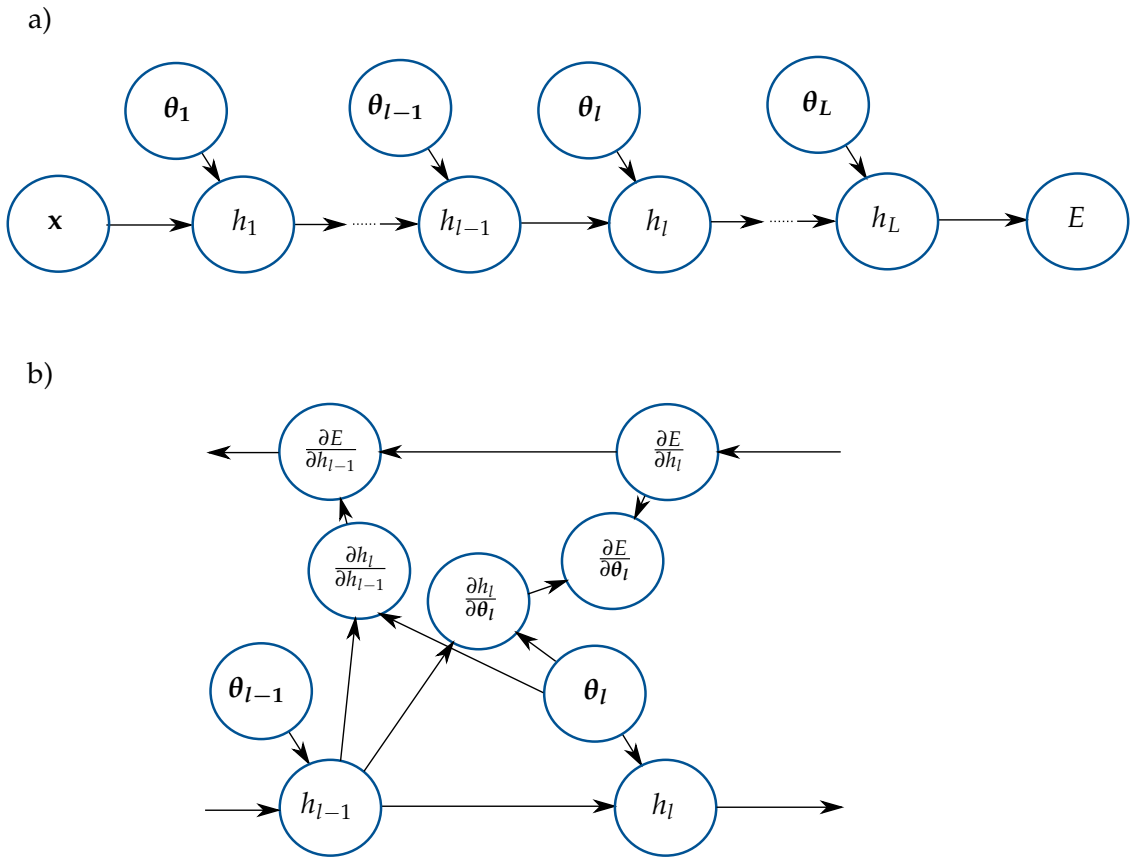

Figure 2.8: Computational graph for error propagation within a neural network. a) shows the dependency of the layers on both the lower layer's output as well as the parameters $\boldsymbol{\theta_l}$ during inference (also known as forward pass). In b) the computational nodes for the error gradient as needed for error backpropagation were added (known as the backward pass).

The error function resulting from the softmax activation function specifically is referred to as the *cross entropy* (equation 2.14). Now the network's parameters may be trained by minimizing its classification error w.r.t. $\boldsymbol{\theta}$. As the logarithm is a convex function and the error function is the *negative* log-likelihood, this is equivalent to a maximum likelihood estimate. The gradient of the error function can be computed using the derivative chain rule (cf. figure 2.8), this method is known as *backpropagation* and forms the computational backbone of CNN architectures.

$$\frac{\partial E_{\mathrm{ML}}(\boldsymbol{\theta})}{\partial \boldsymbol{\theta_{l-1}}} = \frac{\partial E_{\mathrm{ML}}(\boldsymbol{\theta})}{\partial h_{l-1}} \frac{\partial h_{l-1}}{\partial \boldsymbol{\theta_{l-1}}} = \frac{\partial E_{\mathrm{ML}}(\boldsymbol{\theta})}{\partial h_l} \frac{\partial h_l}{\partial h_{l-1}} \frac{\partial h_{l-1}}{\partial \boldsymbol{\theta_{l-1}}} \tag{2.15}$$

The gradient calculated based on the training samples is then used to update the network's parameter.

$$\boldsymbol{\theta}^{(s+1)} = \boldsymbol{\theta}^{(s)} - \eta \nabla E_{\mathrm{ML}}\left(\boldsymbol{\theta}^{(s)}\right) \tag{2.16}$$

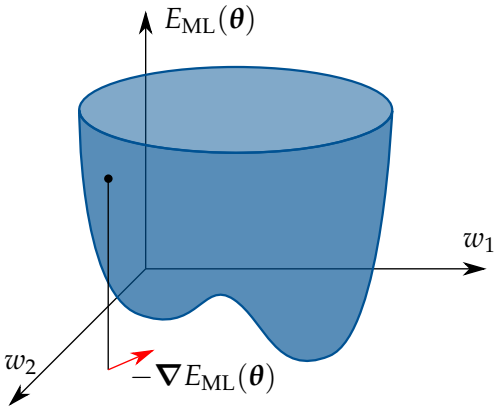

Figure 2.9: Illustration of the surface of a neural network's error function in parameter space. The function is non-convex and contains a local minimum. Note that the resulting minimum depends on both the learning rate and the initialization of the parameters. In this case, following the negative gradient (illustrated as a red arrow) using small steps will converge on a non-optimal local minimum.

Where s counts the number of training epochs (a run through all the samples) and η is a hyperparameter known as the *learning rate*, that effectively describes the size of the step taken in direction of the gradient (cf. figure 2.9). In practise, the gradient is often calculated as an average based on a subset of randomly selected training samples.

$$\boldsymbol{\theta}^{(s+1)} = \boldsymbol{\theta}^{(s)} - \eta \frac{1}{M} \sum_{n=1}^{M} \nabla E_n\left(\boldsymbol{\theta}^{(s)}\right) \tag{2.17}$$

The $\frac{N}{M}$ sets of training samples are called (mini-) *batches* and are trained sequently until all samples have been seen, while training algorithms based on this optimization scheme are known as *stochastic gradient descent* (SGD). It is at this point, that vectorization libraries are used to leverage the power of modern GPUs by processing an entire mini batch in parallel.

State of the art CNN architectures expand upon the building blocks introduced in this chapter in various ways depending on the application. Often it is necessary to regularize the network in order to prevent overfitting. This can be achieved by adding penalties for large weights to the loss function (known as weight decay) or randomly setting some weights to zero during training (so-called dropout) [19, 20, 21, 22]. This can be complemented with adaptive learning schemes (i.e. algorithms that try to adapt learning hyperparameters during training) in order to reliably train neural networks [23, 24, 25, 26, 27, 28, 29]. Finally, both the network's topology and the loss function may be chosen in such a way as to encourage the learning of internal representations to fit the task (e.g. learn the pose of a 3D object) [30, 31, 32, 33]. Even though CNNs have been applied successfully in many fields, there is not yet a theory to derive the topology and hyperparameters best suited for a given problem or predict the network's performance.

2.1.2 Spiking Neural Networks

2.2 Limits of Deep Learning Approaches

3 Capsule Network Architectures

4 Experimental Setup

5 Results

6 Discussion

7 Conclusion

List of Figures

List of Tables

Bibliography

[1] A. Diba, V. Sharma, A. M. Pazandeh, H. Pirsiavash, and L. V. Gool. "Weakly Supervised Cascaded Convolutional Networks." In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017), pp. 5131–5139.

[2] W. Ouyang, X. Zeng, X. Wang, S. Qiu, P. Luo, Y. Tian, H. Li, S. Yang, Z. Wang, H. Li, K. Wang, J. Yan, C. C. Loy, and X. Tang. "DeepID-Net: Object Detection with Deformable Part Based Convolutional Neural Networks." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.7 (July 2017), pp. 1320–1334. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2016.2587642.

[3] A. J. Schofield, I. D. Gilchrist, M. Bloj, A. Leonardis, and N. Bellotto. "Understanding images in biological and computer vision." In: *Interface Focus* 8.4 (2018). ISSN: 2042-8898. DOI: 10.1098/rsfs.2018.0027. eprint: http://rsfs.royalsocietypublishing.org/content/8/4/20180027.full.pdf.

[4] A. Krizhevsky. *Learning Multiple Layers of Features from Tiny Images*. Tech. rep. 2009.

[5] G. Cybenko. "Approximation by superpositions of a sigmoidal function." In: *Mathematics of Control, Signals and Systems* 2.4 (Dec. 1989), pp. 303–314. DOI: https://doi.org/10.1007/BF02551274.

[6] K. Hornik. "Approximation capabilities of multilayer feedforward networks." In: *Neural Networks* 4.2 (1991), pp. 251–257. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(91)90009-T.

[7] Y.-L. Boureau, J. Ponce, and Y. Lecun. "A Theoretical Analysis of Feature Pooling in Visual Recognition." In: *ICML 2010 - Proceedings, 27th International Conference on Machine Learning*. Nov. 2010, pp. 111–118.

[8] D. Scherer, A. Müller, and S. Behnke. "Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition." In: *Artificial Neural Networks – ICANN 2010*. Ed. by K. Diamantaras, W. Duch, and L. S. Iliadis. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 92–101. ISBN: 978-3-642-15825-4.

[9] H. D. H. and W. T. N. "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex." In: *The Journal of Physiology* 160.1 (), pp. 106–154. DOI: 10.1113/jphysiol.1962.sp006837. eprint: https://physoc.onlinelibrary.wiley.com/doi/pdf/10.1113/jphysiol.1962.sp006837.

[10] K. Fukushima. "Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position." In: *Biological Cybernetics* 36.4 (Apr. 1980), pp. 193–202. ISSN: 1432-0770. DOI: 10.1007/BF00344251.

[11] Y. LeCun, Y. Bengio, and G. Hinton. "Deep learning." In: *nature* 521.7553 (2015), p. 436.

[12] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. MIT Press, 2016. Chap. 9. eprint: `http://www.deeplearningbook.org/contents/convnets.html`.

[13] M. Oquab, L. Bottou, I. Laptev, and J. Sivic. "Is object localization for free? - Weakly-supervised learning with convolutional neural networks." In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2015, pp. 685–694. DOI: 10.1109/CVPR.2015.7298668.

[14] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. "Going deeper with convolutions." In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2015, pp. 1–9. DOI: 10.1109/CVPR.2015.7298594.

[15] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. "Densely Connected Convolutional Networks." In: *CVPR*. Vol. 1. 2. July 2017, p. 3.

[16] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi. "Inception-v4, inception-resnet and the impact of residual connections on learning." In: *AAAI*. Vol. 4. 2017, p. 12.

[17] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He. "Aggregated residual transformations for deep neural networks." In: *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*. IEEE. 2017, pp. 5987–5995.

[18] L. Wang, Y. Xiong, Z. Wang, and Y. Qiao. "Towards good practices for very deep two-stream convnets." In: *arXiv preprint arXiv:1507.02159* (2015).

[19] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. "Dropout: a simple way to prevent neural networks from overfitting." In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 1929–1958.

[20] L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, and R. Fergus. "Regularization of Neural Networks using DropConnect." In: *Proceedings of the 30th International Conference on Machine Learning*. Ed. by S. Dasgupta and D. McAllester. Vol. 28. Proceedings of Machine Learning Research 3. Atlanta, Georgia, USA: PMLR, June 2013, pp. 1058–1066.

[21] A. Krogh and J. A. Hertz. "A simple weight decay can improve generalization." In: *Advances in neural information processing systems*. 1992, pp. 950–957.

[22] N. K. Treadgold and T. D. Gedeon. "Simulated annealing and weight decay in adaptive learning: The SARPROP algorithm." In: *IEEE Transactions on Neural Networks* 9.4 (1998), pp. 662–668.

[23] M. D. Zeiler. "ADADELTA: an adaptive learning rate method." In: *arXiv preprint arXiv:1212.5701* (2012).

[24] J. Duchi, E. Hazan, and Y. Singer. "Adaptive subgradient methods for online learning and stochastic optimization." In: *Journal of Machine Learning Research* 12.Jul (2011), pp. 2121–2159.

[25] D. P. Kingma and J. Ba. "Adam: A method for stochastic optimization." In: *arXiv preprint arXiv:1412.6980* (2014).

[26] B. T. Polyak and A. B. Juditsky. "Acceleration of stochastic approximation by averaging." In: *SIAM Journal on Control and Optimization* 30.4 (1992), pp. 838–855.

[27] A. Graves. "Generating sequences with recurrent neural networks." In: *arXiv preprint arXiv:1308.0850* (2013).

[28] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. "On the importance of initialization and momentum in deep learning." In: *International conference on machine learning*. 2013, pp. 1139–1147.

[29] I. Loshchilov and F. Hutter. "Sgdr: Stochastic gradient descent with warm restarts." In: *arXiv preprint arXiv:1608.03983* (2016).

[30] D. Worrall and G. Brostow. "CubeNet: Equivariance to 3D Rotation and Translation." In: *arXiv preprint arXiv:1804.04458* (2018).

[31] D. E. Worrall, S. J. Garbin, D. Turmukhambetov, and G. J. Brostow. "Harmonic networks: Deep translation and rotation equivariance." In: *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*. Vol. 2. 2017.

[32] U. Schmidt and S. Roth. "Learning rotation-aware features: From invariant priors to equivariant descriptors." In: *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE. 2012, pp. 2050–2057.

[33] T. Cohen and M. Welling. "Group equivariant convolutional networks." In: *International conference on machine learning*. 2016, pp. 2990–2999.