# Convolutional Neural Networks in Computer Vision

(MP 021 - The group that has to go deeper.)

Lauri Naalisvaara (78968E) Niklas Strengell (83913L)

January 26, 2018

**Abstract**

Convolutional neural networks have completely revised the world of computer vision. Inspired by the biological eye they perform tasks that were at some point thought to be incredibly tedious. Slowly their development has evolved from reading bank cheques to today's advancements, where the deep learning algorithms are achieving accuracy rates that surpass even humans. Using TensorFlow and Keras toolkits we are able to build and a 12-layer CNN that has 76.7% accuracy rate on the CIFAR-10 database. The performance is on-par with the older state-of-the-art technologies. Furthermore, the CNN utilizes modern state-of-the-art technologies such as dropout and data-augmentation and was trained using only a single laptop without a GPU.

## 1 Introduction

Biological processes have always inspired technological advancements: the artificial neuron was developed not long after the working principle of the biological neurons was found. This groundbreaking innovation of the 1960's is what led the foundations of today's deep learning boom. Later when researchers looked into the human visual system and built a neural network using technologies inspired by the biological eye, they created a computer vision system with before unseen capabilities and accuracy - the convolutional neural networks.

Our research in this paper continues in these same footsteps. We will go trough the working principles of convolutional neural networks (later also CNN) and their history and achievements. We will also build our own model of a CNN and test it's performance using the industry standard database CIFAR-10. We'll go trough our experiments and discuss our results. We will also shortly ponder on the impact of CNNs on computer vision and the future of the field.

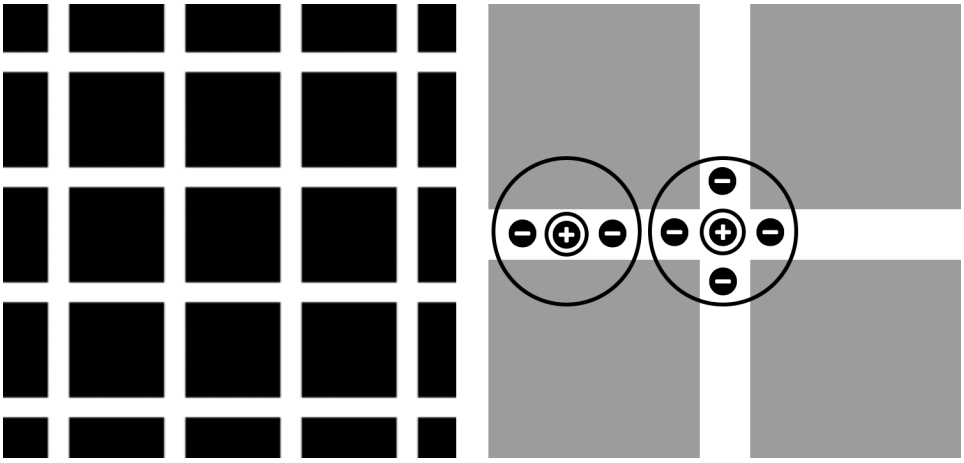

Figure 1: The Hermann grid is a great way to illustrate how the receptive fields work. They are great for detecting discrete changes in the environment, such as edges or lines. However, when the feature is too small for the receptive fields an illusion of a black dot will be created. [7]

## 2 Related work

In 1966, the early days of artificial intelligence, computer vision was thought to be a simple problem that could be solved through a summer project utilizing a camera connected to a computer and "describing what it saw" [15]. The problem turned out to be a tiny bit harder than expected, and although technological advancements were made, computer vision's practical usability remained stagnant until the 1990s. Then CNNs and their possibilities on various problems were introduced by LeCun et al [13][11]. Finally, in 1998 a pioneering convolutional neural network called the LeNet-5 was introduced in a paper. It improved the accuracy of automated classification of hand written digits from 68% to 82%. The system was used by several banks to read cheques proving the usability of CNNs also "in the wild" [12].

The final proof came when a CNN called the AlexNet and developed by Krizhevsky et al won the Imagenet Large Scale Visual Recognition Challenge in 2012. Their method had a top-5 error rate of only 15.3%. By comparison, the second best entry in the competition had a 26.2& error-rate. Ever since convolutional neural networks have been a stable technique in computer vision [9].

As pointed out earlier, their motivation and inspiration lies in the biological eye and the intuition behind their working principle can be best understood trough understanding of the human visual system [11].

The retina of the eye has local and slightly overlapping receptive fields, which send inhibiting or exciting signals when stimulated by light. These inputs are then gathered by the retinal ganglion cells and fed into the visual

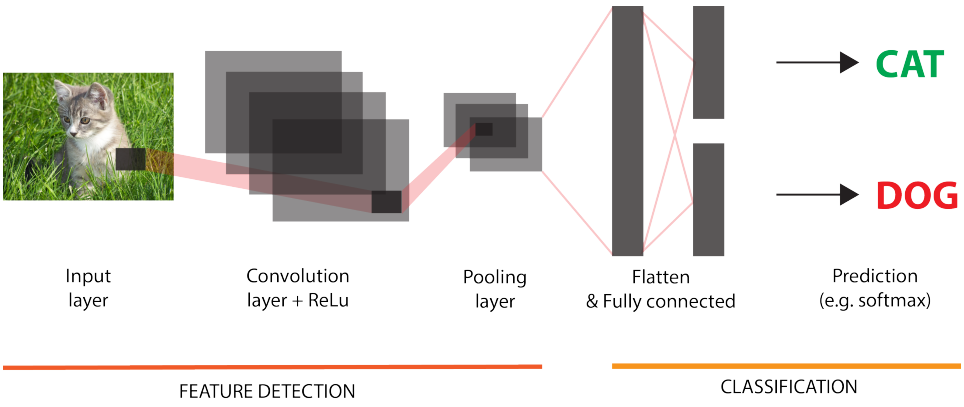


Figure 2: The structure of a stereotypical CNN. Though usually a functional CNN utilizes multiple convolution and pooling layers.

cortex, where through combining the information the neurons can extract visual features such as edges, end points and corners and later conceptual wholes, such as "a car" [2]. In figure 1 the reader can find an example of the receptive fields in the human eye.

A general CNN has a similar structure, where the first layers of the network are used to extract features from the raw data (such as images). The raw input is subjected to various filters at different resolutions at the *convolution layers*. After each convolution layer, there is an *artificial neuron* with a non-linearity function such as a rectified linear unit or hyperbolic tangent. The convolved output is then fed onto the next layers through *a pooling layer*, where the input is downscaled for the next convolution layer. By similar mechanisms as the overlapping receptive fields in the eye and the pooling ganglion cells, these layers in the feature learning stage can learn to spot simple features such as edges or complicated geometrical shapes or even abstract probability distributions. After the feature learning stage, everything is flattened and fed onto a *fully connected layer* after which a prediction of the image's class is produced for example by a soft-max function. Through back-propagation the network then learns which filters are best for feature extractions and which activations correspond to which class.

## 3 Method

Our task is to automatically identify objects in natural photographs. It is a hard problem and before deep learning automation was impractical. Even traditional neural networks break down on this problem, but convolutional neural networks can make the automation possible. We chose this problem as our project, because it highlights the complexity yet the elegance and

possibilities of deep learning algorithms. Further motivation is authors' fascinations with computer vision and appreciation of the animal visual system.

We build our CNN using the deep learning library TensorFlow for Python [1]. However, our code is written using the wrapper library Keras, which makes it possible to use another deep learning toolkit called Theano [5][4]. The final structure is a deep convolutional neural network with a total of 12-layers. Our baseline can be seen in figure 4 and described as the following:

1. Convolutional input layer, 32 feature maps with a size of 5*5 and a rectifier activation function.
2. Dropout layer at 20%.
3. Max Pool layer with size 3*2.
4. Convolutional layer, 32 feature maps with a size of 5*5 and a rectifier activation function.
5. Dropout layer at 20%.
6. Max Pool layer with size 3*2.
7. Flatten layer.
8. Fully connected layer with 1,024 units and a rectifier activation function.
9. Dropout layer at 30%.
10. Fully connected layer with 512 units and a rectifier activation function.
11. Dropout layer at 30%.
12. Fully connected output layer with 10 units and a softmax activation function.

As you can see, compared to a generic CNN presented in figure 2 we also utilized a dropout layer. Dropout is a technique that is utilized to reduce overfitting the data. In it a certain number of randomly selected neurons are ignored during the training. For example, dropout percentage of 20% means that every fifth randomly selected neuron is ignored during training. This increases the performance of the networks because it is not always dependent on some specific neuron for any specific feature activation. This is believed to be due to multiple independent internal representations being learned by the network.[16]

The rest of algorithms used are standard techniques in CNNs nowadays. As a cost function $J$ we use categorical cross entropy. Mathematically, it can be formulated as:

$$J = H(p, q) = -\Sigma_x p(x) log(q(x))$$

The cross entropy measures how many bits we need to encode an event using our predicted probability distribution $q$ (i.e. our prediction of the content in the picture) rather than the "true" distribution $p$ (i.e. the true content of the pictures). The gradient descent $\nabla w$ is now calculated as:

$$\nabla w = w - \alpha \frac{\partial J}{\partial w}$$

Where $\alpha$ is the learning rate. To further reduce overfitting we also used L2 regularisation (also called *weight decay* or *ridge regression*) in the fully connected layers. Ridge regression adds a squared magnitude as a penalty term to our cost function. Then if our cost function $J$ is now calculated with added regularization term $\frac{1}{2}w^2$, it becomes the following:

$$J_{L2} = J + \frac{1}{2}w^2$$

and it can be derived as:

$$\frac{\partial J_{L2}}{\partial w} = \frac{\partial J}{\partial w} + w$$

Now the gradient descent becomes the following:

$$\begin{aligned} \nabla w &= w - \alpha \frac{\partial J_{L2}}{\partial w} \\ &= w - \alpha(\frac{\partial J}{\partial w} + w) \\ &= w - \alpha \frac{\partial J}{\partial w} - \alpha w \\ &= (1 - \alpha)w - \alpha \frac{\partial J}{\partial w} \end{aligned}$$

Our chosen non-linearity was ReLU, which utilizes a rectifier function $f(x) = x^+ = max(0, x)$. ReLUs don't give as good initial results as tanh or sigmoid neurons do, but they are much faster to train and are thus the most used today [10].

The final prediction (output) layer utilizes the softmax function which "squashes" a $K$-dimensional vector $z$ containing real values to a $K$-dimensional vector $\sigma(z)$ with real values in the range $[0, 1]$ that add up to 1 (i.e. $z$-vector contains the likelihood of each class f).

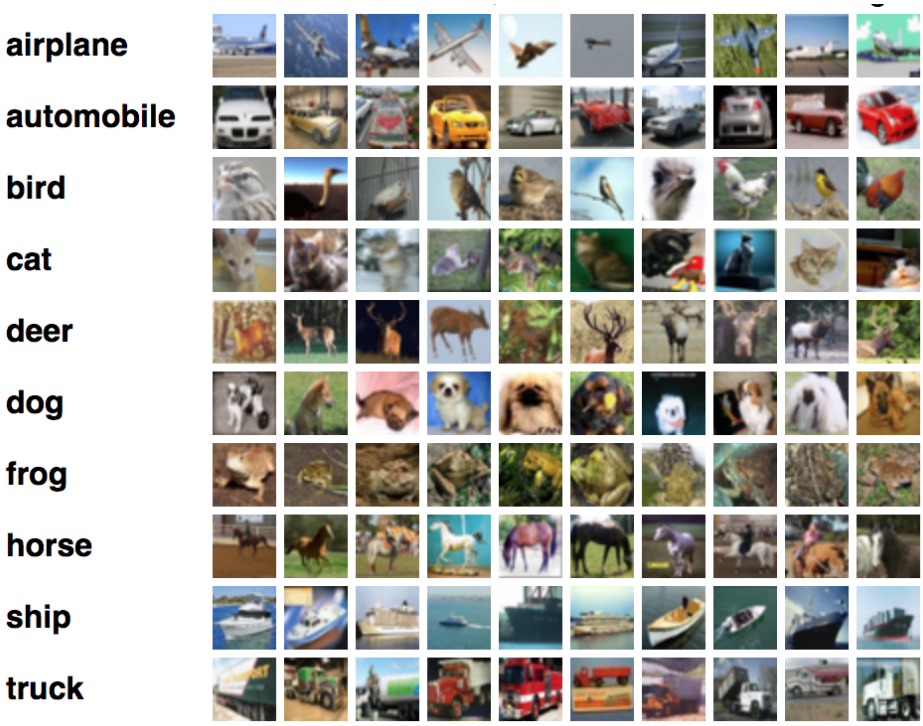


Figure 3: Examples images from the CIFAR-10 dataset [8].

## 4 Data

We used the CIFAR-10-dataset, which is wisely used as a benchmark for computer vision tasks. It consists of 60 000 images. The images are 32 pixels by 32 pixels large and are each labeled with a class. There are 10 different classes (hence the name). In total, there are 50 000 training images and 10 000 test images. This thus equals to 5000 training images and 1000 test images for each class.[8].

The classes are also mutually exclusive, hence an image of a truck is never labeled as an image of an automobile. To further avoid confusion, there are no images with ambivalent class - like pictures of a pick-up truck.

We further applied a horizontal-flipping to our data, thus practically doubling the amount of training picture we had. We also thought about rotating and zooming the pictures, but given that pictures in the dataset we're already so small, it didn't prove out very fruitful. Our total count was thus 120 000 images of which 100 000 were used for training and 20 000 for testing.

Keras proved hear to be a problematic tool as it always processed the data "on demand". We thus needed to preprocess our data each time we trained our networks.

| Network | Accuracy |
|---|---|
| Fractional Max-Pooling (all-time best)[16] | 96.53% |
| The Group that has to do Deeper (our work) | 76.6% |
| Fast-Learning Shallow CNN (all-time 49th best result)[14] | 75.86% |

Table 1: The best results on the CIFAR-10 dataset. All of the results can be seen at the webpage *Are we there yet?* [3].

# 5 Experiments

The goal of our experiment was to build an convolutional neural network for image recognition, which would be accurate enough to be called useful, yet simple enough to be trained on a simple laptop.

We went trough 12 different CNN-architectures and trough try and error we chose the final form, which can also be seen in figure 4 and was discussed in detail earlier.

Interestingly the amount of layers was reduced. We experimented with 3 and 4 convolutional layers, but the accuracy didn't improve much or the model started overfitting. The training was also resource costly. The final form of two convolutional layers was a good compromise on accuracy and training speed.

Dropout layers were added in the later part of development and they proved to a very useful. The initial percentage of dropped out neurons was further increased from the initial 20% to the final 30%. Adding the dropout layers decreased our training accuracy, but increased our testing accuracy.

Filter size was also varied and the 5 times 5 was the best one. We intially trained the neural network with a filter of 3 times 3 but it added overfitting in the training phase. In pooling we always went with 3 times 2.

Minibatch size was also tested with the values of 16, 32, 64 and 128. Final choice was 64. Learning rate was kept at a constant 0.01 and momentum was 0.9. For L2-regularization we used the value 0.02.

Because we had limited resources and training our network took a good 30 to 45 minutes each time, we didn't change and run all the variables for each network architecture but chose the most promising networks and then started playing with the variables.

# 6 Results

We got varying results from all our experiments. The accuracy was always above 60%, most varying between 69% and 77% for the testing set. For the training set we got well above 80%, reaching 85% on our highest try. However, we measured our success by testing set accuracy - not training.

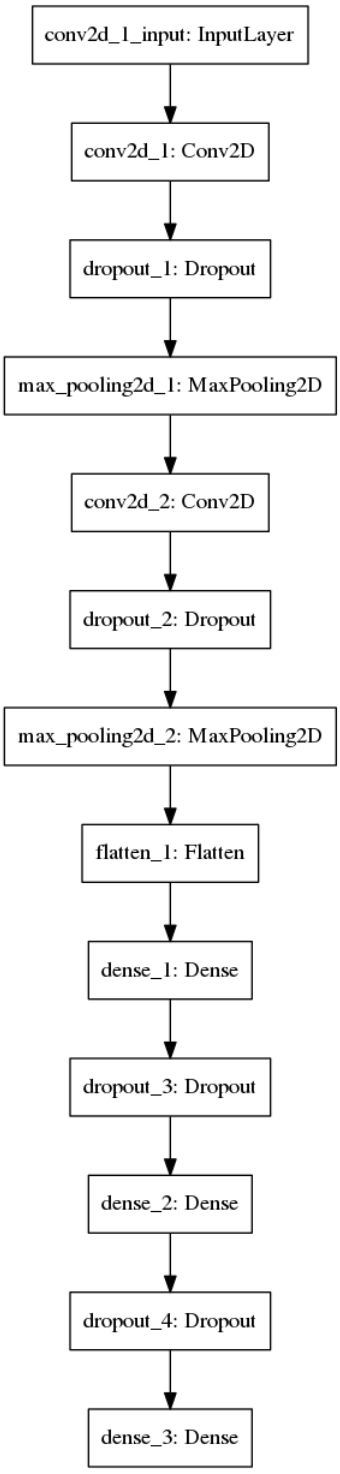


Figure 4: The structure of our final model.

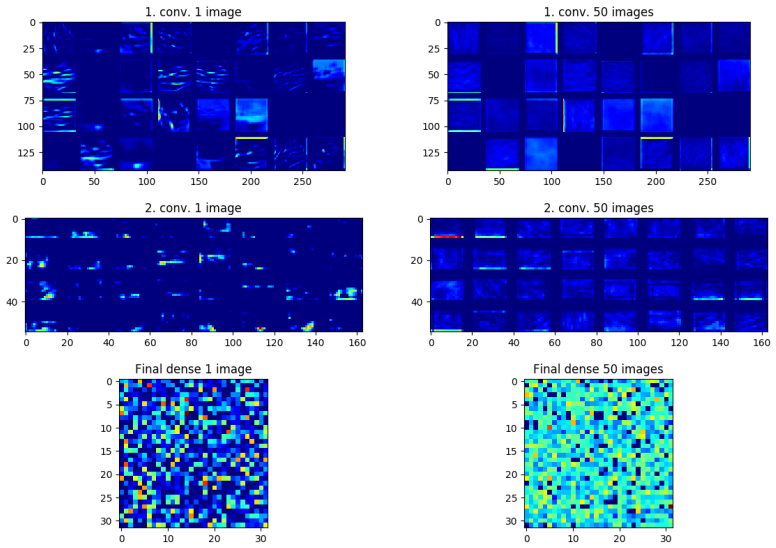


Figure 5: The activations in the layers.

The added dropout layers greatly helped us at this.

However, that 77% proved to be some kind of a hardlimit we coulnd't break - not atleast with our resources. The final architecture seen in the figure 4 gave 76.7% on the testing set.

The activations produced by one single image and a batch of 50 images in our convoluted and dense layers can be seen in the figure 5. Looking at these activations, our convoluted layers seem to be heavily responding to straight lines.

The webpage *Are we there yet?* gathers the best results on the CIFAR-10 dataset. Some of these results are also seen in the table 1. When compared to these results, we are way below the all time best, but we would've landed on the list at the spot 48th!

# 7 Discussion

All in all, our results were very good. As noted, our method is way below the all-time best and way less than human classifier could do. However, it still compares very favorably against other older methods. We could sent this paper to *Are we there yet?*[3] and make it to the list! Granted, most of those papers didn't use the more modern methods as we did - such as data augmentation or dropout.

We noticed the same behaviour in our network as we did on the course.

Very small details can build huge differences in the end - even such a small thing as the random initialization of the weights. The usual problems as with any statistical representations were here too: little noise in the input can totally destroy the results.

A new problem was to realize how computing intensive training the networks is. As noted, we didn't have any super computers or servers at our disposable. All experiments were run on a single laptop. And even more overhead was created by the Keras library which augments the data allway on-demand not into the buffer. Thus all the data needed to be pre-processed new every time.

# 8 Conclusions

We definitely need more GPU power. You can't train any serious neural networks on a single laptop (atleast not without a proper GPU). A possibility would've been to open up a distributed web server, like Amazon Web Services or Heroku. Thus we could've trained our networks on a fast cloud service and not waist our small laptop's resources. This is definitely something to keep up in mind for the next time.

However, the project was very satisfying to build and play with. Truly, the best way to learn how something works is to build it from scratch by yourself. What was surprising, was that the biggest difficulties didn't lie in the programing part itself, but in choosing your network architecture and the hyper-variables. And for this there might not be any shortcuts: as deep learning architectures don't have any rigorous closed form solutions, you just have to find the *do's and dont's* yourself. Developing this kind of intuition requires practice.

During this project, we also pondered on the philosophical aspects of this technology: is there even too much deep learning now? Based on the literature we read, you can't even publish a computer vision paper without deep learning nowadays [6]. Although deep learning and other machine learning technologies are fascinating and will be proven even more usable in the future, we should also look more into the question: what are they actually doing? Because they are not answering the underlying whys, they are just making statistical representations of the data itself.

# References


[1] Martín Abadi et al. "Tensorflow: Large-scale machine learning on heterogeneous distributed systems". In: *arXiv preprint arXiv:1603.04467* (2016).

[2] Mark F Bear, Barry W Connors, and Michael A Paradiso. *Neuroscience*. Vol. 2. Lippincott Williams & Wilkins, 2007.

[3] Rodrigo Benenson. *What is the class of this image?* `http://rodrigob.github.io/are_we_there_yet/build/classification_datasets_results.html`. 2018, accessed January 25th 2018.

[4] James Bergstra et al. "Theano: A CPU and GPU math compiler in Python". In: *Proc. 9th Python in Science Conf.* 2010, pp. 1–7.

[5] François Chollet et al. "Keras: Deep learning library for theano and tensorflow". In: *URL: https://keras. io/k* (2015).

[6] Adnan Darwiche. "Human-Level Intelligence or Animal-Like Abilities?" In: *arXiv preprint arXiv:1707.04327* (2017).

[7] Ludimar Hermann. "Eine erscheinung simultanen contrastes". In: *Pflügers Archiv European Journal of Physiology* 3.1 (1870), pp. 13–15.

[8] Alex Krizhevsky and Geoffrey Hinton. "Learning multiple layers of features from tiny images". In: (2009).

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012, pp. 1097–1105.

[10] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *Nature* 521.7553 (2015), pp. 436–444.

[11] Yann LeCun, Yoshua Bengio, et al. "Convolutional networks for images, speech, and time series". In: *The handbook of brain theory and neural networks* 3361.10 (1995), p. 1995.

[12] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[13] Yann LeCun et al. "Handwritten digit recognition with a back-propagation network". In: *Advances in neural information processing systems*. 1990, pp. 396–404.

[14] Mark D McDonnell and Tony Vladusich. "Enhanced image classification with a fast-learning shallow convolutional neural network". In: *Neural Networks (IJCNN), 2015 International Joint Conference on*. IEEE. 2015, pp. 1–7.

[15] Seymour A Papert. "The summer vision project". In: (1966).

[16] Nitish Srivastava et al. "Dropout: a simple way to prevent neural networks from overfitting." In: *Journal of machine learning research* 15.1 (2014), pp. 1929–1958.

## 9 Roles of the authors

Both authors contributed equal amount of hours to the project. However, given the nature of the project, the work was divided so that Lauri was more in charge of the programming part and Niklas on writing the whitepaper.

To reproduce the results, the code is also available from our GitHub-page `https://github.com/lnaalisv/deeplearning`.


THIS IS YOUR MACHINE LEARNING SYSTEM?
YUP! YOU POUR THE DATA INTO THIS BIG PILE OF LINEAR ALGEBRA, THEN COLLECT THE ANSWERS ON THE OTHER SIDE.
WHAT IF THE ANSWERS ARE WRONG?
JUST STIR THE PILE UNTIL THEY START LOOKING RIGHT.
DATA
ANSWERS