A CONVOLUTIONAL NEURAL NETWORK FOR THE MNIST DATASET USING TENSORFLOW

Project report

FABIO LOCHE - STEFANO TEDESCHI

Corso di Reti Neurali
Anno Accademico 2016/2017

Corso di Laurea Magistrale in Informatica
Scuola di Scienze della Natura
Università degli Studi di Torino

ABSTRACT

Qua scriveremo l'abstract. . .

We have seen that computer programming is an art,
because it applies accumulated knowledge to the world,
because it requires skill and ingenuity, and especially
because it produces objects of beauty.

— Donald E. Knuth Knuth 1974

ACKNOWLEDGMENTS

Put your acknowledgments here.

Many thanks to everybody who already sent me a postcard!

Regarding the typography and other help, many thanks go to Marco Kuhlmann, Philipp Lehman, Lothar Schlesier, Jim Young, Lorenzo Pantieri and Enrico Gregorio[1], Jörg Sommer, Joachim Köstler, Daniel Gottschlag, Denis Aydin, Paride Legovini, Steffen Prochnow, Nicolas Repp, Hinrich Harms, Roland Winkler, Jörg Weber, Henri Menke, Claus Lahiri, Clemens Niederberger, Stefano Bragaglia, Jörn Hees, and the whole LaTeX-community for support, ideas and some great software.

Regarding LyX: The LyX port was intially done by *Nicholas Mariette* in March 2009 and continued by *Ivo Pletikosić* in 2011. Thank you very much for your work and for the contributions to the original style.

1 Members of GuIT (Gruppo Italiano Utilizzatori di TeX e LaTeX)

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

ACRONYMS

CNN	Convolutional Neural Network

MLP	Multylayer Perceptron
ReLU	Rectified Linear Units

1 INTRODUCTION

2 CONVOLUTIONAL NEURAL NETWORKS

Convolutional Neural Networks (CNNs) are a kind of artificial feed-forward neural network in which the organization of the connections between the neurons is inspired by the animal visual cortex. Actually, individual cortical neurons respond to stimuli in a restricted region of space known as the **receptive field**. The receptive fields of different neurons partially overlap such that they tile the visual field. Similarly, in CNNs each neuron is connected with only a small subset of inputs from the previous layer.

These networks are extremely useful when dealing with data with a grid-like topology, such as time-series data or images.

The name "convolutional neural network" indicates that the network employs a specialized kind of linear operation called **convolution**. In short, as stated in [Goodfellow, Bengio, and Courville 2016], *convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers*.

2.1 MAIN FEATURES

While traditional Multylayer Perceptron (MLP) models were successfully used in the past for image recognition, due to the full connectivity between nodes they suffer from the curse of dimensionality and thus do not scale well to higher resolution images.

For instance, in the CIFAR-10 dataset, images are of size 32x32x3 (32 wide, 32 high, 3 color channels), so a single fully connected neuron in the first hidden layer of a regular MLP would have $32 * 32 * 3 = 3,072$ weights. A 200x200 image, however, would lead to neurons that have $200 * 200 * 3 = 120,000$ weights. Such network architecture does not take into account the spatial structure of data, treating input pixels which are far apart or close together exactly in the same way. The full connectivity of neurons is wasteful in the framework of image recognition, and the huge number of parameters quickly leads to overfitting.

As said before, convolutional neural networks are biologically inspired variants of multilayer perceptrons, designed to emulate the behaviour of a visual cortex. These models mitigate the challenges posed by the MLP architecture by exploiting the strong spatially local correlation present in natural images. In particular, CNNs have the following distinguishing features:

3D VOLUMES OF NEURONS The layers of a CNN have neurons arranged in 3 dimensions: width, height and depth. Neurons inside a

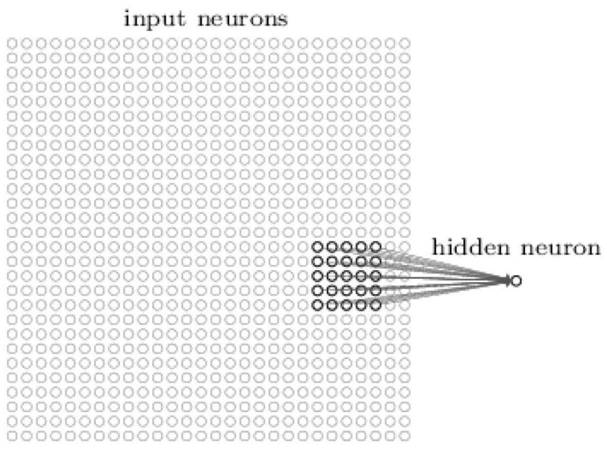

Figure 1: Implementation of a 5x5 receptive field in a CNN

layer are only connected to a small region of the layer before it, called a receptive field. Figure 1, for instance, shows a 5x5 receptive field from the input neurons to the first hidden layer, with a 28x28 input. Distinct types of layers, both locally and completely connected, are stacked to form the CNN architecture.

LOCAL CONNECTIVITY Following the concept of receptive field, CNNs exploit spatially local correlation by enforcing a local connectivity pattern between neurons of adjacent layers. The architecture thus ensures that every **filter** (i. e. weight patch) learnt produces the strongest response to a spatially local input pattern. Stacking many of such layers leads to non-linear filters that become increasingly "global" (i. e. responsive to a larger region of input space). This allows the network to first create good representations of small parts of the input, then assemble representations of larger areas from them.

SHARED WEIGHTS In CNNs, each filter is replicated across the entire visual field. These replicated units share the same parameterization (weight vector and bias) and form a feature map. This means that all the neurons in a given convolutional layer detect exactly the same features. Replicating units in this way allows for features to be detected regardless of their position in the visual field, thus constituting the property of translation invariance.

Together, these properties allow convolutional neural networks to achieve better generalization performances in vision problems. Moreover, th weight sharing helps by dramatically reducing the number of free parameters being learnt, thus lowering the memory requirements for running the network. Decreasing the memory footprint allows the training of larger, more powerful networks.

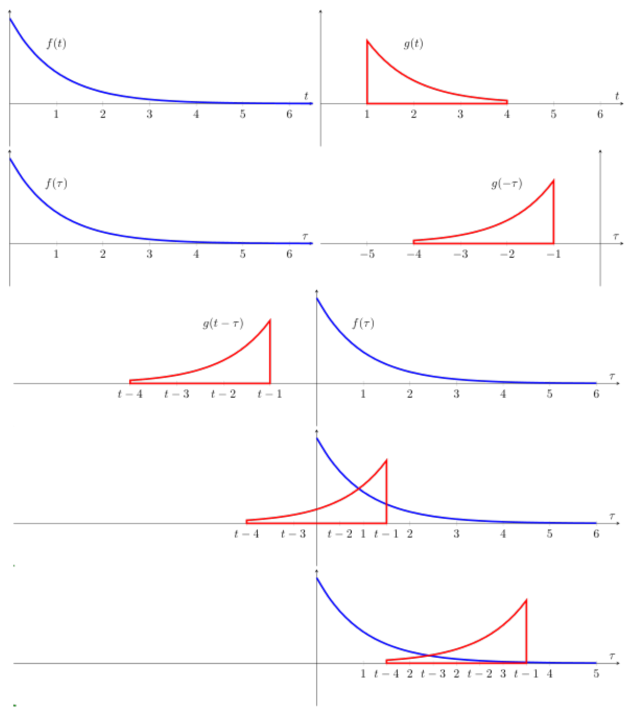

Figure 2: Visual explanation of the convolution operation

2.2 THE CONVOLUTION OPERATION

In its most general form, convolution is an operation on two functions of a real-valued argument. It produces a third function, that is typically viewed as a modified version of one of the original functions, giving the integral of the point-wise multiplication of the two functions as a function of the amount that one of the original functions is translated. It is typically denoted with an asterisk and it is defined as:

$$s(t) = (f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)\, d\tau$$

The convolution formula can be described as a weighted average of the function $f(\tau)$ at moment t where the weighting is given by $g(-\tau)$ simply shifted by amount t. As t changes, the weighting function emphasizes different parts of the input function. The output is sometimes referred to as **feature** or **activation map**.

Figure 2 shows a visual explanation of the operation. Wherever the two functions intersect, the integral of their product is found. In other words, it computes a sliding, i. e. a weighted-sum of function $f(\tau)$, where the weighting function is $g(-\tau)$.

If we now assume that f and g are defined only on integer t, we can define the discrete convolution as:

$$s(t) = (f * g)(t) = \sum_{-\infty}^{\infty} f(\tau)g(t - \tau)$$

In convolutional networks terminology, the first argument of the convolution is often referred to as input and the second as kernel or filter. In machine learning applications, the input is usually a multidimensional array (i. e. a tensor) of data and the kernel is usually a multidimensional array of parameters that are adapted by the learning algorithm.

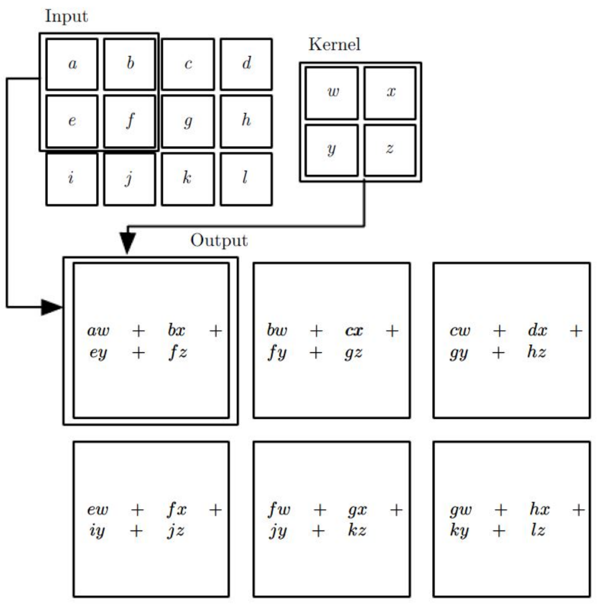

Figure 3: Example of 2-D convolution without kernel-flipping, as reported in [Goodfellow, Bengio, and Courville 2016]

Figure 3 shows an example of convolution (without kernel flipping) applied to a 2-D tensor. In this case, the output is restricted to only positions where the kernel lies entirely within the image.

2.3 ARCHITECTURE

A CNN architecture is formed by a stack of distinct layers that transform the input volume into an output volume through a differentiable function. A few distinct types of layers are commonly used and they are described below.

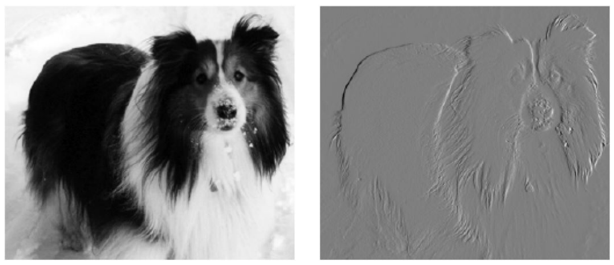

Figure 4: Application of a filter for edge detection (Photo credit: Paula Goodfellow)

2.3.1 *Convolutional layer*

The convolutional layer is the core building block of a CNN. The layer's parameters consist of a set of learnable filters (or kernels), which have a small receptive field, but extend through the full depth of the input volume. During the forward pass, each filter is convolved across the dimensions of the input volume, computing the dot product between the entries of the filter and the input and producing a feature map of that filter.

The amount of units by which the filter shifts is called **stride** and it controls how the filter convolves around the input volume.

As a result of this process, the network learns filters that activate when they detect some specific type of features in a given spatial position in the input.

It is evident that different filters will produce different feature maps for the same input. Figure 4 shows the result of the application of a filter for edge detection to a given image. The image on the right was formed by taking each pixel in the original image and subtracting the value of its neighboring pixel on the left. This shows the strength of all of the vertically oriented edges in the input image, which can be a useful operation for object detection.

Stacking the activation maps for all filters along the depth dimension forms the full output volume of the convolution layer. Every entry in the output volume can thus also be interpreted as an output of a neuron that looks at a small region in the input and shares parameters with neurons in the same activation map.

In fact, in order to recognize an image, we'll need more than one filter. For this reason, the output of a full convolutional layer will be a set of feature maps with a structure similar to the one in figure 5.

Sometimes it is convenient to have an output of the same size of the input. In this case it is necessary to add some additional pixels to the input image, this is called **padding**. For instance, a zero padding like the one in figure 6 pads the input volume with zeros all around the border. The size of this zero-padding is an optional hyperparameter.

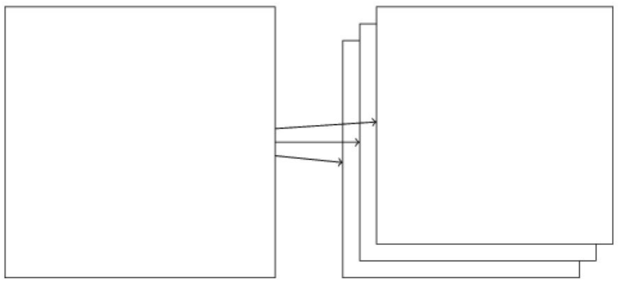

Figure 5: Application of three filters to a given input producing three different feature maps

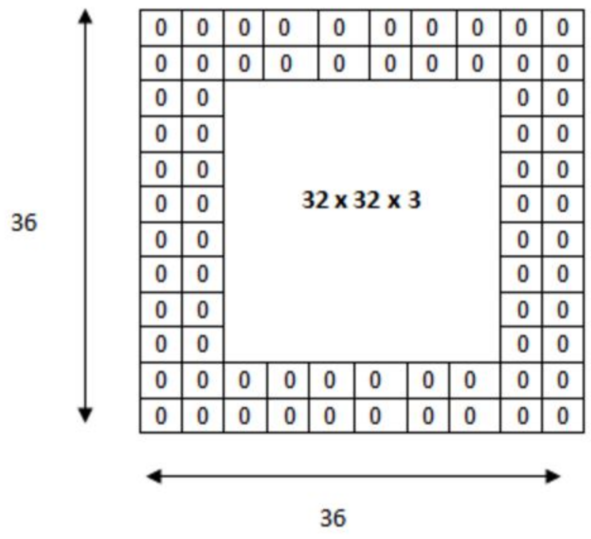

Figure 6: Zero padding of two applied to a 32x32x3 picture

Zero padding, in particular, provides control of the output volume spatial size.

In general, the formula for calculating the output size for any given convolutional layer is:

$$O = \frac{W - K + 2P}{S} + 1$$

where O is the output height/length, W is the input height/length, K is the filter size, P is the padding, and S is the stride. If this number is not an integer, then the strides are set incorrectly and the neurons cannot be tiled to fit across the input volume in a symmetric way.

Setting the padding to

$$P = \frac{K - 1}{2}$$

when the stride is $S = 1$ ensures that the input volume and output volume will have the same size spatially.

2.3.2 *ReLU layer*

After a convolutional layer, it is convention to apply a non-linear layer (or activation layer) immediately afterward. The purpose of this layer is to introduce non-linearity to a system that basically has just been computing linear operations during the convolution operation (just element wise multiplications and summations). This is a layer of neurons that applies the non-saturating activation function:

$$f(x) = \max(0, x)$$

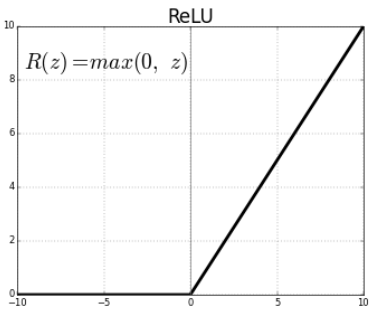

Figure 7: ReLU function

This increases the non-linear properties of the decision function and of the overall network without affecting the receptive fields of the convolution layer. The output of this layer known as **rectified feature map**.

In the past, non-linear functions like $tanh$ and sigmoid were used, but researchers found out that ReLU layers work far better because the network is able to train a lot faster (because of the computational efficiency) without significant reductions in the accuracy [Nair and Hinton 2010]. Using a ReLU function also helps to alleviate the vanishing gradient problem, which is the issue where the lower layers of the network train very slowly because the gradient decreases exponentially through the layers.

The ReLU layer applies the function $f(x) = max(0, x)$ to all of the values in the input volume. In basic terms, this layer just changes all

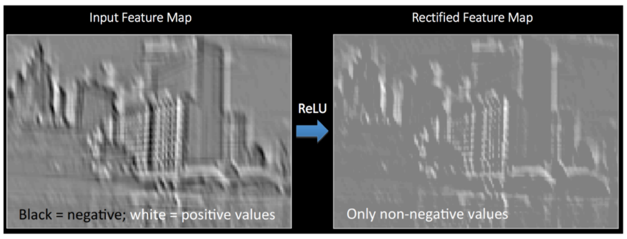

Figure 8: ReLU operation. Source http://mlss.tuebingen.mpg.de/2015/slides/fergus/Fergus_1.pdf

the negative activations to 0. Figure 8 shows the input of a ReLU layer and the resulting output.

2.3.3 *Pooling layer*

Another important concept of CNNs is pooling, which is a form of non-linear down-sampling. There are several non-linear functions to implement pooling among which **max pooling** is the most common. It partitions the input image into a set of non-overlapping regions and, for each of them, outputs the maximum value. The intuition is that once a feature has been found, its exact location isn't as important as its rough location relative to other features. The function of the pooling layer is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network, and hence to control overfitting, too. It is common to periodically insert a pooling layer in-between successive convolutional/ReLU layers in a CNN architecture. The pooling operation also provides a form of translation invariance. This is very powerful since we can detect objects in an image no matter where and how they are located.

The pooling layer operates independently on every depth slice of the input and resizes it spatially.This operation is also known as subsampling or downsampling. The most common form is a pooling layer with filters of size 2x2 applied with a stride of 2. Every operation would in this case be taking a max over 4 numbers. Figure 9 shows an example of this operation.

In addition to max pooling, the pooling units can also perform other functions, such as average pooling and even L2-norm pooling. Average pooling was often used historically but has recently fallen out of favor compared to the max pooling operation, which has been found to work better in practice, as reported in [Scherer, Müller, and Behnke 2010]. Due to the aggressive reduction in the size of the representation (which is helpful only for smaller datasets to control overfitting), the current trend in the literature is towards using smal-

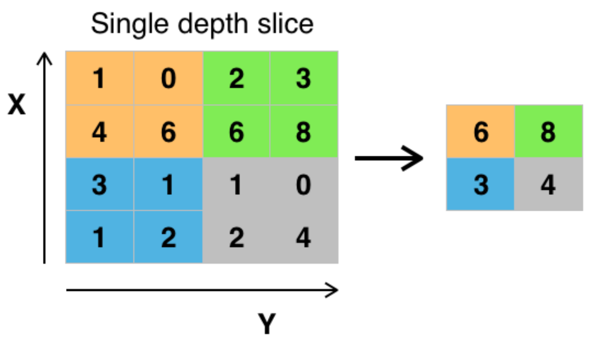

Figure 9: Max pooling with a 2x2 filter and stride $S = 2$

ler filters [Graham 2014] or completely discarding the pooling layer [Springenberg et al. 2014].

Obviously, the pooling operation is applied separately to each feature map.

2.3.4 *Fully connected layer*

Finally, after several convolutional/ReLU and max pooling layers, the high-level reasoning in the neural network is done via fully connected layers. Neurons in a fully connected layer have full connections with all the activations in the previous layer, as in traditional MLPs. Their activations can hence be computed with a matrix multiplication followed by a bias offset.

The output from the convolutional and pooling layers represent high-level features of the input image. The purpose of the fully connected layer is usually to use these features for classifying the input into various classes based on the training dataset. For instance, the image classification aim to put a given image into a precise category.

Apart from classification, adding a fully-connected layer is also a (usually) cheap way of learning combinations of these features. Most of the features from convolutional and pooling layers may be good for the classification task, but combinations of those features might be even better.

2.3.5 *Dropout layer*

Dropout layers have a very specific function in neural networks. The problem of overfitting is very important in machine learning. It happens when after training, the weights of the network are so tuned to the training examples they are given that the network doesn't perform well when given new examples.

The idea of dropout is simplistic in nature. This layer "drops out" a random set of activations in that layer by setting them to zero in the forward pass. At each training stage, individual nodes are either "dropped out" of the net with probability $1 - p$ or kept with probability p, where p is a user-defined parameter so that a reduced network is left. The removed nodes are then reinserted into the network with their original weights.

This pruning, in a way, forces the network to be redundant; the network should be able to provide the right classification or output for a specific example even if some of the activations are dropped out.

This makes sure that the network isn't getting too "fitted" to the training data and thus helps alleviating the overfitting problem. An important note is that this layer is only used during training.

The method also significantly improves the speed of training. This makes model combination practical, even for deep neural nets. The technique seems to reduce the complex, tightly fitted interactions between nodes, leading them to learn more robust features which better generalize to new data. Dropout has been shown to improve the performance of neural networks on tasks in vision, speech recognition, document classification, and computational biology.

An interesting analysis of the use of dropout in order to manage overfitting can be found in [Srivastava et al. 2014]

2.3.6 *Loss layer*

Finally, a loss layer specifies how the network training penalizes the deviation between the predicted and true labels. This is normally the last layer in the network. Various loss functions appropriate for different tasks may be used there.

Softmax loss is used for predicting a class of K different classes (i. e. the digit represented by an image from the MNIST dataset of handwritten digits, see chapter **??** for further details).

The function

$$f_j(z) = \frac{e^{z_j}}{\sum_k e^{z_k}}$$

is called the softmax function: It takes a vector of arbitrary real-valued scores and squashes it to a vector of values between zero and one that sum to one.

Moreover, sigmoid cross-entropy loss is used for predicting K independent probability values in $[0, 1]$. Euclidean loss is used for regressing to real-valued labels in $[-\infty, \infty]$.

These layers are the basic building blocks of any CNN. They can be stacked in order to produce complex architectures. The number of layers depends on the addressed task and on the complexity of the input. A typical convolutional architecture is composed by one or more sets of convolutional, ReLU and pooling layers and ends with a fully connected, a dropout and a loss layer. Again, the loss function to use depends on the task that has to be solved.

Figure 10 shows an example of architechture with two convolutional layers and two pooling layers.

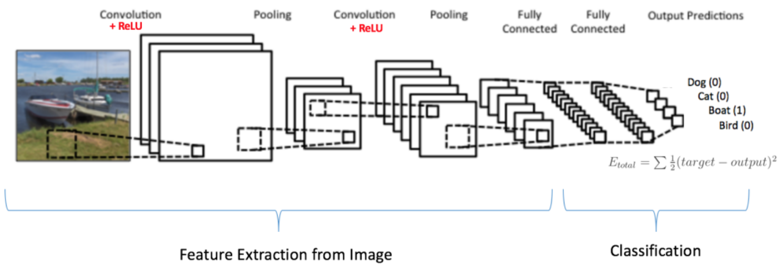

Figure 10: Convolutional neural network for the classification of an image

Many different architectures have been proposed. Among them, LeNet [LeCun et al. 1998] was one of the very first convolutional neural networks which helped propel the field of deep Learning. In 2012, Alex Krizhevsky and others released AlexNet [Krizhevsky, Sutskever, and Hinton 2012] which was a deeper and much wider version of the LeNet and won by a large margin the difficult ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012. The ILSVRC 2014 winner was GoogleNet, a convolutional network from [Szegedy et al. 2014] from Google. Its main contribution was the development of an Inception Module that dramatically reduced the number of parameters in the network. In 2016 GoogleNet (now called Inception) is at its third release.

2.4 TUNING

3 TENSORFLOW BASICS

REFERENCES

Goodfellow, I., Y. Bengio, and A. Courville (2016). *Deep Learning*. http://www.deeplearningbook.org. MIT Press (cit. on pp. 2, 5).

Graham, B. (2014). “Fractional Max-Pooling.” In: *CoRR* abs/1412.6071. URL: http://arxiv.org/abs/1412.6071 (cit. on p. 10).

Knuth, D. E. (1974). “Computer Programming as an Art.” In: *Communications of the ACM* 17.12, pp. 667–673 (cit. on p. iii).

Krizhevsky, A., I. Sutskever, and G. E. Hinton (2012). “Imagenet classification with deep convolutional neural networks.” In: *Advances in neural information processing systems*, pp. 1097–1105 (cit. on p. 12).

LeCun, Y., L. Bottou, Y. Bengio, and P. Haffner (1998). “Gradient-Based Learning Applied to Document Recognition.” In: *Proceedings of the IEEE* 86.11, pp. 2278–2324 (cit. on p. 12).

Nair, V. and G. E. Hinton (2010). “Rectified Linear Units Improve Restricted Boltzmann Machines.” In: *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*. Ed. by J. Fürnkranz and T. Joachims. Omnipress, pp. 807–814. URL: http://www.icml2010.org/papers/432.pdf (cit. on p. 8).

Scherer, D., A. Müller, and S. Behnke (2010). “Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition.” In: *Proceedings of the 20th International Conference on Artificial Neural Networks: Part III*. ICANN’10. Thessaloniki, Greece: Springer-Verlag, pp. 92–101. URL: http://dl.acm.org/citation.cfm?id=1886436.1886447 (cit. on p. 9).

Springenberg, J. T., A. Dosovitskiy, T. Brox, and M. A. Riedmiller (2014). “Striving for Simplicity: The All Convolutional Net.” In: *CoRR* abs/1412.6806. URL: http://arxiv.org/abs/1412.6806 (cit. on p. 10).

Srivastava, N., G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov (2014). “Dropout: A Simple Way to Prevent Neural Networks from Overfitting.” In: *J. Mach. Learn. Res.* 15.1, pp. 1929–1958. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=2627435.2670313 (cit. on p. 11).

Szegedy, C., W. Liu, Y. Jia, P. Sermanet, S. E. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich (2014). "Going Deeper with Convolutions." In: *CoRR* abs/1409.4842. URL: `http://arxiv.org/abs/1409.4842` (cit. on p. 12).