UNIVERSIDADE DA BEIRA INTERIOR
Engenharia

# Deep Learning Model Combination and Regularization using Convolutional Neural Networks

**Xavier Marques Frazão**

Dissertação para obtenção do Grau de Mestre em
**Engenharia Informática**
(2º Ciclo de estudos)

Orientador: Prof. Doutor Luís A. Alexandre

**Covilhã, Junho 2014**

# Acknowledgment

During this year, several persons have contributed in different ways to the development of this dissertation. Fistly, i would like to thank to Professor Luís A. Alexandre, for all the support, expertise, for the constant words of encouragement and for supervising my work.
I am also grateful to all members of SOCIA-LAB, Soft Computing and Image Analysis Lab, particulary Silvio Filipe for all the support and help.
A thanks to my friends for all the friendship and housemates for the good times we had in the last years.
And of course, an enormous thanks to all my family.

# Abstract

Convolutional neural networks (CNNs) were inspired by biology. They are hierarchical neural networks whose convolutional layers alternate with subsampling layers, reminiscent of simple and complex cells in the primary visual cortex [Fuk86a]. In the last years, CNNs have emerged as a powerful machine learning model and achieved the best results in many object recognition benchmarks [ZF13, HSK$^+$12, LCY14, CMMS12].
In this dissertation, we introduce two new proposals for convolutional neural networks. The first, is a method to combine the output probabilities of CNNs which we call Weighted Convolutional Neural Network Ensemble. Each network has an associated weight that makes networks with better performance have a greater influence at the time to classify a pattern when compared to networks that performed worse. This new approach produces better results than the common method that combines the networks doing just the average of the output probabilities to make the predictions. The second, which we call DropAll, is a generalization of two well-known methods for regularization of fully-connected layers within convolutional neural networks, DropOut [HSK$^+$12] and DropConnect [WZZ$^+$13]. Applying these methods amounts to sub-sampling a neural network by dropping units. When training with DropOut, a randomly selected subset of the output layer's activations are dropped, when training with DropConnect we drop a randomly subsets of weights. With DropAll we can perform both methods simultaneously. We show the validity of our proposals by improving the classification error on a common image classification benchmark.

# Keywords

Convolutional Neural Networks, Object Recognition, Network Ensemble, Regularization.

# Contents

# List of Figures

# List of Tables

# Lista de Acrónimos

| | |
|---|---|
| CNN | Convolutional Neural Networks |
| WCNNE | Weighted Convolutional Neural Network Ensemble |
| GPU | Graphics Processing Units |
| MLP | Multi-Layer Perceptrons |
| NN | Neural Network |
| RBF | Radial Base Function |
| MCCE | Multiclass Cross-entropy Loss Error |
| SVHN | The Street View House Numbers |
| MCDNN | Multi-Colum Deep Neural Network |
| GTSRB | The German Traffic Sign Recognition Benchmark |
| MP | Maxpooling layer |
| MPCNN | Maxpooling Layers Convolutional Neural Network |
| MPF | MaxPoolingFragment |
| NIN | Network In Network |

# Chapter 1

## Introduction

Object recognition is a fundamental and challenging problem and a major focus of research for computer vision, machine learning and robotics. The core problem is that each object in the world can cast an infinite number of different 2-D images onto the retina as the object's position, pose, lighting, and background vary relative to the viewer [DC07]. Yet the brain solves this problem effortlessly. Progress in understanding the brain's solution to object recognition requires the construction of artificial recognition systems that ultimately aim to emulate our own visual abilities, often with biological inspiration [DC07, SWB$^+$07].

Such computational approaches are critically important because they can provide experimentally testable hypotheses, and because instantiation of a working recognition system represents a particularly effective measure of success in understanding object recognition. However, a major challenge is assessing the recognition performance of such models. Ideally, artificial systems should be able to do what our own visual systems can.

Convolutional neural networks (CNNs) were inspired by biology and try to solve the problems described. They are hierarchical neural networks whose convolutional layers alternate with subsampling layers, reminiscent of simple and complex cells in the primary visual cortex [Fuk86a]. In the last years, CNNs have emerged as a powerful machine learning model and achieved the best results in many object recognition benchmarks [ZF13, HSK$^+$12, LCY14, CMMS12].

Although these networks are efficient when performing classification, they have the disadvantage of being computationally heavy, which makes their training slow and cumbersome.

With the emergence of parallel programming and taking advantage of the processing power of Graphics Processing Units (GPUs), training these networks takes significantly less time, making it possible to train large networks [JKRL09, CPS06] and also making it possible to train multiple networks for the same problem and combine their results [WZZ$^+$13, HSK$^+$12], an approach that can significantly increase the classification accuracy.

Besides the training time, the major problem of these networks is the overfitting. Overfitting still remains a challenge to overcome when it comes to training extremely large neural networks or working in domains which offer very small amounts of data. Many regularization methods have been proposed to prevent this problem. These methods combined with large datasets have made it possible to apply large neural networks for solving machine learning problems in several domains. Two new approachs have been recently proposed: DropOut [HSK$^+$12] and DropConnect [WZZ$^+$13], which is a generalization of the previous. When training with DropOut, a randomly selected subset of activations is droped. With DropConnect, we randomly drop the weights. Both techniques are only possible for fully connected layers. In this dissertation we propose a generalization of both methods named DropAll. With this approach we were able to train a network with DropOut, DropConnect or both and taking advantage of each method. Model combination improves the performance of machine learning models, i.e combining the results from different networks trained by these techniques significantly improves the classification rate.

Along with the DropAll proposal, we also introduce a new method to combine the results of networks by applying a different weight for each network, instead of using the common method, that involves averaging the output probabilities of several networks.

## 1.1 Motivation and Objectives

Our motivation and objective was the development of one method (at least), that would allow us to contribute to the research area of deep learning, in particular in the sub-area of convolutional neural networks. At the end we contributed with two proposals, introduced in the previous section and for each proposal we submitted a paper.
The first contribution presents a method to combine the output probabilities of CNNs which we call "Weighted Convolutional Neural Network Ensemble". This work was submitted to the 19$^{th}$ Iberoamerican Congress on Pattern Recognition (CIARP) 2014.
The second contribution we propose "DropAll: Generalization of Two Convolutional Neural Network Regularization Methods". This work was submitted to the International Conference on Image Analysis and Recognition (ICIAR) 2014.

## 1.2 Dissertation Outline

The remaining of this dissertation is strucured as follows: chapter 2 gives a state-of-the-art on convolutional neural networks, presenting the architectures, methodologies and other strategies that contributed for the evolution of this area.
Chapters 3 and 4 describe in detail the proposed methods, the validation of the solution and the discussion of the results.
Chapter 5 summarizes all the work performed in this dissertation, presenting the main conclusions.

# Chapter 2

# State-of-the-Art

Multilayer feed-forward Neural Networks like Multi-Layer Perceptrons (MLP), have shown to be a very powerful machine learning technique. Depending on the problem, the choice of the feature extraction algorithm and the features to classify the problem, is often empirical and therefore sub-optimal. A possible solution would be to directly apply the MLP on the "raw" input data and let the training algorithm e.g. backpropagation, find the best feature extractors by adjusting the weights accordingly.

The problem with this approach is that when the input dimension is high, the number of connections, thus the number of free parameters is also high because each hidden unit would be fully connected to the input layer.

The number of training examples, however, might be relatively small compared to the pattern dimension, which means that the NN would have a too high complexity and, thus, would tend to overfit the data.

Another disadvantage comes from the fact that its input layer has a fixed size and the input patterns have to be presented well aligned and/or normalized to this input window, which, in practice, is a rather complicated task. Thus, there is no built-in invariance to small distortions like translations, rotations and scales.

Convolutional Neural Networks (CNN), try to alleviate the above mentioned problems. CNNs automatically learn local feature extractors, are invariant to small distortions in the input pattern and implement the principle of weight sharing, which drastically reduces the number of free parameters, increasing their generalization capacity.

## 2.1 Neocognitron

The first implementation of a convolution neural network, known as the Neocognitron, was proposed by Fukushima [Fuk80, Fuk86b] and originally applied to the problem of handwritten digit recognition. The CNNs were inspired by the visual cortex structure of mammals, proposed by Hubel and Wiesel [HW62]. According to their findings, the visual area of mammals is composed by a set of simple cells and complex cells. While simple cells are responsible for the extraction of features, complex cells make complex sampling of these features. In CNNs, this process is known as convolution and sub-sampling. The Neocognitron also follows this pattern and is therefore recognized as the first CNN model. The basic topology of a Neocognitron is illustrated in Figure 2.1. In this network, we used the concept of receptive fields, i.e. each neuron is connected to a sub-region that corresponds to a number of neighboring neurons in the preceding layer. This network has a $u_{c0}$ input layer and four S and C alternating layers $u_{S1}$, $u_{c1}$, ..., $u_{S4}$, $u_{C4}$ that correspond to convolution and sub-sampling. The weights are shared. $u_{C4}$ is the output layer consisting of single neurons representing the decision of the NN for some input pattern, in this case a digit. A more detailed description of the Neocognitron operation can be found in the referring literature [Fuk80, Fuk86b].

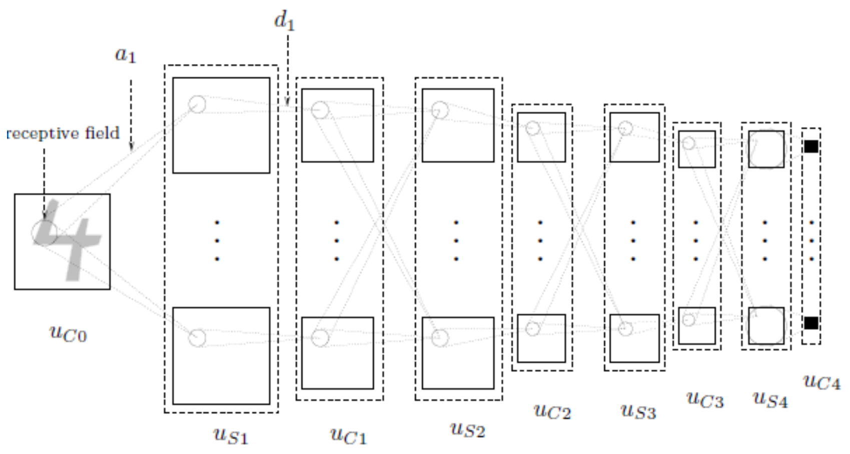


Figure 2.1: The topology of the basic Neocognitron [Fuk80].

## 2.2 Convolutional Neural Networks

A major CNN breakthrough happened with the widespread use of the backpropagation learning algorithm for multi-layer feed-forward NNs [RHW88]. It is a supervised learning algorithm that defines an error function $E$ and applies the gradient descent technique in the weights space in order to minimize $E$.

LeCun $et al.$ [CBD+90] presented the first CNN trained with the backpropagation algorithm and, much like the Neocognitron, it was applied to the problem of handwritten digit recognition. The model proposed by LeCun is simpler and more versatile than the one proposed by Fukushima, and it would eventually serve as the stepping stone for the majority of the variants of CNNs that exist today.

The CNN described by LeCun and the Neocognitron described by Fukushima resemble each other in many aspects. Both use an alternating sequence of convolution layers and sub-sampling. Both models implement the concepts of receptive fields and weight sharing. One of CNNs assets is that the network does not need to be trained layer by layer, since all weights are iteratively adjusted through backpropagation, minimizing the error function. Figure 2.2 shows the topology proposed by LeCun $et al.$ [CBD+90], in later work referred to as LeNet-1.

The input layer $y$ receives a greyscale image containing the digit to be recognized and has a 28 x 28 size. The intensities of the pixels are normalized to values between -1 and 1. The first H1 hidden layer consists of four feature maps $y_j^{(1)}$, each containing 25 weights $w_{j0}^{(1)}(u,v)$ , constituting a 5 X 5 trainable kernel and bias $b_j^{(1)}$. The values of the feature maps $y_j^{(1)}(x,y)$ are obtained by convolving the input map $y^{(0)}$ with the respective kernel $w_{j0}^{(1)}$ and applying a activation function $\phi^{(1)}$ to the result:

$$y_j^{(1)}(x,y) = \phi\left(\sum_{(u,v)\in K} w_{j0}^{(1)} y^{(1)}(x+u,y+v) + b_j^{(1)}\right)$$

where $K = \{(u,v) \in \mathbb{N}^2 | 0 \leq u < 5 \text{ and } 0 \leq v < 5\}$

Each convolution map is followed by a sub-sampling map $y_j^{(2)}$. An average is made in the sub-sampling and the maps size is reduced to half. Hence, the sub-sampling maps of layer H2 are of size 12 x 12. Further, each sub-sampling map has a weight $w_j^{(2)}$ and a bias $b_j^{(2)}$.

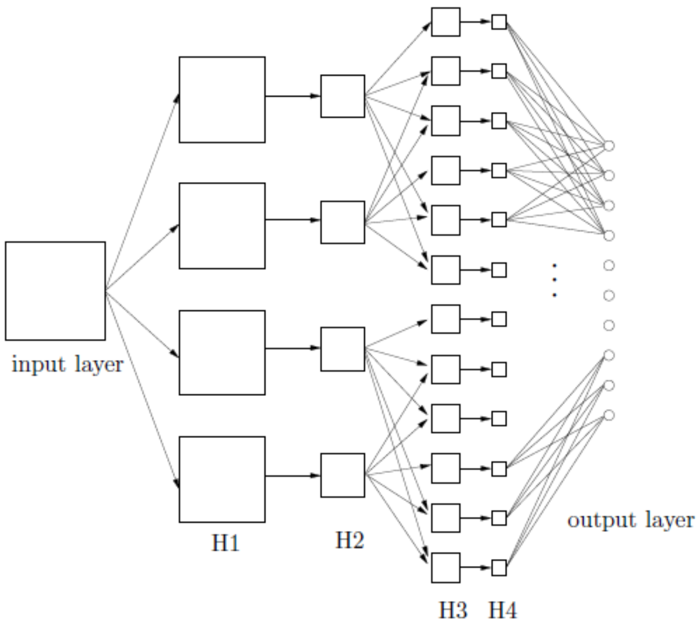


Figure 2.2: The architecture of the CNN LeNet-1 [CBD+90].

$$y_j^{(2)}(x,y) = \phi^{(2)} \left( w_j^{(2)} * \sum_{(u,v)\in(0,1)^2} y_j^{(1)}(2x+u, 2y+v) + b_j^{(2)} \right)$$

$\phi^{(2)}$ is again an activation function that has to be continuous and differentiable, as required by the backpropagation learning algorithm. Figure 2.3 illustrates the process of convolution and sub-sampling.

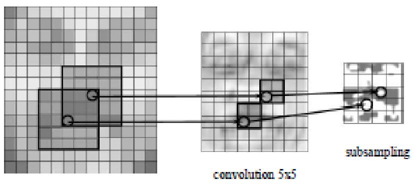


Figure 2.3: An input image followed by a feature map performing a 5 x 5 convolution and a 2 x 2 sub-sampling map

In layers H3 and H4 the process is similar to the process used in H1 and H2, with the difference that the convolution is made of 3 x 3 instead of 5 x 5 filters. Further, one convolution map $j$ in layer H3 can have several kernels $w_{(ji)}^{(3)}$ operating on different maps $i$ in the preceding layer H2. The activation of the respective convolution maps is the sum of the results of all the convolutions and biases. Thus, the general activation formula for the convolution map $j$ in layer $l$ is:

$$y_j^{(l)}(x,y) = \phi^{(1)} \left( \sum_{(i\in l)} \sum_{(u,v)\in K} w_{(ji)}^{(l)}(u,v) y_i^{(l-1)}(x+u, y+v) + b_j^{(l)} \right)$$

where $K = \{(u,v) \in \mathbb{N}^2 | 0 \le u < s_x \text{ e } 0 \le v < s_y\}$, $(s_x, s_y)$ is the dimension of the convolution kernel, and $I$ is the set of maps of the preceding layer to which the convolution map $j$ is connected.

Finally, the output layer contains a set of 10 neurons, fully connected to the previous sub-sampling maps of layer H4, and representing the 10 digits to recognize. The winning neuron responds with the value 1 and the other ones with -1. In total, the network has 4635 units and 98442 connections, but only 2578 independent parameters to learn thanks to the weight sharing.

## 2.3 Training Convolutional Neural Networks

The training of CNNs is very similar to the training of other types of NNs, such as ordinary MLPs. A set of training examples is required, and it is preferable to have a separate validation set in order to perform early stopping to avoid overtraining.
There are some measures that we can take into account to improve the networks generalization capacity. This improvement can be done for instance, by applying small geometric transformations, such as translations, rotations and scale changes to the training set images. The most used training algorithm is the online backpropagation with a real time weight update [LBBH98]. The algorithm is similar to the algorithm used to train the MLPs [RHW88], the only difference to take into account is the weight sharing in the convolution and sub-sampling layers. We also want to minimize the error function $E_p$ after each training example $p$:

$$E_p = \frac{1}{2}\left\|o_p - t\right\|^2 = \frac{1}{2}\sum_{k=1}^{k}(o_{pk} - t_{pk})^2$$

Where $K$ is the number of output units, $o_{pk}$ is the output of the neuron $k$ for the pattern $p$ and $t_{pk}$ is the respective target value $t_{pk} \in [-1, +1]$. For each iteration, the weight update is a small variation to the opposite direction of gradient $\triangledown E$:

$$w_{ji}^{(l)} \leftarrow w_{ji}^{(l)} + \triangle w_{ji}^{(l)} = w_{ji}^{(l)} - \lambda \frac{\partial E_p}{\partial w_{ji}^{(l)}}$$

where $\lambda$ is the learning rate and $w_{ji}^{(l)}$ is the weight from neuron $i$ to neuron $j$ in the $l$ layer.
To the output layer:

$$\triangle W_{kj}^{(l)} = -\lambda \delta_k^{(l)} y_j^{(l-1)}$$

where $y_j^{(l-1)}$ is the activation of neuron $j$ in the $l-1$ layer,
$\delta_k^{(l)} = e_k \phi^{'}(V_k^{(l)})$ is the local gradient,
$e_k = o_k - t_k$ is the error,
$V_k^{(l)} = \sum_j w_{kj}^{(l)} y_j^{(l-1)}$ is the weighted sum of all $y_j^{(l-1)}$ inputs of the neuron $k$.
The activation function $\phi$ of the output neurons is usually the hyperbolic tangent function:

$$\phi(x) = \frac{1 - e^{-x}}{1 + e^{-x}}$$

If there is an additional hidden layer containing neurons between the last sub-sampling and output layer:

$$\delta_j^{(l)} = e_j^{(l)} \phi^{'}(V_j^{(l)}) = \left(\sum_{K=1}^{K} \delta_k^{(l+1)} w_{kj}^{(l+1)}\right) \phi^{'}(V_j^{(l)})$$

to the local gradient of the hidden layer neuron $j$. $e_j^{(l)}$ is the error in neuron $j$ backpropagated from the neurons of the following layer $l+1$.

In the case of a convolution layer, the update of a weight $W_{ij}$ from the features map at position $i$ through $j$ of the kernel $(u, v)$ is performed by the sum of all positions $(x, y)$ of the features map.

$$\triangle W_{ji}^{(l)}(u,v) = -\lambda \sum_{(x,y)} \left(\delta_j^{(l)}(x,y) y_i^{(l-1)}(x+u, y+v)\right)$$

Figure 2.4 illustrates the error backpropagation with convolution maps. The bias $b_j^{(l)}$ of the convolution map $j$ on the layer $l$ is updated as follows:

$$\triangle b_j^{(l)} = -\lambda \sum_{(x,y)} \delta(x,y)$$

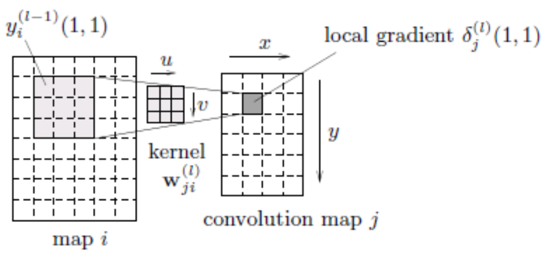


Figure 2.4: Error backpropagation with convolution maps.

For sub-sampling maps the calculation is slightly different. There is only one weight $w_{ij}^{(l)}$ for each connection from the map $i$ to the sub-sampling map $j$. There is only a one to one connection from the convolution to the sub-sampling maps, and so $i = j$. In the general case of a weight $w_{ij}^{(l)}$ of the map $i$ on layer $l-1$ for a sub-sampling map $j$ in layer $l$ with $s_x$ and $s_y$ sub-sampling factors, we have:

$$\triangle w_{ji}^{(l)} = -\lambda \sum_{(x,y)} \delta_j^{(l)}(x,y) \sum_{m=1}^{s_x} \sum_{n=1}^{s_y} Y_i^{(l-1)}(xs_x + m, ys_y + n)$$

The bias updating process is equal to the one made for the convolution maps. Figure 2.5 shows the error backpropagation with sub-sampling maps.

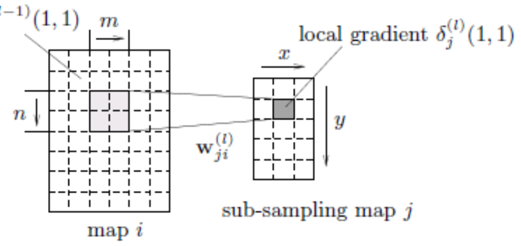


Figure 2.5: Error backpropagation with sub-sampling maps.

Calculus of the local gradient $\delta_j^{(l)}$ of a map $j$ on the layer $l$ depends on the following layer. If the following layer $l+1$ is a neuron layer then:

$$\delta_j^{(l)}(x,y)) = \sum_{k=1}^{K} \sum_{x,y} \delta_k^{(l+1)} w_{kj}^{(l+1)}(x,y)$$

$K$ is the number of neurons on the layer $l+1$ . If the following layer is a convolutional layer, then:

$$\delta_j^{(l)}(x,y)) = \sum_{k \in K_c} \sum_{(u,v)} \delta_k^{(l+1)}(x,y) * w_{kj}^{(l+1)}(u,v)$$

where $K_c$ indicates the set of maps in the layer $l+1$ connected to the convolution map $j$ in the layer $l$. It is necessary to note that the convolution masks cannot cross the border of layer $l$ . It also happens that the following layer can be a sub-sampling layer, and in that case the local gradient is given by:

$$\delta_j^{(l)}(x,y)) = \sum_{k \in K_c} \delta_k^{(l+1)}(\lfloor x/s_x \rfloor, \lfloor y/s_y \rfloor) * w_{kj}^{(l+1)}$$

where $s_x$ and $s_y$ are the sub-sampling factors, $K_s$ indicates the set of maps in the $l+1$ layer that are connected to the sub-sampling map $j$ in layer $l$ and $\lfloor . \rfloor$ is the floor function.

## 2.4 CNN Variants

There are many CNN variants that differ from the traditional model presented by LeCun known as LeNet1. There is no algorithm to automatically determine the optimal architecture of a CNN for a given classification task. The number of layers, the number of feature maps and their dimensions, how they are connected, the sub-sampling method, the regularization methods are all dependent of the given problem. Some variants will be presented, then.

### 2.4.1 LeNet-5

The LeNet-5 CNN is a bit more complex than the LeNet-1 and it was also presented by LeCun et al [LBBH98] and applied to handwritten character recognition. Figure 2.6 shows the topology of LeNet-5 network.

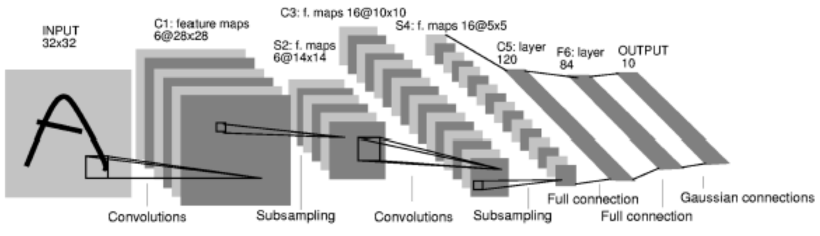


Figure 2.6: The architecture of LeNet-5 [LBBH98].

It is composed of seven layers, not counting the input layer, and the input image is of size 32 x 32 pixels. As in LeNet-1, the first five layers alternate between convolution with filters of size 5 x 5 and a sub-sampling factor of 2 (reduces dimensionality in half). The connections between the units of the respective layers are similar to LeNet-1, i.e. a full connection of the first layer to the input image and one-to-one connections in the sub-sampling layers.
However, layer 3 follows a non-symmetrical connection that is described in Figure 2.7.
The convolution maps in layer 5 have a dimension of only (1,1). Layer 6 is a hidden layer that is fully connected to the previous layer and composed of 84 units. The output layer has 10

|   | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | X |   |   |   | X | X | X |   |   | X | X | X | X |   | X | X |
| 1 | X | X |   |   |   | X | X | X |   |   | X | X | X | X |   | X |
| 2 | X | X | X |   |   |   | X | X | X |   |   | X |   | X | X | X |
| 3 |   | X | X | X |   |   | X | X | X | X |   |   | X |   | X | X |
| 4 |   |   | X | X | X |   |   | X | X | X | X |   | X | X |   | X |
| 5 |   |   |   | X | X | X |   |   | X | X | X | X |   | X | X | X |

Figure 2.7: The connection scheme of layer C3 of Lenet-5: a cross indicates a connection between a particular feature map in S2 (row) and a map in C3 (column) [LBBH98].

neurons that are fully connected, one for each character to be recognized, adding the base radial base function (RBF) [Wik13a]. To calculate the output of a RBF unit $y_i$ we have:

$$y_i^{(6)} = \sum_j (x_j - w_{ij}^{(6)})^2$$

Each RBF unit calculates the squared Euclidean distance between the input vector $x_j$ and its weight vector $w_{ji}^{(6)}$. For the error function $E$, the following was used:

$$E(W) = \frac{1}{P} \sum_{p=1}^{P} \left( y_{pDp}^{(6)} + log(e^{-j} + \sum_i e^{-y_{(pi)}^{(6)}}) \right)$$

where $P$ is the number of training patterns, $y_{(pi)}^{(6)}$ is the activation of the output neuron $i$ when presenting a $p$ pattern and $D_p$ is the desires class of the $p$ pattern.

## 2.4.2 Pooling Methods

The sub-sampling layer, also known as pooling layer, aims to achieve spatial invariance to reduce the dimensionality of the feature maps, preserving important information and discarding that which is irrelevant. Several studies show that the pooling operation can greatly influence the performance of the network. The remaining question is what is the best method to preserve the most important information. The pooling operation is typically a sum, average, maximum or even a combination of various methods. The two most used types of pooling are the average and max. An example of a convolution neural network that uses average pooling is LeCun $et\ al$ network[LBBH98]. Jarret $et\ al$ [JKL] uses max pooling networks. In general terms the pooling layer gets hold of a feature representation and transforms it into a new one, with a smaller dimension, but that preserves important information and discards the irrelevant one. A study [SMB10, BPL10] that evaluates the best performance method, if the average or the maxpooling one, concluded that depending on the data and characteristics, both max and average can be a solution. However, in most cases maxpooling presents the best results [SMB10, BPL10].

## 2.4.3 Stochastic Pooling

A recent study introduced a new type of pooling called stochastic pooling [ZF13]. Due to their large capacity, neural networks are prone to overfitting. There are several methods to solve this problem, like the inclusion of the validation set or the weakening of the weights (weight decay) or the regularization method described above. Otherwise it would be very difficult to train large networks. With stochastic pooling the author proposes a new type of regularization for convolutional layers that allows the training of large networks without the danger of overfit. This method also achieves high performance in several classification tasks. Networks with this

type of pooling use the rectified linear function as activation function, instead of the usual sigmoid or hyperbolic tangent.
In average pooling, all the elements of the region are considered, even those that have low magnitude. When combined with the activation function, they have a low-weight effect. Positive activations and negative activations may cancel each other, leading to weak answers. If the activation function was a hyperbolic tangent this problem would be greater.
Maxpooling is not penalized for these drawbacks, since it always chooses the highest element of the region, however it easily overfits the training set, making it hard to generalize well to test examples.
The proposed pooling scheme has the advantages of maxpooling, but the stochastic process will help prevent overfitting the training.
In stochastic pooling we choose an answer from the map, from a multinomial distribution formed by the activations of each region. We calculate the probability $p$ for each region $j$ by normalizing the activations within the region:

$$p_i = \frac{a_i}{\sum_{k \in R_j} a_k}$$

The output is given by:
$s_j = a_l$ where $l \sim P(p_1, .., p_{|R_j|})$ Figure 2.8 illustrates the process. In the case of maxpooling, it would only capture the strongest activation. However, there may be activations in the same region that should be taken into account. The stochastic procedure ensures that non-maximized activations also have the opportunity to be used.

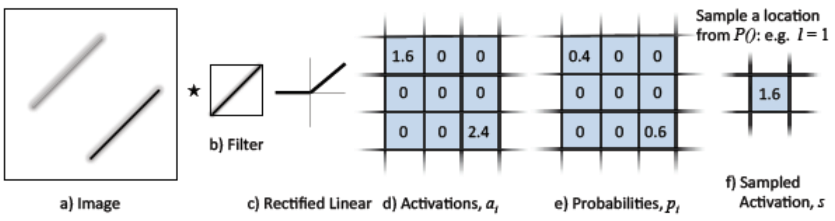


Figure 2.8: Toy example illustrating stochastic pooling. a) Input image. b) Convolutional filter. c) Rectified linear function. d) Resulting activations within a given pooling region. e) Probabilities based on the activations. f) Sampled activation. Note that the selected element for the pooling region may not be the largest element. Stochastic pooling can thus represent multi-modal distributions of activations within a region [ZF13].

The use of stochastic pooling for tests introduces noise and it can degrade performance. As a solution, an average made in a probabilistic way is made, named "Probabilistic Weighting":

$$s_j = \sum_{i \in R_j} p_i a_i$$

This method is different from average pooling, as each element has the potential of having different weights. A comparison will be made in the following section between the various methods.

#### 2.4.3.1 Results

Different datasets were used to test the proposed method. The experiments were performed using a library for C++ with GPU library [Kri]. The network has 3 convolutionary layers with 5 x

5 filters and 64 feature maps per layer with rectified linear units as their outputs. The model was trained for 280 epochs in all experiments. The architecture used for the dataset Street View House Numbers, has 128 feature maps in layer 3 and is trained for 500 epochs.

#### 2.4.3.2 CIFAR-10

The CIFAR-10 dataset [Kri09] consists of 32 x 32 color images drawn from 10 classes split into 50 000 train and 10 000 test images. Using the same network architecture described above, they trained three models using average, max and stochastic pooling respectively and compare their performance. Stochastic pooling avoids overfitting, unlike average and max pooling, and produces less test errors as can be seen in Figure 2.9. Table 2.1 compares the test performance of the three pooling approaches.

| | Train Error % | Test Error % |
|---|---|---|
| Avg Pooling | 1.92 | 19.24 |
| Max Pooling | 0.0 | 19.40 |
| Stochastic Pooling | 3.40 | **15.13** |

Table 2.1: CIFAR-10 Classification performance for various pooling methods.

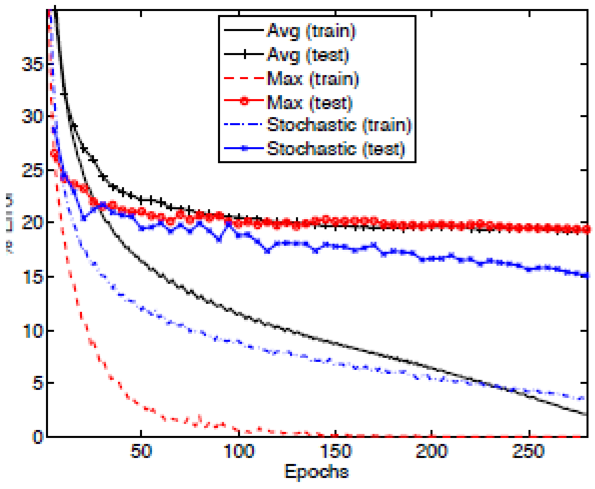


Figure 2.9: CIFAR-10 train and test error rates throughout training for average, max, and stochastic pooling [ZF13].

#### 2.4.3.3 MNIST

The MNIST dataset [LBBH98] consists of 28 x 28 pixel greyscale images of handwritten digits 0-9, with 60,000 training and 10,000 test examples. During training, the error using both stochastic pooling and max pooling dropped quickly, but the latter completely overfit the training data. Weight decay prevented average pooling from over-fitting, but had an inferior performance to the other two methods. Table 2.2 compares the three pooling approaches.

#### 2.4.3.4 CIFAR-100

The CIFAR-100 dataset is another subset of the tiny images dataset, but with 100 classes [Kri09]. There are 50,000 training examples in total (500 per class) and 10,000 test examples. Due to the limited number of training examples per class, typical pooling methods used in convolutional

| | Train Error % | Test Error % |
|---|---|---|
| Avg Pooling | 0.57 | 0.83 |
| Max Pooling | 0.04 | 055 |
| Stochastic Pooling | 0.33 | **0.47** |

Table 2.2: MNIST train and test error rates throughout training for average, max, and stochastic pooling..

networks do not perform well. However, stochastic pooling obtained the lowest results again. Table 2.3 compares the three pooling approaches.

| | Train Error % | Test Error % |
|---|---|---|
| Avg Pooling | 11.20 | 47.77 |
| Max Pooling | 0.17 | 50.90 |
| Stochastic Pooling | 21.22 | **42.51** |

Table 2.3: CIFAR-100 train and test error rates throughout training for average, max, and stochastic pooling.

### 2.4.3.5 Street View House Numbers

The Street View House Numbers (SVHN) dataset is composed of 604,388 images (using both the difficult training set and simpler extra set) and 26,032 test images [NWC$^+$]. Despite having significant amounts of training data, a large convolutional network can still overfit. Two architectures are used: the same architecture used in other datasets and the second with 128 feature maps in the third layer Table 2.4 compares the three pooling approaches.

| | Train Error % | Test Error % |
|---|---|---|
| Conv. net + 2 layer Classefier [SCL12] | - | 5.03 |
| Conv. net + 2 layer Classefier + padding [SCL12] | - | 4.90 |
| 64-64-64 Avg Pooling | 1.83 | 3.98 |
| 64-64-64 Max Pooling | 0.38 | 3.65 |
| 64-64-64 Stochastic Pooling | 1.72 | 3.13 |
| 64-64-128 Avg Pooling | 1.65 | 3.72 |
| 64-64-64-128 Max Pooling | 0.13 | 3.81 |
| 64-64-128 Stochastic Pooling | 1.41 | **2.80** |

Table 2.4: SVHN Classification performance for various pooling methods with 64 or 128 layer 3 feature maps.

### 2.4.3.6 The importance of Stochastic Model and the Probabilistic Weighting

To analyze the importance of stochastic model and the probabilistic weighting in test, a comparison was made between the different methods for training and testing with the CIFAR-10 dataset. Choosing the locations stochastically at test time degrades performance slightly as could be expected, however it still outperforms models where max or average pooling are used at test time. When using probability weighting during training, the network easily overfits and performs sub-optimally at test time using any of the pooling methods. Table 2.5 compares the results for various train and test combinations of pooling methods.

| Train Method | Test Method | Train Errpr % | Test Error % |
|---|---|---|---|
| Stochastic Pooling | Probabilistic Weighting | 3.20 | **15.20** |
| Stochastic Pooling | Stochastic Pooling | 3.20 | 17.49 |
| Stochastic Pooling | Max Pooling | 3.20 | 17.66 |
| Stochastic Pooling | Avg Pooling | 3.20 | 53.50 |
| Probabilistic Weighting | Probabilistic Weighting | 0.0 | 19.40 |
| Probabilistic Weighting | Stochastic Pooling | 0.0 | 24.00 |
| Probabilistic Weighting | Max Pooling | 0.0 | 22.45 |
| Probabilistic Weighting | Avg Pooling | 0.0 | 58.97 |
| Max Pooling | Max Pooling | 0.0 | 19.40 |
| Max Pooling | Stochastic Pooling | 0.0 | 32.75 |
| Max Pooling | Probabilistic Weighting | 0.0 | 30.00 |
| Avg Pooling | Avg Pooling | 1.92 | 19.24 |
| Avg Pooling | Stochastic Pooling | 1.92 | 44.25 |
| Avg Pooling | Probabilistic Weighting | 1.92 | 40.09 |

Table 2.5: CIFAR-10 Classification performance for various train and test combinations of pooling methods.

### 2.4.4 Multi-column Deep Neural Networks

This variant combines several CNNs, a process that the author named Multi-Colum DNN (MCDNN) [CMMS12, Sch12]. This model achieved great results on all datasets in which it was tested, even winning the final stage of a well-known event, "German traffic sign recognition benchmark" [GTR13].

The CNN described here consists of a series of convolutions and sub-sampling where maxpooling is made, where each layer only receives connections from the previous layer. The convolutional layer performs the convolution of $M^{n-1}$ input maps with a filter size $K_x^n * K_y^n$. The resulting activations of the output maps $M^n$ are given by the sum of the $M^{n-1}$, convolutional responses which are passed through a nonlinear activation function:

$$Y_j^n = f(\sum_{i=1}^{M^{n-1}} Y_i^{n-1} * W_{ij}^n + b_j^n)$$

where $n$ indicates the layer, $Y$ is a map of size $M_x$ x $M_y$ and $W_{ij}$ is a filter of size $K_x$ x $K_y$, the output map $Y^n$ is of size $M_x^n = M_x^{n-1} - K_x^n + 1, M_y^n = M_y^{n-1} - K_y^n + 1.$

The pooling/sampling layer runs maxpooling. The softmax function is used as activation function [Wik13b] for the output layer and the hyperbolic tangent for the remaining layers. The network training is shown in Figure 2.10b. A dataset data is processed (P) before training begins, and is continuously distorted (D) during training. Distortions, i.e. translations, rotations and scaling, are made in a random and limited way, being applied to each pre-processed image during training. These values are obtained from a uniform distribution and with a specific variation. The values are $\pm 10\%$ for image translation, 0.9-1.1 for scaling and $\pm 5\%$ for rotation. These distortions allow us to train the network with more parameters without overloading the training and significantly improve the generalization ability of the network.

#### 2.4.4.1 MCDNN Construction

The construction of MCDNN is made by averaging the output activations of several DNN columns, as shown in Figure 2.10c. I.e., for a given sample entry, an average of networks predictions is made:

$$y^i MCDNN = \frac{1}{N} \sum_{j}^{\#columns} y^i DNN_j$$

where $i$ corresponds to the $i^{th}$ class and $j$ runs through all DNNs.
Before training, the weights of all the columns are randomly initialized. A problem is that the error obtained by the different networks tends to be highly correlated, because the training is applied to data which is quite similar to one another. To overcome this warning, the data is normalized in different ways for each network.

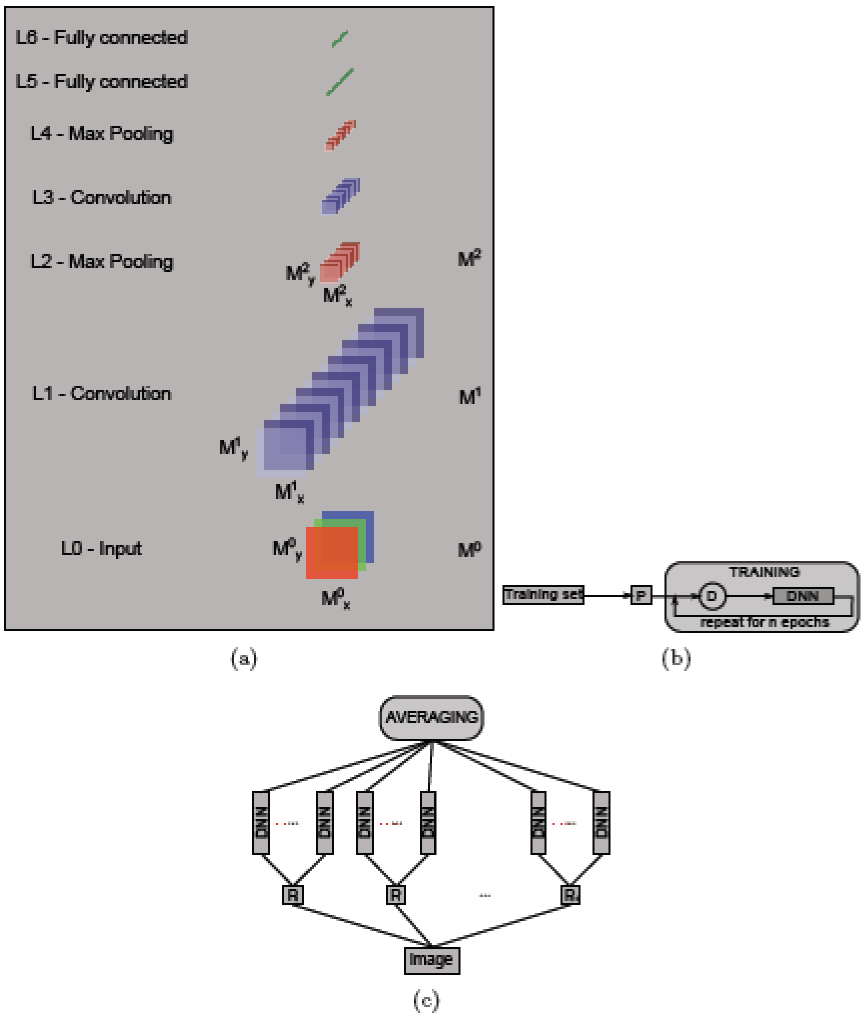


Figure 2.10: (a) DNN architecture. (b) Training a DNN: The dataset is preprocessed (P) before training starts; during training all original or preprocessed images are randomly distorted before each epoch (D). (c) MCDNN architecture: The final predictions are obtained by averaging individual predictions of each DNN [CMMS12].

#### 2.4.4.2 Results

The description of the CNN architecture used for the various experiments is given in the following way: example, 2x48x48-100C5-MP2-100C5-MP2-100C4-MP2-300N-100N-6N. Represents a network with 2 input images of size 48x48, a convolutional layer with 100 maps and 5x5 filters, a max-pooling layer over non overlapping regions of size 2x2, a convolutional layer with 100 maps and 4x4 filters, a max-pooling layer over non overlapping regions of size 2x2, a fully

| trial | W10 | W12 | W14 | W16 | W18 | W20 | Original |
|---|---|---|---|---|---|---|---|
| 1 | 0.49 | 0.39 | 0.40 | 0.40 | 0.39 | 0.36 | 0.52 |
| 2 | 0.48 | 0.45 | 0.45 | 0.45 | 0.50 | 0.41 | 0.44 |
| 3 | 0.59 | 0.51 | 0.41 | 0.41 | 0.38 | 0.43 | 0.40 |
| 4 | 0.55 | 0.44 | 0.42 | 0.42 | 0.39 | 0.50 | 0.53 |
| 5 | 0.51 | 0.39 | 0.48 | 0.48 | 0.36 | 0.29 | 0.46 |
| Avg | 0.52$\pm 0.05$ | 0.44$\pm 0.05$ | 0.43$\pm 0.05$ | 0.40$\pm 0.02$ | 0.40$\pm 0.06$ | 0.39$\pm 0.08$ | 0.47$\pm 0.05$ |
| | | 35-net | average | error: | 0.44$\pm 0.06$ | | |
| 5 coll. m.scale | 0.37 | 0.26 | 0.32 | 0.33 | 0.31 | 0.26 | 0.46 |
| | | | 35 net | MCDNN: | 0.23% | | |

Table 2.6: Test error rate of the 35 NNs trained on MNIST. Wxx - width of the character is normalized to xx pixels

| Method | Paper | Error rate % |
|---|---|---|
| CNN [SSP03] | 7 | 0.40 |
| CNN [PCL06] | 21 | 0.39 |
| MLP [CMGS10] | 35 | 0.35 |
| CNN [CMGS11]committe | 35 | 0.27 |
| MCDNN | 21 | 0.23 |

Table 2.7: Result with MNIST dataset

connected layer with 300 hidden units, a fully connected layer with 100 hidden units and a fully connected output layer with 6 neurons (one per class). A scaled hyperbolic tangent activation function is used for convolutional and fully connected layers, a linear activation function for max- pooling layers and a softmax activation function for the output layer.

During training, images are continually translated, scaled and rotated, whereas only the original images are used for validation.

#### 2.4.4.3 MNIST

The original MNIST digits [LBBH98] are normalized such that the width or height of the bounding box equals 20 pixels. Aspect ratios for various digits vary strongly and we therefore create six additional datasets by normalizing digit width to 10, 12, 14, 16, 18, 20 pixels. They train five CNN columns per normalization, resulting in a total of 35 columns for the entire MCDNN. All 1x29x29-20C4-MP2-40C5-MP3- 150N-10N CNN are trained for around 800 epochs with an annealed learning rate (i.e. initialized with 0.001 multiplied by a factor of 0.993/epoch until it reaches 0.00003).

Results of all individual nets and various MCDNN are summarized in Table 2.6. Table 2.7 compares the results with different numbers of datasets. Table 2.8 shows that most pre-processing results in a lower error on the part of MCDNN.

#### 2.4.4.4 NIST SD 19

The 35-columns MCDNN architecture and preprocess- ing used for MNIST are also applied to Latin characters from NIST SD 19 [GBC$^+$94]. Table 2.9 compare the error obtained in the various subsets of the dataset.

| y | # MCDNN | Average Error % |
|---|---|---|
| 1 | 7 | 0.33$\pm$0.07 |
| 2 | 21 | 0.27$\pm$0.02 |
| 3 | 35 | 0.27$\pm$0.02 |
| 4 | 35 | 0.26$\pm$0.02 |
| 5 | 21 | 0.25$\pm$0.02 |
| 6 | 7 | 0.24$\pm$0.01 |
| 7 | 1 | **0.23** |

Table 2.8: Average test error rate of MCDNN trained on $y$ pre- processed datasets.

| Data task | MCDNN error % | Error rate and paper % | |
|---|---|---|---|
| all(62) | 11.63 | - | - |
| digits(10) | 0.77 | 3.71 [KE95] | 1.88 [ZE02] |
| letters | 21.01 | 30.90 [KK05] | - |
| letters* | 7.37 | 13.00 [CdSBB$^+$06] | 13.66 [KK05] |
| Merged | 7.99 | 0.23 | - |
| uppercase | 1.83 | 10.00 [CdSBB$^+$06] | 6.44 [dSOSM08] |
| lowercase | 7.47 | 10.00 [CdSBB$^+$06] | 13.27 [KK05] |

Table 2.9: Average error rates of MCDNN for all experiments, plus results from the literature. * case insensitive.

#### 2.4.4.5 TRAFIC SIGNS

The GTSRB dataset consists of 26640 train images and 12569 test images [SSSI11]. The original color images contain one traffic sign each, with a border of 10 around the sign. The images vary in size from 15 x 15 to 250 x 250 pixels and are not necessarily square. They crop all images and process only within thebounding box and resized to 48 x 48 pixels. Some normalization are applied to the dataset [ref4]. For each dataset five DNN are trained (ar- chitecture: 3x48x48-100C7-MP2-150C4-150MP2-250C4- 250MP2-300N-43N), resulting in a MCDNN with 25 columns. The results can be seen in Table 2.10.

| trial | Original | Imadjust | Histeq | Adaphisteq | Conorm |
|---|---|---|---|---|---|
| 1 | 98.56 | 98.39 | 98.80 | 98.47 | 96.63 |
| 2 | 98.16 | 98.58 | 98.27 | 98.47 | 98.33 |
| 3 | 98.64 | 98.77 | 98.51 | 98.51 | 98.46 |
| 4 | 98.46 | 98.61 | 98.31 | 98.53 | 98.62 |
| 5 | 98.54 | 98.77 | 98.58 | 98.58 | 98.66 |
| Avg | 98.47$\pm$0.18 | 98.62$\pm$0.15 | 98.48$\pm$0.22 | 98.50$\pm$0.04 | 98.54$\pm$0.14 |
| | Average DNN | recognition | rate: | 98.52$\pm$0.15 | |
| | | | MCDNN: | 99.46 | |

Table 2.10: Accuracy rates of MCDNN and the 25 DNN

### 2.4.5 MaxPoolingFragmentlayer

Despite achieving very good results for different problems to which they are subject, the methods presented so far have one major drawback, the computational cost. To solve this problem, these networks have been using the GPU to boost performance. Even so, the training of the presented datasets can take hours or even days. This problem is a serious limitation for many industries, where large datasets are used and training has to be fast. When speed is a priority, it is necessary to find a solution that substantially reduces the training time. Jonathan Masci et al [MGC$^+$13] proposed an algorithm to train the CNN with maxpooling layers (MPCNN), achieving a very significant increase in speed. The conventional CNNs with maxpooling layers present an excellent performance, but their training operates separately for all the image patches, making it a time-consuming process.

#### 2.4.5.1 Method

MaxPoolingFragment (MPF) is an extension of the maxpooling layer (MP). When receiving a input image $x_i$, a conventional MP layer produces a smaller image in which only the most relevant amount in each non-overlapped region $K$ x $K$ is maintained. While a convolutional layer can be directly applied across the input image and produce results for all patches, an MP layer cannot and that causes redundant calculations. With a MPF layer there will be $k^2$ different offsets in the input map where each produces an output fragment. A MPF layer can be viewed as a collection of MP layers, each of which generates a fragment. If the number of input fragments is $|F_{in}|$, we will have a total of $|F_{in}|k^2$ fragments. All redundant calculations are removed. When considering the training image $x_i$ and their ground truth $t_i$. With the MPF layer we can quickly spread the network across the image in order to generate a output image $\hat{x}_i$, where each pixel is replaced by the output of MPCNN applied to each patch $x_i$. Figure 2.11 illustrates the process by using the MPF layer.

#### 2.4.5.2 Results

The dataset used to test is the ISBI 2102 Electron Microscopy Segmentation challenge [ACH10]. It consists of a volume of 30 gray level images of size 512 x 512 pixels. Networks are trained to minimize the multiclass cross-entropy loss error (MCCE). This implementation runs on the CPU in the Matlab environment. The architectures used N3 and N4 [CGGS12], were those that performed better in the competition. Table 2.11 shows that the speed gain is possible with this approach. This method showed a slightly higher error than N3, 6.8% vs 6.6%, however the speed gain eventually compensate for the low error difference.

|  | Patch (GPU) [CGGS12] patche/s [CGGS12] | Image (CPU, Matlab) patches/s |
|---|---|---|
| N3 | 260 | 4500 (1700 x speed-up) |
| N4 | 130 | 3000 (23x speed-up) |

Table 2.11: Comparison of train times.

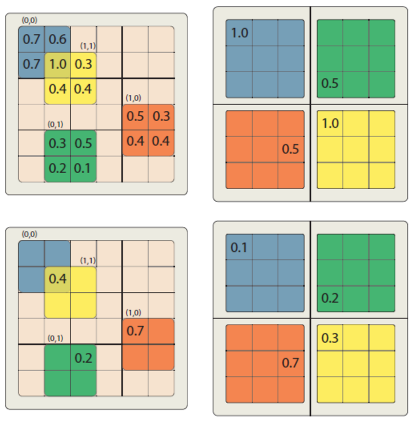


Figure 2.11: MPF layer for the 2 x 2 pooling case. Top: Forward pass, Fragments (0,0) and (1,1) share the same maximal element; Bottom: Backpropation pass where partial derivatives are pushed back to the previous layer in the hierarchy; the partial results of each Fragment are summed together [MGC+13].

### 2.4.6 Regularization Methods

Two approaches for regularizing CNNs have been recently proposed, DropOut [HSK+12] and DropConnect [WZZ+13]. Applying DropOut and DropConnect amounts to sub-sampling a neural network by dropping units. Since each of these processes acts differently as a way to control overfitting, the combination of several of these networks can bring gains.

DropOut is applied to the outputs of a fully connected layer where each element of an output layer is kept with probability $p$, otherwise being set to 0 with probability $(1-p)$. If we further assume a neural activation function with $a(0) = 0$, such as $tanh$ and $relu$, the output of a layer can be written as:

$$r = m * a(Wv)$$

where $m$ is a binary mask vector of size $d$ with each element $j$ coming independently from a Bernoulli distribution $m_j \sim Bernoulli(p)$, $W$ is a matrix with weights of a fully-connected layer and $v$ are the fully-connected layer inputs [WZZ+13].

DropConnect is similar to DropOut, but applied to the weights $W$. The connections are choosen randomly during the training. For a DropConnect layer, the output is given as:

$$r = a((M * W)v)$$

where $M$ is weight binary mask, and $M_{ij} \sim Bernoulli(p)$. Each element of the mask $M$ is drawn independently for each example during training [WZZ+13]. Figure 2.12 illustrates the differences between the two methods.

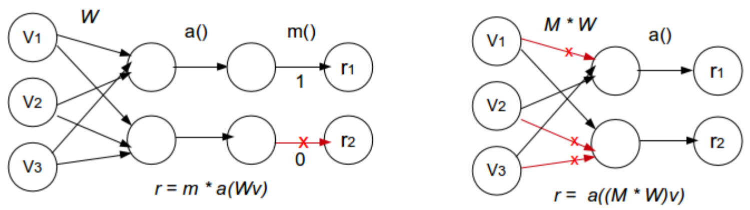


Figure 2.12: The left figure is an example of DropOut. Right figure is an example of DropConnect.

### 2.4.6.1 Results

The following protocol for all experiments unless otherwise stated is used: Augment the dataset by randomly selecting cropped regions from the images, flipping images horizontally and 15% scaling and rotation variations; Train 5 independent networks with DropAll, DropConnect or neither (NoDrop); Manually decrease the learning rate if the network stops improving as in Krizhevsky $et\ al$ [KSH12];
Once the 5 networks are trained we report two numbers: the mean and standard deviation of the classification errors produced by each of the 5 independent networks, and the classification error that results when averaging the output probabilities from the 5 networks before making a prediction.
Three numbers of epochs are used, such as 600-400-200 to define the schedule. Multiply the initial rate by 1 for the first such number of epochs. Then, use a multiplier of 0.5 for the second number of epochs followed by 0.1 again for this second number of epochs. The third number of epochs is used for multipliers of 0.05, 0.01 and 0.005, in that order.

### 2.4.6.2 MNIST

This feature extractor consists of a 2 layer CNN with 32-64 feature maps in each layer respectively. The last layerâs output is treated as input to the fully con- nected layer which has 150 $relu$ units on which No- Drop, Dropout or DropConnect are applied. Original images are cropped 24 x 24 in images from random locations, and rotated and scaled versions of these cropped images. The initial learning rate is 0.01 with a 700-200-100 epoch schedule and preprocess by subtracting the image mean.
Table 2.12 shows the results.

| Model | Error % 5 networks | Voting error % |
|---|---|---|
| NoDrop | 0.30 $\pm 0.035$ | **0.21** |
| DropOut | 0.28 $\pm 0.016$ | 0.27 |
| DropConnect | 0.28 $\pm 0.032$ | **0.21** |

Table 2.12: MNIST classification error. NoDrop and DropConnect achieve a test error of 0.23%, which sets the new state-of-the-art

### 2.4.6.3 CIFAR-10

The networks use a feature extractor with 2 convolutional layers and 2 locally connected layers as described in [HSK$^+$12]. A 128 neuron fully connected layer with $relu$ activations is added between the softmax layer and feature extractor. Following [HSK$^+$12] images are cropped to 24 x 24 with horizontal flips and no rotation or scaling is performed. The networks use

an initial learning rate of 0.001 and train for 700-300-50 epochs with their default weight decay. Model voting significantly improves performance when using Dropout or DropConnect, the latter reaching an error rate of 9.41%. Additionally a training model with 12 networks with DropConnect that achieved 9.32 %.
Table 2.13 shows the results.

| Model | Error % 5 networks | Voting error % |
|---|---|---|
| NoDrop | 11.18 $\pm 0.13$ | 10.21 |
| DropOut | 11.52 $\pm 0.18$ | 9.83 |
| DropConnect | 11.10 $\pm 0.13$ | 9.41 |

Table 2.13: CIFAR-10 classification error. Voting with 12 DropConnect networks produces an error rate of 9.32%.

#### 2.4.6.4 SVHN

Due to the large variety of colors and brightness variations in the images, images are preprocessed using local contrast normalization as in [ZF13]. The feature extractor is the same as the larger CIFAR-10 experiment, but with a larger 512 unit fully connected layer with $relu$ activations between the softmax layer and the feature extractor. After contrast normalizing, the training data is randomly cropped to 28 x 28 pixels and is rotated and scaled. Table 2.14 shows the classification performance for 5 models trained with an initial learning rate of 0.001 for a 100-50-10 epoch schedule.

| Model | Error % 5 networks | Voting error % |
|---|---|---|
| NoDrop | 2.26 $\pm 0.072$ | **1.94** |
| DropOut | 2.25 $\pm 0.034$ | 1.96 |
| DropConnect | 2.23 $\pm 0.039$ | **1.94** |

Table 2.14: SVHN classification error. NoDrop and DropConnect achieve a test error of 1.94%, which sets the new state-of-the-art .

### 2.4.7 MLP Convolution Layers

A new deep network structure called "Network In Network"(NIN) [LCY14] was recently proposed. The conventional convolutional layer uses linear filters followed by a nonlinear activation function to scan the input. In NIN, micro neural networks are built with more complex structures to abstract the data within the receptive field. The micro neural networks are instantiated with a multilayer perceptron.
The micro network structure is called mplconv layer, and is compared with traditional CNN convolutional layer in figure 2.13.
The mlpconv maps the input local patch to the output feature vector with a multilayer perceptron (MLP) consisting of multiple fully connected layers with nonlinear activation functions. The MLP is shared among all local receptive fields. The feature maps are obtained by sliding the MLP over the input in a similar manner as CNN and are then fed into the next layer. The overall structure of the NIN is the stacking of multiple mlpconv layers.
In NIN, Instead of adopting the traditional fully connected layers for classification in CNN, we directly output the spatial average of the feature maps from the last mlpconv layer as the

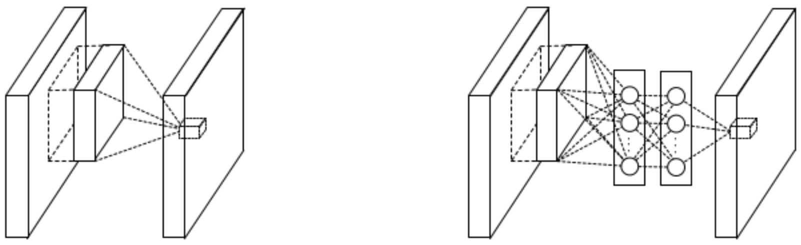


Figure 2.13: Comparison of linear convolution layer (left) and mlpconv layer (right). The linear convolution layer includes a linear filter while the mlpconv layer includes a micro network (multilayer perceptron) [LCY14].

confidence of categories via a global average pooling layer, and then the resulting vector is fed into the softmax layer.
Figure 2.14, shows the structure of NIN.

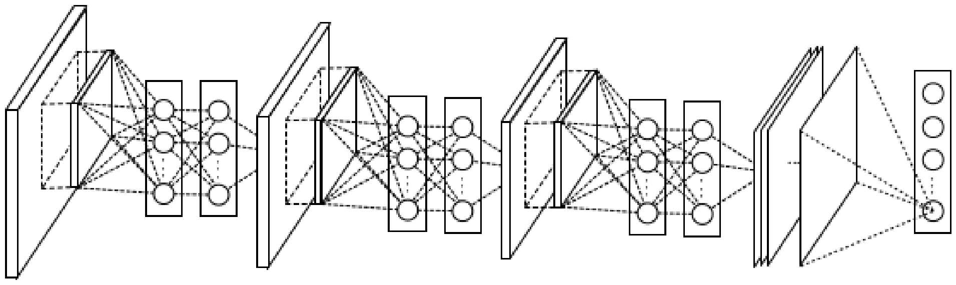


Figure 2.14: The overall structure of Network In Network. Include the stacking of three mlpconv layers and one global average pooling layer [LCY14].

### 2.4.7.1 Global Average Pooling

Global Average Pooling is a strategy to replace the traditional fully connected layers in CNN. The idea is to generate one feature map for each corresponding category of the classification task in the last mlpconv layer. Instead of adding fully connected layers on top of the feature maps, we take the average of each feature map, and the resulting vector is fed directly into the softmax layer. One advantage of global average pooling over the fully connected layers is that it is more native to the convolution structure by enforcing correspondences between feature maps and categories. Thus the feature maps can be easily interpreted as categories confidence maps. Another advantage is that there is no parameter to optimize in the global average pooling thus overfitting is avoided at this layer. Futhermore, global average pooling sums out the spatial information, thus it is more robust to spatial translations of the input.

### 2.4.7.2 Results

The networks used for all the experiments consist of three stacked mlpconv layers, and the mlpconv layers in all the experiments are followed by a spatial max pooling layer which downsamples the input image by a factor of two. As a regularizer, dropout is applied. Unless stated specifically, all the networks used in the experiment section use global average pooling instead of fully connected layers at the top of the network. Another regularizer applied is weight decay as used by Krizhevsky $et\ al$ [KSH12]. The overall training procedure adopted is the same used by Krizhevsky Krizhevsky $et\ al$ [KSH12].

The training process starts from the initial weights and learning rates, and it continues until the accuracy on the validation set stops improving, and then the learning rate is lowered by a scale of 10. This procedure is repeated once such that the final learning rate is one percent of the initial value.

#### 2.4.7.3 CIFAR-10

The number of feature maps for each mlpconv layer in this experiment is 64. Two hyper-parameters are tuned using the validation set, i.e. the local receptive field size and the weight decay. After that the hyper-parameters are fixed and the network are re-trained from scratch with both the training set and the validation set. The resulting model is used for testing. Table 2.15 shows the results.

| Method % | Test Error % |
|---|---|
| NIN | 14.51 |
| NIN + Dropout | 10.41 |
| NIN + Dropout + Data Augmentation | **8.81** |

Table 2.15: Test set error rates for CIFAR-10 using 3 methods.. NIN + Dropout + Data Augmentation method achieve a test error of 8.81%, which sets the new state-of-the-art.

#### 2.4.7.4 CIFAR-100

CIFAR-100 training use the same setting as the CIFAR-10 dataset. The only difference is that the last mlpconv layer outputs 100 feature maps. A test error of 35.68% is obtained for CIFAR-100 which is the current best performance without data augmentation.

# Chapter 3

# Weighted Convolutional Neural Network Ensemble

In this chapter, we introduce a new method to combine the output probabilities of convolutional neural networks which we call Weighted Convolutional Neural Network Ensemble. In this approach, each network has an associated weight that makes networks with better performance have a greater influence at the time to classify an image, in relation to the networks that performed worse. This method produces better results than the stardard method that combines the networks doing just the average of the output probabilities to make the predictions.

## 3.1 Method's Description

Model combination improves the performance of machine learning models. Averaging the predictions of several models is most helpful when the individual models are different from each other, in other words, to make them different they must have different hyperparameters or be trained on different data.

The standard model architecture to combine networks can be seen in figure 3.1. Given some input pattern, the output probabilities from all CNN columns are averaged before making a prediction. For output $i$, the average output $S_i$ is given by:

$$S_i = \frac{1}{n}\sum_{j=1}^{n} r_j(i)$$

where $r_j(i)$ is the output $i$ of network $j$ for a given input patern.

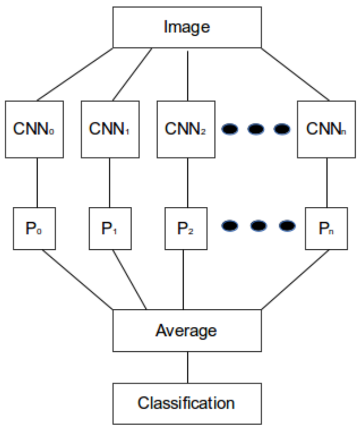


Figure 3.1: The output probabilities are averaged to make the final prediction.

Our approach consists in applying a different weight for each network. In the validation set, networks that had a lower classification error will have a larger weight when combining the results. The model architecture can be seen in figure 3.2. Given some input pattern, the output probabilities from all CNNs are multiplied by a weight before the prediction:

$$S_i = \sum_{j=1}^{n} W_j r_j(i)$$

We use two different approaches to calculate the weight $W$. The first method consists on a weighted mean:

$$W_k = \frac{A_k}{\sum_{i=1}^{n} A_i}$$

where $A_k$ is the accuracy in the validation set for the network $k$, and $i$ runs over the $n$ networks. In the second method, the weight $W_k$ is choosen by rank. The weights are based on the order of accuracy in the validation set. This means that the weights are fixed, independently on the value of the error:

$$W_k = \frac{R(A_k)}{\sum_{i=1}^{n} R(A_i)}$$

where $R()$ is a function that gives the position of the network based on the validation accuracy sorted in increasing order. For example, the network with largest accuracy will have an $R()$ value of $n$, the network with the second largest accuracy an $R()$ value of $n-1$ and so on until the network with lowest accuracy gets an $R()$ = 1.

The first method has weights that are proportional to the accuracy in validation set. If the errors are too close, the weights have values close to each other, and the result will be similar to a simple average.

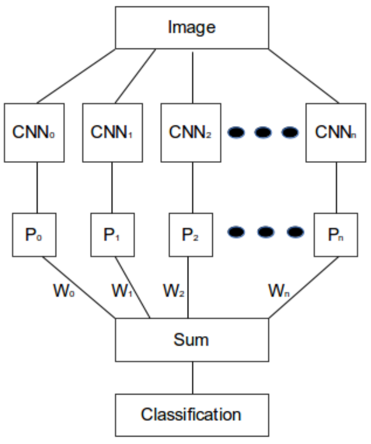


Figure 3.2: The output probabilities are weighted based on the accuracy of the network evalueted on the validation set.

## 3.2 Experiments

Our experiments use a fast GPU based convolutional network library called Cuda-convnet [Kri] in conjunction with Li's code [HSK$^+$12] that allows training networks with DropOut and Dropconnet. We use a NVIDIA TESLA C2075 GPU and a NVIDIA GTX 590 to run the experiments. For each dataset we train five networks with DropConnect, DropOut and NoDrop (five of each). Once the networks are trained we save the mean and standard deviation of the classification errors produced individually by each network and the classification error produced by these networks when combined with our two proposed methods methods and simple average. These results are shown in Tables 3.1-3.4. We use 2 different datasets to evaluate our approach.

### 3.2.1 CIFAR-10

The CIFAR-10 dataset [Kri09] consists of 32 x 32 color images drawn from 10 classes split into 50 000 train and 10 000 test images.

Before feeding these images to our network, we subtract the per-pixel mean computed over the training set from each image as was done in [HSK$^+$12]. The images are cropped to 24x24 with horizontal flips.

We use two feature extractors to perform the experiment. The first, consists in 2 convolutional layers, with 64 feature maps in each layer, 2 maxpooling layers, 2 locally connected layers, a fully connected layer which has 128 $relu$ units on which NoDrop, DropOut or DropConnect are applied and a output layer with $softmax$ units. We train for three stages of epochs, 500-100-100 with an initial learning rate of 0.001, that its reduced by factor 10 between each stage. We chose this fixed number of epochs because it is when the validation error stops improving. Training a network takes around 4 hours. The second feature extractor is similar but with 128 feature maps in each layer and the number of epochs is smaller, 350-100-50. Training a network with 128 maps takes around 20 hours.

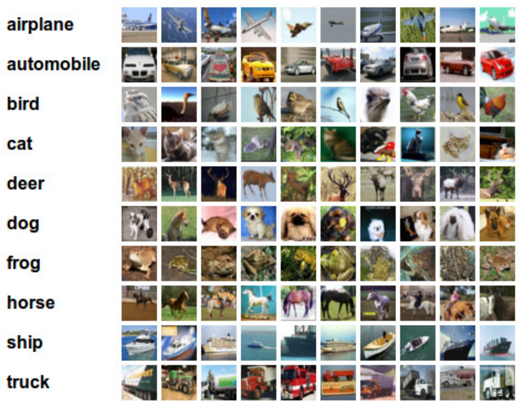


Figure 3.3: CIFAR-10 dataset. Example of 10 random images from each class [Kri09].

In these experiments we compared the results using the three approaches for combining networks described in this chapter. The first experiment used a feature extractor with 64 feature maps (summarized in Table 3.1) and combined networks that were trained with DropConnect, DropOut and NoDrop. NoDrop individually obtained better results and networks with DropOut were the ones with the worst individual results. By combining the nets, DropConnect achieved better results. In addition we combine our 12 best networks (see Table 3.2) and the results were significantly better than the result shown in Table 3.1. With method 2 we achieved an error of $9.37\%$.

The second experiment used a feature extractor with 128 feature maps (summarized in Table 3.3), we also combine networks that were trained with DropConnect, DropOut and NoDrop. DropOut individually achieved better results, and networks trained with DropConnect were the ones with worst result. When combining our 12 best networks, we obtained better results, as can be seen in Table 3.4. Using our second method we obtain $9.12\%$, improving the current state-of-the-art of CIFAR-10.

Table 3.1: CIFAR-10 classification error using 3 types of networks and 3 types of combiners, using 64 feature maps.

| Model | DropConnect | DropOut | NoDrop |
|---|---|---|---|
| Error % 5 networks | 11.18 $\pm 0.15$ | 11.28 $\pm 0.17$ | 10.92 $\pm 0.15$ |
| Method 1 % error | 9.81 | 10.45 | 10.06 |
| Method 2 % error | 9.81 | 10.31 | 10.03 |
| Simple Average % error | 9.84 | 10.48 | 10.06 |

Table 3.2: CIFAR-10 classification error combining our 12 best networks using 3 types of combiners, using 64 feature maps. Previous state-of-the-art is 9.32% [WZZ$^+$13].

| Model | Method 1 | Method 2 | Simple Average |
|---|---|---|---|
| 12 networks % error | 9.50 | 9.37 | 9.47 |

Table 3.3: CIFAR-10 classification error using 3 types of networks and 3 types of combiners, using 128 feature maps.

| Model | DropConnect | DropOut | NoDrop |
|---|---|---|---|
| Error % 5 networks | 10.53 $\pm 0.14$ | 10.53 $\pm 0.13$ | 10.53 $\pm 0.17$ |
| Method 1 % error | 9.77 | 9.68 | 9.63 |
| Method 2 % error | 9.68 | 9.55 | 9.61 |
| Simple Average % error | 9.81 | 9.71 | 9.64 |

Table 3.4: CIFAR-10 classification error combining our 12 best networks using 3 types of combiners, using 128 feature maps. Previous state-of-the-art is 9.32% [WZZ$^+$13].

| Model | Method 1 | Method 2 | Simple Average |
|---|---|---|---|
| 12 networks % error | 9.20 | 9.12 | 9.22 |

### 3.2.2 MNIST

The MNIST dataset [LBBH98] consists of 28 x 28 pixel greyscale images of handwritten digits 0-9, with 60,000 training and 10,000 test examples. Our feature extractor consists in 2 convolutional layers, with 32-64 feature maps in each layer respectively, 2 maxpooling layers, a fully connected layer which has 150 $relu$ units on which NoDrop, DropOut or DropConnect are applied and a output layer with $softmax$ units. We train for three stages of epochs 500-100-50 with an initial learning rate of 0.01, that its reduced by factor 10 between each stage. Training a network takes around 50 minutes. No data augmentation is utilized in this experiment.

Table 3.5: MNIST classification error using 3 types of networks and 3 types of combiners. No data augmentation is used.

| Model | DropConnect | DropOut | NoDrop |
|---|---|---|---|
| Error % 5 networks | 1.32 $\pm 0.03$ | 1.214 $\pm 0.01$ | 1.604 $\pm 0.07$ |
| Method 1 % error | 1.17 | 1.08 | 1.33 |
| Method 2 % error | 1.17 | 1.07 | 1.31 |
| Simple Average % error | 1.17 | 1.08 | 1.33 |

Figure 3.4: MNIST dataset. Example of 100 random images.

## 3.3 Discussion

The experiments showed, that applying a fixed weight to each network (method 2), ensures better results than the other two approaches. We also conclude that when combining a large number of networks, the weighted mean (method 1) performance was similar to the simple average. We also concluded that a feature extractor with 128 feature maps achieves better results than a feature extractor with 64 feature maps. However when using 128 maps we have to use less epochs because the network generalize faster.
In the MNIST experiment, there is no noticeable difference in the results provided by the three methods. The validation error is very low and almost identical in all networks, this means that the results are very similar between the three methods. In datasets where the error is already quite low, there is no great advantage in using our approach, however in datasets whose error value is significant, there is a notable difference if we assign a weight to each network instead of using a simple average.
At the end, with one of our methods, we improved the state-of-the-art of CIFAR-10 for a short period. Currently, the state-of-the-art is 8.81% [LCY14].

# Chapter 4

# DropAll: Generalization of Two Regularization Methods

In this chapter, we introduce DropAll, a generalization of DropOut [HSK$^+$12] and DropConnect [WZZ$^+$13], for regularization of fully-connected layers within convolutional neural networks. Applying these methods amounts to sub-sampling a neural network by dropping units. When training with DropOut, a randomly selected subset of the output layer's activations are dropped, when training with DropConnect we drop a randomly subsets of weights. With this approach we were able to train a network with DropOut, DropConnect or both and taking advantage of each method.

## 4.1 Method's Description

DropAll is a generalization of DropOut [HSK$^+$12] and DropConnect [WZZ$^+$13], for regularizing fully-connected layers within neural deep networks.
As shown in chapter 2, DropOut is applied to the outputs of a fully connected layer where each element of an output layer is kept with probability $p$, otherwise being set to 0 with probability $(1-p)$. If we further assume a neural activation function with $a(0) = 0$, such as $tanh$ and $relu$, the output of a layer can be written as:

$$r = m * a(Wv)$$

where $m$ is a binary mask vector of size $d$ with each element $j$ coming independently from a Bernoulli distribution $m_j \sim Bernoulli(p)$, $W$ is a matrix with weights of a fully-connected layer and $v$ are the fully-connected layer inputs [WZZ$^+$13].
DropConnect is similar to DropOut, but applied to the weights $W$. The connections are choosen randomly during the training. For a DropConnect layer, the output is given as:

$$r = a((M * W)v)$$

where $M$ is weight binary mask, and $M_{ij} \sim Bernoulli(p)$. Each element of the mask $M$ is drawn independently for each example during training [WZZ$^+$13]. Figure 4.1 illustrates the differences between the two methods.

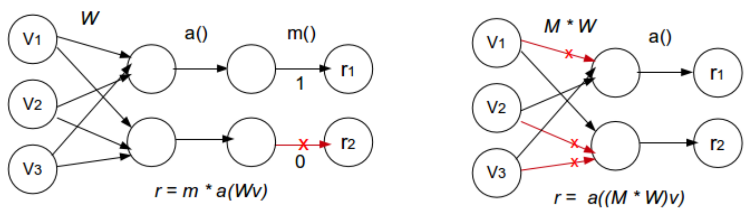


Figure 4.1: The left figure is an example of DropOut. Right figure is an example of DropConnect.

For a DropAll layer, the output is given as:

$$r = m * a((M * W)v)$$

The DropAll model is presented graphically in figure 4.2. This approach has the particularity of being easily adaptable to one of the previous methods. In these two methods, we had only one variable where we choose the percentage of drops, with DropAll we have 2 variables. One variable controls the drop rate of the activation while the other variable controls the drop rate of the weight. If we set one of these variables to zero, we obtain either DropOut or DropConnect.

In both methods the value of the drop rate used is usually 0.5, however if we train with DropAll with 0.5 in both rates, the network discards a lot of information, which is reflected in the results. To solve this problem, the drop rate must be smaller for both variables. When testing the network with different drop rates, we concluded that 0.25 is a good compromise.

In the following section we compare DropAll with DropConnect, DropAll and NoDrop (trained network without dropping units). All of these methods used in conjunction, provide a greater randomness when tested and combining the results from different networks trained by these techniques significantly improves the classification rates.

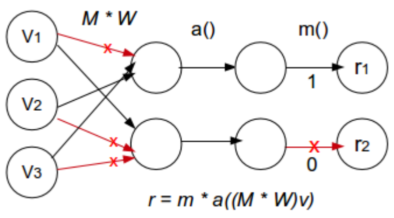


Figure 4.2: Example of DropAll model.

## 4.2 Experiments

Our experiments use a fast GPU based convolutional network library called Cuda-convnet [Kri] in conjunction with Li's code [HSK+12] that allows training networks with DropOut, Dropconnet and DropAll. We use a NVIDIA TESLA C2075 GPU and a NVIDIA GTX 59 to run the experiments. For each dataset we train five networks with DropAll, DropConnect, DropOut and NoDrop (five of each).

Once the networks are trained we save the mean and standard deviation of the classification errors produced individually by each network and the classification error produced by these networks when combined with WCNEE method and simple average. These results are shown in Tables 4.1-4.3. We used the CIFAR-10 dataset [Kri09] to evaluate our approach.

### 4.2.1 CIFAR-10

We use the same data and the same feature extractors to perform the experiment, described in chapter 3. In these experiments we compared the results using (WCNNE) for combining networks and simple average.

The first experiment used a feature extractor with 64 feature maps (summarized in Table 4.1) and combined networks that were trained with DropAll, DropConnect, DropOut and NoDrop.

NoDrop individually obtained better results and networks with DropOut were the ones with the worst individual results. By combining the nets, DropConnect achieved better results.
The second experiment used a feature extractor with 128 feature maps (summarized in Table 4.2), we also combine networks that were trained with DropAll, DropConnect, DropOut and NoDrop. DropOut individually achieved better results, and networks trained with DropAll were the ones with worst result. By combining the nets, DropOut achieved better results.
In addition we join all models and combine our 12 best networks with lowest validation error, and the results were significantly better (see Table 4.3). All of these methods used in conjunction provide a greater randomness and significantly improve the classification rate. If we combine our 12 best networks without DropAll networks the error is slightly worse, 9.12%.

Table 4.1: CIFAR-10 average classification error in percentage and standard deviation using 4 types of networks and 2 types of combiners, using 64 feature maps.

| Model | DropAll | DropConnect | DropOut | NoDrop |
|---|---|---|---|---|
| 5 networks | 11.20 $\pm 0.10$ | 11.18 $\pm 0.15$ | 11.28 $\pm 0.17$ | 10.92 $\pm 0.15$ |
| WCNNE | 10.01 | 9.81 | 10.31 | 10.03 |
| Simple Average | 10.03 | 9.84 | 10.48 | 10.06 |

Table 4.2: CIFAR-10 average classification error in percentage and standard deviation using 4 types of networks and 2 types of combiners, using, using 128 feature maps.

| Model | DropAll | DropConnect | DropOut | NoDrop |
|---|---|---|---|---|
| 5 networks | 10.67 $\pm 0.11$ | 10.53 $\pm 0.14$ | 10.53 $\pm 0.13$ | 10.53 $\pm 0.17$ |
| WCNNE | 9.57 | 9.68 | 9.55 | 9.61 |
| Simple Average | 9.62 | 9.81 | 9.71 | 9.64 |

Table 4.3: CIFAR-10 average classification error combining our 12 best networks using 2 types of combiners, using 128 feature maps. Previous state-of-the-art using the same architecture is 9.32% [WZZ$^+$13]. Current state-of-the art of CIFAR-10 is 8.81% [LCY14].

| Model | WCNNE | Simple Average |
|---|---|---|
| 12 networks | 9.09 | 9.22 |

## 4.3 Discussion

We propose a new method named DropAll that is a generalization of two well-known methods for regularization of convolutional neural networks, used to avoid overfitting. These problem still remains a challenge to overcome when it comes to training extremely large neural networks or working in domains which offer very small amounts of data.
DropAll by itself, did not increase performance when we evaluate a network, however, the flexibility of this method makes it possible to train a network using the potential of DropOut and DropConnect. In general, networks trained with these forms of regularization benefit from an increase randomness, which is a plus when we wish to combine the results of multiple networks. As shown, the combination of all methods significantly improves the classification rate of the problem used in the experiments section to validate our proposal.

# Chapter 5

# Conclusion

In this work the main goal was the development of one method (at least), that would allow us to contribute to the research area of deep learning, in particular in the sub-area of convolutional neural networks.
CNNs have been the subject of extensive study and improvement over the years. New approaches are constantly emerging and many commmon benchmarks are regularly beaten, which makes it a very challenging area.
There are two things that are common to recent approaches, that is the training of the network using GPU power, and model combination to improve the classification rate [JKRL09, CPS06].
Although trained networks using GPUs have been improved in recent years, the way how networks are combined to improve results remains unchanged. And from here came the first idea: to implement a method that combines the networks, improving the stardard method that combines the networks doing the average of the output probabilities to make the predictions.
To improve the results, we apply a weight to each network. Networks that had a lower classification error in the validation set, will have a larger weight when combining the results. We use two different approaches to calculate the weights.
The first method consists on a weighted mean. With this method the weight is variable, which can be a problem if the validation errors are too close, as the weights values will be close to each other, and the result will be similar to a simple average.
To avoid this problem we fixed the weights. The weights are choosed by rank and are based on the order of accuracy in the validation set.
There are other factors that can be addressed to improve the results when combining networks. If the networks are all trained in the same way i.e the same hiperparameters and same data, the examples that the network misses, are very similar between networks. If there is no variability, model combination will not improve the results. Training networks in different ways increases the randomness between data, and is an asset when combining networks predictions.
DropOut and DropConnect are two regularization methods. Training with these methods allows us to use larger networks without overfitting. The dropping of units, increases the variability between networks. And from here came the second idea.
Both methods are similar, however we can not apply the 2 methods together to train a network. Hence we propose DropAll, that is a generalization of both methods. DropAll by itself, did not increase performance when we evaluate a network, however, the flexibility of this method makes it possible to train a network using the potential of DropOut and DropConnect, and benefit from an increase randomness, which in addition with model combination improve the classification rate of a problem.
To achieve such goals, we use a network library called Cuda-convnet [Kri] in conjunction with code by Li [HSK$^+$12]. We choose CIFAR-10, a well known dataset, to evaluate our proposal. This dataset is subject of intense study and has the particularity of not being too big and the classification error is not that big when compared with other common datasets.
We also used the MNIST dataset, but the error is already quite low making it difficult to evaluate our proposals.

In conclusion, the research carried out enabled the development of two methods able to regularize and improve the classification rate when performing model combination. The results attained by our approachs can also be considered an achievement, since they outperform methods previously tested in same architecture. Therefore, we consider all our goals fulfilled.

# Bibliography

[ACH10] Stephan Preibisch Benjamin Schmid Anchi Cheng Jim Pulokas Pavel Tomancak Albert Cardona, Stephan Saalfeld and Volker Hartenstein. An integrated micro- and macroarchitectural analysis of the drosophila brain by computer-assisted serial section electron microscopy. In *PLoS Biol*, pages vol, 8. pp. e1000502., no. 10. 17

[BPL10] Y-Lan Boureau, Jean Ponce, and Yann Lecun. A theoretical analysis of feature pooling in visual recognition. In *27TH INTERNATIONAL CONFERENCE ON MACHINE LEARNING, HAIFA, ISRAEL*, 2010. 9

[CBD$^+$90] Y. Le Cun, B. Boser, J. S. Denker, R. E. Howard, W. Habbard, L. D. Jackel, and D. Henderson. Advances in neural information processing systems 2. chapter Handwritten digit recognition with a back-propagation network, pages 396--404. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990. Available from: `http://dl.acm.org/citation.cfm?id=109230.109279`. ix, 4, 5

[CdSBB$^+$06] Paulo Rodrigo Cavalin, Alceu de Souza Britto, Jr., Flávio Bortolozzi, Robert Sabourin, and Luiz E. Soares Oliveira. An implicit segmentation-based method for recognition of handwritten strings of characters. In *Proceedings of the 2006 ACM symposium on Applied computing*, SAC ’06, pages 836--840, New York, NY, USA, 2006. ACM. Available from: `http://doi.acm.org/10.1145/1141277.1141468`. 16

[CGGS12] Dan C. Ciresan, Alessandro Giusti, Luca Maria Gambardella, and Jurgen Schmidhuber. Deep neural networks segment neuronal membranes in electron microscopy images. In Peter L. Bartlett, Fernando C. N. Pereira, Christopher J. C. Burges, LÃ©on Bottou, and Kilian Q. Weinberger, editors, *NIPS*, pages 2852--2860, 2012. Available from: `http://dblp.uni-trier.de/db/conf/nips/nips2012.html#CiresanGGS12`. 17, 18

[CMGS10] Dan Claudiu Cireşan, Ueli Meier, Luca Maria Gambardella, and Jürgen Schmidhuber. Deep, big, simple neural nets for handwritten digit recognition. *Neural Comput.*, 22(12):3207--3220, December 2010. Available from: `http://dx.doi.org/10.1162/NECO_a_00052`. 15

[CMGS11] Dan Claudiu Ciresan, Ueli Meier, Luca Maria Gambardella, and Jurgen Schmidhuber. Convolutional neural network committees for handwritten character classification. In *ICDAR*, pages 1135--1139. IEEE, 2011. Available from: `http://dblp.uni-trier.de/db/conf/icdar/icdar2011.html#CiresanMGS11`. 15

[CMMS12] Dan C. Ciresan, Ueli Meier, Jonathan Masci, and JÃ$\frac{1}{4}$rgen Schmidhuber. Multi-column deep neural network for traffic sign classification. *Neural Networks*, 32:333--338, 2012. Available from: `http://dblp.uni-trier.de/db/journals/nn/nn32.html#CiresanMMS12`. v, ix, 1, 13, 14

[CPS06] Kumar Chellapilla, Sidd Puri, and Patrice Simard. High Performance Convolutional Neural Networks for Document Processing. In Guy Lorette, editor, *Tenth International Workshop on Frontiers in Handwriting Recognition*, La Baule (France), October 2006. Université de Rennes 1, Suvisoft. http://www.suvisoft.com Université de Rennes 1. Available from: http://hal.inria.fr/inria-00112631. 1, 33

[DC07] James J DiCarlo and David D Cox. Untangling invariant object recognition. *Trends in Cognitive Sciences*, 11(8):333–341, aug 2007. PMID: 17631409. Available from: http://dicarlolab.mit.edu/sites/dicarlolab.mit.edu/files/pubs/dicarlo%20and%20cox%202007.pdf. 1

[dSOSM08] Eulanda Miranda dos Santos, Luiz S. Oliveira, Robert Sabourin, and Patrick Maupin. Overfitting in the selection of classifier ensembles: a comparative study between pso and ga. In Conor Ryan and Maarten Keijzer, editors, *GECCO*, pages 1423--1424. ACM, 2008. Available from: http://dblp.uni-trier.de/db/conf/gecco/gecco2008.html#SantosOSM08. 16

[Fuk80] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*, 36:193--202, 1980. ix, 3, 4

[Fuk86a] Kunihiko Fukushima. A neural network model for selective attention in visual pattern recognition. *Biol. Cybern.*, 55(1):5--16, October 1986. Available from: http://dx.doi.org/10.1007/BF00363973. v, 1

[Fuk86b] Kunihiko Fukushima. A neural network model for selective attention in visual pattern recognition. *Biol. Cybern.*, 55(1):5--16, October 1986. Available from: http://dx.doi.org/10.1007/BF00363973. 3

[GBC$^+$94] Michael D. Garris, James L. Blue, Gerald T. Candela, Gerald T. C, Darrin L. Dimmick, Jon Geist, Patrick J. Grother, Stanley A. Janet, and Charles L. Wilson. Nist form-based handprint recognition system. Technical report, Technical Report NISTIR 5469 and CD-ROM, National Institute of Standards and Technology, 1994. 15

[GTR13] GTRSB. *GTRSB benchmarks*. http://benchmark.ini.rub.de/, 2013. 13

[HSK$^+$12] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012. Available from: http://dblp.uni-trier.de/db/journals/corr/corr1207.html#abs-1207-0580. v, 1, 18, 19, 24, 25, 29, 30, 33

[HW62] D. H. HUBEL and T. N. WIESEL. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 160:106--154, January 1962. Available from: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC1359523/. 3

[JKL] Kevin Jarrett, Koray Kavukcuoglu, and Yann Lecun. What is the best multi-stage architecture for object recognition? 9

[JKRL09] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 2146--2153, Sept 2009. 1, 33

[KE95] James Kennedy and Russell C. Eberhart. Particle swarm optimization. In *Proceedings of the IEEE International Conference on Neural Networks*, pages 1942--1948, 1995. 16

[KK05] Alessandro L. Koerich and Pedro R. Kalva. Unconstrained handwritten character recognition using metaclasses of characters. In *ICIP (2)*, pages 542--545. IEEE, 2005. Available from: http://dblp.uni-trier.de/db/conf/icip/icip2005-2.html#KoerichK05. 16

[Kri] Alex Krizhevsky. Cuda-convnet. http://code.google.com/p/cuda-convnet/. 2012. 10, 24, 30, 33

[Kri09] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009. ix, 11, 25, 30

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C.J.C. Burges, L. Bottou, and K.Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097--1105. Curran Associates, Inc., 2012. Available from: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf. 19, 21

[LBBH98] Yann Lecun, LÃ©on Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278--2324, 1998. ix, 6, 8, 9, 11, 15, 26

[LCY14] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *CoRR*, abs/1312.4400, 2014. v, ix, xii, 1, 20, 21, 27, 31

[MGC$^+$13] Jonathan Masci, Alessandro Giusti, Dan C. Ciresan, Gabriel Fricout, and Jürgen Schmidhuber. A fast learning algorithm for image segmentation with max-pooling convolutional networks. *CoRR*, abs/1302.1690, 2013. ix, 17, 18

[NWC$^+$] Yuval Netzer, Tao Wang, Adam Coates, Ro Bissacco, Bo Wu, and Andrew Y. Ng. Reading digits in natural images with unsupervised feature learning. 12

[PCL06] Christopher Poultney, Sumit Chopra, and Yann Lecun. Efficient learning of sparse representations with an energy-based model. In *Advances in Neural Information Processing Systems (NIPS 2006*. MIT Press, 2006. 15

[RHW88] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Neurocomputing: foundations of research. chapter Learning internal representations by error propagation, pages 673--695. MIT Press, Cambridge, MA, USA, 1988. Available from: http://dl.acm.org/citation.cfm?id=65669.104449. 4, 6

[Sch12] Jurgen Schmidhuber. Multi-column deep neural networks for image classification. In *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, CVPR '12, pages 3642--3649, Washington, DC, USA, 2012. IEEE Computer Society. Available from: http://dl.acm.org/citation.cfm?id=2354409.2354694. 13

[SCL12] Pierre Sermanet, Soumith Chintala, and Yann LeCun. Convolutional neural networks applied to house numbers digit classification. *CoRR*, abs/1204.3968, 2012. Available from: `http://dblp.uni-trier.de/db/journals/corr/corr1204.html#abs-1204-3968`. 12

[SMB10] Dominik Scherer, Andreas Müller, and Sven Behnke. 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, pages 92--101, Berlin, Heidelberg, 2010. Springer-Verlag. Available from: `http://dl.acm.org/citation.cfm?id=1886436.1886447`. 9

[SSP03] Patrice Y. Simard, Dave Steinkraus, and John C. Platt. J.c.: Best practices for convolutional neural networks applied to visual document analysis. In *In: Intâl Conference on Document Analysis and Recognition*, pages 958--963, 2003. 15

[SSSI11] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. The german traffic sign recognition benchmark: A multi-class classification competition. In *IJCNN*, pages 1453--1460. IEEE, 2011. Available from: `http://dblp.uni-trier.de/db/conf/ijcnn/ijcnn2011.html#StallkampSSI11`. 16

[SWB$^+$07] Thomas Serre, Lior Wolf, Stanley Bileschi, Maximilian Riesenhuber, and Tomaso Poggio. Robust object recognition with cortex-like mechanisms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29:411--426, 2007. 1

[Wik13a] Wikipedia. *Introduction to OpenCL Programming*. 2013. Available from: `http://en.wikipedia.org/wiki/Radial_basis_function`. 9

[Wik13b] Wikipedia. *Softmax*. 2013. Available from: `http://en.wikipedia.org/wiki/Softmax_activation_function/bi`. 13

[WZZ$^+$13] Li Wan, Matthew Zeiler, Sixin Zhang, Yann L. Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In Sanjoy Dasgupta and David Mcallester, editors, *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, volume 28, pages 1058--1066. JMLR Workshop and Conference Proceedings, May 2013. Available from: `http://jmlr.org/proceedings/papers/v28/wan13.pdf`. v, xi, xii, 1, 18, 26, 29, 31

[ZE02] Bianca Zadrozny and Charles Elkan. Transforming classifier scores into accurate multiclass probability estimates. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, KDD '02, pages 694--699, New York, NY, USA, 2002. ACM. Available from: `http://doi.acm.org/10.1145/775047.775151`. 16

[ZF13] Matthew D. Zeiler and Rob Fergus. Stochastic pooling for regularization of deep convolutional neural networks. *CoRR*, abs/1301.3557, 2013. Available from: `http://dblp.uni-trier.de/db/journals/corr/corr1301.html#abs-1301-3557`. v, ix, 1, 9, 10, 11, 20

# Appendix A

# Weighted Convolutional Neural Network Ensemble

Xavier Frazão and Luís A. Alexandre

Dept. of Informatics, Univ. Beira Interior
Covilhã, Portugal
xavierfrazao@gmail.com
lfbaa@ubi.pt
http://www.ubi.pt

**Abstract.** We introduce a new method to combine the output probabilities of convolutional neural networks which we call Weighted Convolutional Neural Network Ensemble. Each network has an associated weight that makes networks with better performance have a greater influence at the time to classify a problem in relation to that networks performed worse. This new approach produces better results than the common method that combines the networks doing just the average of the output probabilities to make the predictions. We show the validity of our proposal by improving the classification rate on a common image classification benchmark.

**Keywords:** Convolutional Neural Networks, Object Recognition, Network Ensemble.

## 1 Introduction

Convolutional neural networks (CNNs) are hierarchical neural networks whose convolutional layers alternate with subsampling layers, reminiscent of simple and complex cells in the primary visual cortex [1]. Although these networks are efficient when performing classification, they have the disadvantage of being computationally heavy, which makes their training slow and cumbersome.

With the emergence of parallel programming and taking advantage of the processing power of Graphics Processing Units (GPUs), training these networks takes significantly less time, making it possible to train large networks [2, 3] and also making it possible to train multiple networks for the same problem and combine their results [4, 5], an approach that can significantly increase the classification accuracy.

Besides the training time, the major problem of these networks is the overfitting. Overfitting still remains a challenge to overcome when it comes to training extremely large neural networks or working in domains which offer very small amounts of data. Many regularization methods have also been proposed to prevent this problem. These methods combined with large datasets have made it possible to apply neural large networks for solving machine learning problems in several domains. Two new approachs have been recently proposed, DropOut [5] and DropConnect [4], which is a generalization of the previous. When training with DropOut, a randomly selected subset of activations are droped. With DropConnect, we randomly drop the weights. Both techniques are only possible for fully connected layers. Combining the results from different networks trained by these techniques significantly improves the classification rate. In this paper, we propose a new method to combine the results of networks by applying a different weight for each network, instead of using the common method, that involves averaging the output probabilities of several networks.

## 2 Convolutional Neural Networks

A classical convolutional network is composed of alternating layers of convolution and pooling. The purpose of the first convolutional layer is to extract patterns found within local regions of the input images. This is done by convolving filters over the input image, computing the inner product of the filter at every location in the image and outputting the result as feature maps $c$. A non-linear function $f()$ is then applied to each feature map $c : a = f(c)$. The result activations $a$ are passed to the pooling/subsampling layers. These layers aggregate the information within a set of small local regions, $\{R_j\}_{j=1}^{n}$, producing a pooled feature map $s$ of smaller size as output.

Representing the aggregation function as $pool()$, then for each feature map $c$, we have: $s_j = pool(f(c_i)) \; \forall_i \in R_j$.

The two common choices to perform $pool()$ are average and max-pooling. The first takes the arithmetic mean of the elements in each pooling region, while max-pooling selects the largest element of the pooling region.

A range of functions $f()$ can be used as a non-linearity – *tanh*, *logistic*, $softmax$ and $relu$ are the most common choices.

In a convolutional network model, the convolutional layers, which take the pooled maps as input, can thus extract features that are increasingly invariant to local transformations of the input image.

The last layer is always a fully connected layer with one output unit per class in the recognition task. The activation function $softmax$, is the most common choice for the last layer such that each neurons output activation can be interpreted as the probability of a particular input image belonging to that class.

## 3 Related Work

### 3.1 Ensembles of CNNs

Model combination improves the performance of machine learning models. Averaging the predictions of several models is most helpful when the individual models are different from each other, in other words, to make them different they must have different hyperparameters or be trained on different data.

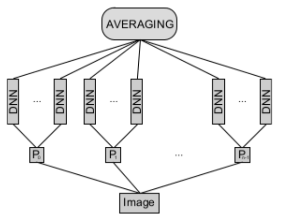


**Fig. 1.** MCDNN architecture. DNNs are trained on inputs preprocessed in different ways. The final prediction is a simple average of all DNNs predictions.

An example of the combination of multiples CNNs appears in [6], as the "Multi-column Deep Neural Networks for Image Classification"(MCDNN). In that model, the input image is preprocessed by blocks. The dataset is preprocessed before training, then, at the beginning of every epoch, the images are distorted (block). An arbitrary number of CNNs can be trained on inputs preprocessed in different ways. The final predictions are obtained by averaging individual predictions of each CNN. Figure 1 shows the architecture of a MCDNN.

### 3.2 Regularization

Two new approachs for regularizing CNNs have been recently proposed, DropOut [5] and DropConnect [4]. Applying DropOut and DropConnect amounts to subsampling a neural network by dropping units. Since each of these processes acts differently as a way to control overfitting, the combination of several of these networks can bring gains, as will be shown below.

DropOut is applied to the outputs of a fully connected layer where each element of an output layer is kept with probability $p$, otherwise being set to 0 with probability $(1-p)$. If we further assume a neural activation function with $a(0) = 0$, such as $tanh$ and $relu$, the output of a layer can be written as:

$$r = m * a(Wv) \tag{1}$$

where $m$ is a binary mask vector of size $d$ with each element $j$ coming independently from a Bernoulli distribution $m_j \sim Bernoulli(p)$, $W$ is a matrix with weights of a fully-connected layer and $v$ are the fully-connected layer inputs [4].

DropConnect is similar to DropOut, but applied to the weights $W$. The connections are choosen randomly during the training. For a DropConnect layer, the output is given as:

$$r = a((M * W)v) \tag{2}$$

where $M$ is weight binary mask, and $M_{ij} \sim Bernoulli(p)$. Each element of the mask $M$ is drawn independently for each example during training [4]. Figure 2 illustrates the differences between the two methods.

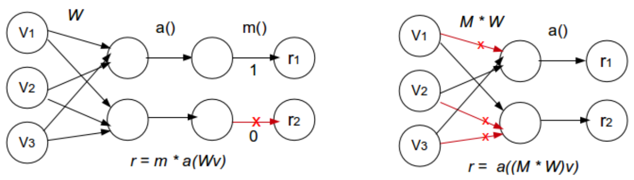


**Fig. 2.** The left figure is an example of DropOut. Right figure is an example of Drop-Connect.

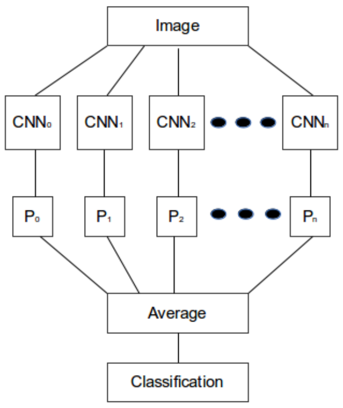


**Fig. 3.** The output probabilities are averaged to make the final prediction.

## 4 Model Description

The standard model architecture to combine networks can be seen in figure 3. Given some input pattern, the output probabilities from all CNN are averaged before making a prediction. For output $i$, the average output $S_i$ is given by:

$$S_i = \frac{1}{n} \sum_{j=1}^{n} r_j(i) \tag{3}$$

where $r_j(i)$ is the output $i$ of network $j$ for a given input patern.

Our approach consists in applying a different weight for each network. In the validation set, networks that had a lower classification error will have a larger weight when combining the results. The model architecture can be seen in figure 4. Given some input pattern, the output probabilities from all CNNs are multiplied by a weight before the prediction:

$$S_i = \sum_{j=1}^{n} W_j r_j(i) \tag{4}$$

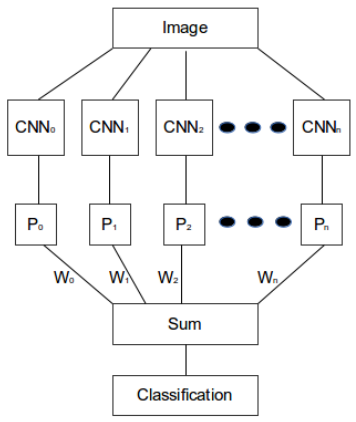


**Fig. 4.** The output probabilities are weighted based on the accuracy of the network evalueted on the validation set.

We use two different approaches to calculate the weight $W$. The first method consists on a weighted mean:

$$W_k = \frac{A_k}{\sum_{i=1}^{n} A_i} \tag{5}$$

where $A_k$ is the accuracy in the validation set for the network $k$, and $i$ runs over the $n$ networks.

In the second method, the weight $W_k$ is choosen by rank. The weights are based on the order of accuracy in the validation set. This means that the weights are fixed, independently on the value of the error:

$$W_k = \frac{R(A_k)}{\sum_{i=1}^{n} R(A_i)} \tag{6}$$

where $R()$ is a function that gives the position of the network based on the validation accuracy sorted in increasing order. For example, the network with largest accuracy will have an $R()$ value of $n$, the network with the second largest accuracy an $R()$ value of $n-1$ and so on until the network with lowest accuracy gets an $R() = 1$.

The first method has weights that are proportional to the accuracy in validation set. If the errors are too close, the weights have values close to each other, and the result will be similar to a simple average.

The second method has the particularity of not looking only at the value of the validation error, but also for the network positions. Even though the difference in error between the two networks might be minimal, the weight value remains fixed, attributing a significantly greater importance to the network that achieved better results in the validation set.

**Table 1.** CIFAR-10 average classification error in percentage and standard deviation using 3 types of networks and 3 types of combiners, using 64 feature maps.

| Model | DropConnect | DropOut | NoDrop |
|---|---|---|---|
| 5 networks | 11.18 ±0.15 | 11.28 ±0.17 | 10.92 ±0.15 |
| Method 1 | 9.81 | 10.45 | 10.06 |
| Method 2 | 9.81 | 10.31 | 10.03 |
| Simple Average | 9.84 | 10.48 | 10.06 |

**Table 2.** CIFAR-10 average classification error in percentage and standard deviation using 3 types of combiners, using 64 feature maps. Previous state-of-the-art using the same architecture is 9.32% [4].

| Model | Method 1 | Method 2 | Simple Average |
|---|---|---|---|
| 12 networks | 9.50 | 9.37 | 9.47 |

In general, both methods give better results than doing just the simple average of the predictions, as will be shown in the results.

## 5 Experiments

Our experiments use a fast GPU based convolutional network library called Cuda-convnet [7] in conjunction with Li's code [5] that allows training networks with DropOut and Dropconnet. We use a NVIDIA TESLA C2075 GPU to run the experiments. For each dataset we train five networks with DropConnect, DropOut and NoDrop (five of each).

Once the networks are trained we save the mean and standard deviation of the classification errors produced individually by each network and the classification error produced by these networks when combined with our two proposed methods methods and simple average. These results are shown in Tables 1-4. We use CIFAR-10 dataset [8] to evaluate our approach.

### 5.1 CIFAR-10

The CIFAR-10 dataset [8] consists of 32 x 32 color images drawn from 10 classes split into 50 000 train and 10 000 test images.

Before feeding these images to our network, we subtract the per-pixel mean computed over the training set from each image as was done in [5]. The images are cropped to 24x24 with horizontal flips.

We use two feature extractors to perform the experiment. The first, consists in 2 convolutional layers, with 64 feature maps in each layer, 2 maxpooling layers, 2 locally connected layers, a fully connected layer which has 128 *relu* units on which NoDrop, DropOut or DropConnect are applied and a output layer with *softmax* units. We train for three stages of epochs, 500-100-100 with

**Table 3.** CIFAR-10 average classification error in percentage and standard deviation using 3 types of networks and 3 types of combiners, using 128 feature maps.

| Model | DropConnect | DropOut | NoDrop |
|---|---|---|---|
| 5 networks | 10.53 ±0.14 | 10.53 ±0.13 | 10.53 ±0.17 |
| Method 1 | 9.77 | 9.68 | 9.63 |
| Method 2 | 9.68 | 9.55 | 9.61 |
| Simple Average | 9.81 | 9.71 | 9.64 |

**Table 4.** CIFAR-10 average classification error in percentage and standard deviation using 3 types of combiners, using 128 feature maps. Previous state-of-the-art using the same architecture is 9.32% [4].

| Model | Method 1 | Method 2 | Simple Average |
|---|---|---|---|
| 12 networks | 9.20 | 9.12 | 9.22 |

an initial learning rate of 0.001, that its reduced by factor 10 between each stage. We chose this fixed number of epochs because it is when the validation error stops improving. Training a network takes around 4 hours. The second feature extractor is similar but with 128 feature maps in each layer and the number of epochs is smaller, 350-100-50. Training a network with 128 maps takes around 20 hours.

In these experiments we compared the results using the three approaches for combining networks described in this paper. The first experiment used a feature extractor with 64 feature maps (summarized in Table 1) and combined networks that were trained with DropConnect, DropOut and NoDrop. NoDrop individually obtained better results and networks with DropOut were the ones with the worst individual results. By combining the nets, DropConnect achieved better results. In addition we combine our 12 best networks (see Table 2) and the results were significantly better than the result shown in Table 1. With method 2 we achieved an error of 9.37%.

The second experiment used a feature extractor with 128 feature maps (summarized in Table 3), we also combine networks that were trained with DropConnect, DropOut and NoDrop. DropOut individually achieved better results, and networks trained with DropConnect were the ones with worst result. When combining our 12 best networks, we obtained better results, as can be seen in Table 4. Using our second method we obtain 9.12%, improving the classification rate of CIFAR-10.

## 6 Conclusions

In this paper, we propose two methods to combine the results of CNNs by applying different weights for each network, instead of simply averaging the output predictions of several networks.

The experiments showed, that applying a fixed weight to each network (method 2), ensures better results than the other two approaches. We also conclude that when combining a large number of networks, the weighted mean (method 1) performance was similar to the simple average. We also concluded that a feature extractor 128 feature maps achieves better results than a feature extractor with 64 feature maps. However when using 128 maps we have to use less epochs because the network tends to overfitt, losing its ability to generalize. We also tested more maps (not presented in results), but the networks overfitted very quick, producing worst results. At the end, with one of our methods, we improved the classification rate of CIFAR-10.

In summary, we demonstrate with our contribution that there are better alternatives to combine the results than doing just the simple average of the networks predictions.

## References

1. Kunihiko Fukushima, "A neural network model for selective attention in visual pattern recognition," *Biol. Cybern.*, vol. 55, no. 1, pp. 5–16, Oct. 1986.
2. K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?," in *Computer Vision, 2009 IEEE 12th International Conference on*, Sept 2009, pp. 2146–2153.
3. Kumar Chellapilla, Sidd Puri, and Patrice Simard, "High Performance Convolutional Neural Networks for Document Processing," in *Tenth International Workshop on Frontiers in Handwriting Recognition*, Guy Lorette, Ed., La Baule (France), Oct. 2006, Université de Rennes 1, Suvisoft, http://www.suvisoft.com Université de Rennes 1.
4. Li Wan, Matthew Zeiler, Sixin Zhang, Yann L. Cun, and Rob Fergus, "Regularization of neural networks using dropconnect," in *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, Sanjoy Dasgupta and David Mcallester, Eds. May 2013, vol. 28, pp. 1058–1066, JMLR Workshop and Conference Proceedings.
5. Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *CoRR*, vol. abs/1207.0580, 2012.
6. Jurgen Schmidhuber, "Multi-column deep neural networks for image classification," in *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Washington, DC, USA, 2012, CVPR '12, pp. 3642–3649, IEEE Computer Society.
7. Alex Krizhevsky, "Cuda-convnet," `http://code.google.com/p/cuda-convnet/`, 2012.
8. Alex Krizhevsky, "Learning multiple layers of features from tiny images," Tech. Rep., 2009.

# DropAll: Generalization of Two Convolutional Neural Network Regularization Methods

Xavier Frazão and Luís A. Alexandre

Dept. of Informatics, Univ. Beira Interior
Covilhã, Portugal
`xavierfrazao@gmail.com`
`lfbaa@ubi.pt`
`http://www.ubi.pt`

**Abstract.** We introduce DropAll, a generalization of DropOut [1] and DropConnect [2], for regularization of fully-connected layers within convolutional neural networks. Applying these methods amounts to subsampling a neural network by dropping units. Training with DropOut, a randomly selected subset of activations are dropped, when training with DropConnect we drop a randomly subsets of weights. With DropAll we can perform both methods. We show the validity of our proposal by improving the classification error of networks trained with DropOut and DropConnect, on a common image classification dataset. To improve the classification, we also used a new method for combining networks, which was proposed in [3].

## 1 Introduction

Convolutional neural networks (CNNs) are hierarchical neural networks whose convolutional layers alternate with subsampling layers, reminiscent of simple and complex cells in the primary visual cortex [4]. Although these networks are efficient when performing classification, they have the disadvantage of being computationally heavy, which makes their training slow and cumbersome.

With the emergence of parallel programming and taking advantage of the processing power of Graphics Processing Units (GPUs), training these networks takes significantly less time, making it possible to train large networks [5, 6] and also making it possible to train multiple networks for the same problem and combine their results [2, 1], an approach that can significantly increase the classification accuracy.

Besides the training time, the major problem of these networks is the overfitting. Overfitting still remains a challenge to overcome when it comes to training extremely large neural networks or working in domains which offer very small amounts of data. Many regularization methods have been proposed to prevent this problem. These methods combined with large datasets have made it possible to apply large neural networks for solving machine learning problems in several domains. Two new approachs have been recently proposed: DropOut [1] and DropConnect [2], which is a generalization of the previous. When training with DropOut, a randomly selected subset of activations is droped. With DropConnect, we randomly drop the weights. Both techniques are only possible for fully connected layers.

In this paper, we propose a generalization of both methods named DropAll. With this approach we were able to train a network with DropOut, DropConnect or both and taking advantage of each method.

## 2 Convolutional Neural Networks

A classical convolutional network is composed of alternating layers of convolution and pooling. The purpose of the first convolutional layer is to extract patterns found within local regions of the input images. This is done by convolving filters over the input image, computing the inner product of the filter at every location in the image and outputting the result as feature maps $c$. A non-linear function $f()$ is then applied to each feature map $c : a = f(c)$. The resulting activations $a$ are passed to the pooling/subsampling layers. These layers aggregate the information within a set of small local regions, $\{R_j\}_{j=1}^{n}$, producing a pooled feature map $s$ of smaller size as output.

Representing the aggregation function as $pool()$, then for each feature map $c$, we have: $s_j = pool(f(c_i))\ \forall_i \in R_j$.

The two common choices to perform $pool()$ are average and max-pooling. The first takes the arithmetic mean of the elements in each pooling region, while max-pooling selects the largest element of the pooling region.

A range of functions $f()$ can be used as a non-linearity – *tanh*, *logistic*, $softmax$ and *relu* are the most common choices.

In a convolutional network model, the convolutional layers, which take the pooled maps as input, can thus extract features that are increasingly invariant to local transformations of the input image.

The last layer is always a fully connected layer with one output unit per class in the recognition task. The activation function $softmax$, is the most common choice for the last layer such that each neuron output can be interpreted as the probability of a particular input image belonging to that class.

## 3 Related Work

### 3.1 Ensembles of CNNs

Model combination improves the performance of machine learning models. Averaging the predictions of several models is most helpful when the individual models are different from each other, in other words, to make them different they must have different hyperparameters or be trained on different data.

The standard model architecture to combine networks can be seen in figure 1. Given some input pattern, the output probabilities from all CNN are averaged before making a prediction. For output $i$, the average output $S_i$ is given by:

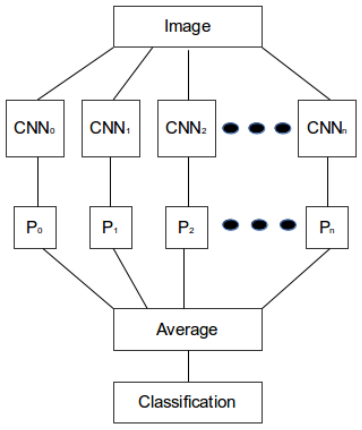


**Fig. 1.** The output probabilities are averaged to make the final prediction.

$$S_i = \frac{1}{n}\sum_{j=1}^{n} r_j(i) \tag{1}$$

where $r_j(i)$ is the output $i$ of network $j$ for a given input patern.

We recently proposed a new approach to combining neural networks called Weighted Convolutional Neural Network Ensemble (WCNNE)[3] that presented better results than doing just the simple average of the predictions. This method consists in applying a different weight for each network. Networks that had a lower classification error in the validation set, will have a larger weight when combining the results. The model architecture can be seen in figure 2. Given some input pattern, the output probabilities from all CNNs are multiplied by a weight before the prediction:

$$S_i = \sum_{j=1}^{n} W_j r_j(i) \tag{2}$$

The weights $W_k$ is choosen by rank and are based on the order of accuracy in the validation set. This means that the weights are fixed, independently on the value of the error:

$$W_k = \frac{R(A_k)}{\sum_{i=1}^{n} R(A_i)} \tag{3}$$

where $R()$ is a function that gives the position of the network based on the validation accuracy sorted in increasing order. For example, the network with largest accuracy will have an $R()$ value of $n$, the network with the second largest accuracy an $R()$ value of $n-1$ and so on until the network with lowest accuracy gets an $R() = 1$.

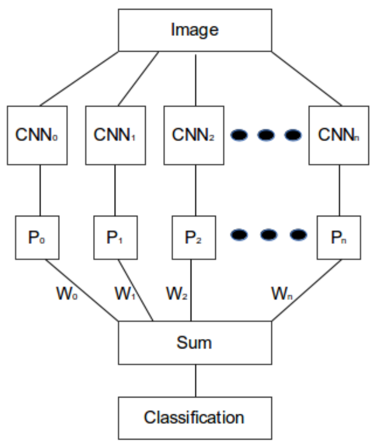


**Fig. 2.** The output probabilities are weighted based on the accuracy of the network evalueted on the validation set.

This method has the particularity of not looking only at the value of the validation error, but also for the network positions in terms of the ranked error list. Even though the difference in error between the two networks might be minimal, the weight value remains fixed, attributing a significantly greater importance to the network that achieved better results in the validation set.

## 3.2 Regularization

Two approachs for regularizing CNNs have been recently proposed, DropOut [1] and DropConnect [2]. Applying DropOut and DropConnect amounts to subsampling a neural network by dropping units. Since each of these processes acts differently as a way to control overfitting, the combination of several of these networks can bring gains, as will be shown below.

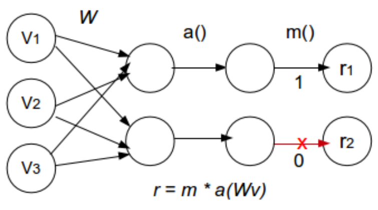
DropOut is applied to the outputs of a fully connected layer where each element of an output layer is kept with probability $p$, otherwise being set to 0 with probability $(1-p)$. If we further assume a neural activation function with $a(0)=0$, such as $tanh$ and $relu$, the output of a layer can be written as:

$$r = m * a(Wv) \tag{4}$$

where $m$ is a binary mask vector of size $d$ with each element $j$ coming independently from a Bernoulli distribution $m_j \sim Bernoulli(p)$, $W$ is a matrix with weights of a fully-connected layer and $v$ are the fully-connected layer inputs [2].

DropConnect is similar to DropOut, but applied to the weights $W$. The connections are choosen randomly during the training. For a DropConnect layer, the output is given as:

$$r = a((M * W)v) \tag{5}$$



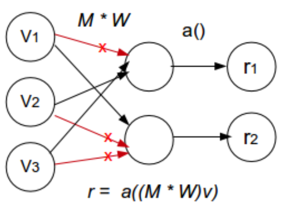


**Fig. 3.** The left figure is an example of DropOut. Right figure is an example of DropConnect.

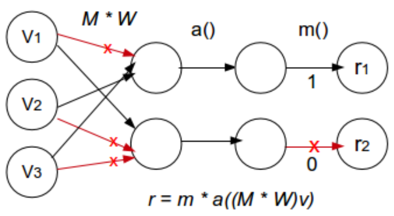


**Fig. 4.** Example of DropAll model.

where $M$ is weight binary mask, and $M_{ij} \sim Bernoulli(p)$. Each element of the mask $M$ is drawn independently for each example during training [2]. Figure 3 illustrates the differences between the two methods.

## 4 DropAll

DropAll is a generalization of DropOut [1] and DropConnect [2], for regularizing fully-connected layers within neural deep networks. In the previous section we saw that DropOut is described by equation 4 and DropConnect is described by equation 5. For a DropAll layer, the output is given as:

$$r = m * a((M * W)v) \tag{6}$$

The DropAll model is presented graphically in figure 4. This approach has the particularity of being easily adaptable to one of the previous methods. In these two methods, we had only one variable where we choose the percentage of drops, with DropAll we have 2 variables. One variable controls the drop rate of the activation while the other variable controls the drop rate of the weight. If we set one of these variables to zero, we obtain either DropOut or DropConnect.

In both methods the value of the drop rate used is usually 0.5, however if we train with DropAll with 0.5 in both rates, network discards a lot of information, which is reflected in the results. To solve this problem, the drop rate must be smaller for both variables. When testing the network with different drop rates, we concluded that 0.25 is a good compromise.

**Table 1.** CIFAR-10 average classification error in percentage and standard deviation using 4 types of networks and 2 types of combiners, using 64 feature maps.

| Model | DropAll | DropConnect | DropOut | NoDrop |
|---|---|---|---|---|
| 5 networks | 11.20 ±0.10 | 11.18 ±0.15 | 11.28 ±0.17 | 10.92 ±0.15 |
| WCNNE | 10.01 | 9.81 | 10.31 | 10.03 |
| Simple Average | 10.03 | 9.84 | 10.48 | 10.06 |

**Table 2.** CIFAR-10 average classification error in percentage and standard deviation using 4 types of networks and 2 types of combiners, using, using 128 feature maps.

| Model | DropAll | DropConnect | DropOut | NoDrop |
|---|---|---|---|---|
| 5 networks | 10.67 ±0.11 | 10.53 ±0.14 | 10.53 ±0.13 | 10.53 ±0.17 |
| WCNNE | 9.57 | 9.68 | 9.55 | 9.61 |
| Simple Average | 9.62 | 9.81 | 9.71 | 9.64 |

In the following section we compare DropAll with DropConnect, DropAll and NoDrop (trained network without dropping units). All of these methods used in conjunction, provide a greater randomness when tested and combining the results from different networks trained by these techniques significantly improves the classification rates.

## 5 Experiments

Our experiments use a fast GPU based convolutional network library called Cuda-convnet [7] in conjunction with Li's code [1] that allows training networks with DropOut, Dropconnet and DropAll. We use a NVIDIA TESLA C2075 GPU to run the experiments. For each dataset we train five networks with DropAll, DropConnect, DropOut and NoDrop (five of each).

Once the networks are trained we save the mean and standard deviation of the classification errors produced individually by each network and the classification error produced by these networks when combined with our proposed method [3] and simple average. These results are shown in Tables 1-3. We used the CIFAR-10 dataset [8] to evaluate our approach.

### 5.1 CIFAR-10

The CIFAR-10 dataset [8] consists of 32 x 32 color images drawn from 10 classes split into 50 000 train and 10 000 test images.

Before feeding these images to our network, we subtract the per-pixel mean computed over the training set from each image as was done in [1]. The images are cropped to 24x24 with horizontal flips.

We use two feature extractors to perform the experiment. The first, consists in 2 convolutional layers, with 64 feature maps in each layer, 2 maxpooling

**Table 3.** CIFAR-10 average classification error combining our 12 best networks using 2 types of combiners, using 128 feature maps. Previous state-of-the-art using the same architecture is 9.32% [2]. Current state-of-the art of CIFAR-10 is 8.81% [9].

| Model | WCNNE | Simple Average |
|---|---|---|
| 12 networks | 9.09 | 9.22 |

layers, 2 locally connected layers, a fully connected layer which has 128 *relu* units on which NoDrop, DropOut, DropConnect or DropAll are applied and a output layer with *softmax* units. We train for three stages of epochs, 500-100-100 with an initial learning rate of 0.001, that its reduced by factor 10 between each stage. We chose this fixed number of epochs because it is when the validation error stops improving. Training a network takes around 4 hours. The second feature extractor is similar but with 128 feature maps in each layer and the number of epochs is smaller, 350-100-50. Training a network with 128 maps takes around 20 hours. In these experiments we compared the results using our approach (WCNNE) [3] for combining networks and simple average, both described in this paper.

The first experiment used a feature extractor with 64 feature maps (summarized in Table 1) and combined networks that were trained with DropAll, DropConnect, DropOut and NoDrop. NoDrop individually obtained better results and networks with DropOut were the ones with the worst individual results. By combining the nets, DropConnect achieved better results.

The second experiment used a feature extractor with 128 feature maps (summarized in Table 2), we also combine networks that were trained with DropAll, DropConnect, DropOut and NoDrop. DropOut individually achieved better results, and networks trained with DropAll were the ones with worst result. By combining the nets, DropOut achieved better results.

In addition we join all models and combine our 12 best networks with lowest validation error, and the results were significantly better (see Table 3). All of these methods used in conjunction provide a greater randomness and significantly improve the classification rate. If we combine our 12 best networks without DropAll networks the error is slightly worse, 9.12%.

## 6 Conclusions

In this paper, we propose a new method named DropAll that is a generalization of two well-known methods for regularization of convolutional neural networks, used to avoid overfitting. These problem still remains a challenge to overcome when it comes to training extremely large neural networks or working in domains which offer very small amounts of data.

DropAll by itself, did not increase performance when we evaluate a network, however, the flexibility of this method makes it possible to train a network using the potential of DropOut and DropConnect. In general, networks trained with these forms of regularization benefit from an increase randomness, which is a plus when we wish to combine the results of multiple networks. As shown, the combination of all methods significantly improves the classification rate of the problem used in the experiments section to validate our proposal.

## References

1. Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov, "Improving neural networks by preventing co-adaptation of feature detectors," *CoRR*, vol. abs/1207.0580, 2012.
2. Li Wan, Matthew Zeiler, Sixin Zhang, Yann L. Cun, and Rob Fergus, "Regularization of neural networks using dropconnect," in *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, Sanjoy Dasgupta and David Mcallester, Eds. May 2013, vol. 28, pp. 1058–1066, JMLR Workshop and Conference Proceedings.
3. Xavier Frazao and Luís A. Alexandre, "Weighted convolutional neural network ensemble," in *submitted*, 2014.
4. Kunihiko Fukushima, "A neural network model for selective attention in visual pattern recognition," *Biol. Cybern.*, vol. 55, no. 1, pp. 5–16, Oct. 1986.
5. K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?," in *Computer Vision, 2009 IEEE 12th International Conference on*, Sept 2009, pp. 2146–2153.
6. Kumar Chellapilla, Sidd Puri, and Patrice Simard, "High Performance Convolutional Neural Networks for Document Processing," in *Tenth International Workshop on Frontiers in Handwriting Recognition*, Guy Lorette, Ed., La Baule (France), Oct. 2006, Université de Rennes 1, Suvisoft, http://www.suvisoft.com Université de Rennes 1.
7. Alex Krizhevsky, "Cuda-convnet," `http://code.google.com/p/cuda-convnet/`, 2012.
8. Alex Krizhevsky, "Learning multiple layers of features from tiny images," Tech. Rep., 2009.
9. Min Lin, Qiang Chen, and Shuicheng Yan, "Network in network," *CoRR*, vol. abs/1312.4400, 2013.