Data Science 2

Convolution Neural Networks I

Ondřej Týbl

Charles University, Prague

Outline

1. Motivation – Temporal Data
2. Convolution
3. Network Architecture
4. Batch Normalization
5. Regularization Revisited

Motivation – Temporal Data

Motivation – Temporal Data

We compare a fully connected layer to the special case when some parameters are forced to zero and some parameter values are shared.

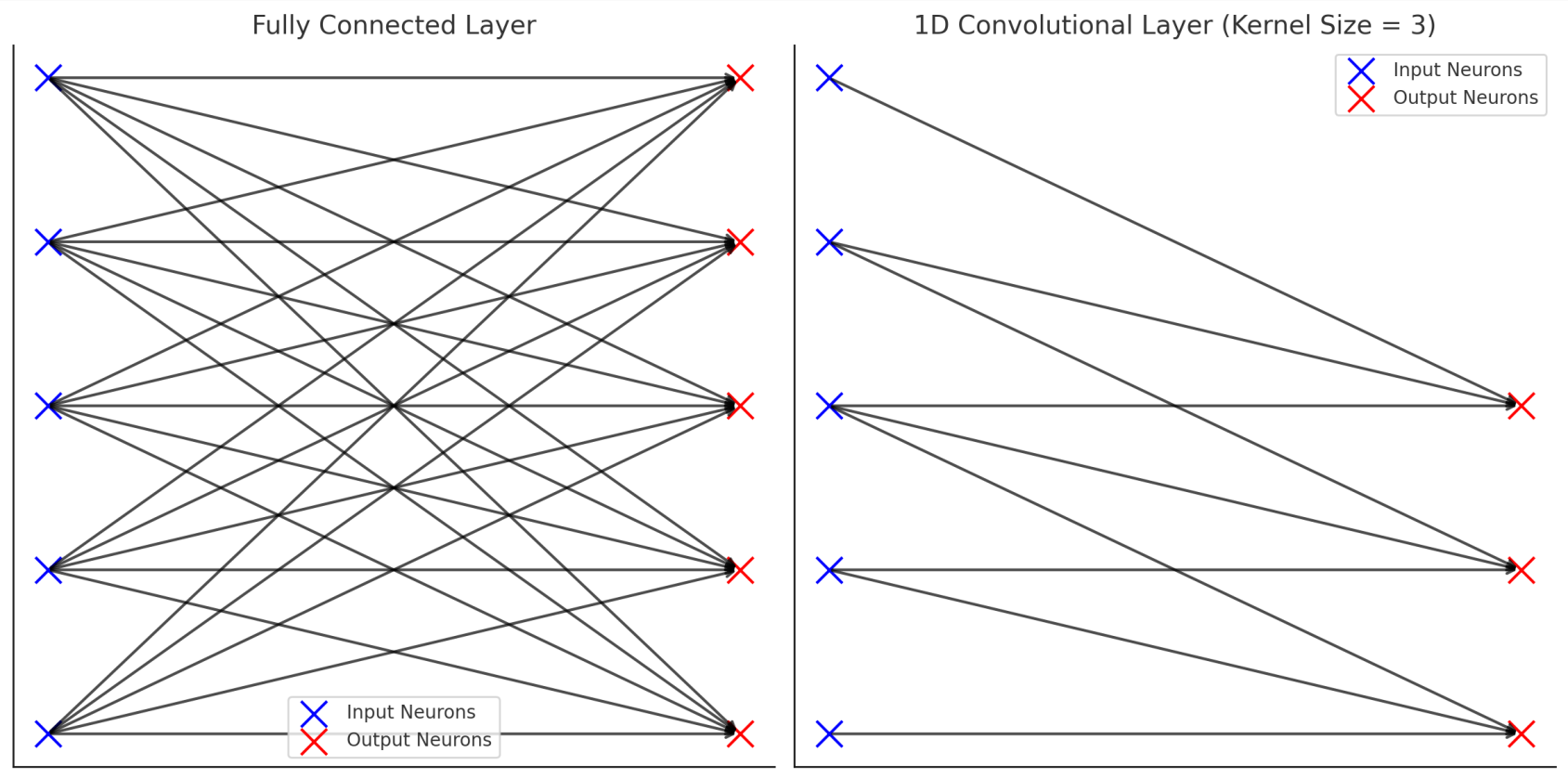


Figure 1: Graph comparison for a fully connected layer and convolution in 1 dimension.

Motivation – Temporal Data

The fully connected layer is represented by a matrix multiplication

$$Wx + w_0. \tag{1}$$

Similarly, convolution in 1 dimension can be written as (1) with

$$W = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ k_1 & k_2 & k_3 & 0 & 0 \\ 0 & k_1 & k_2 & k_3 & 0 \\ 0 & 0 & k_1 & k_2 & k_3 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad w_0 = \begin{bmatrix} 0 \\ b \\ b \\ b \\ 0 \end{bmatrix}$$

where $K = (k_1, k_2, k_3) \in \mathbb{R}^{1\times 3}$ is called a *kernel* and $b \in \mathbb{R}$ is a shared bias term.

Motivation – Temporal Data

Now if our input data had some temporal structure, the disadvantage of using (1) in its full generality is

- the information on what is *close* is not a priori encoded in the computation graph, it might be difficult to train the network,
- there are no shared parameters and so the information is treated differently at each time step

Overall, fully connected network does not use the structure to decrease the parameter count and improve trainability.

Convolution

Convolution

Similarly, we define a convolution for matrices representing black-and-white images.

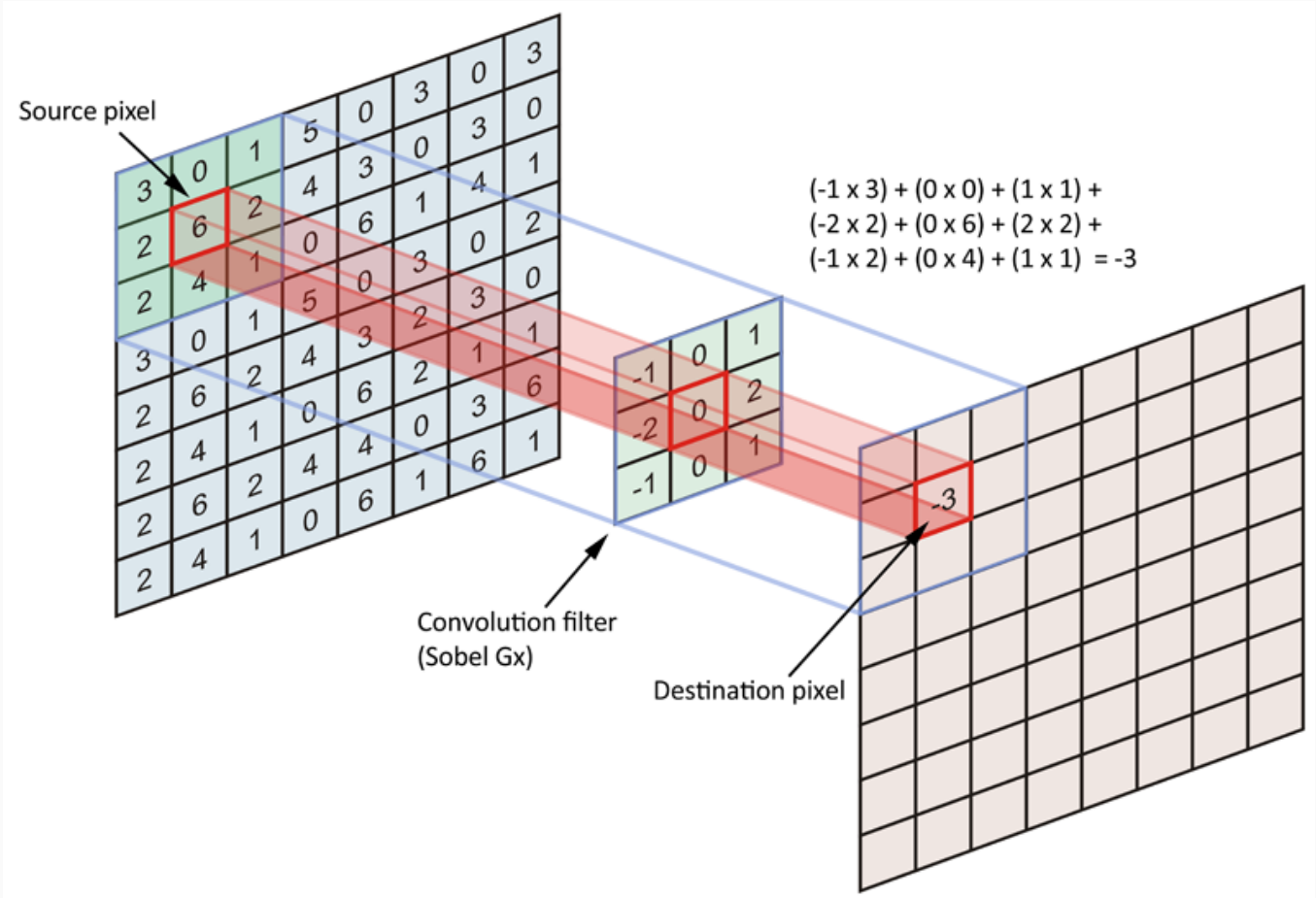


Figure 2: Computation graph for a convolution in 2 dimensions.

Convolution

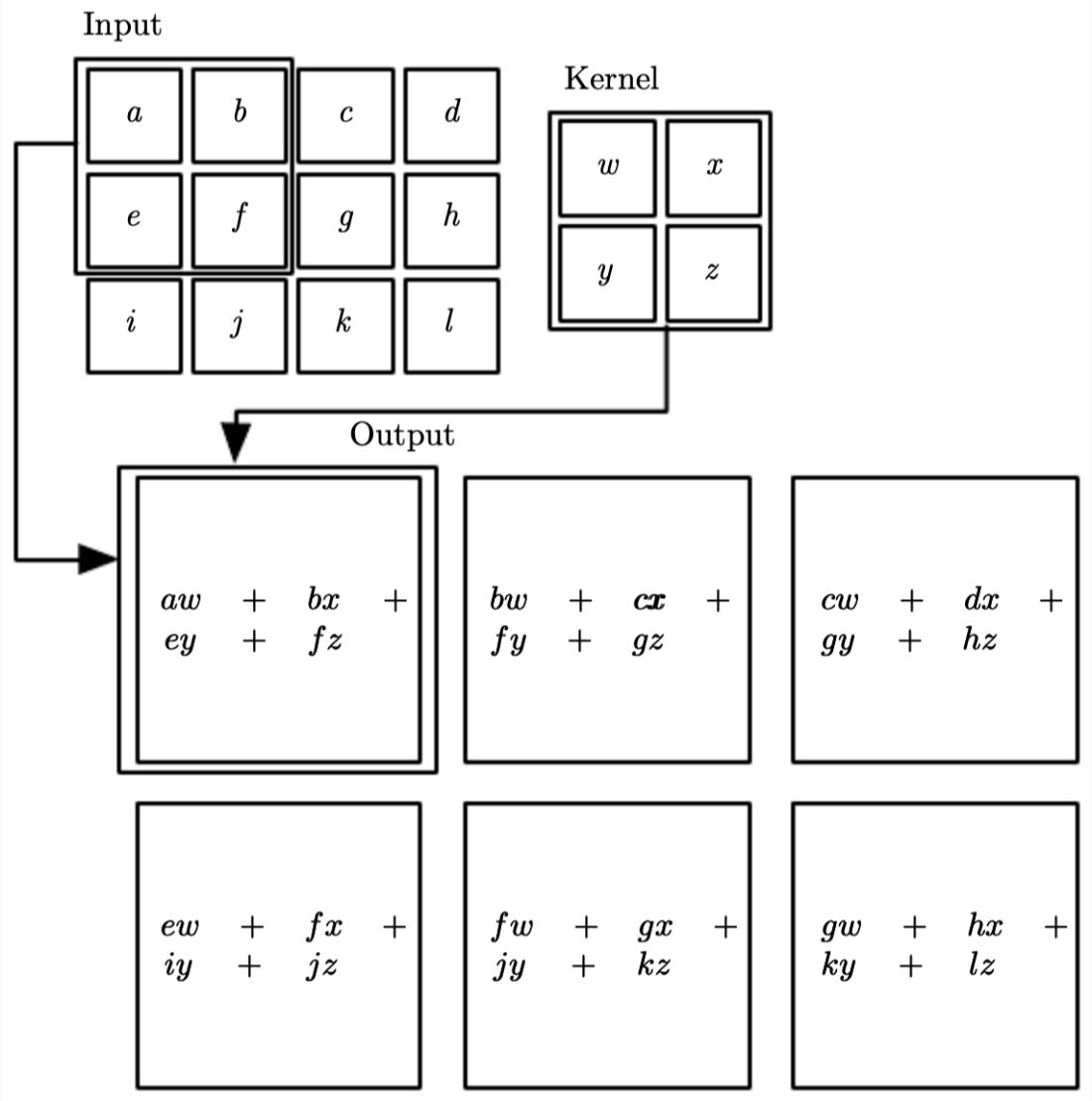


Figure 3: Computation scheme for a convolution in 2 dimensions.

Convolution

Key ideas coming from computer vision:

- *shift invariance* a pattern has the same meaning irrespective of the position; useful property if we care more about whether some feature is present than exactly where it is.
- *local interactions* joint information from neighboring pixels is more important than from pixels far away

Convolution

Definition (Convolution)

For a measure μ on $\mathbb{R}^d$ and two square-integrable real functions $f, g \in L^2(\mu)$ we define their convolution $f * g : \mathbb{R}^d \mapsto \mathbb{R}$ by

$$f * g(x) = \int_{\mathbb{R}^d} f(y)\, g(x - y)\, d\mu(y)$$

Click for example in one dimension.

Convolution

In the special case when dealing with matrices[1] $K : \{-k, \dots, k\}^2 \mapsto \mathbb{R}$ and $I : \{0, \dots, n\}^2 \mapsto \mathbb{R}$ we obtain that $K * I$ is a $n \times n$ matrix

$$(K * I)_{ij} = \sum_{n_1=1}^{k} \sum_{n_2=1}^{k} K_{n_1 n_2} I_{i-n_1, j-n_2}, \quad i, j \in \{0, \dots, n\}$$

- We adopted the notation $K(n_1, n_2) = K_{n_1 n_2}$.
- We define $I_{i-n_1, j-n_2} = 0$ if $i - n_1$ or $j - n_2$ are not in $\{0, \dots, n\}$ we simplicity[2].

[1] That is, functions of two variables supported on a subset of integers.

[2] Corresponds to the option *same* below

Convolution

Usually, I represents black-and-wite image and K (denoted as *filter* or *kernel*) represents some image processing. For example, if

$$K = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix},$$

we obtain the edge detector.

Convolution

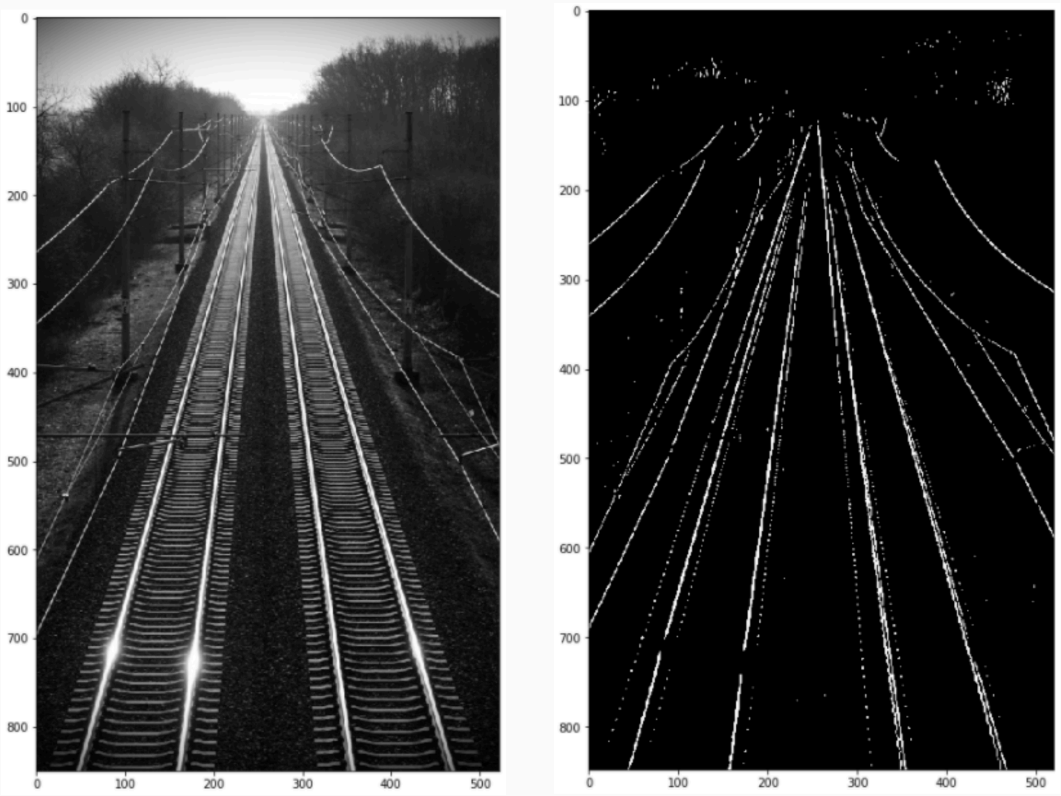


Figure 4: Example of an input image before and after edge detector application.

Convolution

In practice, an $h_{input} \times w_{input}$ image is represented by a sequence of matrices (i.e. a 3 dimensional tensor)

$$I : \{0, \dots h_{input} - 1\} \times \{0, \dots, w_{input} - 1\} \times \{0, \dots c_{input} - 1\} \mapsto \mathbb{R},$$

so that each pixel value is in fact a vector.

Each matrix

$$I_{\cdot,\cdot,c} \in \mathbb{R}^{h_{input} \times w_{input}}, \quad c = 0, \dots, c_{input} - 1$$

is called a *channel*.

Convolution

We need to specify what happens on the borders of the image I.

- *same*: I is defined to be zero outside of its domain
- *valid*: the convolution is computed only for indices for which all the necessary input pixels exist[3]

[3] That is, the output image is smaller depending on the kernel size.

Convolution

We also use *stride* S which is a (small) integer (typically just 2) to skip some pixels if we want to control the output dimensionality.

$$(K * I)_{ij} = \sum_{n_1=1}^{k} \sum_{n_2=1}^{k} K_{n_1 n_2} I_{iS-n_1, jS-n_2}$$

That is, if $S = 2$ the output image is of half the size of the input image.

Convolution

Click to see illustrative examples

Convolution

Definition (Convolution for Neural Networks)

Suppose an input tensor $I : \{0, \dots h_{input} - 1\} \times \{0, \dots, w_{input} - 1\} \times \{0, \dots c_{input} - 1\} \mapsto \mathbb{R}$ and a kernel tensor $K : \{-(h-1)/2 \dots (h-1)/2\} \times \{-(w-1)/2, \dots, (w-1)/2\} \times \{0, \dots c_{input} - 1\} \times \{0, \dots, c_{output} - 1\} \mapsto \mathbb{R}$. Let $S \in \mathbb{N}$. The *convolution*[4] of K and I with stride S is defined as

$$(K * I)_{i,j,o} = \sum_{m=-(w-1)/2}^{(w-1)/2} \sum_{n=-(h-1)/2}^{(h-1)/2} \sum_{c=0}^{c_{input}-1} K_{m,n,c,o} I_{iS+m,jS+n,c}$$

[4] In contrast to the convolution as defined above we add the indices instead of subtracting them. This operation is in some literature called *cross-correlation*. We stick to the definition often used in neural networks as it actually is just a matter of notation as cross-correlation is just a convolution with transposed kernel K which is a trainable parameter for which we do not care about the indexing.

Convolution

- A rigorous definition would also include the specification of a padding scheme. We neglect it for simplification.
- The size of the convolution $K * I$ depends not only on the input I and kernel K but also on the padding scheme and stride. Usually, stride $S = 2$ (i.e. a spatial reduction by a factor 2 comes with an increase of channels $c_{output} = 2c_{input}$ so that more the features get global the higher is their representation dimension.
- Typically, the kernel height and weight are usually the same being between 3 and 7. The overall number of trainable parameters in a convolution is $w \times h \times c_{input} \times c_{output}$.
- The number of input and output channels should be a power of 2 for efficient parallelization on GPUs.

Convolution

A convolution layer consists of a convolution operation with bias term (applied per output channel) and an activation function.

Definition (Convolution Layer)

Suppose $\varphi : \mathbb{R} \mapsto \mathbb{R}$ is an activation function and let $K : \{-(h-1)/2 \dots (h-1)/2\} \times \{-(w-1)/2, \dots, (w-1)/2\} \times \{0, \dots c_{input} - 1\} \times \{0, \dots, c_{output} - 1\}$ and $b : \{0, \dots, c_{output} - 1\} \mapsto \mathbb{R}$. Convolution layer is a tensor function defined for $I : \{0, \dots, h_{input} - 1\} \times \{0, \dots, w_{input} - 1\} \times \{0, \dots, c_{input} - 1\} \mapsto \mathbb{R}$ as[5]

$$CL_{i,j,o} = \varphi\left((K * I)_{i,j,o} + b_o\right)$$

[5] As above the convolution $K * I$ may contain some stride S which influences the output dimension.

Network Architecture

Network Architecture

We also introduce *pooling*. Instead of convolution we take either *maximum* or *average* over a specified window in each channel.

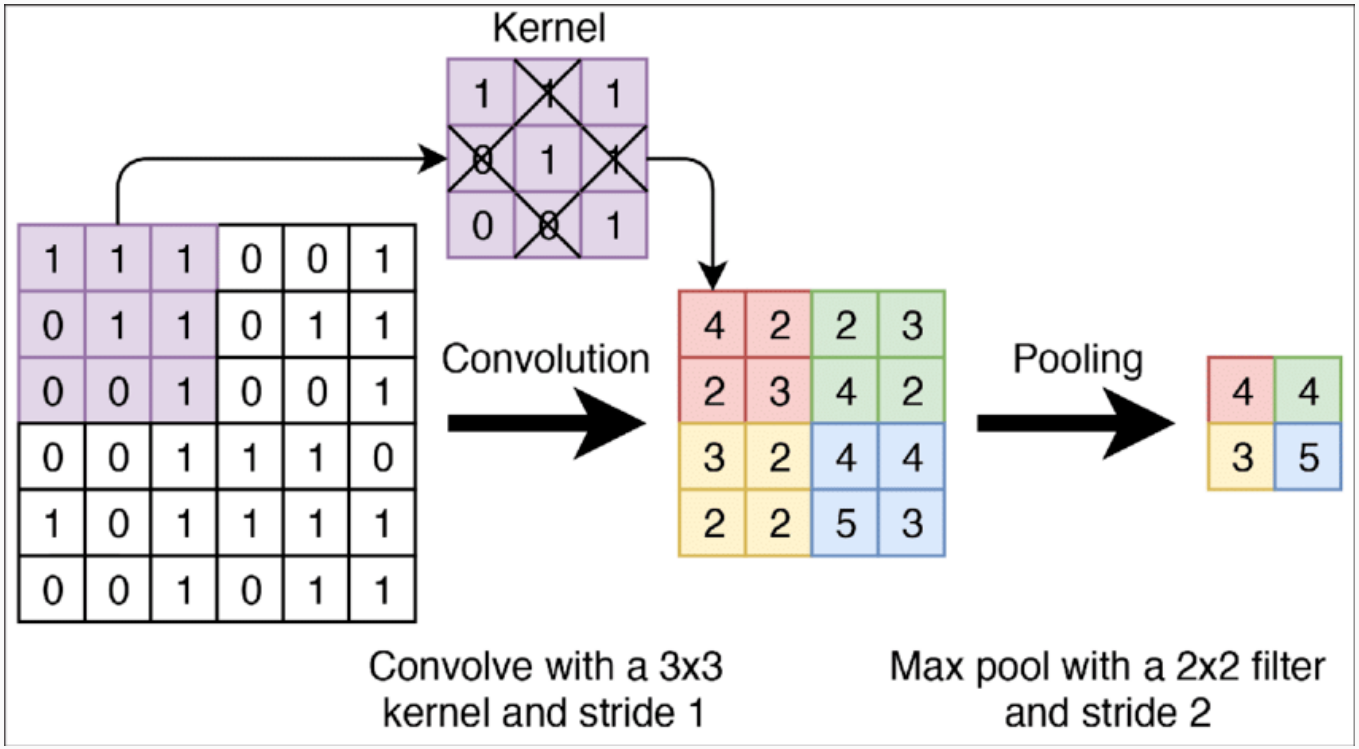


Figure 5: Schema showing a convolution followed by a pooling operation.

Pooling is parameter-free but reduces the image dimension.

Network Architecture

Maximum pooling simulates the *or* function:

$$\max(x, y) = x \vee y, \quad x, y \in \{0, 1\}.$$

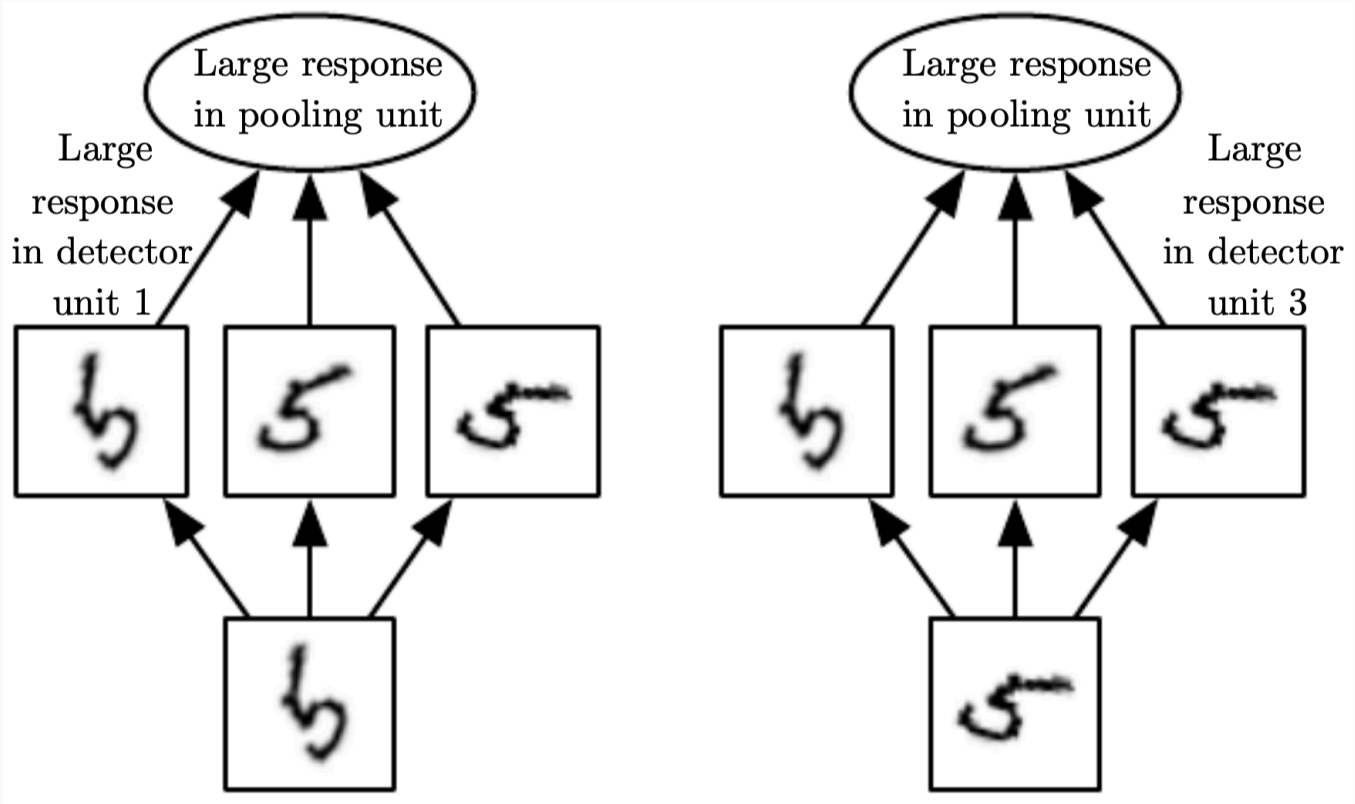


Figure 6: Network that detects the digit 5: 3 networks, each able to detect 5 in a different position, are stacked and then maximum is taken, see [2].

Network Architecture

Example on Figure 6 corresponds to pooling layer in the dimension of channels – detection if a feature is present at a location.

In convolution networks the pooling layer is commonly used in the spatial dimensions (for each channel separately) to detect if the features represented by different channels are present at some neighboring location.

Network Architecture

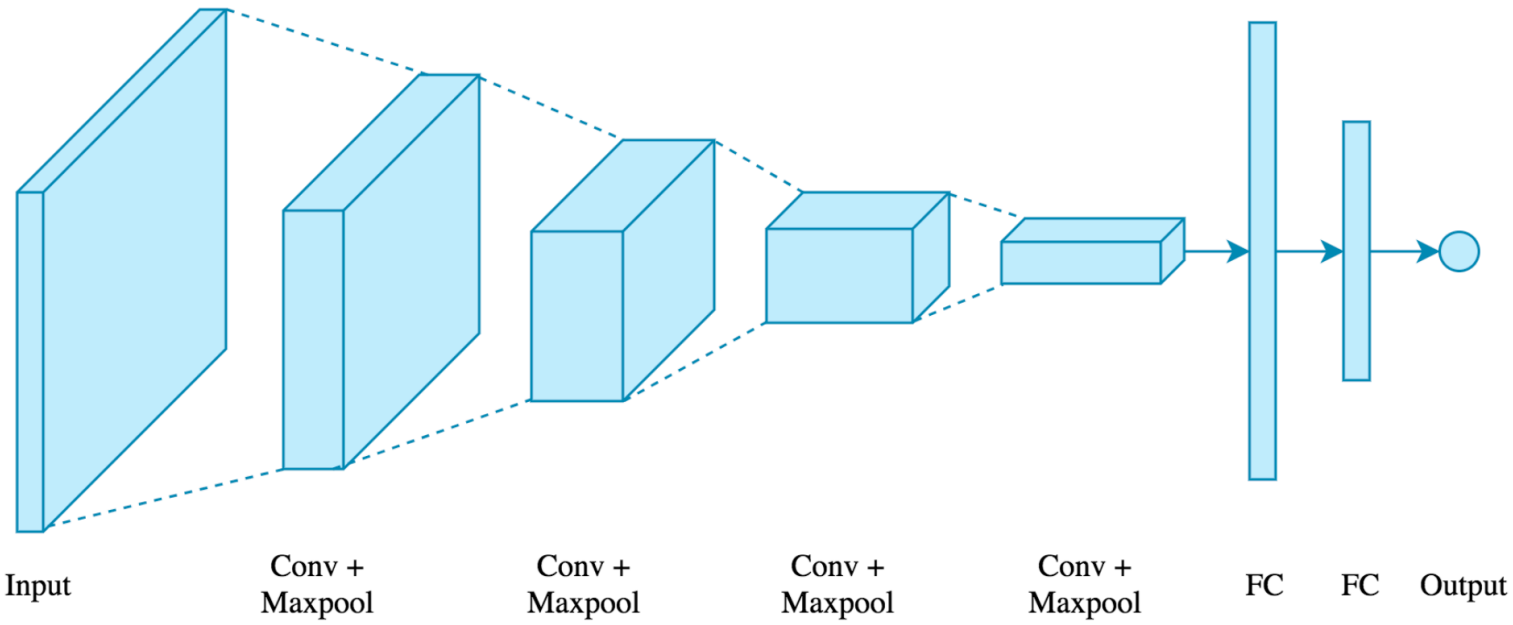


Figure 7: Basic convolution neural network architecture.

We repeat blocks of a) convolution, b) non-linear activation and optionally c) pooling. Number of channels is increasing – more global information uses higher dimensions. Finally, image is flattened and fully connected (FC) layers create an output vector.

Network Architecture

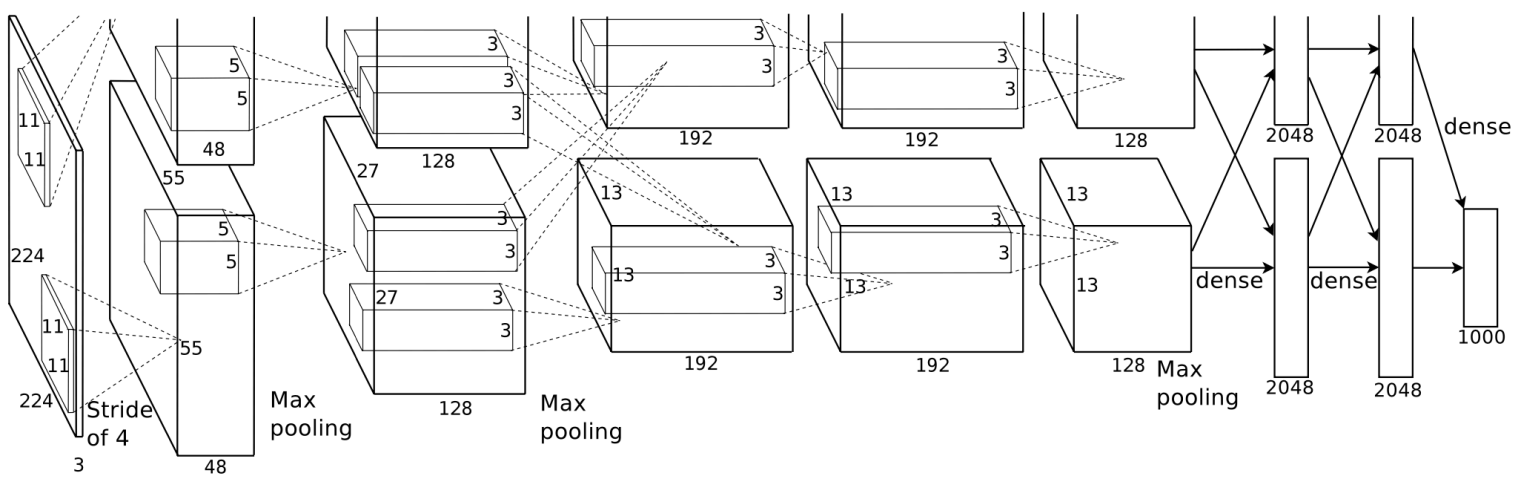


Figure 8: AlexNet network from [4] with around 60*M* parameters. The computation graph is separated into two lines which interact only at specific points for easy GPU parallelization.

In 2012, the AlexNet obtained 84.7% top-5 accuracy[6] on ImageNet [1].

[6] It is defined as the share of test examples for which the correct class is among the 5 classes with highest predicted probabilities.

Batch Normalization

Batch Normalization

During training we observe *internal covariate shift* – as the initial layers change, the distribution of their output changes as well and consequently the latter layers are inputted with data of previously unseen distribution.

It has been empirically observed [3] that internal covariate shift slows down training.

Batch Normalization

As we train using batches of data, each convolution layer produces a sequence $I^1, \ldots, I^B \in \mathbb{R}^{H \times W \times C}$, where h and w denote the spatial dimensions and C is the number of channels.

First, we center across the spatial dimensions and the batch

$$\tilde{I}^b_{\cdot,\cdot,c} = \left(I^b_{\cdot,\cdot,c} - \bar{I}^{\cdot}_{\cdot,\cdot,c}\right) / \sigma_{I^{\cdot}_{\cdot,\cdot,c}}, \quad b \in \{0, \ldots, B-1\}, c \in \{0, \ldots, C-1\}$$

where $\bar{I}^{\cdot}_{\cdot,\cdot,c}$ and $\sigma_{I^{\cdot}_{\cdot,\cdot,c}}$ denote the average and standard deviation over spatial and batch dimensions. Then we normalize

$$\tilde{\tilde{I}}^b_{\cdot,\cdot,c} = \left(\tilde{I}^b_{\cdot,\cdot,c} - \alpha_c\right) / \beta_c, \quad b \in \{0, \ldots, B-1\}, c \in \{0, \ldots, C-1\},$$

where $\alpha_c \in \mathbb{R}$ and $\beta_c \in (0, \infty)$ are trainable parameters.

Batch Normalization

The normalization is applied right after the convolution before the non-linear activation as suggested empirically [3].

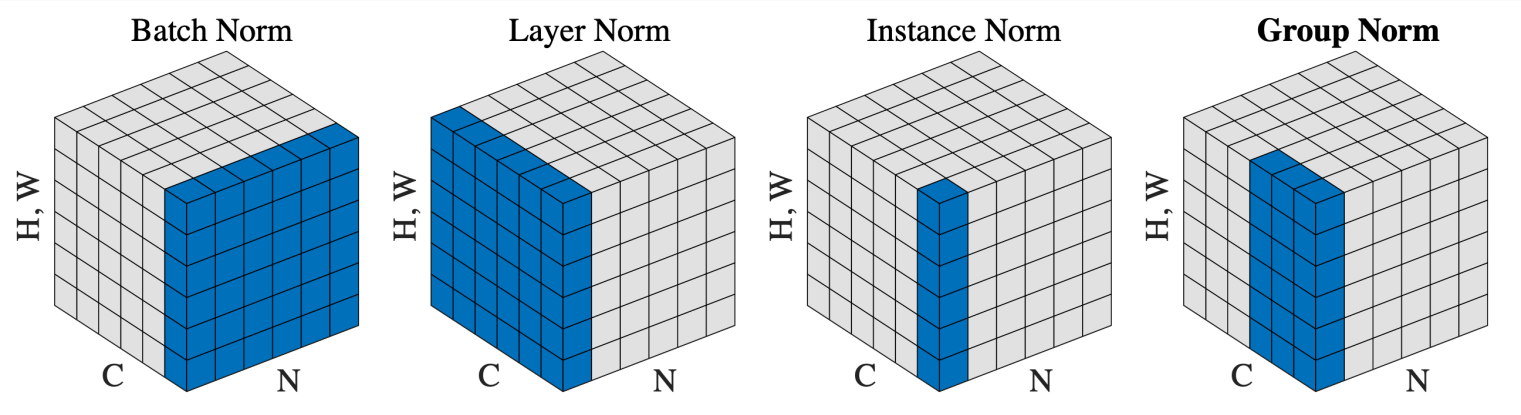


Figure 9: Diagram from [6] showing different possibilities for data normalization. In convolution networks batch normalization is the standard choice but this may vary in different architectures.

Here H, W are the spatial dimensions, C is the dimension of channels and N is the number of examples in batch. In batch normalization we obtain C averages and standard deviations for each channel.

Batch Normalization

During inference, it is not desirable that the prediction for one example would depend on other examples from the batch[7].

That is, we replace $\bar{I}_{:,:,:,c}$ and $\sigma_{I_{:,:,:,c}}$ by some other estimates of mean and standard deviation.

Commonly, mean and standard deviation are estimated empirically using exponential moving average during the training.

[7] This is a serious limitation of batch normalization and it has lead to a decline of its applications whenever possible. But for convolution networks it is still the best practice to use it.

Regularization Revisited

Regularization Revisited

Regularization technique to make the features (intermediate neuron outputs) universal, i.e. to have a valuable information irrespective to the other neuron values, via *dropout*, see [5].

During forward pass, each neuron is dropped[8] with certain probability[9] $p \in (0, 1)$.

[8] Simulated by setting the respective convolution weights to zero

[9] Typically, $p = 0.5$.

Regularization Revisited

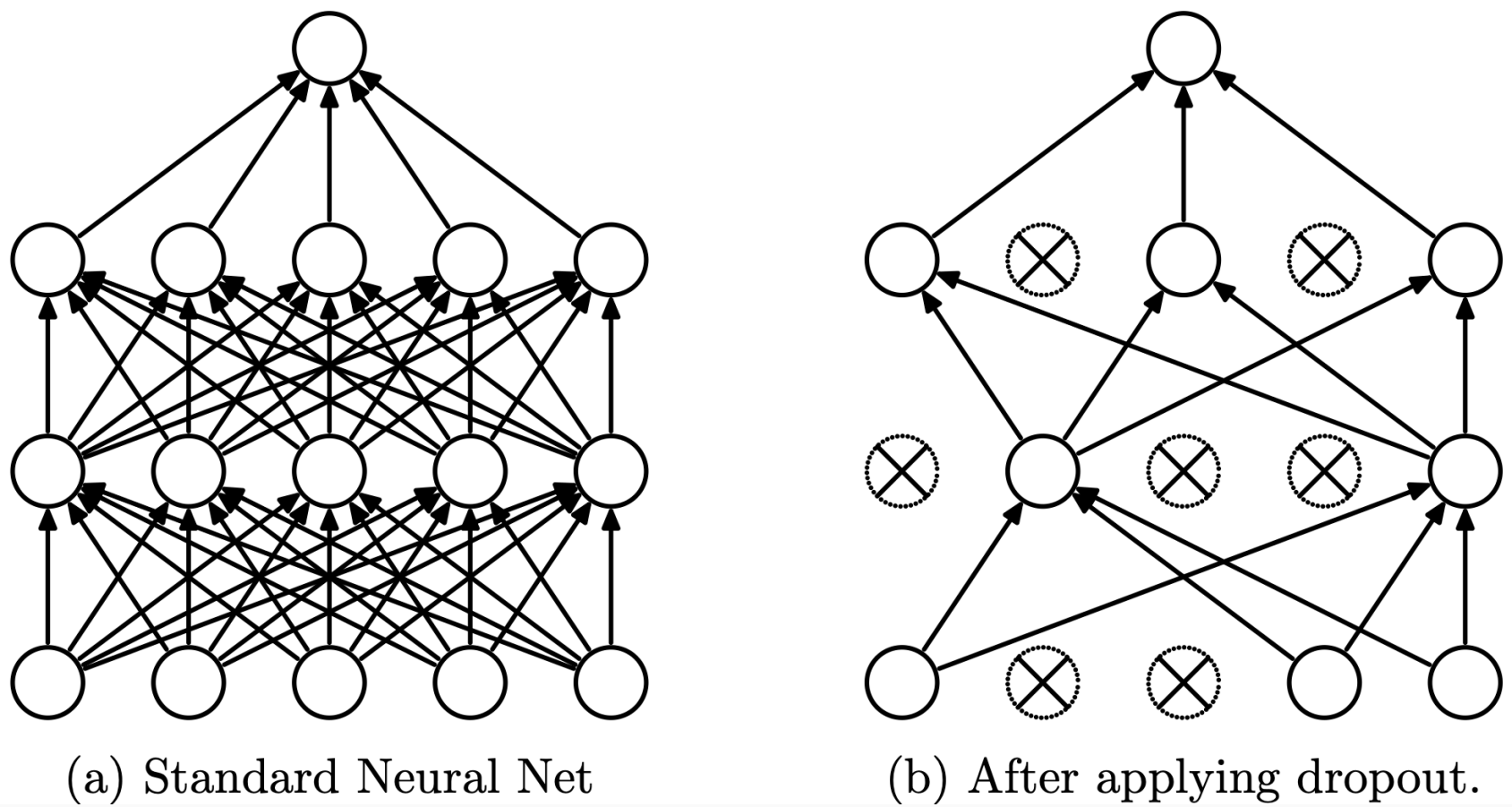
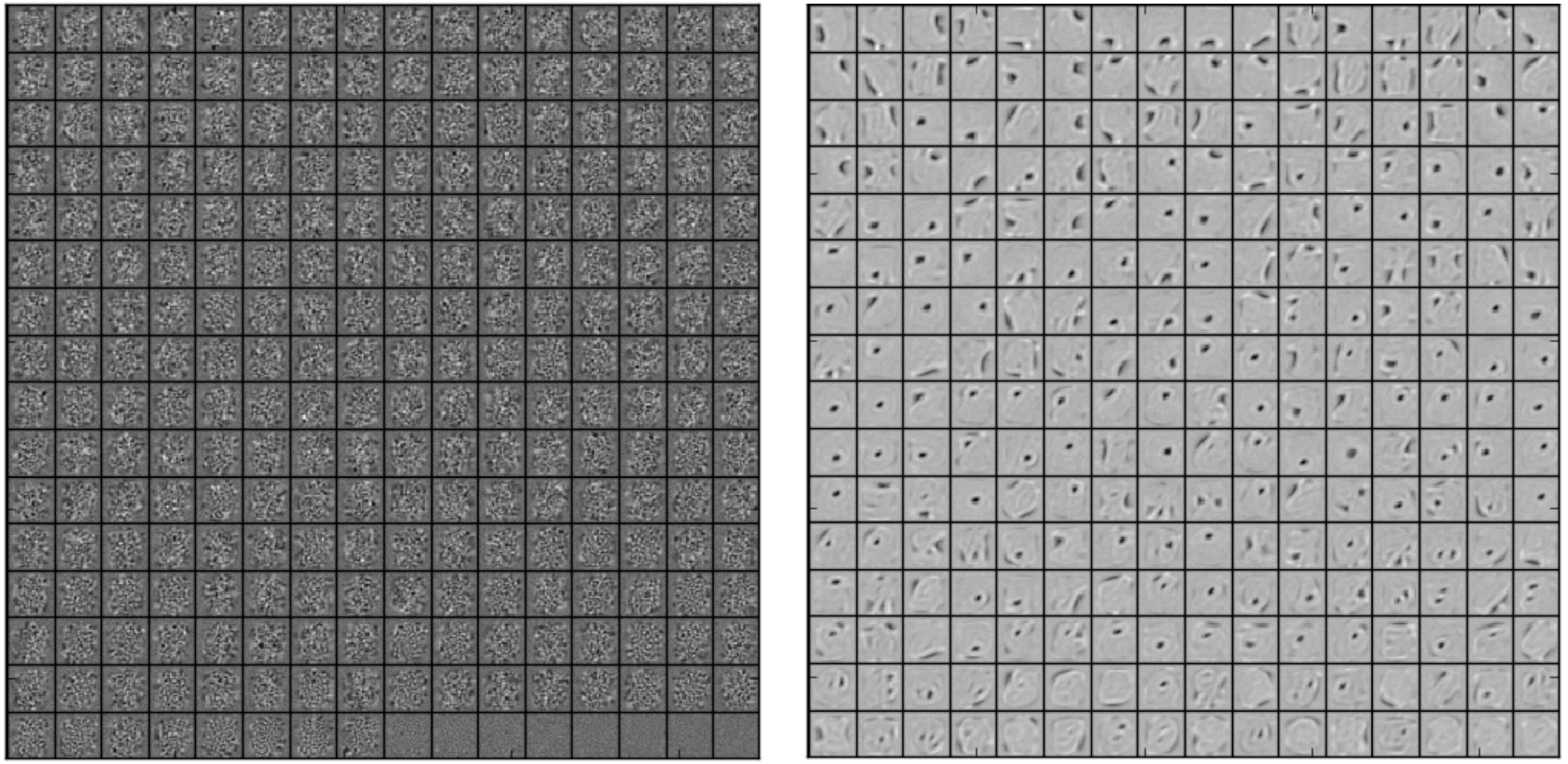


Figure 10: Two layer neural network with and without application of dropout.

Regularization Revisited



(a) Without dropout

(b) Dropout with $p = 0.5$.

Figure 11: Network with and without from [5]. The $23 \times 28 = 784$ pixel digit is flattened and connected to a hidden layer of size 256 with rectified linear unit followed by an output layer of size 784 trained with mean square loss to predict again the input image. Learned weight matrices of the first layer are compared.

Regularization Revisited

Similar test errors for both cases on Figure 11 can be observed.

Without dropout the neurons have co-adapted to produce good reconstructions. However, each hidden unit does not detect a meaningful feature.

With dropout, the hidden units detect edges, strokes and spots in parts of the image.

Regularization Revisited

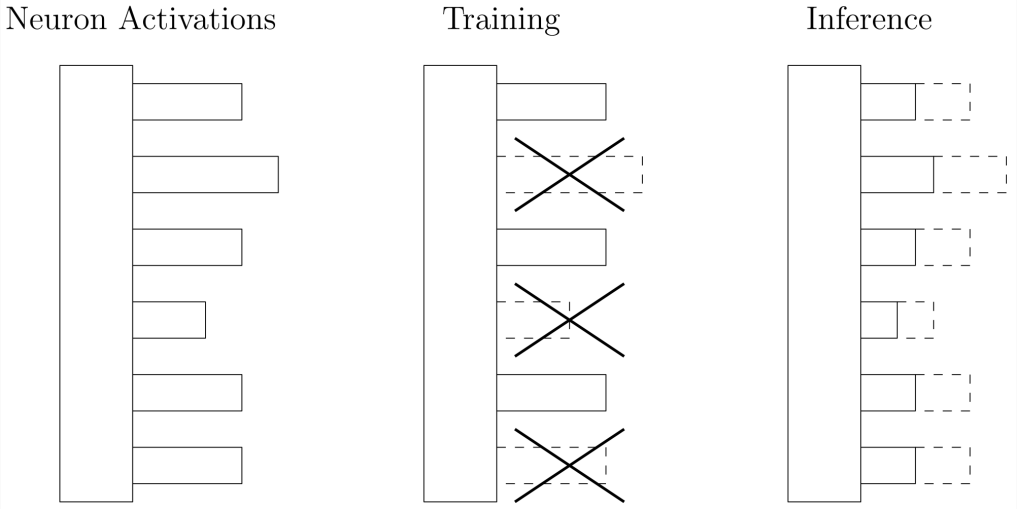


Figure 12: Comparison of neuron outputs with and without dropout.

Dropout is performed during training only as during the inference we do not want to limit the network capacity.

Dropout effectively decreased the sum of outputs in each neuron roughly by a factor of $1 - p$. Therefore, during training all the outputs are scaled by $1/(1 - p)$ so that the network can adapt to values which are expected during the inference.

Regularization Revisited

Another type of regularization technique is so called *data augmentation*: we intentionally damage/adjust the input while keeping the label unchanged. The augmentation usually depends on some random seed which is drawn independently for each training iteration.

This helps to lower the generalization error as

- we perform the augmentation differently in each iteration differently which leads to much more rich training dataset
- the network can not memorize the dataset as easily as it comes slightly modified in each iteration which enforces learning of general features.

Regularization Revisited

Examples of data augmentation in computer vision

- translations,
- rotations,
- scaling
- color adjustments
- MixUp: see [7], where two input images are combined using convex combination[10]
- CutMix: similar to MixUp but we choose some pixel regions from the first image and some from the other one

Each domain (speech recognition, video processing,...) has its own data augmentation techniques.

[10] Here, also the labels are adjusted – convex combinations of the original labels.

References i

J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei.
Imagenet: A large-scale hierarchical image database.
In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

I. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio.
Maxout networks.
In *International conference on machine learning*, pages 1319–1327. PMLR, 2013.

References ii


S. Ioffe and C. Szegedy.
Batch normalization: Accelerating deep network training by reducing internal covariate shift.
In *International conference on machine learning*, pages 448–456. pmlr, 2015.


A. Krizhevsky, I. Sutskever, and G. E. Hinton.
Imagenet classification with deep convolutional neural networks.
Advances in neural information processing systems, 25, 2012.


N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov.
Dropout: a simple way to prevent neural networks from overfitting.
The journal of machine learning research, 15(1):1929–1958, 2014.

References iii

Y. Wu and K. He.
Group normalization.
In *Proceedings of the European conference on computer vision (ECCV)*, pages 3–19, 2018.

H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz.
mixup: Beyond empirical risk minimization.
arXiv preprint arXiv:1710.09412, 2017.