KU LEUVEN

Reducing Redundancy in Deep Convolutional Neural Networks

Pau Rodríguez López

Thesis submitted for the degree of Master of Science in Artificial Intelligence, option Engineering and Computer Science

Thesis supervisor:
Prof. Bart de Moor

Assessor:
Peter Roelants

Mentor:
Peter Roelants

Academic year 2014 – 2015

Preface

I would like to thank Peter Roelants and professor Bart de Moor for giving me the freedom to work on my own ideas as well as giving useful advise to work on them. I'd also like to thank the STADIUS group for letting me access to their server and NVIDIA for providing a GPU for free. My gratitude to Aida and all my family, who gave me moral support from the distance.

Pau Rodríguez López

Contents

Abstract

After the breakthrough of Alex Krizhevsky's convolutional neural network for object recognition (CNN) in 2012 [17], there has been a lot of emphasis in finding a model that has the same or better performance using less parameters in order to reduce the need of huge computational resources. This has been tackled from the point of view of the network filters structure in ILSVRC2013 by Mathew Zeiler (2013) [45], and also building very deep models with very small convolutional kernels in ILSVRC2014 (Szegedy *et al.* , 2014) [37] (Simonyan and Zisserman, 2014) [33]. However, the problem of the redundancy in the parameters of CNNs is not directly studied in any of the mentioned papers and nowadays, it is important that our models are as efficient as possible in order to adapt to the trend of mobile computing. Thus, this thesis focuses on the problem of finding the source of redundancy in CNNs and proposes some methods to visualize and measure it, as well to reduce it. These methods will be based on the filter weights since it is much more efficient than performing a statistical analysis over the filter activations and will result in a framework for filter analysis, pruning, and regularization. The researched techniques will finally be tried for pruning over Imagenet, showing that it is possible and opening the door for trying it over the state-of-the-art models. Regularization techniques have also been tried over Cifar10 and Imagenet, showing that Cifar10 is not complex enough to be regularized but obtaining more promising results on a pre-learnt Imagenet model present in the Caffe framework [14], which encourage to work on their verification and to do further research.

List of Figures and Tables

List of Figures

List of Tables

List of Abbreviations and Symbols

Abbreviations

ANN	Artificial Neural Network
CNN	Convolutional Neural Network
DNN	Deep Neural Network
MSE	Mean Square error
SOM	Self Organising Map
IMED	Image Euclidean Distance

Chapter 1

Introduction

Convolutional Neural Networks (CNNs) are becoming one the most popular techniques for image classification nowadays, not because it is a new technique, but we have the computational resources and big enough databases to use them successfully. An example for such big databases is Imagenet [30], with more than 1M of images. However, although there are new and more powerful hardware and improved algorithms, training a winner network for this challenge can take a lot of time. For example, it lasted six days to train the neural network which won the ILSVRC2012 [17]. Hence, computational resources are still an issue on this learning technique and one of the causes is the huge number of parameters of the network. Moreover, it was clearly shown by Mathew Zeiler *et al.* (Clarifai) in [45] that a lot of these parameters (concretely the filter weights) were highly redundant using as example the learned filters from the former ILSVRC2012 competition [17]. Observing the filters of the winner CNN in figure 1.1 on the following page, it is clear that some of them are really similar and others do not extract any important feature at all. Thus, Clarifai team proposed to solve the problem implicitly by reducing the convolution kernel size and the stride of the kernel. Effectively, their solution was able to increase the amount of useful information captured by the filters and it improved the result with respect to Krizhevsky's architecture, see figure 1.1 on the next page.

This information raises the question if there is still margin to reduce this redundancy to improve performance. The latest results of the ILSVRC2014 [30] show it is possible and thus, the aim of this Masters Thesis is to try to answer this question through the study the redundancy in CNNs as well as methods to reduce it in order to use the existent parameters as efficiently as possible and, therefore, take the maximum advantage of the hardware. In fact, this is also following the trend of adapting the software to the mobile computing requirements. Small devices with less capacity than desktop computers, such as smartphones and tablets with their own low-end GPUs, are colonizing the market. Thus, it is crucial that AI algorithms can adapt to them in order to be ubiquitous.

To tackle this problem, a literature review will be presented in order to take into account the work that has already been done in this field, as well as some metrics that could be useful to assess this redundancy further in this thesis. Next, the concrete

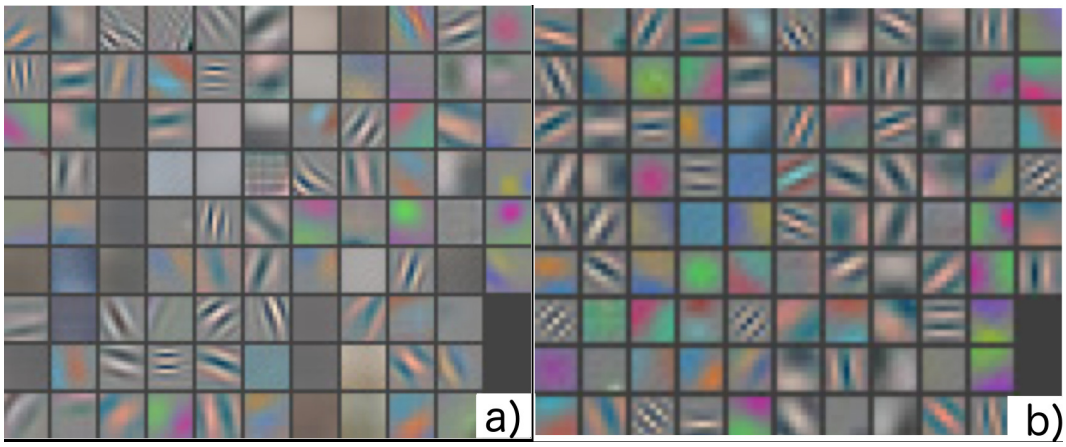


Figure 1.1: a) represents the learned weights in the first layer of Krizhevsky's architecture in ILSVRC2012 while b) is from Zeiler's team architecture in ILSVRC2013. It is possible to appreciate how the first architecture has more "dead" filters than the second one.

methodology for trying to solve it will be presented. This will consist in a series of filter weight visualization techniques, a metric to assess their spatial distribution and an algorithm for pruning at the filter level. Then some algorithms based on similarity metrics will also be provided in order to specialize the filters better with the aim of creating a better model instead of a smaller one. All this will be finally presented in the form of a framework to analyze and to regularize and prune the weight filters of CNNs. These algorithms have been prototyped in python, but the most feasible ones have been implemented on the Caffe CNN framework [14] using the NVIDIA CUDA parallel platform.

Finally, the results of applying filter pruning over an Imagenet model will be presented. They show that it is indeed possible to reduce the number of network filters without losing performance. On the other hand, the results of applying the filter specialization techniques over the Cifar10 and Imagenet datasets are also shown. For the first dataset, no improvements are observed but it seems possible that they have some effect over the latter. Namely, it seems that it could be moving the filters to a better close local minimum. However, more evidence will be needed and trials over Imagenet are very expensive. Thus, further research out of the time scope for this project can be done in order to verify and improve these results.

Chapter 2

Literature Review

This thesis is almost entirely based on the Deep Neural Networks and, especially, Convolutional Neural Networks. Thus, is important to understand these techniques in order to follow this text.

In a nutshell, an Artificial Neural Network is a bioinspired machine learning technique which simulates the interactions between layers of neurons coding the information in the strength of the connections between them. In fact, the first ANN dates from 1958 and was called Perceptron, Rosenblatt (1958) [27], see figure 2.1. However, Minsky and Papert (1969) [21] showed it is a weak method since it cannot approximate the XOR function.

After the XOR critics, ANNs lost their popularity until 1986, after the XOR problem was solved through the use of multiple layers of neurons. To do this, a new training algorithm called backpropagation was developed [4][42][28]. When an ANN has more than one hidden layer, it is called Deep Neural Network (DNN). CNNs are a variant of DNNs that were developed by Yann LeCun *et al.* on 1998 [18] based on the work of Kunihiko Fukushima (1980) [10]. This kind of DNN supposed an advance

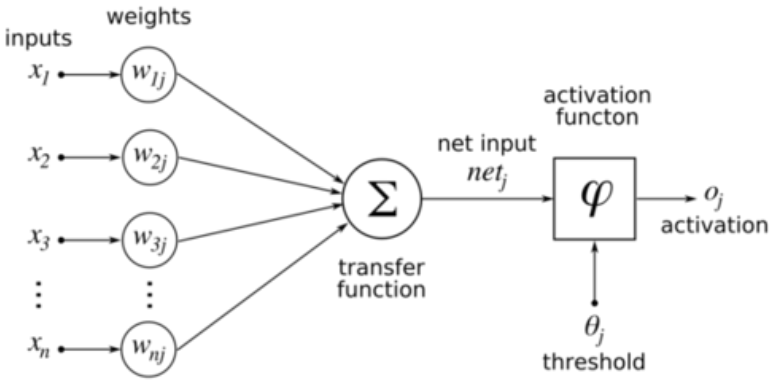


Figure 2.1: Scheme of a perceptron (http://nl.wikipedia.org/wiki/Perceptron). As it can be seen it tries to simulate a neuron which receives input through its dendrites and, is the stimulus surpasses a threshold, it send an output through the axon.

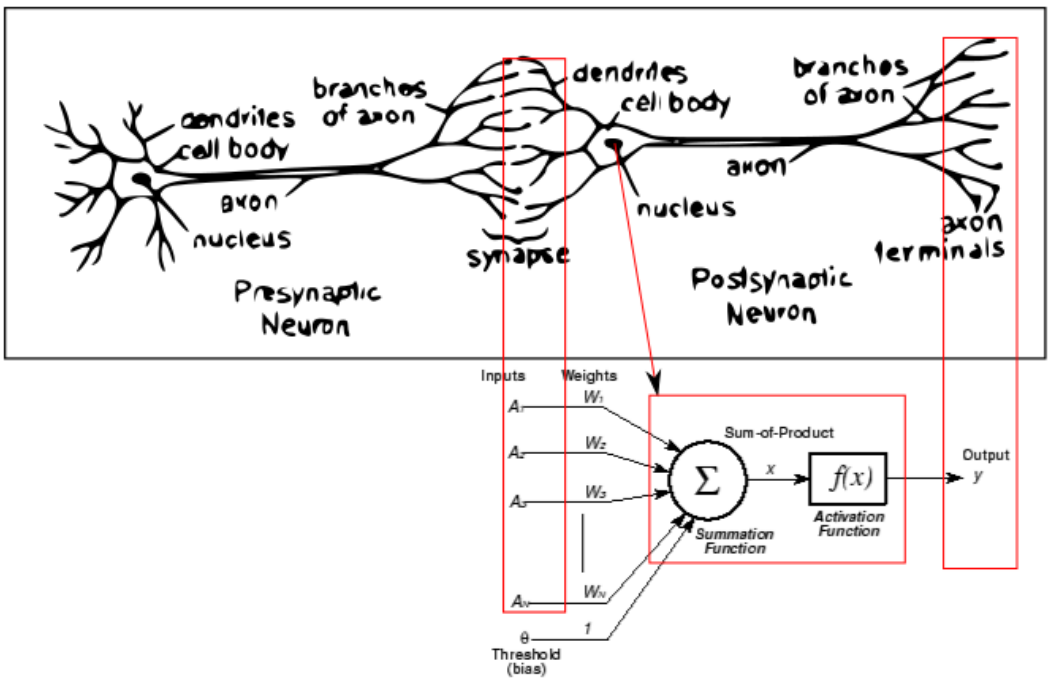


Figure 2.2: Schematic comparison between biological neurons (above) and artificial neurons (below).

in image recognition as it takes into account the spatial information in images. Next, the concept of Neuron, DNN and CNN will be explained.

2.1 The Neuron Model

As seen before, the neurons used in ANN are inspired by biological neurons, see figure 2.2.

So, using the notation of the figure 2.2, an artificial neural network is a device which receives N numeric inputs $[A_1 \dots A_N]$ and a bias term b and computes $h_{W,b}(A) = f(W^t A)$ where f is called the activation function, which is an approximation to the Heaviside step function such as the sigmoid:

$$f(z) = \frac{1}{1 + e^{-z}}$$

or the hyperbolic tangent:

$$f(z) = tanh(x) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$$

Although it is not possible to compute the XOR function, with a neuron we can start learning some logic gates such as AND or OR. For example, an AND gate can be created if there are two inputs with weight 0.5, a bias with weight -0.9 and a

x_1	x_2	$W \cdot X + b$	$f(W \cdot X + b)$
0	0	−0.9	0
0	1	−0.4	0
1	0	−0.4	0
1	1	0.1	1

Table 2.1: Example of AND gate using a neuron.

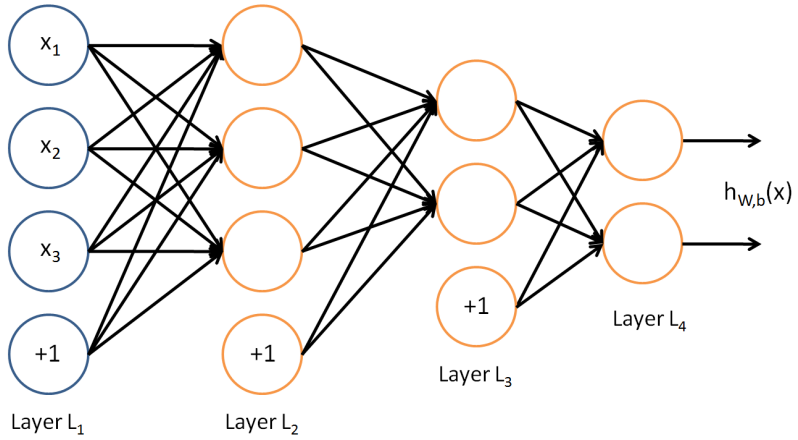


Figure 2.3: A 4-layer DNN scheme. The +1 are the bias terms.
(http://ufldl.stanford.edu/wiki/index.php/Neural_Networks)

threshold function $f(x)$, which outputs 1 when $x > 0$ and 0 when $x <= 0$. The resulting truth table would be the one on the table 2.1.

If more than one neuron is used (without connections between them), the model will be called an ANN and the set of parallel neurons will be called layer. These layers can be stacked so that the output of one layer is the input of the next one. When there are more than three layers, the convention is to say that the layers between the input and the output layers are *hidden layers*. If more than one hidden layers are present, the convention is that the ANN will be called DNN. This last model together with its training algorithm will be explained below.

2.2 Deep Artificial Neural Networks

Given the description above, a DNN can be represented with the graphical model of figure 2.3.

To obtain the output given an input, The process to obtain the output given an input is layer-wise. Namely, one has to compute first the outputs of the first layer and use them as input for the second one and so on until reaching the output layer. This is called forward propagation. The formulas for the forward propagation are the following:

$$z^{(l+1)} = W^{(l)}a^{(l)} + b^{(l)}$$

Where z is the vector of the weighted sum of the inputs of each neuron, $W^{(l)}$ is the weight vector between the current layer and the previous one, $a^{(l)}$ is the output vector of the previous layer and $b^{(l)}$ is the vector containing the bias terms. After that, the activation function has to be applied to calculate the output of the layer:

$$a^{(l+1)} = f(z^{(l+1)})$$

Where $a^{(l+1)}$ is the final output of the current layer; except for the last layer which is $h_{W,b}(X)$.

To learn the parameters of the network (W, and b), the backpropagation algorithm has to be applied. To apply the algorithm, a forward step has to be computed in order to compare the current output h with the desired one d and propagate the error until the first layer. For cross-entropy and MSE, this is:

$$\delta^{(n)} = -(d - h) \cdot f'(z^{(n)})$$

Where n means the last layer. For the other layers:

$$\delta^{(l)} = ((W^{(l)})^T \delta^{(l+1)}) \cdot f'(z^{(n)})$$

If J is the cost function, the partial derivatives used to update the weights to descend it are:

$$\Delta_{W^{(l)}} J(W, b; x, y) = \delta^{(l+1)}(a^{(l)})^T$$

$$\Delta_{b^{(l)}} J(W, b; x, y) = \delta^{(l+1)}$$

If the learning is doing with batches of examples, the average of the gradients of all the examples will be used to update the weights. Finally, the update step only consists in subtracting the gradient from the weights for each layer:

$$W^{(l)} = W^{(l)} - \alpha(\Delta W^{(l)} + \lambda W^{(l)})$$

$$b^{(l)} = b^{(l)} - \alpha \Delta b^{(l)}$$

Where α is called the *learning rate*, and regulates the size of the steps in the gradient descent. λ is the regularization term called *weight decay* which prevents overfitting and decreases the magnitude of the weights.

DNNs have been proved to work well in multiple fields such as character recognition, Image compression, Robotics, etc. But their performance in Computer Vision, for example, object recognition is not so good because they don't give much invariability and are prone to overfitting. Therefore, CNNs were created to overcome this problem as they take into account the spatial structure of images and the fact that the same pattern can be detected in multiple patches of a given image.

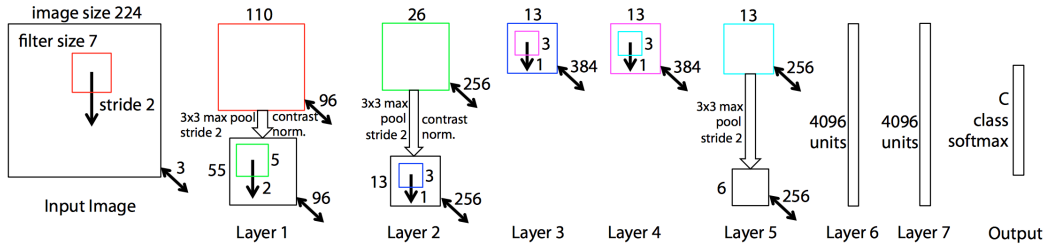


Figure 2.4: 8 layers CNN. This is the model used by Zeiler et al. *at the ILSVRC2014 to win the competition [45].*

2.3 Convolutional Neural Networks

Figure 2.4 is an example of CNN. As it can be seen, a set of weights are arranged as a 7×7 filter or window. These weights are used in different parts of the image as the window slides, providing one output per position as they were the weights of a neuron with 49 inputs for each position. Thus, these weights will be shared among all the neurons corresponding to all the window positions. Normally, multiple $m \times m$ filters are used so that each one specializes to detect a particular feature in the whole image. After each convolution layer, there is usually a pooling layer. Pooling layers also use a window to extract the maximum (normally) value from all the positions where it is on. This helps to reduce the amount of information because normally it slides without overlapping and to provide some invariance because it will not matter where in the window the activation is. These two kind of layers keep alternating merging lower level features into higher level features until the information is decoded using conventional neuron layers (fully connected) to achieve a classification. In fact, an additional layer called softmax is used in order to convert the outputs of the last fully connected layer into a probability distribution. The softmax function is the following:

$$P(Y = i|x, W, b) = \frac{e^{W_i x + b_i}}{\sum_j e^{W_j x + b_j}}$$

Normally the weights and biases referenced previously are initialized randomly for symmetry breaking. The convolution filters are no exception and normally multiple filters for the same layer are randomly initialized. Over the learning iterations, each filter specializes to detect a concrete feature but sometimes a group of filters converges into similar or the same feature, see figure 1.1 on page 2.

2.4 Information Theory

Neural networks process information. Moreover, we are interested in how we can maximize the information captured by the neurons of the network. This makes important to have a good metric to estimate how much information is being captured.

As this problem has been already addressed by Shannon's information theory [31] (1948), the **mutual information**, see equation (2.1), will be the metric used in this MSc Thesis.

$$I(X;Y) = \sum_{x,y} p(x,y) \log \frac{p(x,y)}{p(x)p(y)} \tag{2.1}$$

This equation expresses how much information gives the X distribution given the distribution Y in bits (if the log_2 has been used). So, if both distributions give the same information $I(X;Y) = 0$. However, if they are completely different, $I(X;Y) = H(X)$ where $H(X)$ is the **entropy** (in bits) of the X distribution. The entropy, as said by Thomas M. Cover and Thomas Joy (1991) [6], is *a measure of uncertainty of a random variable*. It is expressed by the equation (2.2) and it is 0 bits when there is no uncertainty ($p(X = x_i) = 1$) or 1 bit when it is random ($p(X = x_i) = \frac{1}{\#i}$).

$$H(X) = -\sum_{x \in X} p(x) \log(p(x)) \tag{2.2}$$

However, we are interested in the case where there is more than one distribution as we might want to know the entropy of a group of neurons. For this we can use the **joint entropy**:

$$H(X) = -\sum_{x \in X} \sum_{y \in Y} p(x,y) \log(p(x,y)) \tag{2.3}$$

When the joint entropy is maximum it guarantees that neurons are uncorrelated as all the states of the neurons are equally probable but a more accurate measure is the conditional entropy:

$$H(Y|X) = -\sum_{x \in X} p(x) \log(p(Y|X = x)) \tag{2.4}$$

When the conditional entropy is 0 bits, it means that the distribution of the activations of the neuron Y completely depends on the distribution X, so there is redundancy. The opposite case is when the conditional entropy is 1. However, this measure is neuron-wise. If a global measure is wanted we can use the above explained mutual information although its formula is usually seen as in the equation (2.5).

$$I(X;Y) = I(Y;X) = H(X) - H(X|Y) = H(X) + H(Y) - H(X,Y) \tag{2.5}$$

The mentioned book [6], gives a bivariate formula like the one above but there is no multivariate formula specified for mutual information. In fact, it is a current topic of research. However, the two most accepted formulas [43] [38] [25] [34] are the **total correlation** [41] and **interaction information** [20]. However, Williams *et al.* (2010) [43], claim that both formulas are problematic because (i) total correlation does not provide any information about the structure of the mutual information, so we are blind to the possible dependencies between variables; and (ii) interaction

information can be negative and sometimes confuses redundancy and *synergy*. That's why they decomposed mutual information and found a general formula but it is not scalable at all. In addition, there is a third metric proposed by Han [12] (1978), called **dual total correlation**, that differs from total correlation by the upper boundary, which is the joint entropy instead of the sum of the entropies. This metric is said to be more correct than the total correlation.

In detail, the formula for the total correlation is:

$$C(X_1, X_2, \ldots, Xi_n) = \sum_{x_1 \in X_1} \sum_{x_2 \in X_2} \cdots \sum_{x_n \in X_n} p(x_1, x_2, \ldots, x_n) \log \frac{p(x_1, x_2, \ldots, x_n)}{p(x_1)p(x_2)\ldots p(x_n)} \tag{2.6}$$

This formula calculates the redundancy or dependency in a set of n random variables.

The formula for the dual total correlation is:

$$D(X_1, X_2, \ldots, X_n) = H(X_1, X_2, X_n) - \sum_{i=1}^{n} H(X_i | X_1, X_2, \ldots, X_n) \tag{2.7}$$

Finally, the formula for the interaction information is the following:

$$I(X_1; X_2; \ldots, X_n) = I(V) = \sum_{k=1}^{N} (-1)^{k-1} \sum_{X \subset \{X_1, X_2, \ldots, X_n\}, |X|=k} H(X) \tag{2.8}$$

2.5 Redundancy in Neural Networks

As explained before, one of the problems of the CNN used for ILSVRC2012 seemed to be that its size was limited by the maximum hardware capacity as it had so many parameters [17]:

> *A single GTX 580 GPU has only 3GB of memory, which limits the maximum size of the networks that can be trained on it.*

However, it has been also said that Matthew D. Zeiler *et al.* [45] noticed that the filters of [17] were not as good as they could be because (i) the first layer was capturing high and low frequencies but not the middle ones, (ii) the stride of the filter convolutions was too big and caused aliasing, (iii) there were "dead" filters that simply represented a uniform background. The solution proposed was to modify Krizhevsky's architecture so that the filters at the first layer shrank from 11×11 to 7×7 and the stride from 4 to 2. More filters were added as well and the final number of parameters was $65M$.

The latest results from the ILSVRC2014 show how GoogLeNet team [37] has been able to improve the detection results of ILSVRC2013 using much fewer parameters and thus getting better computational performance than its predecessor. The same

happens with the NUS team, matching [17] with only **7.5M** parameters and not using fully connected layers.

This shows that learning properly the given parameters is better than expecting that using a lot of them randomly initialized, all the important information will be automatically captured. Given this, a good way to know that the parameters are capturing as much information as possible is to try to reduce their redundancy. This means that a group of parameters can't be capturing the same information but all different features. A good example of redundancy reduction is the already explained difference between the filters of the CNN winner of the ILSVRC2012 [17] and ILSVRC2013 [45],

In fact, the concept of reducing redundancy of the parameters is related to the concept of having low-correlated neurons, which leads to the concept of sparsity. This last property is very important because that is how neuroscientists define our brain works [2]. Although these two concepts are related, they are not the same. For instance, it is possible to have decorrelated neurons which are not activated sparsely. To merge both concepts, code theory must be used as we want the shortest codes (neuron activation patterns) which contain the maximum information about the source (input data), see Cover and Thomas (1991) [6]. There is a lot of literature trying to link Information Theory and Artificial Neural Networks but the computational complexity of Information Theory methods makes it difficult to apply it to big networks. However, it can still be used as a metric to compare learning procedures. In fact, this has already been tried for sparsity using, for instance, the KL divergence between neurons as a regularization term [3][23]. In [11][23] it is also stated the improvement of sparsity due to the use of Rectifier Linear Units and L1 normalization. These two methods were shown to be really important as the ReLU units force absolute 0s in neurons and L1 normalization promotes sparsity but also prevents unbounded activations. However, in the next chapter, simpler statistics are going to be exposed in order to deal not with the activations but with the filter weights.

Chapter 3

Methodology

Provided the information in the previous chapter, the first problem that arises from trying to add a regularization term to reduce redundancy, is to find a feasible and differentiable objective function to descend. The second problem would be where to apply that function. Three possible applications would be: (i) between neurons of the same layer, (ii) between weights (force the convolution filters to be as different as possible) (iii) between layers. Information theory procedures in (iii) have already been tried in speech recognition to maximize the mutual information between the input and the output [26] but, unlike (i) and (ii), this method doesn't ensure that the internal representation of the probability distributions in the network is optimal.

The solution has to reach a compromise between the overhead produced by the redundancy reduction algorithms, the speedup due to the reduction of network parameters and, in particular, its classification performance. That means that the most suitable solution is to try to reduce the redundancy between the weights because, unlike (i), it does not require to calculate the probability distributions of the activations of the neurons and, unlike (iii), it does take into account internal representations.

In fact, when regularizing, one is trying to force the filters to converge into a better local minimum. As said, one possibility is to include this regularization directly into the gradient but another possibility is to check the redundancy in filters every N iterations and make a "de-correlation step". Figure 3.1 on the following page illustrates the aim of these solutions.

Another possibility if filter redundancy is found is to try to delete those filters which are considered redundant. This has already been tried at the single-weight level by the Optimal Brain Damage technique from Yann Le Cun *et al.* (1990) [8]. However, in the typical parallel architectures used nowadays to execute CNNs, removing individual weights doesn't suppose a real change in performance. For instance, in the Caffe Framework filter convolutions are treated as parallel matrix multiplications, which will take the same time if some of the weights in them were 0 or deleted. Sometimes the process can be even faster if the matrix has a power-of-two size, even if it's bigger, due to the efficient use of the hardware. On the other hand, Caffe loops through the filters to compute the convolutions in the GPU so it would

Figure 3.1: This figure illustrates which is the aim for the filter regularization. One wants to change the filters enough to move to a better local minimum without "unlearning too much", namely, without moving too far where the current minimum was. Thus, one expects to have first a loss in performance but, after fine-tuning, to have an actual improvement over the original performance.

be an actual improvement of performance if we could prune at the filter level, this would also reduce the required memory for the network. After pruning, one would expect the accuracy to drop because the upper layers still count on the non-pruned features, but after some training iterations, it should recover up to a point dependent on the number of pruned filters. Mobility is a very important factor nowadays due to the expansion of the smartphones and other small devices in the market. Thus, although the accuracy of a pruned network can be worse than the original one, it is very interesting to reduce big networks, in terms of memory and CPU usage, so that they can be used in lighter hardware.

One way to see if filters are really similar or not is to use a clustering algorithm to see if they form clusters or not. There exists a data visualization algorithm that allows us to project the filters into a 2-D plane to see if they are spread or agglomerated called T-sne [39]. If we apply it to the network filters we obtain the figure 3.2 on the next page. As it can be seen in the picture, there is a cluster in the center with similar filters and there are smaller groups in the periphery.

It could be the case that having redundancy in the filter weights was not resulting in redundancy over the activations, that is why T-sne has also been performed over the activations of the previous filters using the Pascal database [9]. The results in figure 3.3a on the facing page and figure 3.3b on the next page show that clusters are still present in filter activations. Thus, it is likely that reducing weight redundancy, activation redundancy is reduced as well.

During the next sections, some proposed methods for pruning or increasing the effective number of parameters will be exposed.

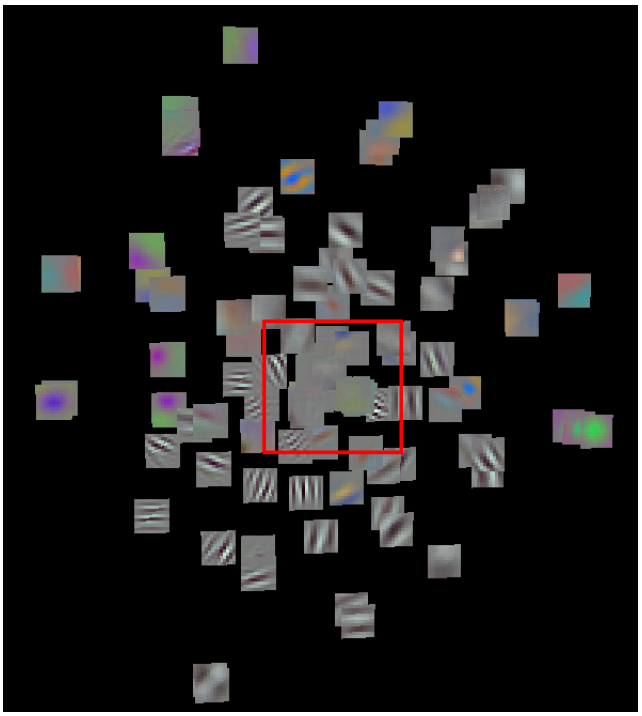
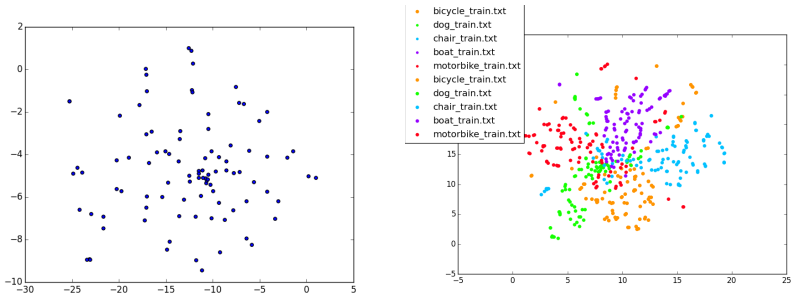


Figure 3.2: 2d projection of the first convolution layer filters of the pre-learnt CaffeNet using T-sne. The red square shows how there's an agglomeration of very similar and "dead" filters in the center.



(a) 2d projection of the activations of the first convolution layer filters of the pre-learnt CaffeNet using T-sne and images from the dog class present in Pascal VOC database.

(b) 2d projection of the activations of the first convolution layer filters of the pre-learnt CaffeNet using T-sne and images from 5 classes present in Pascal VOC database.

Figure 3.3: It can be seen it presents exactly the same shape as the filter weights. This means that the filter activations might be highly related to the filter activations and thus, two filters which are very close in the weights plot are very likely to activate at the same time. This is a sign of redundancy because only one filter would be enough to cover these two.

Algorithm 1 Basic filter regularization algorithm

```
1: procedure FilterPruning(network, layer, num)  ▷Num is the desired final number of filters.
2:     nfilters = length(network[layer].filters)
3:     while nfilters > num do
4:         dist_matrix = calc_neigh_dists(network[layer].filters)  ▷Calculate distance between every pair of filters with our preferred distance function.
5:         f1, f2 = argmin(dist_matrix)  ▷Indices of the nearest filters.
6:         network[layer].filters[f1] = mean(network[layer].filters[(f1,f2)])  ▷Filter f1 becomes the average of filters f1 and f2.
7:         network[layer].filters[f2].delete()  ▷Delete f2.
8:         network[layer+1].filters.channels[f1] =
9:             mean(network[layer+1].filters.channels[(f1,f2)])
10:        network[layer+1].filters.channels[f2].delete()  ▷Do the same to next layer's filters.
11:        nfilters = length(network[layer].filters)  ▷Update the filter count.
12:    end while
13: end procedure
```

3.1 Filter pruning

As seen previously, one can determine when two filters are very likely to always activate at the same time just by looking at their similarity. One example is their euclidean distance distance when projected in a 2D space with a visualization technique like T-sne. Thus, one obvious procedure would be to merge those filters that are very similar, given their 2D projection or an arbitrary distance function. When this is done to a layer, though, it has to be taken into account that the next layer will have to be modified so that it expects fewer inputs. Concretely, every filter in layer l represents a channel in the feature map received by layer $l+1$. Hence, if filters $f1$ and $f2$ in layer l are merged, the weights in layer $l+1$ that point to this channels should be treated as well, for example, merging them. For this project, the average function will be considered as a merging function. The procedure is thus simple, and it is described in algorithm 1.

In ILSVRC2012 Alex Krizhevsky introduced a feature in CNNs called group [17] that made that half of filters of the layer $l+1$ only used a half of the channels of the feature map from layer l to optimize the code for two GPUs. This complicates the original scheme because its implementation forces the number of filters to be pair and thus, pruning an odd number of filters is not possible. There is also the problem that it can unbalance the number of filters between GPUs (it is not allowed by the implementation). For this, a naive solution has been used, which consists in considering the halves of filters and pruning the same amount of filters in each of them (so it is not possible to merge filters between both halves).

This algorithm, extended and adapted to work with Caffe's data structures has been implemented in python and its results will be shown in the next chapter. For a

graphical representation of the pruning algorithm, see figure A.1 on page 48. Next, it will be presented an algorithm to reduce the number of redundant filters without pruning.

3.2 Filter regularization

As said previously, the other way to reduce redundancy in a CNN could be to make the filters as much independent as possible so that "dead filters" would barely appear. For that, distance measures have to be considered in order to compare filters and create a good regularization term. Some of the distances that can be considered are the Mahalanobis distance [19], Image Euclidean Distance (IMED) [40], tangent distance [32], PCA-Based distances, 2-D Entropy [44], Image Normalized Cross-Correlation [22], 2-D Mutual Information [29] or approaches in the frequency domain. However, this thesis will be focused on using IMED and Fourier distances since they are cheap to compute and don't have too many parameters to tune. Regarding Entropy metrics, Agrawal *et al.* (2014) [1] show that entropy has already been used to evaluate the CNN filter activation exclusiveness for each class. It shows that effectively, most of the filters only turn on when a certain class is predicted, but it doesn't mention the "dead filters" nor multiple filters doing the same task. It also would be good to be able to learn and evaluate them with unsupervised learning. Jaderberg *et al.* (2014) [13], also show there is redundancy in convolution filters and that they can be linearly decomposed in smaller filters that are faster to convolve. The following sections will discuss how to apply these distance metrics in the domain of CNNs to successfully compare filters in order to deal with this redundancy.

Mahalanobis distance

The Mahalanobis distance is a distance metric that uses the variance to weight the distances of the different dimensions. So, the Mahalanobis distance between the weight vectors $\vec{x}$ and $\vec{y}$ is:

$$D(\vec{x}, \vec{y}) = \sqrt{(\vec{x} - \vec{y})^T S^{-1} (\vec{x} - \vec{y})} \tag{3.1}$$

Where S is the covariance matrix. For the case of the image filters, the covariance matrix is proportional to $X * X^T$ where X is a matrix where each column is a flattened filter and each row is mean-centered.

The problem of this metric is the need to invert the covariance matrix. For the case of the *conv1* filters of the pre-learnt network for ILSVRC2012 present in the Caffe framework [14], this inverse matrix cannot be computed because the determinant of the covariance matrix is 0. A possible alternative to solve this problem is to use the (Moore-Penrose) pseudo-inverse of a matrix but the results are not very intuitive and thus, this distance metric will be discarded for the moment.

There is another way to take into account the variances in each dimensions which are to compute the distance in the *Principal Components Analysis (PCA) space* [24].

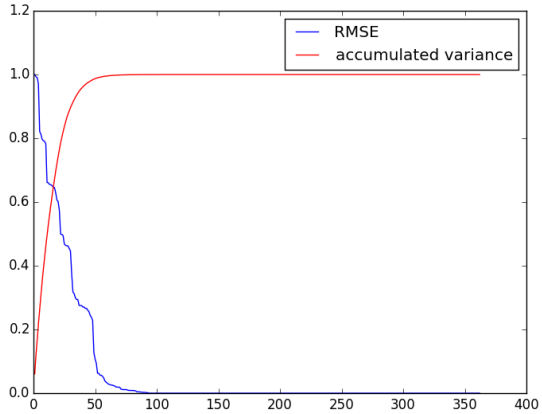


Figure 3.4: RMSE of PCA reconstruction using the k eigenvectors with the highest k eigenvalues. It can be seen that with 100 eigenvectors we have almost 0 error. However, this is a bad metric to know how many eigenvectors should be chosen. Normally, it is wanted that the k chosen eigenvectors capture a high percentage of the variance. That is expressed by the red line. This shows that with 60 eigenvectors is enough to capture the 99% of the variance.

PCA distance

Another way to compute the distance between filters is to project them to a space where only the dimensions with most of the variance are kept and the distance is computed on the relevant information. To convert our filters to the PCA space, we need the covariance matrix of the data (see prev. subsection). Then the *singular value decomposition* must be performed over the covariance matrix $\sum$ so that the *eigenvectors* and *eigenvalues* corresponding to these eigenvectors are obtained. The eigenvectors allow us to change the filters to the PCA space and the corresponding eigenvalues are proportional to the variance in the direction represented by each eigenvector. Thus, the dimensions with the least variance can be ignored as the values in that direction will always be around the mean. The figure 3.4 shows the reconstruction RMSE of the ILSVRC2012 filters using the k eigenvectors with the largest associated eigenvalues as well as the accumulated variance captured by these eigenvectors.

Unfortunately, it is possible that the presented distances are too affected by small deformations and translations. To overcome this problem, we can use the IMED distance, Wang *et al.* (2005) [40], which it is designed to be robust to this small transformations. Another problem that makes the PCA distance not feasible to be applied during the training phase is the fact that the covariance matrix must be calculated each epoch since the weights are constantly changing.

IMED distance

It has been demonstrated that IMED distance can be reduced to a linear transformation of the two images to compare and then calculate the normal Euclidean

distance between the images [5]. This is because the distance between two pixels is the weighted distance between them and all the neighbors. These weights are taken from a Gaussian distribution and they can be put into a circulant matrix G so that:

$$d_{imed}(x, y) = (x - y)G(x - y)^T$$

This matrix G can then be decomposed so that we obtain:

$$d_{imed}(x, y) = (x - y)^T G^{\frac{1}{2}} G^{\frac{1}{2}} (x - y)$$

Then, if $u = xG^{\frac{1}{2}}$ and $v = yG^{\frac{1}{2}}$, the IMED distance can be reduced to:

$$d_{imed}(x, y) = (u - v)^T (u - v)$$

This is nothing else than applying a smoothing operation over both filters before computing the euclidean distance. This smoothing is called by Wang *et al.*, the standarizing transform $ST(\cdot)$. Another nice property of this transform is that weights decay very fast when we move away from the target pixel. This means that instead of storing the G matrix, which has $N^2 \times N^2$ positions ($N = \#px_{filter}$), a smaller 5×5 or 3×3 Gaussian kernel can be convolved with the filters, providing almost the same result as the full $ST(\cdot)$.

Frequency domain
It has already been explained that one of the issues argued in the solution of ILSVRC2013 [45] was the fact that the filters captured very high frequencies or low frequencies, hence skipping the mid-range frequencies. This can be proved by applying the Fourier Transform to the filters and plot an amplitude histogram. However, in figure 3.5 on the following page, it can be seen it is not completely true because they are clearly capturing low and mid frequencies more than the higher ones. Up to a point, this is normal because very high frequencies can represent noise.

We can try to make the captured frequencies more unique between filters by descending the gradient of the similarity between filters using the coefficients of the frequencies in the Fourier domain. Since the Fourier coefficients are translation-invariant, we can simply descend their Euclidean distance.

3.2.1 Regularization procedure

Once known which distance or similarity functions to try, we need to think of a procedure to maximize them. Since we don't want to affect too much the training time, one way to do it might be to pick a random pair of filters per iteration, and add the gradient of the distance function to the backpropagation gradient. Hence, the basic algorithm could be the one presented on algorithm 2.

The target of the method is to redistribute the filters so that they are spread like a uniform distribution so that there are not too many identical filters, which would be a source of redundancy. Thus, an important question is if it is safe to assume that moving random pairs of filters will eventually leave a uniform distribution after some

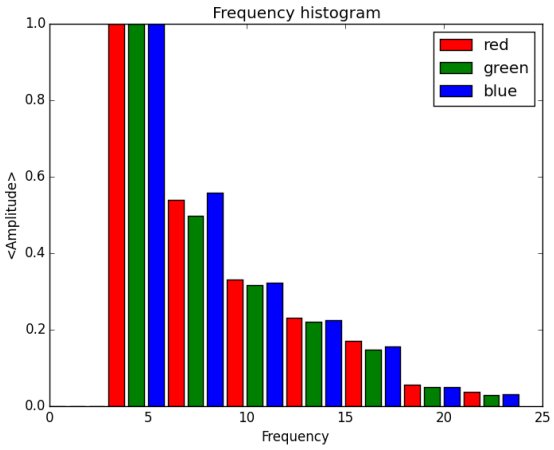


Figure 3.5: Histogram of the Fourier transform coefficients of the filters without the DC component for the R, G and B components. However, it can be seen that most of the filters capture low frequencies, not high nor mid range ones.

Algorithm 2 Basic filter regularization algorithm

procedure RegBackpropagation1
  for each backpropagation iter **do**
    diff = backpropagation_gradient() ▷Calculate the backpropagation gradient.
    for ($step = 0$; step<regularization_steps; ++step) **do**
      x, y = random_filter_pair() ▷Current filter pair.
      $xgradient, ygradient$ = simfunc_gradient(x, y) ▷Calculate the gradient of the similarity of the filters.
      $diff[x] = diff[x] + \alpha * xgradient$ ▷Add the gradient to the backpropagation gradient.
      $diff[y] = diff[y] + \alpha * ygradient$
    end for
    backpropagation_update_weights(diff, global_lr) ▷Continue with the modified gradient.
  end for
end procedure

iterations. One way to illustrate it, is to try this basic algorithm for maximizing the euclidean distance of a set of randomly distributed 2d points. Thus, for each random pair of points (X_i, X_j), the function to maximize will be $\sqrt{(X_i - X_j)^T(X_i - X_j)}$. Since the maximum is infinite, the problem will be turned into the minimization of a similarity function between 0 and 1. This function can be:

$$d(X_i, X_j) = \frac{1}{1 + (X_i - X_j)^T(X_i - X_j)}$$

The gradient of the function with respect to X_k^d with $k \in \{i, j\}$ and $d \in 1, 2$ since there are two dimensions, is:

$$\Delta d(X_i, X_j) = \frac{d}{dx_k^d} \frac{1}{1 + (X_i - X_j)^T (X_i - X_j)} = \frac{d}{dx_k^d} \frac{1}{1 + \sum_d (x_i^d - x_j^d)^2}$$

$$= ((1 + \sum_d (x_i^d - x_j^d)^2)^{-1})'(1 + \sum_d (x_i^d - x_j^d)^2)' dx_k^d = \frac{(-1)^k 2(x_i^d - x_j^d)}{(1 + \sum_d (x_i^d - x_j^d)^2)^2}$$

$$= \frac{(-1)^k 2(X_i - X_j)}{(1 + (X_i - X_j)^T (X_i - X_j)}$$

Then, the positions can be updated by adding (it is already multiplied by -1) this gradient multiplied by a "learning rate" constant α. We need however a metric to assess the performance of the method or even to know when to stop it. Namely, we want a measure of how uniform is the distribution of the filters across an n-dimensional space (for the points is 2D). There is plenty of literature (mostly for geological data) about metrics to assess the spatial distributions of points. Kam *et al.* (2013) [15] explain there are three main categories: Quadrant-based, which divide the space in a grid and performs point statistics inside each square; Distance-based metrics, which use the neighboring distances to perform the statistical analysis; others like the Projection Index (PI). They also explain that the first kind of metrics do not take into account the global spatial distribution while the second one does. This is important for the filter case since we don't want to have "impossible filters", meaning that moving a filter to a region which was empty because that pattern doesn't exist in the training data might be a bad move. Cressie and Noel (1993) [7] make a complete analysis of this subject in the book "Statistics for spatial data". However, for this problem, it can be seen that using a simple metric such as the nearest neighbor variance, does perform well enough. This is because if all the filters are at the same distance, this variance should be 0, but the global appearance of the distribution would be similar since we are only looking at the nearest neighbors (which represent the filters that are overlapping and thus, the ones we want to fix). If, after the regularization, the variance of the nearest neighbor distance has increased, this will mean that either the points are now very far from each other and thus a small difference in their distances is amplified or that the mean nearest neighbor distance has been preserved but simply the points still aren't uniformly distributed. On the other hand, if it decreases, it only can mean that the points are uniformly distributed (preserving the global shape) or that they all are at the same place (extreme case to be taken into account).

Figure 3.6 on the next page shows the results of applying such algorithm on 2d random data. It can be seen that after regularization, the points that were already near are still near, and the points that were far have increased their distance. This is happening because the function that is being minimized (figure 3.7 on the following page) is not penalizing enough the distance. Namely, when two points are very far, the gradient should be 0. This can be solved by using a Gaussian RBF as similarity target function (figure 3.8 on page 21):

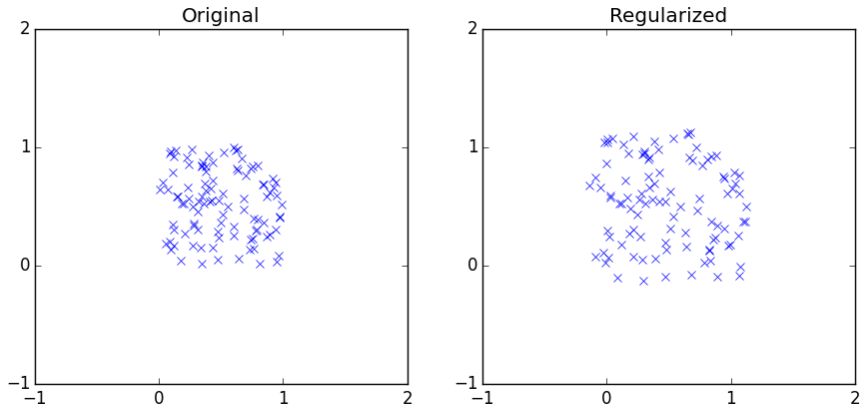


Figure 3.6: 1v1 regularization example for 100 2D points. The learning rate is set to 0.001 *and* 1000 *iterations. Increasing the number of iterations or learning rate only makes the expansion area smaller or bigger but the spatial homogeneity of the points is still the same.*

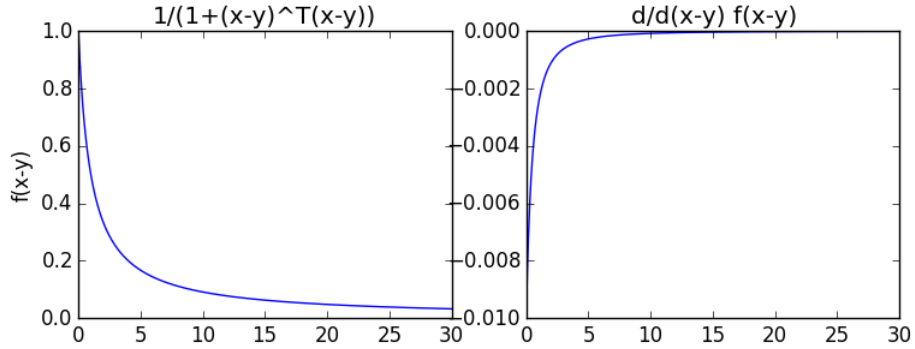


Figure 3.7: Target function (left) and its derivative (right).

$$d_{RBF}(X_i, X_j) = e^{-\frac{(X_i - X_j)^T (X_i - X_j)}{2 * \sigma^2}}$$

$$\Delta d_{RBF}(X_i, X_j) = \frac{d}{dX_k^d} e^{-\frac{(X_i - X_j)^T (X_i - X_j)}{2 * \sigma^2}} = \frac{d}{dx_k^d} e^{-\frac{\sum_d (x_i^d - x_j^d)^2}{2 * \sigma^2}}$$

$$= (-\frac{\sum_d (x_i^d - x_j^d)^2}{2\sigma^2})' e^{-\frac{\sum_d (x_i^d - x_j^d)^2}{2\sigma^2}} dx_k^d = \frac{(-1)^k (x_i^d - x_j^d)}{\sigma^2} e^{-\frac{\sum_d (x_i^d - x_j^d)^2}{2\sigma^2}}$$

$$= \frac{(-1)^k (X_i - X_j)}{\sigma^2} e^{-\frac{(X_i - X_j)^T (X_i - X_j)}{2\sigma^2}}$$

Tuning its σ value can make it to only affect the nearest neighbors. This highly improves the final result, as it can be seen on figure 3.9 on the facing page.

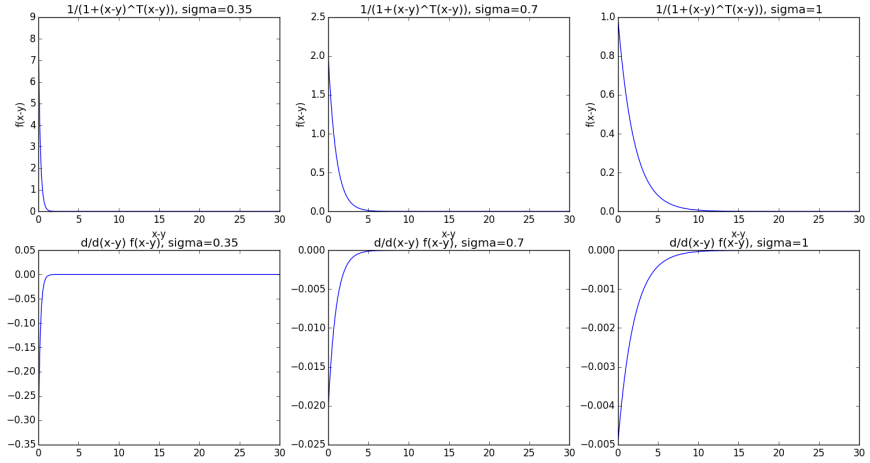


Figure 3.8: Target RBF for different σ values. The gradient turns rapidly into 0 when the distance increases.

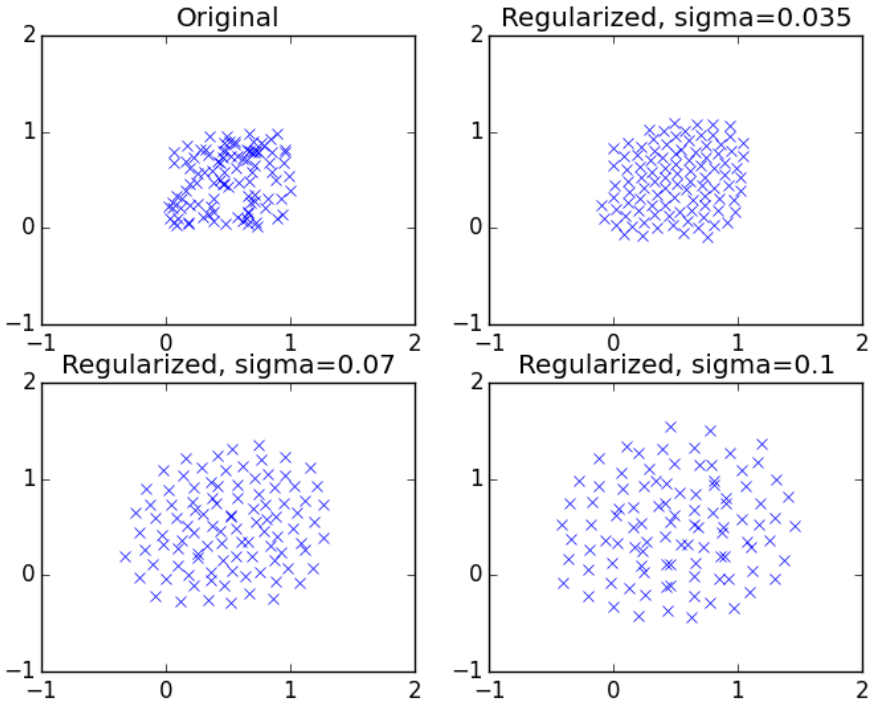


Figure 3.9: 1v1 2D regularization with RBF, $\alpha = 0.003$ and 50K iterations.

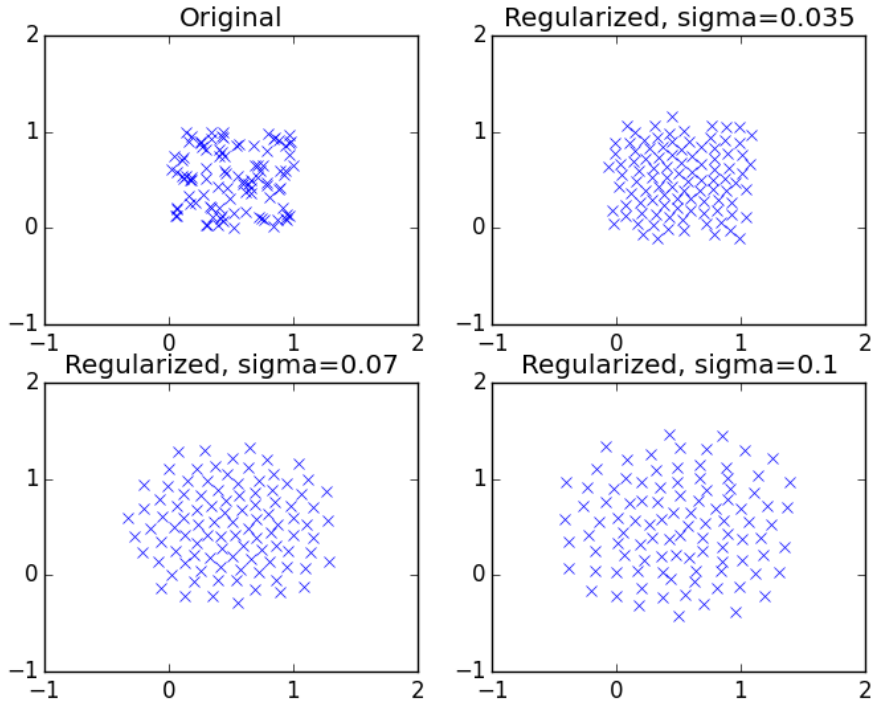
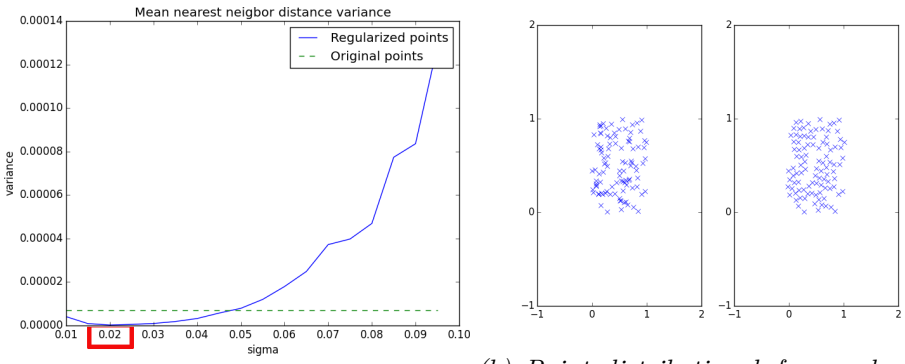


Figure 3.10: 2D regularization with RBF, 1vAll strategy and $\alpha = 0.005$ and 1K iterations, which is equivalent to 10K iterations since there are 100 points.

It is possible that $1v1$ is not the best strategy, results might improve if gradients for one filter are calculated with respect to the rest of the filters, this $1vAll$ procedure is illustrated on figure 3.10.

The nearest neighbor variance for different σ values is plot on figure 3.11 on the facing page. This shows that regularization is actually improving the homogeneity of the point distribution.



(a) Variance with respect to σ.

(b) Point distribution before and after regularization with $\sigma = 0.02$.

Figure 3.11: These plots show how the variance of the nearest neighbors can tell how good is the regularization performing. (a) shows the sigma that makes the nearest neighbor variance of the distance be below the original value and, when applied to the points in (b), they spread in a more uniform way, without identical pairs and without covering unrealistic values. These are the characteristics that one expects from non-redundant filters.

IMED gradient descent

As explained by Sun and Feng (2008) [36], IMED is no more than performing a smoothing to the both filters to compare and then, the euclidean distance between both:

$$IMED(W^{(0)}, W^{(1)}) = (GW^{(0)} - GW^{(1)})^T (GW^{(0)} - GW^{(1)})$$

Where G is a Gaussian-like weight Toeplitz matrix equivalent to convolve a small smoothing filter.

The similarity function will be such that has a minimum when the filters are completely different:

$$sim(W^{(0)}, W^{(1)}) = \frac{1}{1 + (GW^{(0)} - GW^{(1)})^T (GW^{(0)} - GW^{(1)})}$$

Thus, the gradient $\frac{d}{dw^f_{i,j}} sim(W^{(0)}, W^{(1)})$ is

$$\Delta sim(W^{(0)}, W^{(1)}) = \frac{d}{dw^{(k)}_{i,j}} \frac{1}{1 + \sum_{x_1,y_1} (\sum_{x_2,y_2} g^{(x_1,y_1)}_{x_2,y_2} w^{(0)}_{x_2,y_2} - \sum_{x_2,y_2} g^{(x_1,y_1)}_{x_2,y_2} w^{(1)}_{x_2,y_2})^2}$$

$$= \frac{(-1)^k * (-1) * 2(\sum_{x_1,y_1} g^{x_1,y_1}_{i,j} (\sum_{x_2,y_2} g^{(x_1,y_1)}_{x_2,y_2} w^{(0)}_{x_2,y_2} - \sum_{x_2,y_2} g^{(x_1,y_1)}_{x_2,y_2} w^{(1)}_{x_2,y_2}))}{(1 + \sum_{x_1,y_1} (\sum_{x_2,y_2} g^{(x_1,y_1)}_{x_2,y_2} w^{(0)}_{x_2,y_2} - \sum_{x_2,y_2} g^{(x_1,y_1)}_{x_2,y_2} w^{(1)}_{x_2,y_2})^2)^2}$$

Then, in vectorial form:

$$|\Delta sim(W^{(0)}, W^{(1)})| = \frac{2G(GW^{(0)} - GW^{(1)})}{(1 + (GW^{(0)} - GW^{(1)})^T (GW^{(0)} - GW^{(1)}))^2}$$

For the case of the Gaussian function:

$$|\Delta sim(W^{(0)}, W^{(1)})| = \frac{G(GW^{(0)} - GW^{(1)})}{\sigma^2} e^{\frac{-(GW^{(0)}-GW^{(1)})^T(GW^{(0)}-GW^{(1)})}{2\sigma^2}}$$

and an iteration consists in

$$W_{t+1}^{(0)} = W_t^{(0)} + \alpha|\Delta sim(W_t^{(0)}, W_t^{(1)})|$$
$$W_{t+1}^{(1)} = W_t^{(1)} - \alpha|\Delta sim(W_t^{(0)}, W_t^{(1)})|$$

By changing the sum and the subtraction from $W^{(0)}$ to $W^{(1)}$ and vice-versa it can be seen how it switches from minimization to maximization.

PCA distance gradient descent

For PCA there is the problem that the correlation matrix has to be computed and decomposed every iteration, which is very costly. To compensate this, the gradient will be computed over a dimensionally reduced space and then returned to the original space. Since PCA is a linear application from one space to the other, we will assume the recovered gradient is proportional to the original gradient. An interesting fact about PCA is that if we choose the first k eigenvectors (with the highest eigenvalues) and descent de gradient in the resulting space of using these eigenvectors, we will be forcing the pixels that already have a high variance to be different. So, if it is done over the dimensions with the least variance, noise will be introduced but the filters will further increase their difference and we can trust the neural network to reduce the effect of the noise. The similarity measure in PCA space will be then

$$sim(W_{pca}^{(0)}, W_{pca}^{(1)}) = \frac{1}{1 + (W_{pca}^{(0)} - W_{pca}^{(1)})^T(W_{pca}^{(0)} - W_{pca}^{(1)})}$$

the gradient in PCA space is

$$|\Delta sim(W_{pca}^{(0)}, W_{pca}^{(1)})| \approx \frac{(W_{pca}^{(0)} - W_{pca}^{(1)})}{(1 + (W_{pca}^{(0)} - W_{pca}^{(1)})^T(W_{pca}^{(0)} - W_{pca}^{(1)}))^2}$$

So, to recover the original gradient we have to undo the PCA transform

$$|\Delta sim(W^{(0)}, W^{(1)})| = PCA^{-1}(W_{pca}^{(0)}, W_{pca}^{(1)})$$

Again, using a Gaussian function:

$$|\Delta sim(W_{pca}^{(0)}, W_{pca}^{(1)})| \approx \frac{(W_{pca}^{(0)} - W_{pca}^{(1)})}{\sigma 2} e^{-\frac{(W_{pca}^{(0)}-W_{pca}^{(1)})^T(W_{pca}^{(0)}-W_{pca}^{(1)})}{2\sigma^2}}$$

and thus, the update rule is

$$W_{t+1}^{(0)} = W_t^{(0)} + PCA^{-1}(\Delta sim(W_t^{(0)}, W_t^{(1)}))$$
$$W_{t+1}^{(1)} = W_t^{(1)} - PCA^{-1}(\Delta sim(W_t^{(0)}, W_t^{(1)}))$$

Frequency domain

We can try to make the captured frequencies more unique between filters by descending the gradient of the similarity between filters using the coefficients of the frequencies in the Fourier domain. So, given the magnitudes $w_{i,j}$ of the FT of the filters $W^{(0)}$ and $W^{(1)}$, the equations for calculating the gradient are the following:

$$sim(W^{(0)}, W^{(1)}) = \frac{1}{1 + (W^{(0)} - W^{(1)})^T (W^{(0)} - W^{(1)})}$$

in the non-vectorial form

$$sim(W^{(0)}, W^{(1)}) = \frac{1}{(1 + \sum_{i,j} (w_{i,j}^{(0)} - w_{i,j}^{(1)})^2}$$

and the derivative with respect to a magnitude $w_{i,j}^{f}$ is

$$\frac{d}{dw_{i,j}^{(0)}} sim(W^{(0)}, W^{(1)}) = -\frac{(w_{i,j}^{(0)} - w_{i,j}^{(1)})}{(1 + \sum_{i,j} (w_{i,j}^{(0)} - w_{i,j}^{(1)})^2)^2}$$

and

$$\frac{d}{dw_{i,j}^{(1)}} sim(W^{(0)}, W^{(1)}) = \frac{(w_{i,j}^{(0)} - w_{i,j}^{(1)})}{(1 + \sum_{i,j} (w_{i,j}^{(0)} - w_{i,j}^{(1)})^2)^2}$$

Then, in vectorial form, the gradient is:

$$|\Delta sim(W^{(0)}, W^{(1)})| = IFFT(\frac{(W_{FFT}^{(0)} - W_{FFT}^{(1)})}{(1 + (W_{FFT}^{(0)} - W_{FFT}^{(1)})^T (W_{FFT}^{(0)} - W_{FFT}^{(1)}))^2})$$

Again, using a Gaussian RBF:

$$|\Delta sim(W^{(0)}, W^{(1)})| = IFFT(\frac{(W_{FFT}^{(0)} - W_{FFT}^{(1)})}{\sigma^2} e^{\frac{(W_{FFT}^{(0)} - W_{FFT}^{(1)})^T (W_{FFT}^{(0)} - W_{FFT}^{(1)})}{2\sigma^2}})$$

The update steps are

$$W_{t+1}^{(0)} = W_t + |\Delta sim(W_t^{(0)}, W_t^{(0)})|$$
$$W_{t+1}^{(1)} = W_t - |\Delta sim(W_t^{(1)}, W_t^{(1)})|$$

Combinations

All these distance functions can be combined to get the best of each. IMED and

PCA could be combined by performing the weighted sum of both. The same holds for any combination of the previous.

$$W_{t+1}^{(0)} = W_t^{(0)} + \alpha|\Delta IMED(W^{(0)}, W^{(1)})| + \beta|\Delta D_{PCA}(W^{(0)}, W^{(1)})|$$
$$W_{t+1}^{(1)} = W_t^{(1)} - \alpha|\Delta IMED(W^{(0)}, W^{(1)})| - \beta|\Delta D_{PCA}(W^{(0)}, W^{(1)})|$$

Another way to do it is to regularize completely using different techniques in a loop until the improvement of the nearest neighbor variance is less than a threshold.

3.3 Preliminary results

Adding and tuning regularization term like this during the training of a neural network is very time-consuming. Hence, a good practice could be to assess the regularization performance over pre-trained filters to choose the actual model that will be used on the final neural network. Furthermore, the same experiment as with the 2D points could be used to check if the algorithm changes the filters in the desired directions. First, the performance will be assessed over a big pre-trained network like the one used by Krizhevsky *et al.* over ILSVRC2012 [17]. However, it is not likely that the regularization could be tried on the actual network during the training phase within the time of this project, so a Cifar10 network will also be studied and compared with the Imagenet one and, finally, it will be possible to try it over a real Cifar10 [16] network model during the training phase.

3.3.1 Imagenet

As already said, Imagenet is a very big dataset, for which it can take weeks to fit a model. However, there exist pre-trained models online, one of them provided by the Caffe framework [14], Jia. *et al.* (2013). For example, from this model (figure 3.12 on the next page), we can extract the first convolution layer filters to use for regularization. This first "conv1" layer is composed by 96 filters of $11 \times 11 \times 3$ weights. These are going to be compared pairwise in order to force similar filters to be different using the algorithm 2 and the Gaussian RBF target function. Table 3.1 on page 29 shows the amount of regularization achieved by IMED and Fourier for the already trained filters for different combinations of hyper-parameters. For more detail, see figure A.2 on page 49, where the whole the search space has been plotted.

Figure 3.14 on page 28 shows how are the final filters after the best regularizations, using the parameters found in table 3.1 on page 29, in comparison to the original filters shown on again in figure 3.13 on the facing page for convenience. Regarding the frequency histogram, in figure 3.15 on page 30 it can also be observed that the frequencies are slightly better balanced than in figure 3.5 on page 18.

Given the mentioned figures, the regularization seems to succeed in reducing the number of dead filters and to create a better redistribution of them. On the other hand, all these results are found over a huge neural network, which is more prone to have redundancy than a small one. Thus, the next subsection will be about how these techniques generalize for Cifar10 [16] models, which tend to be much smaller.

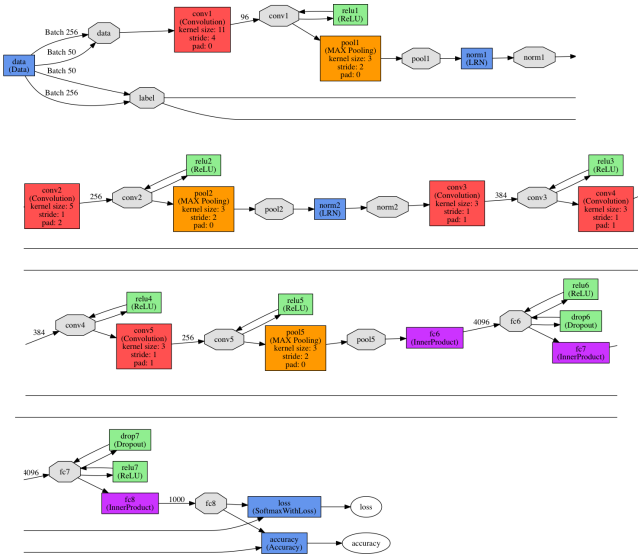
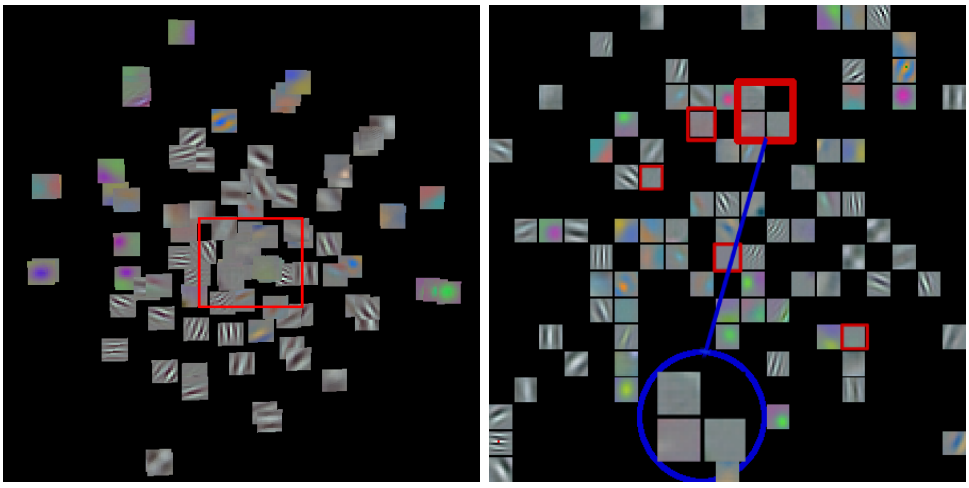
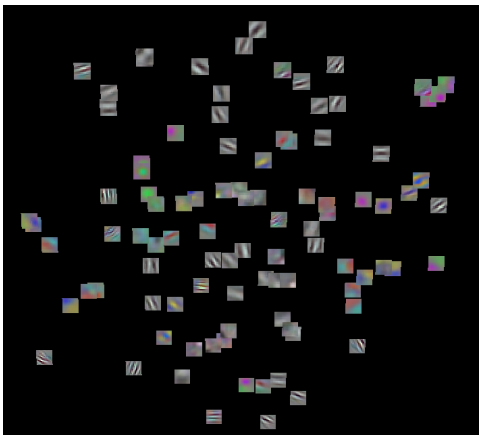


Figure 3.12: "Caffenet" model for Imagenet.

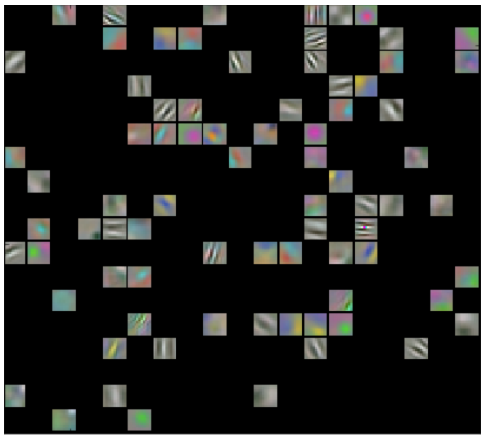


(a) T-sne of the original filters. *(b) Kohonen SOM of the original filters.*

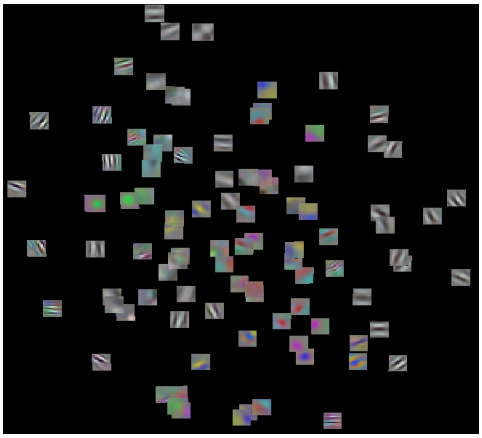
Figure 3.13: Visualizations of the original caffenet filters. Some of the "dead" and similar filters are highlighted in red. T-sne, seems to give a more meaningful representation than the Kohonen SOM.



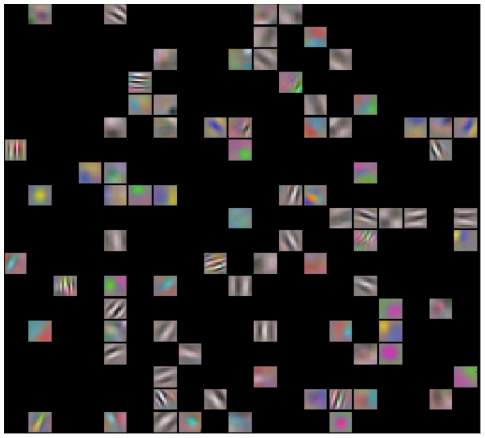
(a) T-sne with $IMED_{3x3}$, $\sigma^2 = 0.24$, $\alpha = 0.1$ and 5000 iters.



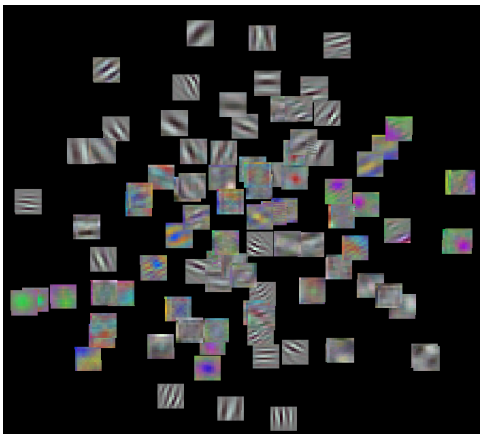
(b) Kohonen SOM with $IMED_{3x3}$, $\sigma^2 = 0.24$, $\alpha = 0.1$ and 5000 iters.



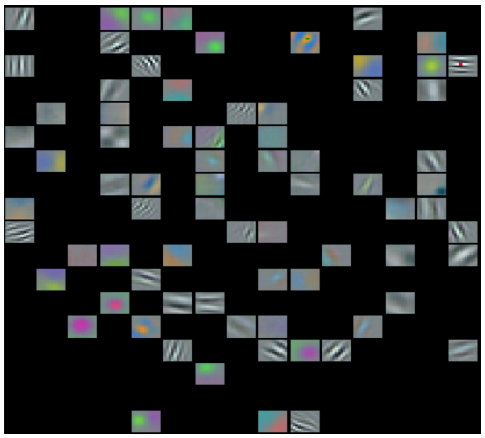
(c) T-sne with $IMED_{5x5}$, $\sigma^2 = 0.4$, $\alpha = 0.05$ and 5000 iters.



(d) T-sne with $IMED_{5x5}$, $\sigma^2 = 0.4$, $\alpha = 0.05$ and 5000 iters.



(e) T-sne with frequency domain regularization, $\sigma^2 = 0.56$, $\alpha = 1$ and 10^4 iters.



(f) T-sne with frequency domain regularization, $\sigma^2 = 0.56$, $\alpha = 1$ and 10^4 iters.

Figure 3.14: Visualization of the Imagenet filters after regularization. It is important to point out that the relative improvement measure seems to work although the Fourier regularization seems to have introduced some noise and edge effects when α or $\#iters$ are big.

	α	σ^2	Iters	Rel. Improvement
$IMED_{3x3}$	0.1	0.24	5K	56.41%
$IMED_{5x5}$	0.01	0.4	20K	50%
Fourier	1	0.56	100K	71.42%

*Table 3.1: Best hyper-parameters found for the different regularization methods over Imagenet. One way to compare them is to see which is the relative improvement w.r.t. their base variances, i.e. a **56.41%** for $IMED_{3x3}$, a **50%** for $IMED_{5x5}$ and a **71.42%** for Fourier. This means that there might be more margin to reduce the variance in te frequency domain.*

Algorithm 3 Automatic hyper-parameter determination using a grid search in the hyper-parameter space. In this pseudocode, a recursive call has been used since it is more "natural" for this problem. It works better if filters are standarized due to the RBF kernel.

```
1: function FIND_BEST_HYPER_PARAMS(FILTERS, INITIAL_RANGE, ITERS)
2:     if iters is 0 then
3:         return center(initial_range)
4:     end if
5:     grid = create_hyperparameter_grid(initial_range)
6:     for each cell in grid do
7:         new_filters = regularize(filters, cell.hyperparams)
8:         cost[cell] = nearest_neighbor_std(distance_space(filters)) ▷Set the cost of
       the current choice of hyper-parameters (if we are using IMED, the cost has to be calculated
       into the IMED space. The same for the FT.)
9:     end for
10:    best_cell = grid[argmin(cost)]                ▷Retrieve the best combination so far.
11:    grid = convert_cell_to_grid(grid, best_cell)  ▷Divide the space to create a new
       grid of the same size as the previous one centered on the best values and having the previous
       adjacent cells as extreme values.
12:    return find_best_hyper_params(filters, grid, iters - 1)
13: end function
```

3.3.2 Cifar10

The CIFAR-10 dataset consists of 60000 32×32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.[1] Caffe already provides some predefined networks to train over this dataset ensuring to achieve a predetermined performance. From these, the tests will use the one called `cifar10_full`, represented in figure 3.16 on the next page and which reaches a $\sim 82.2\%$ accuracy in $6h$ in an NVIDIA Tesla K40c in 70K iterations. On table 3.2 on page 31 one can see that the results for variance reduction on the first layer of this pre-trained model, which only has 32 filters of 5×5 size, are not as good as

[1] http://www.cs.toronto.edu/~kriz/cifar.html

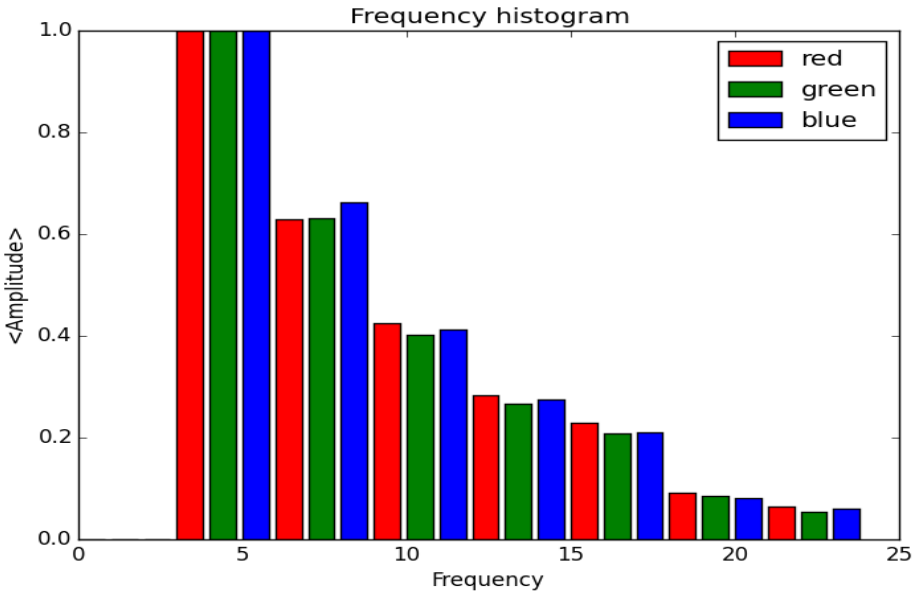


Figure 3.15: Frequency histogram for the regularized filters. The medium and higher frequencies are slightly higher than in the original filters, however, it is not as balanced as one would expect. This happens because, in most of the filters, these frequencies are so similar that the gradient becomes 0 because it reaches the maximum in the similarity function.

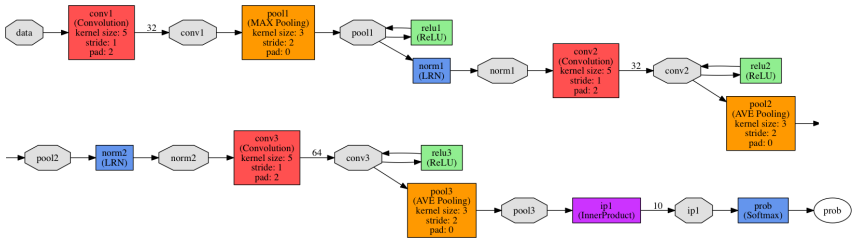


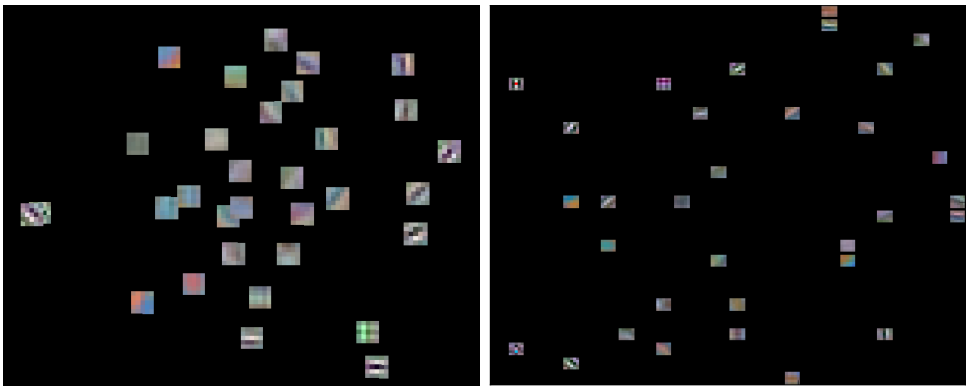
Figure 3.16: `cifar10_full` model for Cifar10.

they are for Imagenet. The full hyper-parameter search can be seen in figure A.3 on page 50. Figure 3.17 on the facing page shows visualizations of the resulting filters.

Since there is no margin for improvement on this model, a natural step would be to make the model bigger and try to use the regularization to increase the number of non-redundant parameters. Table 3.3 on the next page shows the results of applying the regularization techniques over the same network but with 96 filters instead of 32 in the first layer. Figure 3.18 on page 32 shows T-sne visualizations of the 96-filter network after different regularizations. The accuracies after training without regularization are exactly the same for 32 and 96 filters. Again, the full search can be seen in figure A.4 on page 51.

	α	σ^2	Iters	Rel. Improvement
$IMED_{3x3}$	0.01	0.13	10	0.43%
$IMED_{5x5}$	0.01	0.13	10	0.36%

Table 3.2: Best hyper-parameters found for the `cifar10_full` *model. It can be seen there is not enough redundancy and thus, the margin to reduce the redundancy is so narrow that it is even possible to make it worse accidentally.*



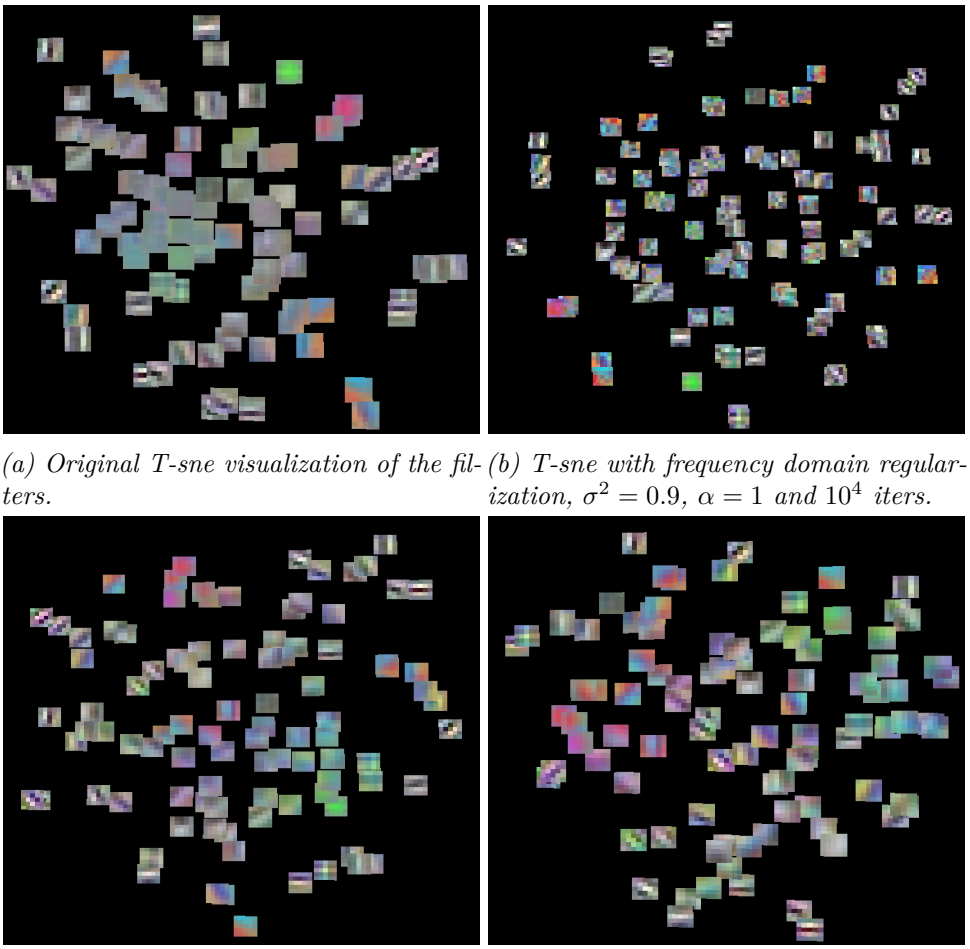
(a) T-sne of the original filters. *(b) Kohonen SOM of the original filters.*

Figure 3.17: (a) and (b) are the projection of the filters in a 2D space using T-sne and Kohonen SOM maps respectively.

	α	σ^2	Iters	Rel. Improvement
$IMED_{3x3}$	0.01	0.12	20K	30.39%
$IMED_{5x5}$	0.001	0.16	20K	40.9%
Fourier	0.1	0.85	100K	52%

Table 3.3: Best hyper-parameters found for regularizing the first convolutional layer of `cifar10_full` *modified to have 96 filters instead of 32 in the mentioned layer. It can be seen that now there is more margin to distribute the filters and thus, one can see better improvements.*

Since these figures show that there is still margin to specialize filters, the next question that arises is how this regularization should be introduced in the backpropagation algorithm. The easiest possible way would be to fine-tune the network over the regularized filters. However, it might be possible that it is too late to specialize them and the fine-tuning makes them converge again to the original ones. Another easy possibility could be to do it on the initial filters, which are randomly initialized with a normal distribution $\mathcal{N}(0, 0.0001)$. This means that most of the points will be really close, around the 0, and it is possible that they overlap, so it might be important to specialize them from early stages. The next section will show the results of applying the regularization at different stages.



(a) Original T-sne visualization of the filters.

(b) T-sne with frequency domain regularization, $\sigma^2 = 0.9$, $\alpha = 1$ and 10^4 iters.

(c) T-sne with $IMED_{3x3}$, $\sigma^2 = 4.3$, $\alpha = 1$ and 5000 iters.

(d) T-sne with $IMED_{5x5}$, $\sigma^2 = 1.98$, $\alpha = 1$ and 5000 iters.

Figure 3.18: Visualization of the `cifar10_full` model filters after regularization. This time, the regularization in the frequency domain seems to create too noisy filters. It is important to remark that the regularized filters seem to have less "dead filters" and the central agglomeration is "dissolved".

Chapter 4

Results

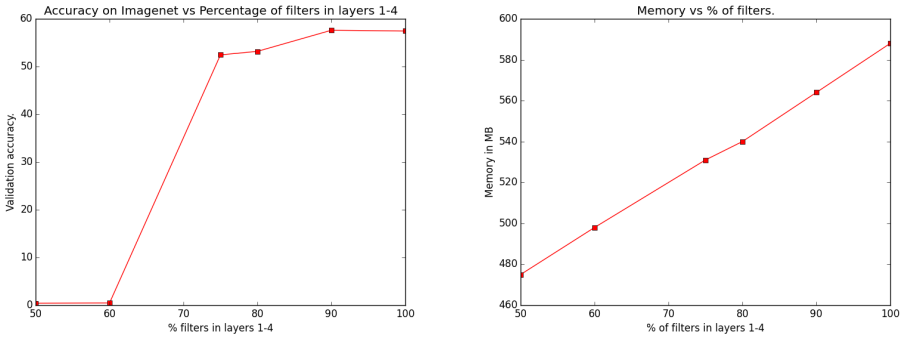
In this chapter, it will be shown the results obtained by applying the previously explained techniques over well-known datasets such as CIFAR10 and Imagenet. Concretely, Imagenet will be first pruned and its performance will be assessed. Next the results of filter regularization are going to be shown. For the latter, 3 set-ups are going to be introduced: continuous regularization, single-stage, and multi-stage regularization. However, the results without applying any regularization are going to be shown first in order to have a reference to compare with.

4.1 Network pruning

Since Imagenet models can take weeks to be learned, the pre-learnt `caffenet` model present in the Caffe framework [14] will be used. This model has been trained by Jeff Donahue and its performance is already specified in a `README` file:

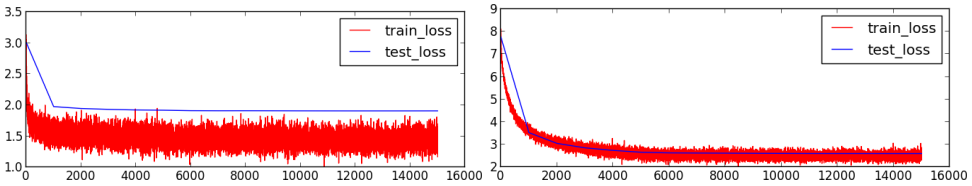
```
This model is snapshot of iteration 310,000.
The best validation performance during training was iteration 313,000
with validation accuracy 57.412% and loss 1.82328.
```

Because this thesis is mostly focused on CNNs and there is now the tendency to suppress the stacked fully-connected layers for only one of them (see GoogLeNet solution for ILSVRC2014 [37]), the pruning will be performed over the first four convolution layers of `caffenet`. The fifth is not going to be changed just for simplicity so that it won't be required to make any change to the next layer, which is fully-connected. After pruning, the model is fine-tuned for 15K iterations with a learning rate that goes from 10^{-4} to 10^{-6}, being divided by 10 every 10K iterations (even the non-pruned model for control). Known all this, the results are presented in figure 4.1 on the following page.



(a) Comparison of the accuracy vs the number of filters. *(b) Comparison of the number of filters vs the memory used in MB.*

Figure 4.1: This figure presents the `caffenet` *pruning results. (a) shows the accuracy vs the remaining % of filters in layers 1-4 while (b) relates the latter with the MB needed to deploy the model in a GPU (to train it, the difference is much bigger). This figure demonstrates that the network has indeed redundancy since we can reduce the number of filters to the 90% without having the redundancy affected.*



(a) Fine-tuning `caffenet` *with 90% of the filters on the first four layers.* *(b) Fine-tuning* `caffenet` *with 50% of the filters on the first four layers.*

Figure 4.2: This figure shows how pruning filters reduces overfitting by decreasing variance of the model and introducing some bias. This is illustrated by the difference of train and validation loss, which is higher when only the 10% of the filters are pruned (a) than when the 50% of them are pruned (b) (for the 1-4 conv layers).

To sum up, a new criteria to prune filters has been proposed and successfully tried over the `caffenet` model. Although the results are modest, with only a 10% of the filters can be pruned using this model without losing performance, this already represents 24MB of weights represented by float numbers in the GPU.

It is also interesting to see that this method is, in fact, reducing the overfitting of the model. Figure 4.2 illustrates it. In fact, since there is a link between redundancy and overfitting, one could rethink the use of the methods proposed during this thesis as a criteria to prune or to add filters to the network during the training to control it.

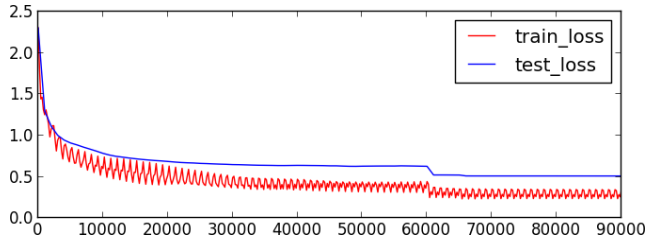


Figure 4.3: Original cifar10_full training accuracy and loss for 90K iterations. The accuracy over the test set reaches a plateau when it is 0.822. The "steps" that the curve does some time are due to a change of the learning rate.

4.2 Network regularization

In order to be able to compare the regularized model with a reference model, it is useful to first illustrate how are the results for the base `cifar10_full` model with 96 filters in the first convolutional layer. Figure 4.3 shows how an original training process looks like.

In fact, this is the result for one random initialization but on average the model reaches an 82.3% of accuracy with a standard deviation of 0.28%. Another thing to take into account is the fact that the model has small bias since both the train and test errors are low and the test error is slightly higher than the training one, which is usually an indicator of higher variance or overfittng. However the small gap between training and test errors show that variance is also very small. Thus, it is very probable that the regularization is not able to perform well on this dataset since it is aiming to increase the bias and decrease the variance by making the filters more uniformly distributed and hence, more general. Once this is known, one can proceed to verify all these hypotheses by comparing with the regularized versions in the next sections.

4.2.1 Continuous regularization

One way to regularize the filters is the one explained in algorithm 2. This is nothing else than choosing a random pair of filters every epoch and the gradient of their similarity function to the backpropagation gradient. The advantage of this method is that, since it can be applied all the time, its effects are not reversed by the backpropagation gradient. Another good reason for regularizing during all the training is that one can make the filters start specializing from the very beginning, when it is less likely that two filters are exactly the same because they are randomly generated. As it has already been explained, the problem of having two identical filters is that their gradient will be 0 and the regularization will fail unless there is another similar filter that is not identical and can push one of the identical filters to another state that allows it to start a specialization.

The regularization has been implemented within the Caffe CNN framework [14] using the NVIDIA Cuda library in the form of a new type of convolutional neural

network for which the following parameters have to be specified in the network model:

	Description
sigma	The σ^2 for the RBF
gamma	To decrease sigma
wlr	The "learning rate" for the weight regularization.
blr	The "learning rate" for the bias regularization.
steps	The number of random pairs to regularize per epoch.
IMED-ksize	The IMED kernel size. Zero is equivalent to the euclidean distance.

The regularization in the frequency domain has been implemented as well using the cuFFT library, but the results are very unstable due precision errors during the conversion from the Caffe data types to the cuFFT ones. Nevertheless, the frequency domain regularization has been implemented on python and some results will be shown as well in the next sections.

Thousands of configurations have been tried during the thesis but, for the sake of clarity, a very brief summary of the continuous regularization algorithms are shown on table 4.1.

Model	σ	α	**Kernel Size**	**Layer**	**accuracy**
cifar10_full	0	0	0	-	82.3% ($\pm$0.28%)
cifar10_full	Auto	10^{-5}	3	conv1	82.3% ($\pm$0.34%)
cifar10_full	Auto	10^{-5}	5	conv1	82.3% ($\pm$0.28%)
cifar10_full	Auto	10^{-4}	3	conv1	82.3% ($\pm$0.15%)
cifar10_full	Auto	10^{-4}	5	conv1	82.3% ($\pm$0.05%)

Table 4.1: Brief summary of the results. When σ = Auto, this value starts being the best found for a normal distribution and gradually changes to be the best one found for the already learned filters (with grid-search). This method has been used because the grid-search is too expensive to run it every iteration. Finally, it can be seen that the continuous regularization does not improve the best accuracy. There are different explanations for this, one of them being that there is not enough redundancy in this model in contrast to the Imagenet one.

4.2.2 Single-stage regularization

As it has been seen, the continuous regularization presents the problem of finding good hyper-parameters for regularizing during each of the stages of the learning. To solve this problem, it has been implemented a grid-search algorithm that determines the best number of iterations, the σ^2 and the learning rate so that the mean nearest neighbor variance is minimized. Then the Caffe python interface has been used to regularize the filters of a given model before continuing the training. As said before, another problem of regularizing only once during all the training process could be that, when done during initial stages, the rest of the training will revert the regularization and no improvement or deterioration will be appreciated in the final

performance. On the other hand, if the regularization is performed in late stages, it is possible that some filters have already converged to very similar values and the regularization cannot differentiate them enough. As said, this regularization doesn't seem to have any effect on the accuracy:

Model	σ	α	**Method**	**Layer**	**It. bef. reg.**	**accuracy**
cifar10_full	-	-	-	-	80K	82.2%
cifar10_full	Auto	Auto	Any	conv1	Any	82.2%
cifar10_full	Auto	Auto	Any	all	Any	82.2%

Table 4.2: Results of applying automatic regularization after normal training of the network for 10K, 20K, 30K, 40K, 50K, 60K, 65K and 70K and continue learning until reaching the 80K iterations. We can see the regularization is not changing the accuracy at all. The different applied methods are the Fourier transform, $IMED_{3x3}$, $IMED_{5x5}$ and mixtures.

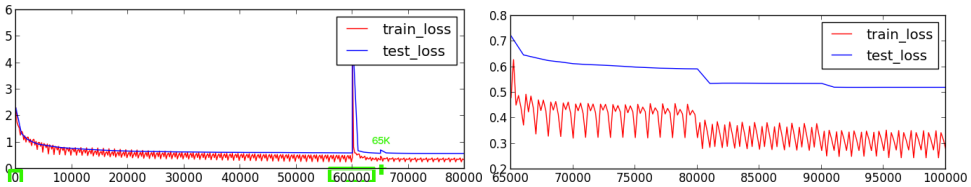
4.2.3 Multi-stage regularization

To overcome the problems of single-stage regularization and the continuous one, one can stop the training more than once to rectify the filters and continue training normally. Since the regularization is a fast process in comparison to the rest of the training, it can be included without increasing the training time too much. For this kind of regularization, one would expect to have a loss peak just after regularization and a fast recovery since the non-redundant filters that have already been learned do not suffer from dramatic changes using this regularization technique. This is due to the use of the RBF function to limit the area of influence. This hypothesis is confirmed on figure 4.4 on the following page although the ideal result would be a fast recovery that eventually surpasses the original performance. Again, it is possible that for this model this is not possible since there is no variance to reduce. Figure 4.4 on the next page shows how the training process is when using multistage regularization.

Seen all this, there doesn't seem to exist any improvement after regularizing the filter weights. However, the hypothesis for this thesis cannot be discarded until it's tried in a network that evidences much more redundancy than the Cifar one. So, the next step should be to try the regularization over the Imagenet Model.

Model	IMED Kernel	Freq. domain	Layer	Accuracy
cifar10_full	-	-	-	82.29% ($\pm, 0.28\%$)
cifar10_full	-	Yes	all	79.51%
cifar10_full	3	No	all	80.22%
cifar10_full	5	No	all	79.54%
cifar10_full	5	Yes	all	78.97%
cifar10_full	-	Yes	conv1	81.44%
cifar10_full	3	No	conv1	80.16%
cifar10_full	5	No	conv1	80.99%
cifar10_full	5	Yes	conv1	80.30%

Table 4.3: Results after regularizing the initial weights, after 60K iterations and 65K iterations. The results seem to underperform the original model but, in fact, if one lets the network run for more iterations, it arrives at the original performance (but not better). The standard deviation for the rest of models is not given because, given the difference of accuracy, it was not worthy to run them more than twice.



(a) Full training with regularization at 0, 60 and 65K iterations.

(b) Fine-tuning after the 65K regularization in (a).

Figure 4.4: (a) shows the multistage cifar10_full training accuracy and loss for 80K iterations with $IMED_{5x5}$ regularization on all the layers. One can see the expected peaks where the regularization was performed. It is also interesting to see how fast it recovers after the weights were regularized. (b) shows that, letting the network train for some more iterations, it returns to the original non-regularized accuracy. The fact that the second regularization peak is so small, even if the network has already recovered from the first one, means that the filters are in effect converging to a different local optima without having an impact to the performance. All this shows that the regularization is indeed working like with the 2D toy example, specializing the similar filters but preserving the global shape of their distribution and thus, not destroying much of the already-learned non-redundant information.

4.2.4 Imagenet Regularization

For this section, the `caffenet` pre-learned model will be regularized in single-stage mode and it will be fine-tuned to see if it converges to a better local minimum. The regularization has been applied to all the "conv" layers of `caffenet` using $IMED_{5x5}$, since it is the technique that produces better visualizations with good mean nearest neighbors variance in the previous chapter. Then the network has been fine-tuned for 300K iterations with decreasing learning rate from 10^{-3} to 10^{-5}, descending every 100K iterations, achieving a performance of **57.97%** of accuracy over the validation set during the 287K iteration. Although this seems a lot of iterations, when looking

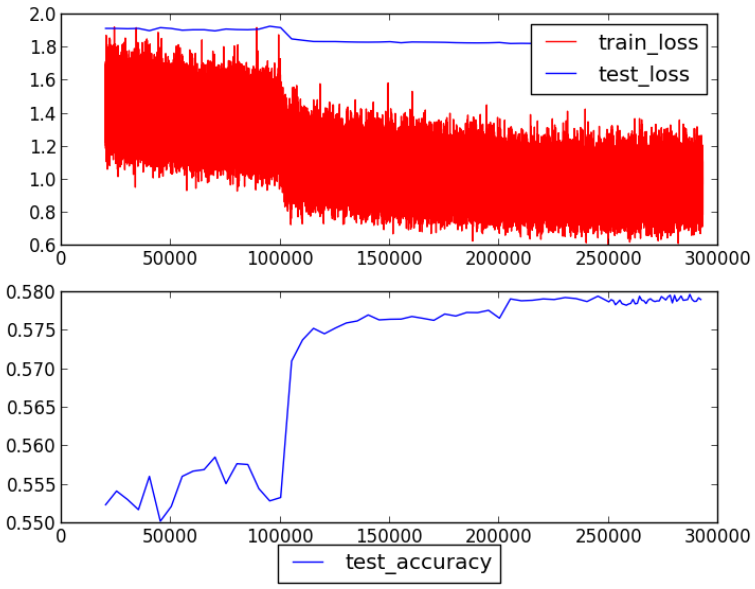
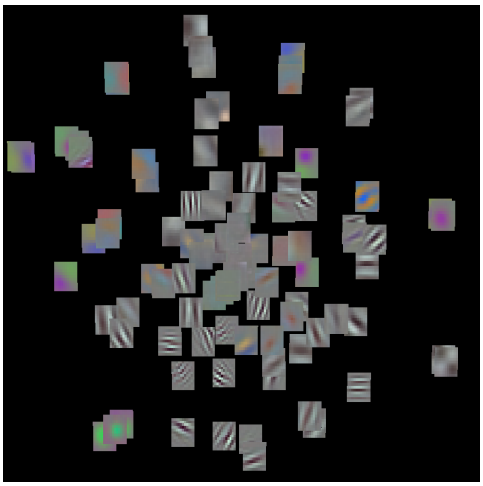


Figure 4.5: Plot of the accuracy over the imagenet dataset. It can be seen how the 57.97% of accuracy is obtained in the 287K iteration. It can also be seen that the network is overfitting and thus, it might have margin to reduce the variance present on the convolution layers.

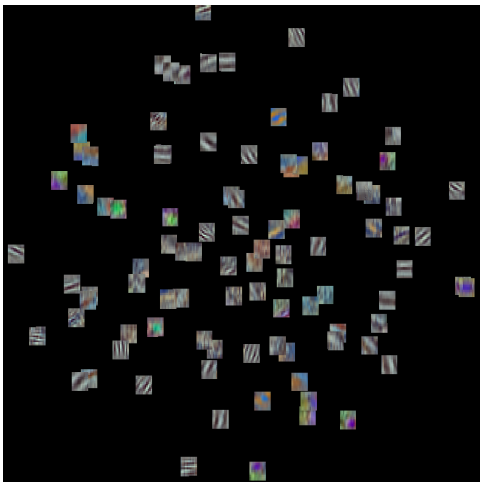
at the plot of the accuracy on figure 4.5 we can see that it is possible that it could be reduced by starting with a smaller learning rate. In fact, there is the possibility that starting with such high learning rate might be undoing the regularization, however, trials are too expensive to discard this possibility within the schedule of the project.

To sum up, the regularized Cifar10 models do not seem to be improved by the regularization because they might not have as much margin in terms of bias and variance as such a big network as Imagenet. However, the latter could have been improved by regularization since its accuracy after fine-tuning (57.97%) is higher than the accuracy after performing the same fine-tuning on the original network (57.8%). However, this needs a further verification because, even the reference and the regularized model share the same initial filters, dropout (Nitish Srivanstava *et al.* , 2012) [35] is introducing some randomness to the learning process.

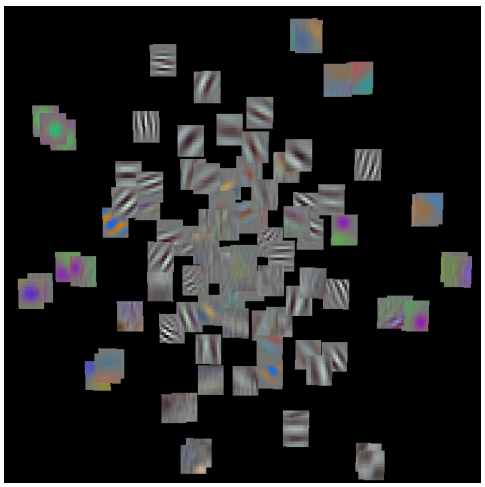
In the case that the regularization is the real cause for this improvement, it should be tried on the state-of-the-art models to see if it can improve their performance. It is also possible that with a different setup, with multi-stage regularization or the continuous one, even maybe with different hyper-parameters, one could get better improvements.



(a) Original Imagenet pre-learned filters.



(b) Regularized Imagenet pre-learned filters.



(c) Imagenet filters after 280K iterations.

Figure 4.6: T-sne visualization of the filters at different stages. This shows that in iteration 280K, the filters are converging to a state that is similar to the original one, but the filters are more uniformly spread in the center.

Chapter 5

Conclusion

This thesis has exposed the problem of quantifying and reducing the redundancy of the filters of the Convolutional Neural Networks by (i) using visualization techniques to illustrate the problem, (ii) defining a measure to quantify this redundancy and (iii) proposing and testing regularization methods to reduce it.

For the first one, the T-sne technique released by L. van der Maaten and G. Hinton (2008) [39] has resulted to be a good indicator for filter redundancy since the similar filters in the projected 2D space are always near.

For the second one, some literature has been reviewed in order to assess the spatial distribution of high dimensional data but finally, the use of the nearest neighbor variance has shown to be easy and powerful enough for this problem and it has been proved to work over 2D toy examples.

Finally, a literature review has been performed in order to find a good differentiable similarity measure. This is crucial since it has been shown that the euclidean distance is not likely to work well for this problem because it is very sensitive to small translations, deformations, and scaling. From all the reviewed distances, the Image Euclidean Distance (IMED) from Bing Sun and Jufu Feng (2008) [36] and the euclidean distance in the frequency domain have been selected because they are more robust than the euclidean distance, they are easy to implement and they are faster to compute than the rest of the metrics. A PCA distance has been considered as well, but it has been discarded like the others. These similarity metrics has been adapted and differentiated in order to use them to descend the gradient of the similarity between filters. Furthermore, the algorithm 2 has been proposed to introduce the regularization in the backpropagation gradient. This algorithm can and also has been applied directly on the filters in batch mode before fine-tuning. In addition to all this, the algorithm 3 has also been provided in order to find the best hyper-parameters that minimize the mean nearest neighbor variance objective. It is important to remark the fact that applying a RBF function to limit the radius of influence of the compared filters has been crucial to preserve the weights of the non-redundant filters and to prevent the filters from expanding to regions (in the n-dimensional weight space) that are unlikely to be good.

All these methods have been implemented using the python language, resulting in

a complete framework for filter redundancy analysis. On the top of this, for efficiency reasons, some of the regularizations have been implemented in C++, resulting in a new kind of convolutional layer in the Caffe framework [14] using the CUDA parallel computing platform.[1]

Focusing into the experimental results, it has been successfully shown that there indeed exists redundancy in the Imagenet filters and the regularization methods have been shown to dissolve filter clusters in a sandbox. However, when tried over the real Cifar10 [16] training, it doesn't seem to have any impact on its performance. This can be due to the fact that the network used to train Cifar10 is too small to have such redundancy problems. It has also been shown as well that the model does not have enough variance that can be reduced using the explained regularization techniques.

In the case of pruning, a method for removing redundant filters based on the T-sne visualization has been used to merge the most similar filters. Note that T-sne has been used because its results seemed visually intuitive but any distance function, like IMED, could be used in its place. The pruning has been tried over the `caffenet` model, a pre-trained CNN provided by the Caffe framework on the Imagenet dataset, showing that it is possible to reduce the redundancy by pruning the 10% of the filters of the first four layers, reducing the network size in 20MB, without losing accuracy. It has also been shown that pruning results in a reduction of the overfitting.

Finally, the best regularization technique has been tried over Imagenet achieving a slightly better results than the best result obtained by the original reference model. This improvement can still be explained by the randomness introduced by dropout [35] and it is also possible that the effects of regularization have been removed by using a too big learning rate for fine-tuning. In the case that it is verified that this improvement is due to regularization, there is still a huge margin to further improve it since there can still be tried the multi-stage and continuous regularizations and there are still many parameters to tune. This leads us to the further research that can be done after this project since it requires so much time that it goes out of the timing possibilities for this Master's thesis:

- Study the effects of pruning over the state-of-the-art neural networks, which are more based on convolution layers.
- Apply continuous and multi-stage regularizations over Imagenet.
- Try to find a better and more robust similarity function.
- Find better of hyper-parameters like the IMED sigma, per-layer learning rate, etc.
- Perform the analysis over the activations instead of the filters.
- Try to have better insight of how is redundancy affecting higher convolution layers.

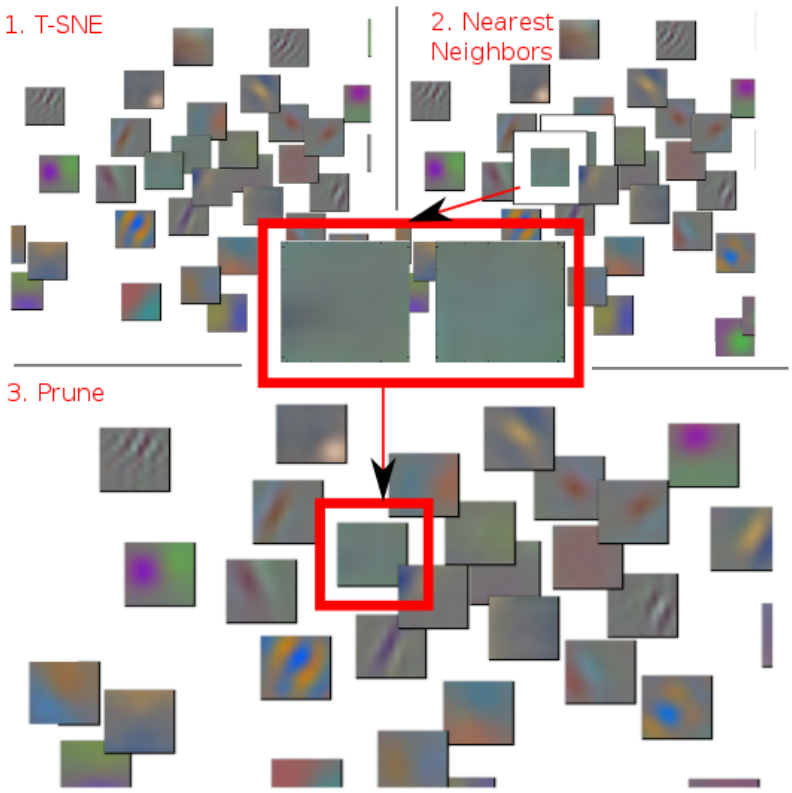
[1] http://www.nvidia.com/object/cuda_home_new.html

- Assess the effects of the network depth on the network redundancy.

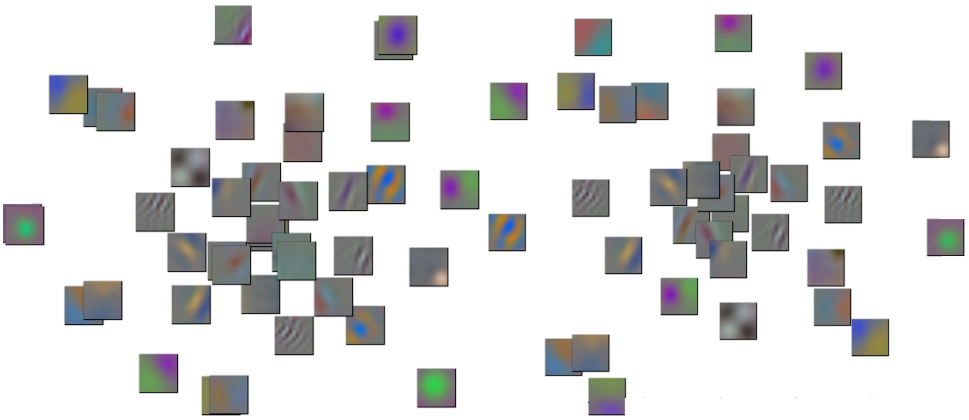
Appendices

Appendix A

Additional Figures



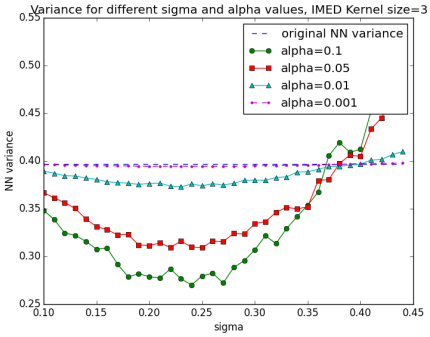
(a) Illustration of the pruning algorithm (real execution time visualizations).



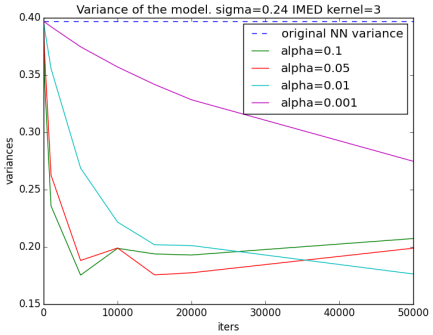
(b) Visualization of `caffenet` *filters before pruning.*

(c) Visualization of `caffenet` *filters after pruning.*

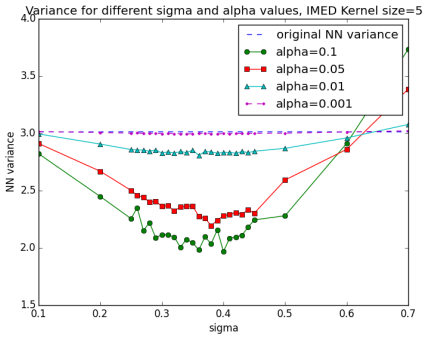
Figure A.1: Illustration of the pruning procedure. (a) shows the step that is repeated n times in order to prune the network. (b) shows the original 48 filters. (c) shows the pruned layer that now has 38 filters, instead of the initial 48.



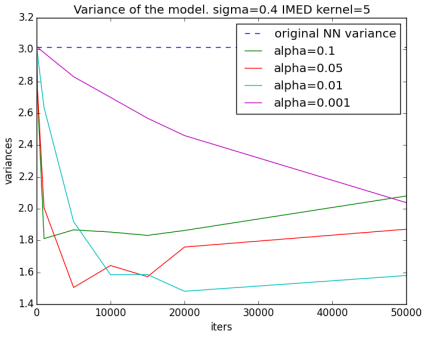
(a) Finding best σ^2 for $IMED_{3x3}$



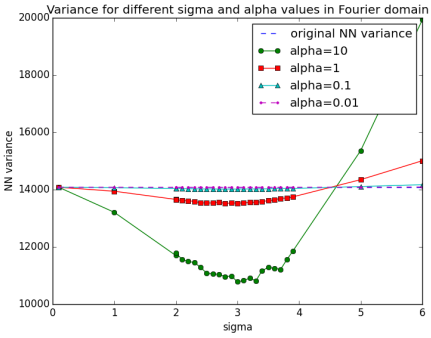
(b) Finding best α and #iters for $IMED_{3x3}$



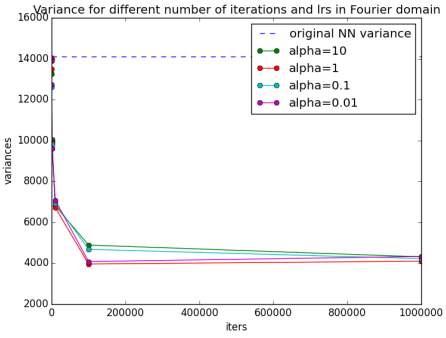
(c) Finding best σ^2 for $IMED_{5x5}$



(d) Finding best α and #iters for $IMED_{5x5}$

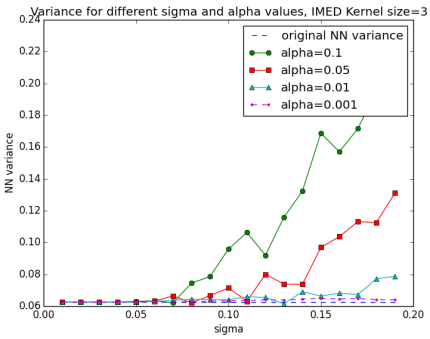


(e) Finding best σ^2 with FFT.

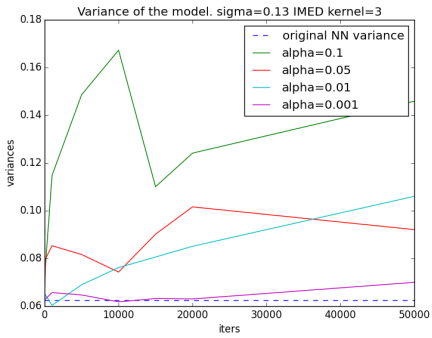


(f) Finding best α and #iters with FFT, $\sigma^2 = 0.56$.

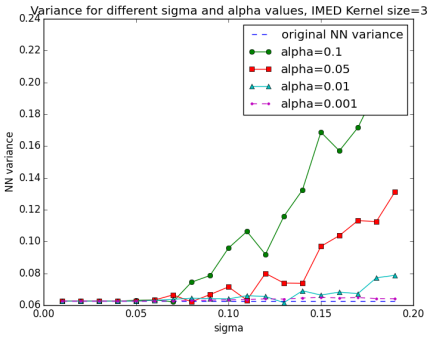
*Figure A.2: Imagenet variance optimization and hyper-parameter tuning. Figures (a) and (b) show the tuning of the hyper-parameters and the best-achieved variance using an IMED kernel of 3x3. (c) and (d) show the same for a 5x5 kernel. (e) and (f) use the regularization on the frequency domain. One way to compare them is to see which is the relative improvement w.r.t. their base variances, i.e. a **56.41%** for $IMED_{3x3}$, a **50%** for $IMED_{5x5}$ and a **71.42%** for Fourier. This means that there might be more margin to reduce the variance in te frequency domain. The above-mentioned "sigmas" are in fact σ^2.*



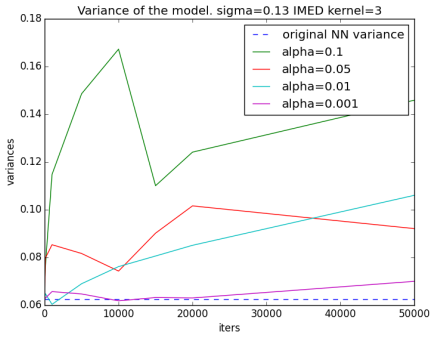
(a) Finding best σ^2 for $IMED_{3x3}$



(b) Finding best α and #iters for $IMED_{3x3}$

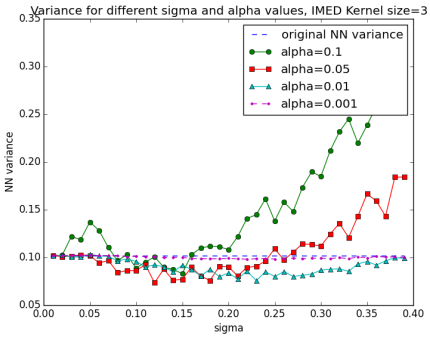


(c) Finding best σ^2 for $IMED_{5x5}$

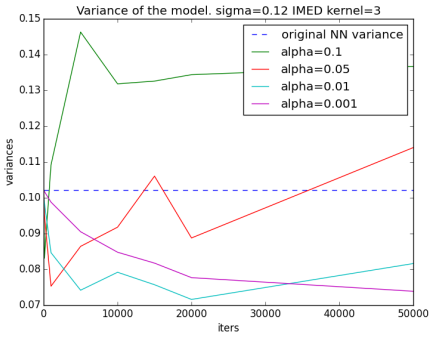


(d) Finding best α and #iters for $IMED_{5x5}$

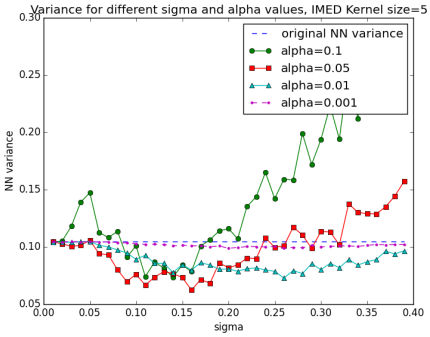
Figure A.3: Cifar10 variance optimization and hyper-parameter tuning. Figures (a) and (b) show the tuning of the hyper-parameters and the best-achieved variance using an IMED kernel of 3×3. (c) and (d) show the same for a 5×5 kernel. The same happens using the Fourier transform.



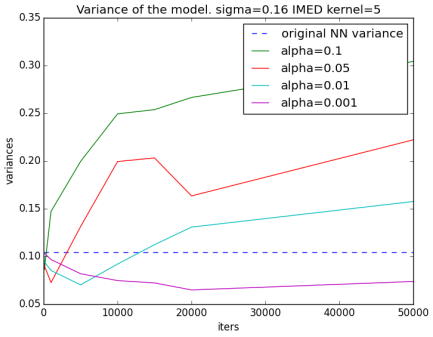
(a) Finding best σ^2 for $IMED_{3x3}$



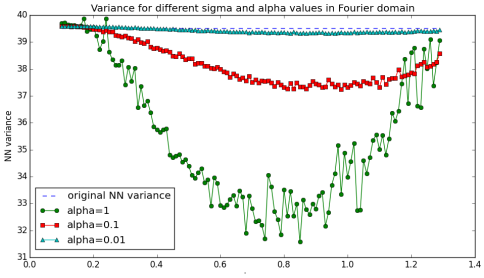
(b) Finding best α and #iters for $IMED_{3x3}$



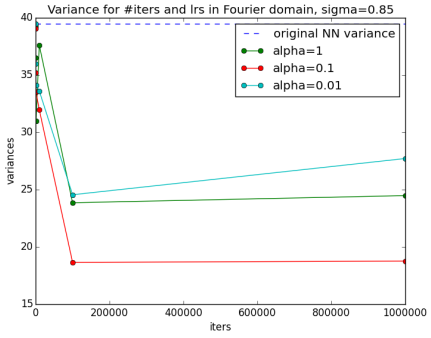
(c) Finding best σ^2 for $IMED_{5x5}$



(d) Finding best α and #iters for $IMED_{5x5}$



(e) Finding best σ^2 with FFT.



(f) Finding best α and #iters with FFT, $\sigma = 3$.

*Figure A.4: Cifar10 variance optimization and hyper-parameter tuning. Figures (a) and (b) show the tuning of the hyper-parameters and the best-achieved variance using an IMED kernel of 3x3. (c) and (d) show the same for a 5x5 kernel. (e) and (f) use the regularization on the frequency domain. One way to compare them is to see which is the relative improvement w.r.t. their base variances, i.e. a **30.39%** for $IMED_{3x3}$, a **40.9%** for $IMED_{5x5}$ and a **52%** for Fourier. This means that there might be more margin to reduce the variance in te frequency domain and that 96 filters present indeed more redundancy than 32. However, it is still less than the margin with the Caffenet model. This could be accounted by the fact that filters are smaller as well.*

Bibliography

[1] P. Agrawal, R. Girshick, and J. Malik. Analyzing the performance of multilayer neural networks for object recognition. 2014.

[2] D. Attwell and S. B. Laughlin. An energy budget for signaling in the grey matter of the brain. *J Cereb Blood Flow Metab*, 21(10):1133–45, 2001.

[3] J. A. Bagnell and D. M. Bradley. Differentiable sparse coding. In *NIPS*, pages 113–120. Curran Associates, Inc., 2008.

[4] A. E. Bryson and Y. C. Ho. *Applied Optimal Control*. Blaisdell, New York, 1969.

[5] L. Chen and H. Y. H. Chuang. A fast algorithm for euclidean distance maps of a 2-d binary image. *Inf. Process. Lett.*, 51(1):25–29, 1994.

[6] T. Cover and J. Thomas. *Elements of Information Theory*. Wiley and Sons, 1991.

[7] N. A. C. Cressie. *Statistics for spatial data*. Wiley series in probability and mathematical statistics. J. Wiley & Sons, New York, Chichester, Toronto, 1993.

[8] Y. L. Cun, J. S. Denker, and S. A. Solla. Optimal brain damage. In *Advances in Neural Information Processing Systems*, pages 598–605. Morgan Kaufmann, 1990.

[9] M. Everingham, L. Van Gool, C. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, 88(2):303–338, 2010.

[10] K. Fukushima. Neocognitron: A self-organizing neural network model for a mechanish of pattern recognition unaffected by shifts in position. *Biological Cybernetics*, 36:193–202, 1980.

[11] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In G. J. Gordon, D. B. Dunson, and M. Dudk, editors, *AISTATS*, volume 15 of *JMLR Proceedings*, pages 315–323. JMLR.org, 2011.

[12] T. S. Han. Nonnegative entropy measures of multivariate symmetric correlations. *Information and Control*, 36(2):133–156, February 1978.

[13] M. Jaderberg, A. Vedaldi, and A. Zisserman. Speeding up convolutional neural networks with low rank expansions. In *British Machine Vision Conference*, 2014.

[14] Y. Jia. Caffe: An open source convolutional architecture for fast feature embedding, 2013.

[15] Z. L. Z. Q. T. R. Y. J. Kam, Kin Ming. Technical paper. *Journal of Manufacturing Systems*, 32(1):154–166, 2013.

[16] A. Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. Burges, L. Bottou, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[18] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, volume 86, pages 2278–2324, 1998.

[19] P. C. Mahalanobis. On the generalized distance in statistics. *Proceedings of the National Institute of Sciences (Calcutta)*, 2:49–55, 1936.

[20] W. J. McGill. Multivariate information transmission. *Trans. of the IRE Professional Group on Information Theory (TIT)*, 4:93–111, 1954.

[21] M. Minsky and S. Papert. *Perceptrons*. MIT Press, Cambridge, MA, 1969.

[22] A. Nakhmani and A. Tannenbaum. A new distance measure based on generalized image normalized cross-correlation for robust video tracking and image recognition. *Pattern Recognition Letters*, 34(3):315–321, 2013.

[23] A. Ng, J. Ngiam, C. Y. Foo, Y. Mai, and C. Suen. UFLDL Tutorial. http://ufldl.stanford.edu/wiki/index.php/UFLDL_Tutorial, 2013. [Online; accessed 22-October-2014].

[24] K. Pearson. On lines and planes of closest fit to points in space. *Philos. Mag*, 2,:559–572, 1901.

[25] T. H. Pham, T. B. Ho, Q. D. Nguyen, D. H. Tran, and V. H. Nguyen. Multivariate mutual information measures for discovering biological networks. In *RIVF*, pages 1–6. IEEE, 2012.

[26] G. Rigoll. Maximum mutual information neural networks for hybrid connectionist-hmm speech recognition systems. *IEEE Transactions on Speech and Audio Processing*, 2(1):175–184, 1994.

[27] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[28] D. E. Rumelhart, G. E. Hinton, and R. J. Wilson. Learning representations by back-propagating errors. *Nature*, 323:533–536, 1986.

[29] D. B. Russakoff, C. Tomasi, T. Rohlfing, and C. R. M. Jr. Image similarity using mutual information of regions. In *ECCV (3)*, volume 3023 of *Lecture Notes in Computer Science*, pages 596–607. Springer, 2004.

[30] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge, 2014.

[31] C. E. Shannon and W. Weaver. A mathematical theory of communication. *The Bell System Technical Journal*, 27:379–423,623–656, July, October 1948.

[32] P. Simard, Y. LeCun, and J. S. Denker. Efficient pattern recognition using a new transformation distance. In *Advances in Neural Information Processing Systems 5, [NIPS Conference]*, pages 50–58, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.

[33] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[34] S. Srivansa. A review on multivariate mutual information. In *Notre Dame EE-80653 Information Theory Tutorials*. Notre Dame, 2005.

[35] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, Jan. 2014.

[36] B. Sun and J. Feng. A fast algorithm for image euclidean distance, Oct. 2008.

[37] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *CoRR*, abs/1409.4842, 2014.

[38] T. Van de Cruys. Two multivariate generalizations of pointwise mutual information. In *Proceedings of the Workshop on Distributional Semantics and Compositionality*, DiSCo '11, pages 16–20, Stroudsburg, PA, USA, 2011. Association for Computational Linguistics.

[39] L. van der Maaten and G. Hinton. Visualizing high-dimensional data using t-sne. 2008.

[40] L. Wang, Y. Zhang, and J. Feng. On the euclidean distance of images. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(8):1334–1339, 2005.

[41] S. Watanabe. Information theoretical analysis of multivariate correlation. *IBM J. Res. Dev.*, 4(1):66–82, Jan. 1960.

[42] P. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. PhD thesis, Harvard University, 1974.

[43] P. L. Williams and R. D. Beer. Nonnegative decomposition of multivariate information. *CoRR*, abs/1004.2515, 2010.

[44] M. Wu. *Entropy-based 2D Image Dissimilarity Measure*. University of Texas–Pan American, 2005.

[45] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. *CoRR*, abs/1311.2901, 2013.

Master thesis filing card

Student: Pau Rodríguez López

Title: Reducing Redundancy in Deep Convolutional Neural Networks

UDC: 621.3

Abstract:
After the breakthrough of Alex Krizhevsky's convolutional neural network for object recognition (CNN) in 2012 [17], there has been a lot of emphasis in finding a model that has the same or better performance using less parameters in order to reduce the need of huge computational resources. This has been tackled from the point of view of the network filters structure in ILSVRC2013 by Mathew Zeiler (2013) [45], and also building very deep models with very small convolutional kernels in ILSVRC2014 (Szegedy et al. , 2014) [37] (Simonyan and Zisserman, 2014) [33]. However, the problem of the redundancy in the parameters of CNNs is not directly studied in any of the mentioned papers and nowadays, it is important that our models are as efficient as possible in order to adapt to the trend of mobile computing. Thus, this thesis focuses on the problem of finding the source of redundancy in CNNs and proposes some methods to visualize and measure it, as well to reduce it. These methods will be based on the filter weights since it is much more efficient than performing an statistical analysis over the filter activations and will result in a framework for filter analysis, pruning and regularization. The researched techniques will finally be tried for pruning over Imagenet, showing that it is possible and opening the door for trying it over the state-of-the-art models. Regularization techniques have also been tried over Cifar10 and Imagenet, showing that Cifar10 is not complex enough to be regularized but obtaining more promising results on a pre-learnt Imagenet model present in the Caffe framework [14], which encourage to work on their verification and to do further research.

Thesis submitted for the degree of Master of Science in Artificial Intelligence, option Engineering and Computer Science

Thesis supervisor: Prof. Bart de Moor

Assessor: Peter Roelants

Mentor: Peter Roelants