

Impact of coherence and occurrence in the training of CNN for anomaly detection

Dissertation presented by
Cécile Hautecoeur

for obtaining the Master's degree in
Mathematical Engineering

Supervisor(s)
Laurent Jacques
Christophe de Vleeschouwer

Reader(s)
Simon Carbonnelle,

Academic year 2017-2018

Abstract

The CNNs and other Deep Learning methods are more and more used to solve supervised machine learning tasks such as classification. Most of the researches about these methods concern the improvement of the model itself or the analyse of the behaviour of CNNs for a given problem. However, in this work, we consider the impact of the training data on the problem of anomaly detection using CNNs. We constructed for this purpose synthetic images that consist of a background similar for all images and a foreground, or anomaly, which appears only on certain images. We observed the impact of two main parameters on the classification accuracy of a CNN model. First, we looked at the influence of the coherence between background and foreground signals. The coherence is a kind of measure of the similarity between two sets of signals. Then, we looked at the importance of the frequency of occurrence of each class in the training set. We noticed that images with high coherence are faster to learn but have an unstable generalisation on unknown data, while images with low coherence, a priori easier to distinguish, have a slower learning but a good generalisation on unknown data. An in-depth analysis of the learning behaviour, through notably the performances of a trained network on images containing only foreground, highlighted that the different behaviours are due to a difference in the features learned by the CNN. On images with low coherence, the used CNN learns essentially the foreground, while on images with high coherence it learns the background as well. Surprisingly, it is not evident that the observed behaviour of the CNN is linked to the coherence between background and foreground. Other indicators such as the evolution of the coherence inside the network, or the spatial support of the foreground, or even the frequency of occurrence of discriminating elements in the training dataset, seem more appropriate. Nevertheless, this work illustrates that the content of the training dataset impacts significantly the behaviour of a CNN.

Résumé

Les réseaux de neurones convolutifs, ou CNN en anglais, ainsi que les autres méthodes d'apprentissage profond, sont de plus en plus utilisés en apprentissage automatique, notamment pour classifier des données. La plupart des recherches sur ces méthodes concernent l'amélioration du modèle utilisé ou l'analyse du comportement des CNN pour un problème donné. Cependant, ce mémoire va s'intéresser à l'impact des données d'entraînement sur un CNN construit pour détecter des images contenant une anomalie. Pour cela, nous avons construit des images synthétiques constituées d'un arrière-plan toujours similaire et d'un éventuel avant-plan, ou anomalie. Nous avons analysé l'impact de deux paramètres principaux sur la précision atteinte par le CNN dans sa classification. D'abord, nous avons regardé l'influence de la cohérence entre les signaux d'arrière et d'avant plan. La cohérence est en quelque sorte une mesure de la similarité entre deux ensembles de signaux. Nous avons également regardé l'importance de la fréquence d'occurrence de chaque classe, à savoir les images avec ou sans avant-plan, dans l'ensemble d'images d'entraînement. Nous avons pu observer que les images avec une grande cohérence demandent moins de temps d'apprentissage par le CNN, mais celui-ci généralise mal sur des images inconnues. Au contraire, les images avec une faible cohérence, à priori plus faciles à classer, nécessitent un plus long apprentissage, mais le CNN généralise très bien. Une analyse plus approfondie du comportement du CNN durant et après l'entraînement, à travers notamment l'observation de ses performances sur des images contenant uniquement une anomalie sans arrière-plan, a mis en avant que le CNN ne discrimine pas de la même manière les deux classes dans ces deux cas. En effet, les CNN apprennent principalement l'avant plan quand la cohérence est faible, alors qu'il apprend également beaucoup de caractéristiques de l'arrière-plan dans le cas contraire. Néanmoins, il n'est pas évident que les comportements observés sont liés à la cohérence entre l'arrière et l'avant plan. D'autres indicateurs, tels que l'évolution de la cohérence dans le CNN ou le support spatial de l'avant-plan ou encore la fréquence d'occurrence d'éléments discriminants dans l'ensemble d'entraînement paraissent plus appropriés. Cependant, ce travail montre que le contenu de l'ensemble d'entraînement influence de manière significative le comportement des CNN.

Contents

1	Introduction	2
2	Preliminaries	5
2.1	Notation	5
2.2	Deep neural networks	5
2.3	Convolutional neural networks	6
2.4	Image processing tools	12
2.4.1	Sparse image representation using dictionaries	12
2.4.2	Mutual coherence	14
3	Generation of synthetic datasets	18
4	Classification results for different generative models	20
4.1	Impact of the mutual coherence between background and foreground dictionaries	20
4.1.1	Observation of the behaviour of the CNN	20
4.1.2	The features learned by the CNN	23
4.2	Impact of the frequency of occurrence	25
4.2.1	When the foreground and the background have incoherent dictionaries . .	25
4.2.2	When the foreground and the background have coherent dictionaries . . .	28
4.3	Observations with a limited range of possible foreground	30
5	Dissecting the CNN training	32
5.1	The evolution of the learned weights	32
5.2	Evolution of the gradients back-propagated	37
5.3	The filters contents	42
6	Analyse of the generalised coherence inside the network	45
6.1	The generalised coherence	46
6.2	Experimental results	47
6.3	Comments on the choice to use mutual coherence to characterise the datasets . .	50
7	Conclusion	52
	Appendices	58
A	Theoretical bound on the coherence	59
B	Complete description of the experimental steps	63
B.1	Detailed first tests	63
B.2	Frequency results	66

Chapter 1

Introduction

Nowadays, computers and other machines are full members of our daily life. They help us in several tasks, common or not. Typically, to solve a certain problem using a machine, humans must formalise it first and find appropriate algorithms. Then, the problem must be encoded in the machine as well as the instructions that form the algorithm. Once this is done, the machine is finally able to solve the problem. However, these days we produce more and more machines with an "artificial intelligence" that are able to learn from their surroundings.

For example, if you create an account on a web-site allowing you to score movies, the site will soon advise you a few other movies that you are likely to appreciate too, based on your preferences but also on the ratings of the other users. Often, the suggestions of the website are quite accurate. In this example, no-one took the time to model your preferences, it is done by an algorithm. Moreover, the suggested movies change if you add new scores, but also following their success. Indeed, if two comparable movies have been noted differently by the users of the website, you are much more likely to see the best noted in your suggestions.

From the traditional approach of problem-solving, this method skips the problem-based modelling by using the same model for all problems, where only the inputs given to the algorithm change. Moreover, the algorithm is able to adapt and improve its performances using new inputs. The branch of computer science that develop algorithms able to learn from their inputs is named machine learning.

Of course, there is not algorithm able to solve all the problems. Nevertheless, a set of methods named the Deep Neural Networks (DNNs) revolutionised the machine learning and brought back the dream to construct an artificial intelligence as good as, or even better, than the human's one [22]. The Deep Neural Networks are now used in a lot of different fields where they equalise or outperform traditional methods. For example, they are employed in image classification [20] or detection [13], face or speech recognition [27] [16], text translation [32], games playing [31], and for many other applications. When the input data are images, it is common to use Convolutional Neural Networks (CNNs) that are a part of DNNs.

In this work, we focus on supervised CNN in the content of image classification. The word "supervised" indicates that the selected method is trained on examples with know class membership. The CNNs are called "Neural" Networks as they were inspired by the human brain. Each component of such method is thus often called a neuron. Two distinct CNN methods can differ from each other by the optimisation scheme used to learn, but also by the disposition of the "neurons" or basic elements. Often, researchers attempt to optimise the selected model with respect to their problem [36]. However, Arpit and al. have recently pointed that input data themselves influence the performances of the algorithm [1].

For example, we have learned a CNN, described in Sec. 2.2 to classify two classes of images in two different cases: a first test without background, and a second with a periodic background. The two different classes of images were constructed by summing three elements from the Haar wavelet dictionary defined in 2.2 at scale 3 (first class) and at scale 5 (second class). The backgrounds of the second test were constructed by summing 16 elements from the DCT basis defined in 2.1 and adding it to the image. Fig.1.1 shows examples of images from each test. For humans, the second case is harder than the first one, but is it the same for Neural Networks?

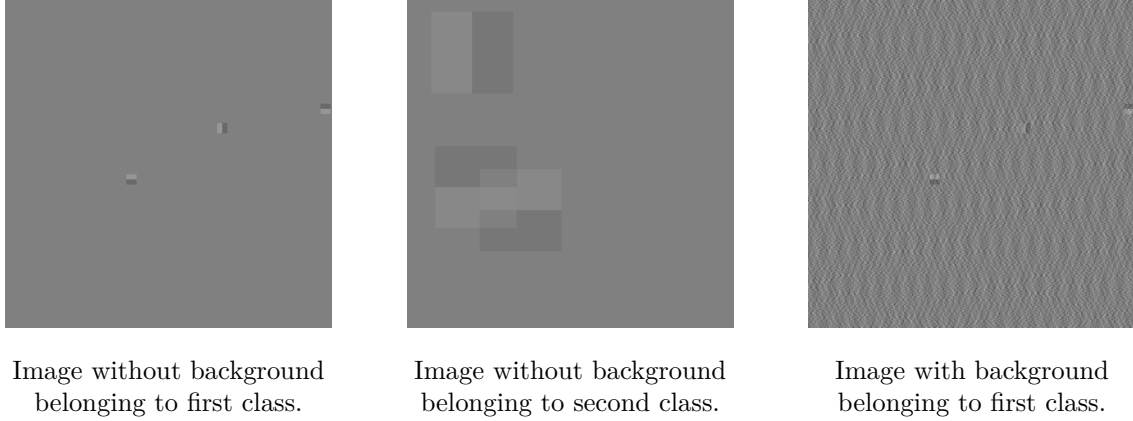


Figure 1.1: Examples of images from each class. Note that the intensities of these images have been slightly increased to distinguish them more easily.

We can observe in Fig.1.2 that the chosen network learns better its task when there is no background added to the images, and obtains a higher *accuracy* during the training phase. During this phase, the network receives images and their real classification. Each time a network has seen all the training images, we said that it has done an *epoch*. The training phase lasts usually a few dozen of epochs, and the accuracy is the percentage of training images correctly classified by the algorithm. The training phase is very important in supervised machine learning, since it is during this phase that the algorithm fixes all its intern parameters and learns its task. The learning of CNNs is explained in details in Sec. 2.2.

However, when we provide unknown data, i.e test data, to the CNN we can observe that the accuracy of the network trained on images without background stays very low, while the network trained on images with background is able to *generalise*, to classify with high accuracy unknown data. An algorithm only able to classify known data is doing *memorisation* instead of learning the task, and this is not at all the goal of machine learning. Therefore, we can conclude that classifying images with a background is easier for the chosen network.

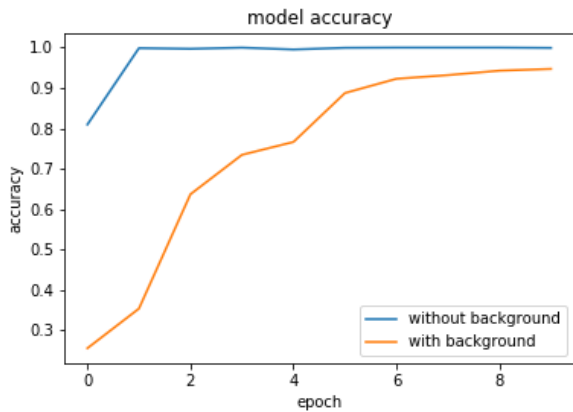


Figure 1.2: Evolution of the accuracy of the network during training.

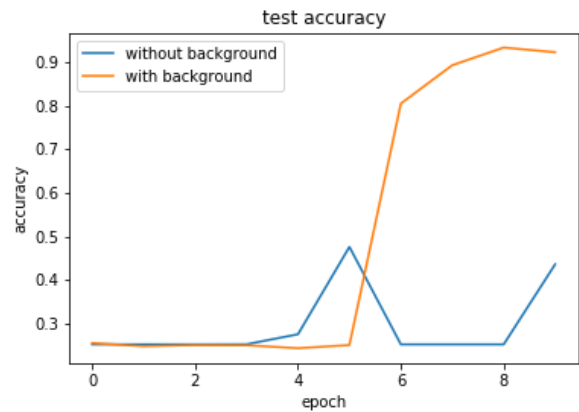


Figure 1.3: Evolution of the accuracy of the network on unknown data.

We observed that for a fixed problem the aspect of the data to analyse influences the performances of the algorithm as well as the model chosen. In this master thesis, we consider the influence of the input data for the following problem: determine the presence or the absence of a foreground, or anomaly, on images with similar backgrounds. In practice, this problem can be linked to the defect-detection or anomaly-detection problem, where the defects, or anomaly, are the foreground and the normal objects the background.

For this purpose, we decided to use a fixed model that has been designed for this problem and is thus efficient. We then compared the performances of the network when different kind of inputs are provided to it. CNNs in general and the network that we chose in particular are explained in detail in Chapter 2.

To differentiate the inputs, we constructed datasets using mutually coherent or incoherent dictionaries [9] for the background and the foreground. In a few words, the mutual coherence between two sets of functions (or dictionaries) is a measure of their similarities. Having mutually coherent dictionaries is an issue for several algorithms in image processing, to describe sparsely images in a combination of dictionaries (as in Basis Pursuit [9]), or to separate the background from the foreground (as in Morphological Component Analysis [2]). But Xin and al. demonstrated that it is possible to get over this issue using Neural Networks with hand-crafted structure [37]. Quite naturally, we wonder what is the impact of the mutual coherence between background and foreground dictionaries on the performances of general CNNs.

We used a Discrete Cosine Transform dictionary to create the backgrounds and a Haar wavelet dictionary at fixed scale to create the foregrounds. This choice has been motivated by the results of Krahmer and Ward that demonstrated that the coherence between cosine-based dictionaries and wavelet ones can be bounded locally [19]. On the other hand, Bruna and Mallat demonstrates that the use of Scattering Convolution Networks leads to results comparable to those obtained using Deep Neural Networks [3]. These networks are basically Deep Neural Networks whose neurons have been designed to perform wavelets transforms, and they can be used to understand better the performance of general networks [25].

The rest of this master project is structured as follows: the used dictionaries as well as the concept of mutual coherence and the CNNs are presented in Chapter 2. Chapter 3 describes in details the datasets created to test the behaviour of the network and the observations are presented in Chap. 4.

We first observe in Sec. 4.1 the performances of the algorithm when the input data are constructed using coherent or incoherent dictionaries. We analyse thus the ability of the network to distinguish images containing a foreground from images having only a background. In general, a higher coherence between the dictionaries allows the network to learn faster, but a too high coherence makes the network less reliable. We also observe the performances of the network on unbalanced classes in Sec. 4.2. It is interesting to note that providing a higher percentage of images with foreground to the network helps it to learn faster. Then in Sec. 4.3, we constrain the possible appearance of the foreground, and analyse again the behaviour of the network on databases with different coherence. Even though this problem seems easier than the problem analysed in Sec.4.1, it appears that the networks benefits of this simplification mainly when the foreground and the background dictionaries are very incoherent or very coherent.

In Chap. 5, we investigate more deeply the behaviour of the chosen network during the training phase and try to explain the differences observed in previous chapter. We can observe that the two firsts convolutional layers are essentials for the learning.

In Chap. 6 we present the concept of generalised coherence, based on the mutual coherence. We show how this concept can be used to analyse CNNs.

Finally, Chap. 7 contains a discussion about the observations made in previous chapters.

Chapter 2

Preliminaries

In this chapter, we present the notations and the concepts that we use in this thesis. First, we explain the Deep Neural Networks in general and the Convolutional Neural Networks (CNN) in particular, as the network that we use is a CNN. We also describe the used network in detail.

The goal of this thesis is to analyse the influence of the input data on a foreground detection algorithm. We present thus idea and concepts from image processing that allow us to construct datasets of images with similar background but distinct foreground from one dataset to the other. These datasets allow us to compare the performances of the chosen network with different kind of inputs whose content is perfectly known.

2.1 Notation

In this thesis we will use the following conventions:

- ◇ The vectors are represented using lowercase bold letters: $\mathbf{x}$. The notation $x[i]$ designs the i^{th} entry of vector $\mathbf{x}$.
- ◇ The matrices are represented using uppercase bold letters: $\mathbf{X}$. The notation $X[i, j]$ designs the element at row i and column j in the matrix $\mathbf{X}$.
- ◇ The dictionaries are represented using uppercase bold letters as the matrices: $\mathbf{D}$. The notation D_i represent the i^{th} atom of dictionary $\mathbf{D}$.
- ◇ The symbol $*$ is used to replace another symbol. For example, if we have three matrices $\mathbf{W}_1$, $\mathbf{W}_2$ and $\mathbf{W}_3$, the notation $\mathbf{W}_*$ designs all the matrices.
- ◇ $|x|$ is the absolute value of x if it is a scalar, or the size of the set x .
- ◇ $\langle \cdot, \cdot \rangle$ is the scalar product.
- ◇ $\mathbf{X}^{(t)}$ designs the value of matrix $\mathbf{X}$ at iteration t in an iterative algorithm.
- ◇ $f = O(g)$ means that the function $|f|$ is asymptotically bounded by the function g multiplied by a positive constant.

2.2 Deep neural networks

These last years, a set of methods in machine learning has known a rapid boom: the Deep Learning using Neural Networks. Those networks demonstrated their accuracy in various domains. In image classification, for example, the Convolutional Neural Networks, more suited for images, lead today to most state of the art results. They were popularised in 2012 with the ImageNet challenge [20], where they increased the state of the art of about 10%.

The idea behind Neural Networks is to train parameters using a set of labelled data, named training data, in order to be able to label unknown data. It is thus a supervised learning method for classification. There exist extensions that permit to do regression [28] or unsupervised learning [29] [35], but they are less developed, and we will not focus on them in this work.

The Neural Networks are a succession of layers performing different functions. Each layer's output is the input of one or more layer. The disposition of the layers and their type determines the **architecture** of the network. The input layer is the image to classify and the output layer is its labelling. The Convolutional Neural Networks are a set of Deep Neural Networks particularly adapted to classify images. Let us thus describe the most common layers that form a Convolutional Neural Network.

2.3 Convolutional neural networks

The Convolutional Neural Networks are a set of methods especially suited for images. They are composed of several layers that are presented below.

Dense layer: This layer is the most traditional in Neural Networks. It contains several neurons. Each neuron combines linearly the inputs that it receives, and apply a non-linear activation on it, to define the output of the neuron. This is illustrated in Fig.2.1.

The non-linear activation function is necessary to add non-linearity to the method as most of the problem in real-life cannot be described linearly.

Mathematically, let $\mathbf{W}_l$ be the weight matrix of layer l with $W_l[j, i]$ the weight applied by neuron $h[i]$ on input $I[j]$, and $\sigma_l(\cdot)$ be the non-linear activation function. We have

$$h[i] = \sigma_1 \left(\sum_j W_l[j, i] I[j] \right) \Rightarrow \mathbf{h} = \sigma_1(\mathbf{W}_l^\top \mathbf{I})$$

The trainable parameters in this layer are the weights $W_*[j, i]$.

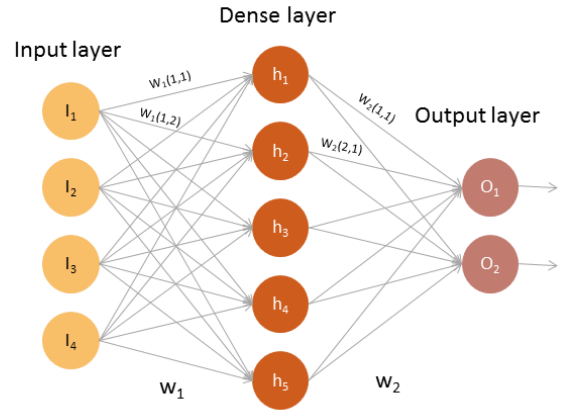


Figure 2.1: Example of a network with 2 dense layers.

Convolutional layer: This layer has given its name to the entire model, because it brings something new and fundamental to the Neural Networks: the possibility to analyse the pertinence of a combination of neighbour pixels at low cost. This property is important when we are dealing with images. Indeed, in natural image, the variation of the intensity between two nearby pixels is most of the time small, except when they correspond to a singularity or a visual structure (e.g. an edge) in the image. Therefore, it is more interesting to consider a small rectangle of pixels especially when the images are

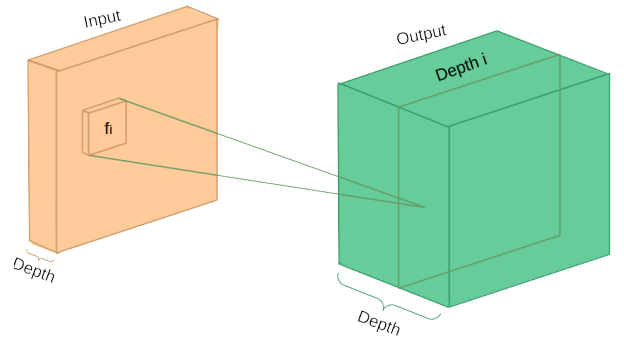


Figure 2.2: Example of convolutional layer

noised. This allows also to identify the discriminant structures independently of their position in the image.

To take that into consideration, the convolutional layers apply several filters on the input data. A filter is a small rectangle of weights applied on a sub-part of the input image with the same size. At each application, the dot product between the filter and its input is computed. In order to travel the input image entirely, the filters are translated, or convoluted, over the input. A convolutional layer is illustrated in Fig.2.2.

Formally, the convolutional layer is characterised by 4 main parameters:

- ◊ **The depth:** the depth determines the number of filters. Usually, the closer is the layer to the output layer, the higher is the depth of the convolution.
- ◊ **The size:** this parameter defines the size of the filters. The most used filters have size 1×1 , 3×3 , 5×5 or 7×7 . Using bigger filters is usually difficult as it asks for a lot of computational resources [34].
- ◊ **The padding:** as the filters have a small size, they are most of the time smaller than the input image of the layer. Each filter is thus translated over the image to travel it entirely. To give the same significance to the sides of the image than the inner pixels, it is possible to define a padding around the input image. This allows also to output an image of the same size than the input. When we use padding, 0 are added on the side of the image, as illustrated in Fig.2.3.
- ◊ **The stride:** this parameter determines the translations made by each filter to travel the input. It is a tuple $(\text{stride}_1, \text{stride}_2)$. Concretely, the network applies first the filter on the upper left corner of the input and translate the filter by stride_1 and apply it again. The algorithm repeats this operation until it reaches the end of the input image. At the end of a line, it returns to the left border but goes down of stride_2 pixels. This is illustrated in Fig.2.4.

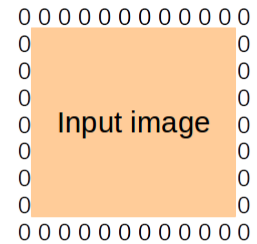


Figure 2.3: Example of a padding of size 1

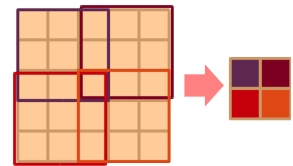


Figure 2.4: Example of a stride of size (2,2) using a filter 3×3 on a 5×5 input image

The trainable parameters in a convolutional layer are the filters. For input images with a depth d on a layer with size (s_1, s_2) , each filter contains $s_1 \times s_2 \times d$ elements. Consequently, if the layer has depth d_l they are $s_1 \times s_2 \times d \times d_l$ parameters to train in this layer. Moreover, as for the dense layer, the convolutional layer applies an activation on its output to add non-linearity to the network.

(Max) Pooling layer: The goal of the pooling layer is to resume the data and to add invariance to small shifts in orientation, position or scale to the network. This layer is important as it makes the CNN more robust in its generalisation and allows to reduce the size of the input data.

Add robustness to the algorithm is of course primordial, but reducing the input data size is important too. First, even though the computers are more and more powerful and have access to more and more memory, these resources are not infinite and not free. Maintaining a low memory cost is very important especially for algorithms like CNN that must run on a large amount of potentially huge data. On the other hand, reducing the input size also reduce the number of

parameters in the next layers. Even though we could think that having more parameters allows to handle more complicated functions, in practice, it has been shown that a too huge number of parameters with respect to the complexity of the problem produce inaccurate results. Indeed, in this case, the networks tend to overfit: they fit too well the training data and are not able to generalise on unknown data.

In practice, the pooling layers have the same parameters than the convolutional layers except that they do not have a depth parameter. They are composed of one grid with fixed size that travels the image following the rules given by the padding and stride parameters. The convention is to take a stride equal to the size of the grid. At each step, the layer returns for each depth of the input data the maximal value in the grid, or the average value for an average pooling layer, that is the second main pooling layer type.

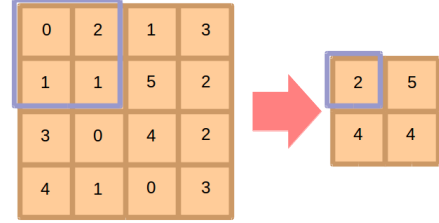


Figure 2.5: Example of max pooling layer

Fig.2.5 illustrates a Max Pooling layer. This layer does not contain trainable parameters.

In convolutional neural networks, each convolutional layer is often followed by a pooling layer. As the pooling layers are not trainable, they are usually not considered while counting the number of layers of a CNN. Usually, a CNN contains several convolutional layers (followed by pooling ones) that extracts the main features from the images. Then, some dense layers do the classification [39].

The architecture of a CNN must be defined by the user as well as the nonlinear **activation function** for each dense and convolutional layer. They are 2 main activation functions used in CNNs: the **Rectified Linear Units** or ReLU and the **SoftMax** activation.

- ◊ **ReLU**: The ReLU function is a function that is activated only for positive entries.

$$\text{ReLU}(x) = \max(0, x)$$

Despite its simplicity, it has been shown experimentally that using the ReLU as activation function is very effective in neural networks [26]. This function is often used for the intern layers of the network.

- ◊ **SoftMax**: The SoftMax function is a function designed for the output layer. In this layer, the goal is to output the probability to belong to each class. Consequently, this layer has to output a vector with values in $[0, 1]$ and with sum equal to 1, that is a stochastic vector. The softmax function is defined as follows, where $1 \leq l \leq L$ with L the number of labels:

$$\text{Softmax}[l](x) = \frac{e^{x[l]}}{\sum_k e^{x[k]}}$$

Once the architecture is fixed, the network must be trained. For that, it is important to measure its quality. This is done through the **loss function**. This function computes how far is the output provided by the algorithm from the expected output. A traditional loss function for classification is the **crossentropy loss**. If $\hat{y}_{i,l}$ denotes the probability that image x_i belongs to class l according to the CNN and $y_{i,l}$ the true labelling of this image (that is 1 if the image has label l and 0 otherwise), the crossentropy loss on a batch of images is defined as

$$\text{Loss} = -\frac{1}{|\text{Batch}|} \sum_{i \in \text{Batch}} \sum_l y_{i,l} \log(\hat{y}_{i,l}).$$

Once the loss function is chosen, the network tries to minimise it during the training phase using **stochastic gradient descent**. This algorithm is a gradient descent applied on random **batches**. A batch is a set of inputs, randomly taken from the training set. Once the network considered all the data we said that it has done an **epoch**. To apply a gradient descent algorithm, we must compute the gradient of the loss with respect to each weight ($= \frac{\partial \text{Loss}}{\partial \mathbf{W}_i}$). This gradient can be computed using an algorithm named **backpropagation**.

This algorithm works in two phases: first, it does a forward pass where it computes the output of each layer for a batch of data, and then a backward pass where it computes the gradient using the activation's values computed in the forward pass. The computation of the gradient is then based on the chain rule [15]. This rule claims that if $F = f(g(x))$, we have $F'(x) = f'(g(x)) \cdot g'(x)$. As the CNN are a succession of layers, it makes sense to use such algorithm. Finally, using the computed gradient, it is possible to apply one step of the gradient descent:

$$\mathbf{W}_i^{(t+1)} = \mathbf{W}_i^{(t)} - \eta \frac{\partial \text{Loss}^{(t)}}{\partial \mathbf{W}_i}$$

Where η is called the **learning rate** of the algorithm. One iteration, or one step, of the stochastic gradient descent is illustrated on figure 2.6. We say that the algorithm converges when $\lim_{t \rightarrow \infty} \text{Loss}(t) = c$ with $c \in \mathbb{N}$.

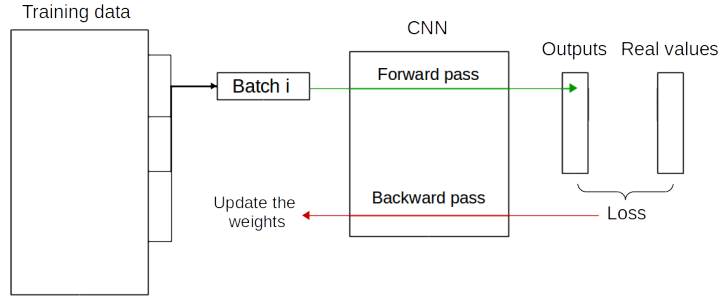


Figure 2.6: One step of the stochastic gradient descent algorithm for a CNN.

The learning rate can be simply a fixed constant value, but there exist algorithms that adapt it during the training. This allows to take in account the evolution of weights. Usually, having a larger training rate in the beginning of training is better as the CNN learns a lot during the first epochs. But having a too large training rate can make the network going over the optimum. That is why once we approach the minimum, the learning rate must often be decreased. There are also more sophisticated algorithms to adapt the learning rate.

For example, the **AdaGrad** method [10] computes a learning rate for each parameter. The goal is to adapt the learning rate such that parameters with small gradients and thus small updates have a larger learning rate and parameters with bigger gradients have a smaller learning rate. Moreover, it takes in account the previous evolution of the gradients. Mathematically, if η is the initial learning rate and $g_t = \frac{\partial \text{Loss}^{(t)}}{\partial x}$ is the gradient of the loss function with respect to parameter x at iteration t , we update each parameter as

$$x^{(t+1)} = x^{(t)} - \frac{\eta}{\sqrt{\sum_{j=1}^t g_j^2}} g_t.$$

This method improves the performances of the algorithm but has some drawbacks. First, the gradients are always summed, and the learning rate decreases at each iteration and potentially goes to zero. A learning rate equal to zero prevents the learning. On the other hand, if the gradients are very high in the beginning of the learning, the learning rate will be small during the rest of the training.

The **AdaDelta** method fixes these issues by accumulating the gradients with an exponential decay. Formally, if we keep the same notation as for AdaGrad, we define two accumulative variables u and G and a decay rate p (p is usually 0.95). We can update the parameters in the following way:

$$\begin{aligned} G_t &= pG_{t-1} + (1-p)g_t^2 & u_t &= pu_{t-1} + (1-p)\Delta x_t^2 \\ \Delta x_t &= \frac{\sqrt{u_{t-1}}}{\sqrt{G_t}} g_t & x^{(t+1)} &= x^{(t)} - \Delta x_t. \end{aligned}$$

We used the AdaDelta method in our algorithm.

Once the CNN has been trained, it is important to verify that it is able to generalise on unknown data. Indeed, being able to classify known data is not very remarkable, the interest of CNN is their ability to generalise. Unfortunately, it is not always the case. When the network is too complex regarding the problem to analyse, or when the number of data is too small (both are obviously linked), a phenomenon named overfitting may occurs. Overfitting networks are networks that are unable to generalise despite their good results on the training set.

To avoid this problem, it is important to keep a set of data, named **test set**, on the side. This set must never be used during the training. Once the training is done, the test set is labelled by the network, and the accuracy obtained is compared with the one obtained on the training set. The accuracy on the test set is almost always slightly lower than on the training set, but a huge difference is a sign of overfitting. Note that the loss function is used to train the network, but the **accuracy** is often used to determine its quality. By accuracy, we mean the percentage of images properly labelled.

If we look now at the features learned by the algorithm, we can observe that in CNN, the first layers extract the basic features from the images, and are thus often similar whatever the network has been trained for. This property is the foundation of the transfer learning paradigm: using the final weights of a CNN as initial weights of another CNN that solves a similar but different problem may accelerate the learning and improve the accuracy on the test set [30] [38]. This implies also that a network that has been built for a given problem can be used for another close problem. Unfortunately, this is not always true. For some problems, it is very hard to find a structure that make the CNN converge and generalise properly in a reasonable time, even when the problem does not look complicated and seems close to another problem easy to solve with CNNs.

This remark points a very important issue about CNNs: they may work very well, but their performances are not guaranteed and we are still lacking fundamental understanding to design and train models in a way that converges and generalises properly. There are not yet theoretical optimal architectures and associated optimisation (training) algorithms that ensure the convergence of the network, except for specific problems [23]. Moreover, even though the CNNs can extract basic features from images, they are very sensitive to adversarial perturbations. That means that for any network it is possible to construct images that the CNN miss-classify with high confidence, despite their visual appearance correctly interpreted by humans. Jo and Bengio, pointed that this problem was due to trend of the CNN to learn only low-level abstraction [18]. On the other hand, the convolutional neural networks ask for a good balance between the different class. Even if there are ways to avoid this problem, having 80% of training data in one class and 20% in the other is often an issue when working with CNN [21] [4].

In this work, we are going to try to better understand CNNs. We are not going to search for an ideal structure for CNNs, but instead we are going to analyse the influence of the input data on its performances. Consequently, we fixed our investigated CNN architecture as well as the associated optimisation algorithm. Before to present it, let us explain an improvement that we used :

The Batch Normalisation: As indicates its name, the goal of the batch normalisation is to normalise the activation of a layer for each batch. It can be applied or not at each layer, and maintains the mean activation of this layer close to 0 and the standard deviation close to 1. This enables higher learning rates and may thus accelerate the algorithm. Moreover, it regularises the model too thereby favouring better generalisation properties [17].

To construct our CNN that is illustrated in Fig.2.7, we get inspired by the architecture suggested by Weimer and al. [36]. Indeed, they designed an architecture adapted for defaults detection on tissues. But the detection of defaults on images can be linked to the detection of foreground, if we consider the tissues as background and the defaults as foreground. Since the aim of this master thesis is not to find the best architecture but to analyse the influence of the data, we decided to use 3 convolutional layers, with depth 64, 128 and 128, followed by 2 dense layers with 1024 neurons. The convolutional layers use filters 3×3 . Remark that all the convolutional layers are followed by a 2×2 MaxPooling layer except the last one.

Unfortunately, this network appears to be very inefficient to classify our images. Indeed, it was designed for 32×32 pixels, while our constructed images contain 256×256 pixels. To avoid this problem, we used the beginning of the GoogLeNet architecture [33]. Indeed, this model has been designed for a complex problem with 256×256 pixels.

Consequently, we added 2 convolutional layers with MaxPooling in the beginning of the CNN. Note that the first layer uses a filter 7×7 with stride of size 2, that results also in a dimension reduction. The second convolutional layer uses 3×3 filters. But both layers are followed by a MaxPooling of size 3×3 with a padding of 2×2 . Moreover, we did not use padding and that reduce slightly the dimensions too. Moreover, the first convolutional layer is followed by a batch-normalisation to regularise the model. These two layers were also trained during the learning phase, with random initial weights, such as the layers presented before.

We used to crossentropy loss function and the AdaDelta update of the learning rate with parameter $p = 0.95$ on batches with 128 images.

In practise, we used Keras [7] to implement our network and do the manipulations presented in the next chapters.

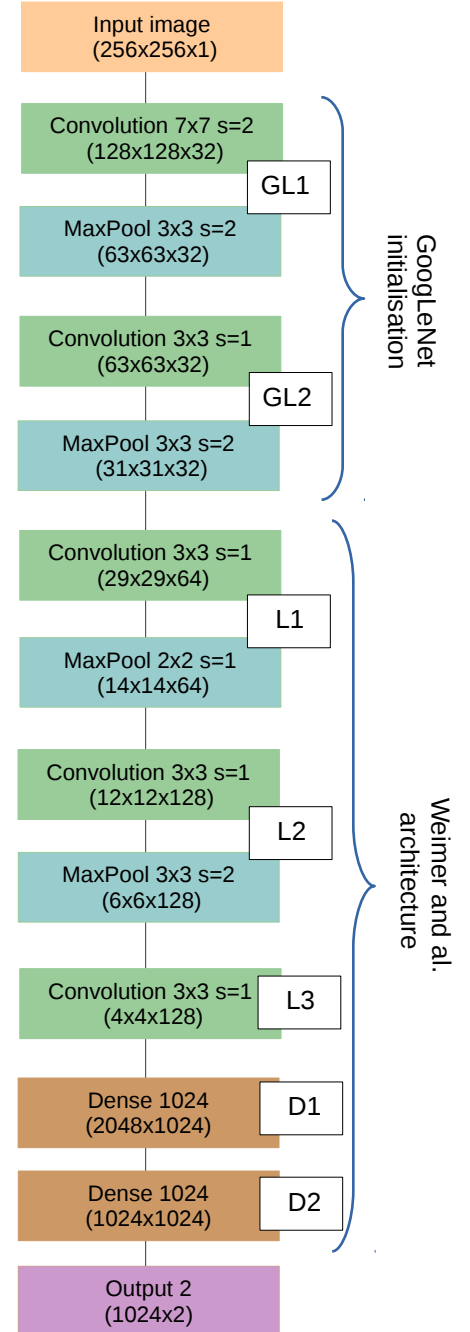


Figure 2.7: The architecture used for the CNN

2.4 Image processing tools

The image processing field contains methods to analyse images and extract their useful information. In this work, we are going to use concepts from image processing not to extract features from images but instead to construct images with distinct features. We present in this section the concept used that are the sparse image representation using dictionaries and the mutual coherence between dictionaries.

2.4.1 Sparse image representation using dictionaries

A good image model is often the key to solve image processing problems. Indeed, a good prior knowledge about the image content allows to denoise or deblur it, but also to do effective compression or edge-detection.

The sparse representation scheme is a modelling approach that proved its efficiency in various image processing problems like compressed sensing, deblurring or image inpainting [11]. This scheme supposes that the images can be described sparsely in dictionaries, that are sets of basic images, named atoms. An image is thus a linear combination of a few number of basic images.

In this work, we use standard dictionaries to construct synthetic images. This section presents those dictionaries and the reason of their selection. First, let us define mathematically the sparse representation scheme.

Formally, let $\mathbf{x} \in \mathbb{R}^{n^2}$ be the image to model. For practical reasons, the $n \times n$ image has been reshaped in a vector n^2 .

Let $\mathbf{D} \in \mathbb{R}^{n^2 \times d}$ be a dictionary, with d atoms $\mathbf{D}_{*,i}$, $i \in \{1, \dots, d\}$. To be able to describe all the images in this dictionary, $\mathbf{D}$ must generate $\mathbb{R}^{n^2}$, and thus $d \geq n^2$.

The image $\mathbf{x}$ can be sparsely described in $\mathbf{D}$ if there exists a sparse vector $\boldsymbol{\alpha} \in \mathbb{C}^d$ such that

$$\mathbf{x} = \mathbf{D}\boldsymbol{\alpha}. \quad (2.1)$$

A lot of dictionaries have been built to match natural or more specific images. Some of the oldest but also the most known/used are the Cosine-based and the Wavelets dictionaries. In this work, we consider the Discrete Cosine Dictionary and the Haar wavelet dictionary.

Definition 2.1. *The **Discrete Cosine Dictionary (DCT) dictionary** has several definitions. In this work, we consider an orthonormal dictionary, that is equivalent to an orthonormal basis of $\mathbb{R}^{n^2}$. The most common basis for the DCT is the following, named DCT-II:*

In one dimension: For $N = 2^p$, the one-dimensional orthonormal discrete cosine basis $\boldsymbol{\Gamma}$ in $\mathbb{R}^N$ consists of the following vectors:

$$\gamma_k[n] = \sqrt{\frac{2 - \delta_{k,0}}{N}} \cos\left(\frac{\pi(2n+1)k}{2N}\right), \quad \text{with } n = 0, \dots, N-1,$$

where $\delta_{k,0}$ is the Kronecker delta, that is 1 for $k = 0$ and 0 otherwise. The parameter $k \in \mathbb{N}$ determines the frequency of the vector and varies between 0 and $N-1$.

In two dimensions: For $N = 2^p$, the two-dimensional orthonormal discrete cosine basis $\boldsymbol{\Gamma}$ in $\mathbb{R}^{N \times N}$ is comprised of the matrices (that are then reshaped into vectors, to form the dictionary):

$$\gamma_{k_1, k_2}[n_1, n_2] = \gamma_{k_1}[n_1] \cdot \gamma_{k_2}[n_2],$$

with $n_1, n_2, k_1, k_2 \in \mathbb{N}$ and $0 \leq n_1, n_2, k_1, k_2 \leq N-1$.

On the other hand, the Haar wavelet dictionary comes from the **Haar system** [14], that defines a scale function $\phi(t)$ and a wavelet function $\psi(t)$ in $\mathbb{R}$ as follows:

$$\phi(t) = \begin{cases} 1 & \text{if } 0 \leq t < 1, \\ 0 & \text{otherwise.} \end{cases} \quad \psi(t) = \begin{cases} 1 & \text{if } 0 \leq t < \frac{1}{2}, \\ -1 & \text{if } \frac{1}{2} \leq t < 1, \\ 0 & \text{otherwise.} \end{cases}$$

The dictionary is then defined as translations, dilation and multiplications of these functions, that are called the mother wavelets.

For $N = 2^p$, the **standard orthonormal Haar basis** in $\mathbb{R}^N$ consists of atoms $h_{j,l,v}$ with $j, l \in \mathbb{N}$ and $v \in \{0, 1\}$. Parameter j is the **scale**, or dilation at which we consider the function, parameter l is the **translation** and v is the **mother wavelet** used. That means that $h_{j,l,1}[n] = 2^{-\frac{j}{2}} \psi(2^{-j}n - l)$ and $h_{j,l,0}[n] = 2^{-\frac{j}{2}} \phi(2^{-j}n - l)$ with $n \in \mathbb{N} = [0, N - 1]$ [5]. More specifically we have

$$h_{j,l,v} = \begin{cases} h_{j,l,1} & \text{if } 1 \leq j \leq J \leq p - 1, \ 0 \leq l \leq 2^{p-j} - 1, \\ h_{J+1,l,0} & \text{if } 0 \leq l \leq 2^{p-J-1} - 1. \end{cases}$$

A way to define a Haar wavelet **dictionary** is to fix J and to increase the number of translations, by playing with the parameter l . In our case, we fixed $J = p - 1$ and we used parameter l' for the translations such that $l = 2^{-j}l'$ with $0 \leq l' \leq 2^p - 2^j$, $l' \in \mathbb{N}$. This choice takes in account all the possible translations of the Haar wavelet non-zero pattern.

Indeed, at a given scale j , a Haar wavelet basis atom is non-zero on $2^j \times 2^j$ pixels. By our choice of l , we do not modify this non-zero pattern, it is just translated on the image. This leads us to the following definition :

Definition 2.2. For $N = 2^p$, our **Haar wavelet dictionary** for $\mathbb{R}^N$ consists of atoms $h_{j,l,v}$ with $j, l \in \mathbb{N}$ and $v \in \{0, 1\}$. The atoms are defined as $h_{j,l,1}[n] = 2^{-\frac{j}{2}} \psi(2^{-j}(n - l))$ for $1 \leq j \leq p - 1$, $0 \leq l \leq 2^p - 2^j$ and $h_{p,0,0}[n] = h_0[n] = 2^{-\frac{p}{2}}$, with $n \in \mathbb{N} = [0, N - 1]$.

Note that this definition implies

$$h_{j,l,1}[n] = \begin{cases} 2^{-\frac{j}{2}} & \text{if } l \leq n < l + 2^{j-1}, \\ -2^{-\frac{j}{2}} & \text{if } l + 2^{j-1} \leq n < l + 2^j, \\ 0 & \text{otherwise.} \end{cases}$$

To describe a bivariate Haar wavelet dictionary, let us describe the window functions $h_{j,l,0}[n]$ more precisely:

$$h_{j,l,0}[n] = \begin{cases} 2^{-\frac{j}{2}} & \text{if } l \leq n < l + 2^j, \\ 0 & \text{otherwise.} \end{cases}$$

In two dimensions: For $N = 2^p$, our bivariate Haar dictionary for $\mathbb{R}^{N \times N}$ consists of the following atoms:

$$\begin{cases} h_{(0,0)}[n_1, n_2] = h_0[n_1] \cdot h_0[n_2] = 2^{-p} \\ h_{j,(l_1,l_2),(v_1,v_2)}[n_1, n_2] = h_{j,l_1,v_1}[n_1] \cdot h_{j,l_2,v_2}[n_2] \end{cases}$$

with $0 < j < p$; $0 \leq l_1, l_2 \leq 2^p - 2^j$, and $(v_1, v_2) \in \{\{1, 0\}, \{0, 1\}, \{1, 1\}\}$.

Actually, this is the wavelet dictionary with mother wavelets $\psi^1[n_1, n_2] = \psi[n_1] \cdot \phi[n_2]$, $\psi^2[n_1, n_2] = \phi[n_1] \cdot \psi[n_2]$ and $\psi^3[n_1, n_2] = \psi[n_1] \cdot \psi[n_2]$ [24], also named Daubechies Wavelet 1 or db1.

The DCT and the Haar wavelet dictionaries were constructed to describe sparse natural images. In this work, we reverse their utility. Indeed, we construct synthetic images using a random sparse vector multiplied by a fixed dictionary. In practice, we chosen to use the DCT dictionary to model the backgrounds and the Haar wavelet one to model the foregrounds. Indeed, the DCT dictionary produces images with redundant patterns while the Haar wavelet one produces singular or large patterns (depending of the scale) but without redundancy. Figure 2.8 illustrates an image with foreground and background. The goal of our algorithm is to determine if there is a foreground on a given image. In real case situation, this could correspond to defect or anomaly detection for example.



Figure 2.8: An example of background, foreground and image combining both. The intensity of the foreground has been increased to see it more easily. The images have been created using the methodology described in Chapter 3 and then cropped.

2.4.2 Mutual coherence

Intuitively, the mutual coherence is a kind of measure of the similarity between sparse images in two bases or dictionaries. It computes the proximity of the two closest elements between each dictionary. In practise, the higher is the coherence, the more similar are the dictionaries.

The mutual coherence of two orthonormal bases is linked to the sparse representation, and more precisely to the sparse recovery problem. The sparse recovery challenge is to recover the vector α from equation 2.1 where $\mathbf{x}$ and $\mathbf{D}$ are known. This is not trivial as the dictionary can be hard or impossible to be inverted. That is why iterative algorithms such as basis pursuit have been developed. When $\mathbf{D} = [\mathbf{D}_1, \mathbf{D}_2]$ with $\mathbf{D}_1$ and $\mathbf{D}_2$ two orthonormal basis, the uniqueness of the basis pursuit algorithm can be determined with respect to the mutual coherence between $\mathbf{D}_1$ and $\mathbf{D}_2$ [9].

On the other hand, this relation allows to define compression strategies that ensure a stable and robust image reconstruction, using compressive sensing [19]. Moreover, the mutual coherence defines a threshold for Morphological Component Analysis (MCA) that optimises the convergence of the algorithm [2]. As the MCA algorithm is used to separate the foreground from the background in an image, it is natural to suppose that the coherence can also have an impact on algorithms that try to determine the existence of the foreground.

Formally, it is defined as follows.

Definition 2.3. The *mutual coherence* of two orthonormal bases or dictionaries of $\mathbb{R}^N$ $\Phi = \{\phi_j\}_{j=1}^N$ and $\Psi = \{\psi_k\}_{k=1}^N$ is defined by

$$\mu(\Phi, \Psi) = \sup_{1 \leq j, k \leq N} |\langle \phi_j, \psi_k \rangle|.$$

The mutual coherence varies between $N^{-1/2}$ and 1. We said that two dictionaries are mutually incoherent when $\mu = O(N^{-1/2})$. The definition of the mutual coherence can be extended to non-orthonormal dictionaries [8]. It can also be defined locally, to refine the analyse. Indeed,

the previous definition computes a supremum. That means that even if most scalar products are low, the mutual coherence can be high due to one or two vectors in the basis. Krahmer and Ward formalised a local definition [19]:

Definition 2.4. The *local coherence* of an orthonormal basis $\Phi = \{\phi_j\}_{j=1}^N$ of $\mathbb{R}^N$ with respect to the orthonormal basis $\Psi = \{\psi_k\}_{k=1}^N$ of $\mathbb{C}^N$ is the function $\mu^{\text{loc}}(\Phi, \Psi) \in \mathbb{R}^N$ defined coordinate-wise by

$$\mu_j^{\text{loc}}(\Phi, \Psi) = \sup_{1 \leq k \leq N} |\langle \phi_j, \psi_k \rangle|.$$

Note that knowing $\mu^{\text{loc}}(\Phi, \Psi)$ allows to recover the mutual coherence of a sub-dictionary $\Phi_\Omega \subseteq \Phi$ with respect to an orthonormal dictionary Ψ . Indeed $\mu(\Phi_\Omega, \Psi) = \sup_{\phi_j \in \Phi_\Omega} \mu_j^{\text{loc}}(\Phi, \Psi)$.

Krahmer and Ward demonstrated that the local coherence of the orthonormal two-dimensional Fourier basis $\Phi = \{\phi_{k_1, k_2}\}$ with respect to the orthonormal bivariate Haar wavelet basis $\Psi = \{h_{j, l, v}\}$ in $\mathbb{R}^{N \times N}$ is bounded [19]: $\mu_{k_1, k_2}^{\text{loc}} \leq \min\left(1, \frac{18\pi}{\max(|k_1|, |k_2|)}\right)$. This bound is arbitrarily small for large $|k_1|, |k_2|$. That means that for large parameters k_1 and k_2 , ϕ_{k_1, k_2} becomes nearly orthogonal to all elements of Ψ . This result is interesting because it theoretically allows to construct a set of sub-Fourier basis with a common characteristic $\max(|k_1|, |k_2|)$ but a different coherence with respect to the Haar wavelet basis. This would permit to test the influence of the coherence without having to create a lot of different dictionaries.

In this work, we chosen to describe the background as a sparse element in the DCT dictionary and a foreground as a sparse element in the Haar wavelet dictionary. Therefore, we adapted the theoretical bound for the DCT basis, instead of the Fourier one, and analysed the local coherence of the Haar wavelet with respect to the DCT (the other way around than in [19]). Intuitively, this will allows us to analyse the coherence of each kind of foreground with respect to the background and latter to analyse the influence of the coherence on foreground detection.

Note that the Haar wavelet dictionary that we use in this work is not orthonormal. Indeed, it is an overcomplete dictionary. It is possible to recover a chosen atom from this dictionary using a linear combination of other atoms. Nevertheless, we can still compute the local coherence for each atom of the dictionary with respect to the DCT dictionary that is an orthonormal basis of $\mathbb{R}^{N \times N}$. Indeed, even if they are not orthogonal, the atoms of our dictionary are unitary. Moreover, we chosen to define the mutual coherence between a sub-dictionary of the Haar wavelet one and the DCT dictionary as the maximal value of the mutual coherence of each atom of this sub-dictionary with respect to the DCT. This is formalised in definition 2.5.

Definition 2.5. The *mutual coherence* between a dictionary in $\mathbb{R}^N$ containing unitary atoms (but not necessarily orthonormal) $\Phi = \{\phi_j\}_{j=1}^d$ and an orthonormal basis of $\mathbb{R}^N$ $\Psi = \{\psi_k\}_{k=1}^N$ is defined by

$$\mu(\Phi, \Psi) = \sup_{1 \leq j \leq d} \mu_j^{\text{loc}}(\Phi, \Psi),$$

where $\mu_j^{\text{loc}}(\Phi, \Psi)$ is the local coherence of atom ϕ_j with respect to Ψ .

This definition implies that $\mu(\Phi, \Psi) = \sup_{1 \leq j \leq d, 1 \leq i \leq N} |\langle \phi_j, \psi_i \rangle|$.

To compute a theoretical bound on the mutual coherence between the Haar wavelet dictionary and the DCT one, we follow the same logic as Krahmer and Ward. That means that we first bound the scalar product in one dimension for $k \neq 0$. If $h_{j,l,v}$ is an atom of the Haar wavelet dictionary at scale j , as defined in definition 2.2 and γ_k is an atom of the DCT dictionary defined in 2.1, we can write :

$$|\langle h_{j,l,v}, \gamma_k \rangle| = 2^{\frac{-1-p-j}{2}} \sum_{z=\pm k} e^{iz\pi(2^{-p-1}+2^{j-pl})} (1 + (-1)^v e^{2^{j-p-1}}) \left(\frac{1 - e^{iz\pi 2^{j-p-1}}}{1 - e^{iz\pi 2^{-p}}} \right).$$

We then use Lemma A.1, A.2 and A.3 described in appendix A. These lemma states that $|1 - e^{i\pi 2^{-j}k}| \leq \min(2, \pi 2^{-j}|k|)$ and that $|1 - e^{i\pi 2^{-p}k}|$ is lower-bounded by $2^{-p+3/2}|k|$ when $|k| \leq 2^{p-1}$ and by $2^{-p+1}|k|$ when $|k| < 2^p$.

This allows us to bound the fraction $\left(\frac{1 - e^{iz\pi 2^{j-p-1}}}{1 - e^{iz\pi 2^{-p}}} \right)$ and to conclude that in one dimension

$$|\langle h_{j,l,v}, \gamma_k \rangle| \leq \begin{cases} \pi 2^{\frac{j-p}{2}-1} & j > 1, \\ \pi 2^{\frac{-p}{2}} & j = 1. \end{cases}$$

Considering now the two-dimensional case, we can easily update the bound using the fact that $h_{j,(l_1,l_2),(v_1,v_2)}[n_1, n_2] = h_{j,l_1,v_1}[n_1] \cdot h_{j,l_2,v_2}[n_2]$ and $C_{k_1,k_2}[n_1, n_2] = C_{k_1}[n_1] \cdot C_{k_2}[n_2]$. Indeed, this implies that $\langle h_{j,(l_1,l_2),(v_1,v_2)}, C_{k_1,k_2} \rangle = \langle h_{j,l_1,v_1}, C_{k_1} \rangle \cdot \langle h_{j,l_2,v_2}, C_{k_2} \rangle$. We have thus when $k_1, k_2 \neq 0$ that

$$|\langle h_{j,(l_1,l_2),v}, \gamma_{k_1,k_2} \rangle| \leq \begin{cases} \pi^2 2^{j-p-2} & j > 1, \\ \pi^2 2^{-p} & j = 1. \end{cases}$$

As the analyse with k_1 and/or k_2 equal to 0 provided a lower upper bound, we can build the Thm 2.1 presented below. All the computational details for the construction of this theorem can be find in appendix A.

Theorem 2.1. *Let $N = 2^p$ with $p \in \mathbb{N}$. In $\mathbb{R}^{N \times N}$, the local coherence of the bivariate Haar wavelet dictionary $\mathbf{H} = \{h_{j,l,v}\}$ with respect to the orthonormal two-dimensional DCT dictionary $\mathbf{\Gamma} = \{\gamma_{k_1,k_2}\}$ as defined respectively in Def. 2.2 and 2.1, is bounded by*

$$\mu_{j,l,v}^{\text{loc}}(\mathbf{H}, \mathbf{\Gamma}) \leq \begin{cases} \pi^2 2^{j-p-2} & \text{When } j > 1, \\ \pi^2 2^{-p} & \text{When } j = 1. \end{cases}$$

We can observe that the local coherence between an element of the Haar wavelet dictionary with respect to the DCT dictionary is bounded by a value that relies only on the scale j of the Haar wavelet element. Consequently, by taking all the elements at a fixed scale j , we can define several dictionaries (one for each scale) with a theoretical different coherence with respect to the DCT orthonormal basis.

Fig.2.9 shows the evolution of the mutual coherence according to the scale for image with 256×256 pixels. We can observe in this figure that the mutual coherence between the Haar wavelet dictionary at a fixed scale and the DCT orthonormal dictionary increases effectively with the scale. The rate of increasing is more or less a factor two, as expected by the theory. However, we can see that the bound computed when one of the parameters k_i of the DCT dictionary is fixed to 0, is more accurate. This bound is the following:

$$|\langle h_{j,(l_1,l_2),(v_1,v_2)}, \gamma_{0,k_2} \rangle| \leq \begin{cases} \pi 2^{j-p-1} & \text{When } j > 1, \\ \pi 2^{\frac{1-2p}{2}} & \text{When } j = 1. \end{cases}$$

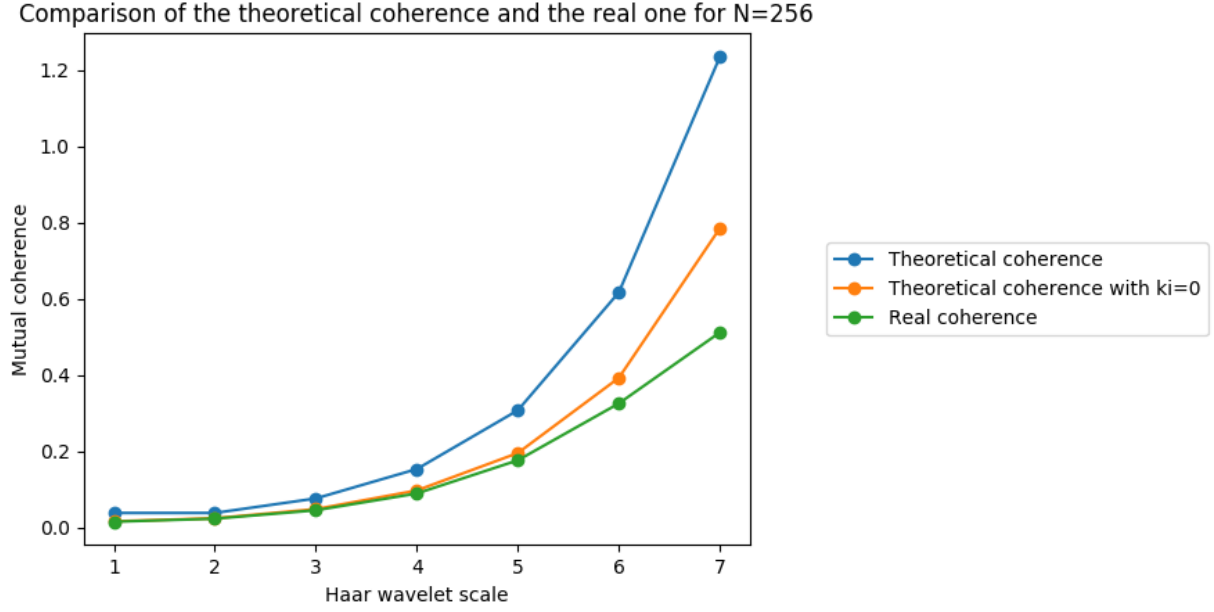


Figure 2.9: Comparison of the mutual coherence between the Haar wavelet dictionary at a fixed scale and the DCT orthonormal basis.

In conclusion, if we consider the set of bivariate Haar wavelet atoms at a given scale, H_i $2 \leq i \leq p-1$, we form a dictionary whose coherence with respect to the two-dimensional DCT dictionary grows with i by a factor two. When $p=8$, H_1 has also a smaller coherence than H_2 . Consequently, in this case we can consider seven sub-dictionaries that have a different mutual coherence with respect to the DCT dictionary. This allows us to analyse the impact of seven different coherence on the performances of the chosen CNN.

Fig.2.10 illustrates images whose background has been built using the DCT dictionary and the foreground has been built using the Haar wavelet dictionary at scale 2,4 and 6.

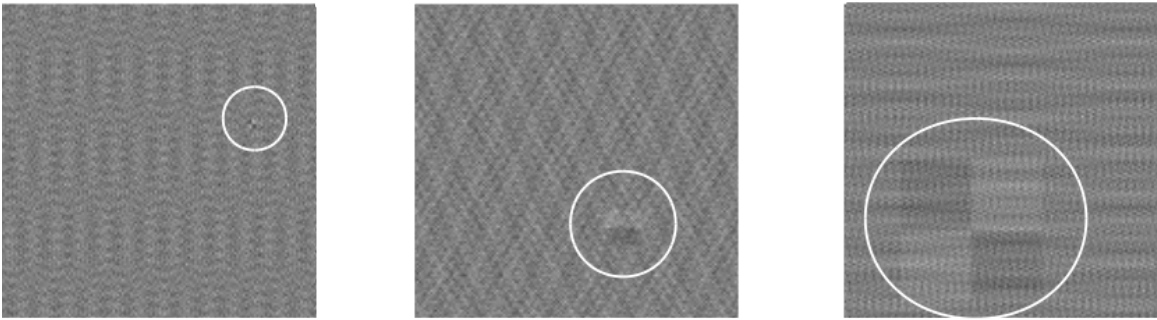


Figure 2.10: Example of images with DCT background and Haar foreground for scales 2,4 and 6, the intensity of all foreground has been slightly increased to distinguish them better. The images have been constructed using the methodology described in Chapter 3.

Chapter 3

Generation of synthetic datasets

We used the concepts presented in Sec. 2.4 to build different datasets of images. Each image consists of a background and a possible foreground built using coherent or incoherent dictionaries. We chosen to build images containing 256×256 pixels.

- ◇ The background: the background is the sum of 16 elements taken randomly from the DCT dictionary defined in Def. 2.1. Each atom is multiplied by a coefficient equal to ± 1 . The pixel values are then re-scaled: we divide each pixel value by the maximal absolute value that can take a pixel in a combination of 16 DCT atoms, named **maxDCT**. This allows to centre the pixel values of the background between -1 and 1 . In the case of 256×256 images with 16 DCT basis elements, **maxDCT** is equal to $\frac{16 \cdot 2}{256} = 0.125$.

- ◇ The foreground: the foreground is the sum of three atoms taken randomly from the Haar wavelet dictionary defined in Def. 2.2 at a given scale. Each atom is also multiplied by ± 1 . The foreground image has pixels values that are most of the time between -1 and 1 . As the images contain 256×256 pixels, they are 7 scales in the Haar wavelet dictionary at which the non-zero patterns is smaller than the image. We chosen to use these 7 scales to build 7 different type of foregrounds. An example of background is presented in Fig. 3.1.

On the other hand, it is interesting to notice that each atom from the Haar wavelet dictionary consists of a non-zero pattern that can take three distinct appearances, illustrated on Fig. 3.2. This is due to the definition of the Haar wavelet atoms : $h_{j,(l_1,l_2),(v_1,v_2)} = h_{j,l_1,v_1} \cdot h_{j,l_2,v_2}$ where h_{j,l_i,v_i} is non-zero on a limited range and $(v_1, v_2) \in \{\{1, 0\}, \{0, 1\}, \{1, 1\}\}$ (see Def. 2.2 for more details).

- ◇ The image: when it contains a foreground, the image is defined as the sum of its background and its foreground, clipped between $[-1, 1]$. Otherwise, it is simply the background. Note that the clipping doesn't affect the image in most of the cases. Examples of images with a foreground at each scale are presented in Fig. 3.3. The image has a probability p to have or not a foreground. Formally, if we name $\mathbf{B} \in \mathbb{R}^{256 \times 256}$ the background and $\mathbf{F} \in \mathbb{R}^{256 \times 256}$ the foreground, we construct images $\mathbf{X} \in \mathbb{R}^{256 \times 256}$ such that

$$\mathbf{X} = \mathbf{B} + \alpha \mathbf{F} \quad \text{with } \alpha = \begin{cases} 0 & \text{with probability } p \\ 1 & \text{with probability } 1 - p. \end{cases}$$

The goal of the algorithm is then to recover the value of α adopted for a given sample.

On each dataset, we constructed 20000 images for the training set. We also constructed a test set with 5000 images. However, the images in the test set have always a probability of 50% to have or not a foreground. Each dataset contains thus 80% of images used for training and 20% used for testing. This is quite usual to train CNNs.

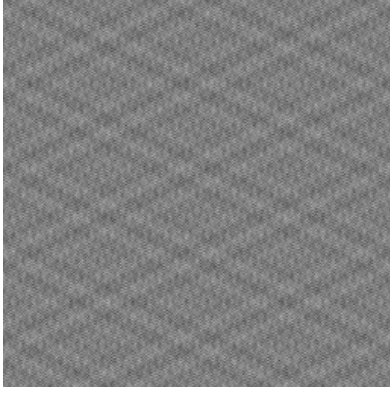


Figure 3.1: Example of image containing only a background.

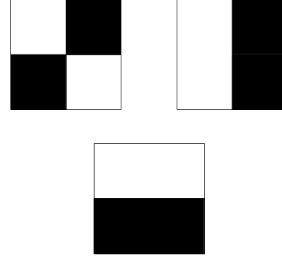
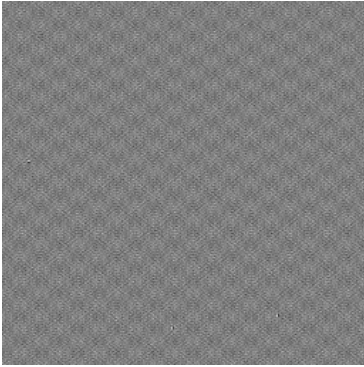
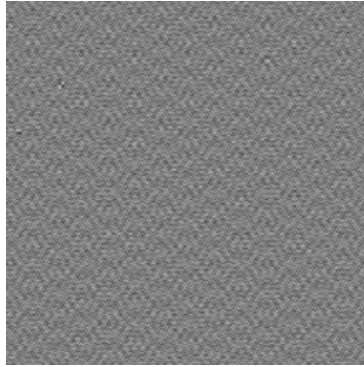


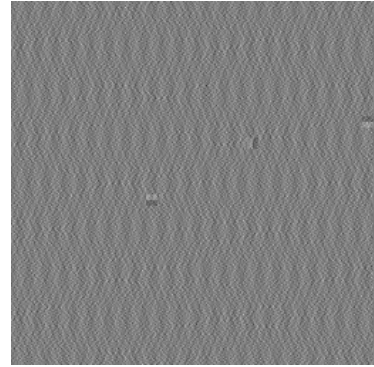
Figure 3.2: The 3 Haar wavelet patterns.



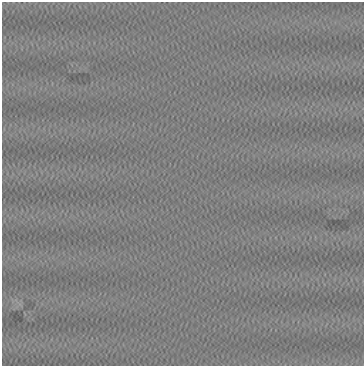
Scale 1.



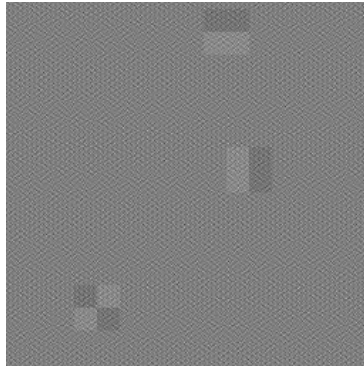
Scale 2.



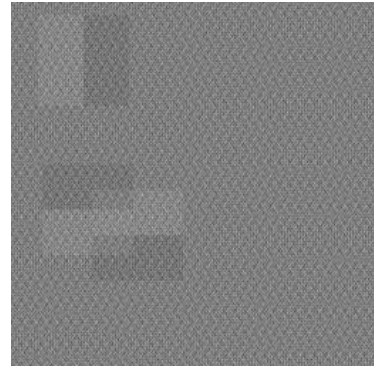
Scale 3.



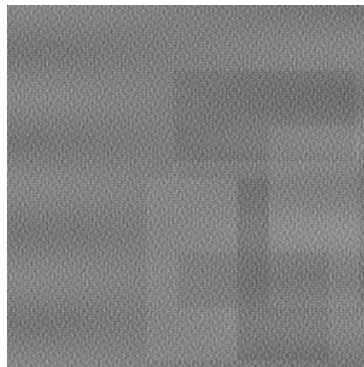
Scale 4.



Scale 5.



Scale 6.



Scale 7.

Figure 3.3: Examples of images from each dataset. The intensity of the foreground has been slightly increased to distinguish it more easily.

Chapter 4

Classification results for different generative models

4.1 Impact of the mutual coherence between background and foreground dictionaries

In section 2.2 we presented a CNN built to detect anomalies on images. In other word, the goal of the CNN is to determine if a 2D-signal contains or not a foreground. We first tested our model on datasets of images built with mutually coherent or incoherent background and foreground dictionaries. The images have been built using the methodology presented in Chapter 3, and each dataset contains 50% of images in each class. In practice, the 2D-signals contain a background and a potential foreground built using Haar wavelets elements at a fixed scale i . The mutual coherence between the background and the foreground grows with the scale used. For simplicity we often denote images with foreground built using wavelets atoms at scale i as "images at scale i ", and designate each dataset by its scale.

4.1.1 Observation of the behaviour of the CNN

We applied our CNN on datasets with foreground at different scales. Fig. 4.1 shows its average performances on five tests for each scale. Each test has a random initialisation identical for each scale, but different from a test to another, to be able to compare the performances of networks with same initialisation for different input data. We ran each test during 30 epochs and computed the performances on the test set after each epoch. The different tests had slightly different performances (see Appendix. B.1), but some interesting behaviours can already be pointed out.

We can observe that the algorithm reaches almost the same accuracy for each scale, but scales 6 and 7 are clearly learned faster than the others. On the other hand, scales 1 and 3 are slowly learned, and the CNN reaches a lower accuracy on images at first scale after the training phase. This is quite opposite to the intuition. Indeed, the smaller scales produce a foreground with a small non-zero area with high intensity that is quite incoherent with the background and should then be easier to distinguish, as discussed in Sec. 2.4.2. Here, the images constructed with a foreground basis having a high coherence with respect to the background basis seems on the contrary the easiest to classify.

Moreover, the CNN needs a few epoch before being able to learn anything at the first scales. Indeed, during the first epochs, the accuracy stays at 50%, as if the networks had not learned at all. The loss function also stays more or less at the same value. This is quite unusual. Indeed, normally, a neural network improves its accuracy until more or less 80 % quite fast and then gains the last improvements more slowly. Usually, when a network has not learned anything after 5 epochs, we consider that the network is not appropriate for the problem and will not be able

to learn. This test proves the contrary. It is possible that the network does not show any sign of learning, but is then suddenly able to get out of this situation, as if something clicked.

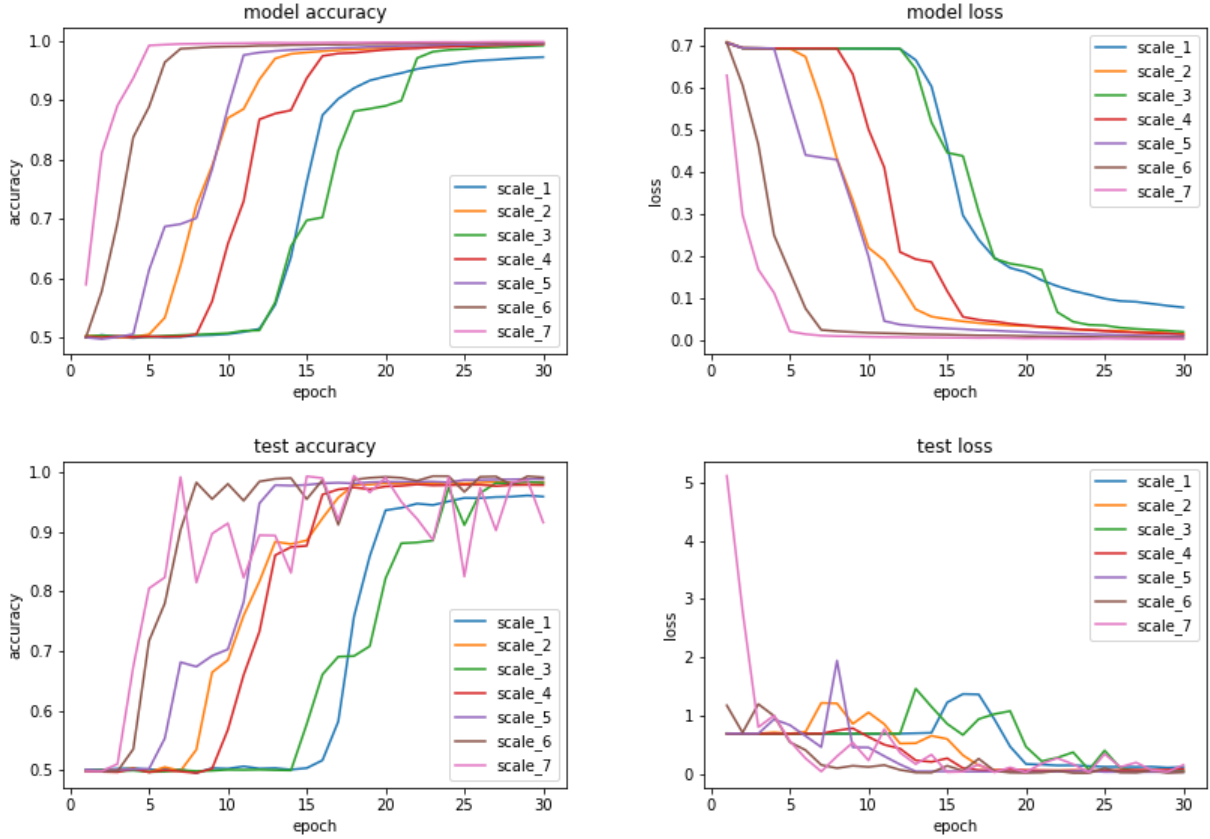


Figure 4.1: Performances of the CNN on images with different scale on the training and the test set.

If we look at the tests on unknown data, we can observe for all scales that the final accuracy is more or less the same than the one obtained during training. Moreover, if we inspect the evolution of the loss function, we can see a peak just before the learning phase. Fig. 4.2 illustrates it for a test. This peak appears when the network begin to learn on the training set. That means that the CNN is unable to generalise when it begins to learn. Fortunately, this peak last maximum one or two epochs and the network generalises well after.

Again, this result is quite unusual. For example, to avoid overfitting it is common to do early stopping using the test set. That means that we stop the learning when the results on the test set start worsening and/or are too poor in comparison with the results obtained on the training set. In this case, this method would have prevented the learning. This illustrates again that the CNNs are quite unpredictable.

On the other hand, networks trained for images at scale 6 and especially scale 7 seem to be very unstable in their generalisation. Indeed, as illustrated by Fig. 4.3, the performances of the networks on the test set are very variable. The accuracy goes from 50% to 100%. We cannot really speak of overfitting in this case as the networks are able to learn anyway.

Moreover, it is often possible to determine if the networks generalise well or not using a set of unlabelled data. Indeed, remember that the CNNs return an output containing the probabilities for the input image to belong to each class. The image is then labelled with the class with highest probability. If we consider this probability as the certainty level of the algorithm, we can observe that the mean certainty level of the network on the test set is almost always lower when it obtains a poor accuracy, as illustrated by Fig. 4.4.

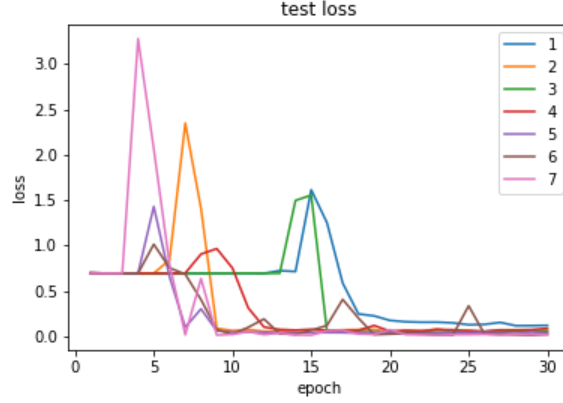


Figure 4.2: Peak in the test loss.

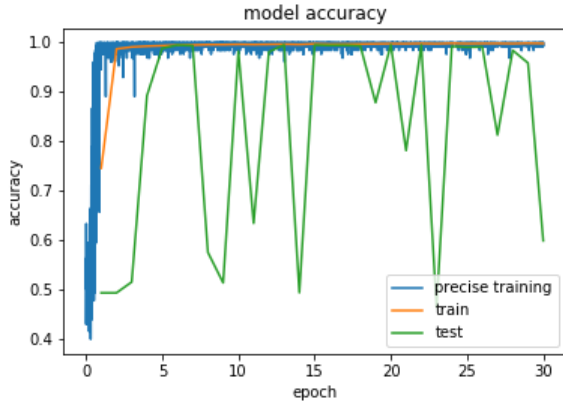


Figure 4.3: Instability of the 7th scale. The blue curve shows the evolution of the accuracy after each batch during training, while the orange curve is the summary of this evolution after each epoch.

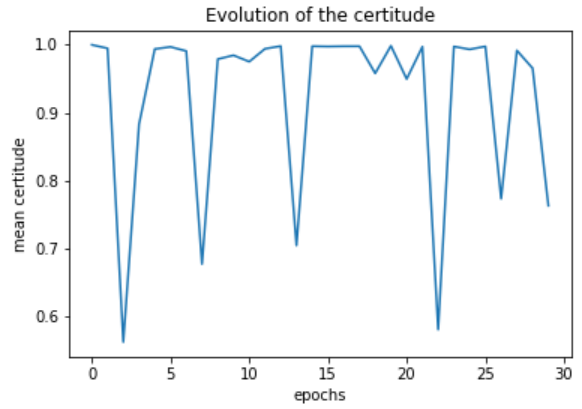


Figure 4.4: Certitude on the 7th scale.

We looked also at the generalisation on a second test set and we could observe that it was like the generalisation on the first test set. That means that the generalisation is not too dependent to the test set used (if it is different from the training set, of course). The problem here is naturally the fact that the certainty on the test set is "almost always" low when the network has a low accuracy on it, there is no sure way to verify that the CNN generalises well. Our CNN has thus an unstable generalisation on images at scale 6 and 7 that can be characterised using a dataset labelled or not, in a quasi certain way.

In general, there is no obvious link between the performances of the algorithm and the mutual coherence of the foreground dictionary with respect to the background one. Nevertheless, the concept of mutual coherence allows us to determine datasets on which the CNN has very different behaviours. Indeed, on the smallest scales, the network has a stagnation period where it is unable to learn and has performances like networks with random weights, while the higher scales are learned very quickly, but with an unstable generalisation on unknown images. Moreover, the first scale has an accuracy lower than the others.

Consequently, it is interesting to analyse these differences and try to explain them using or not the mutual coherence between the foreground and the background dictionaries.

4.1.2 The features learned by the CNN

To analyse the behaviour of the learning algorithm, we first looked at the features that it is discriminating. For that, we tested the transferability of each network on unlearned scales, or in other words, we tested each learned network on all the datasets available. That means that we tested the network learned on images with foreground at scale 1 using the tests set of scales 1 to 7, for example. Fig. 4.5 shows the median transferability on 5 tests for each scale. The lines are the performances of the CNN trained on images with foreground at scale i .

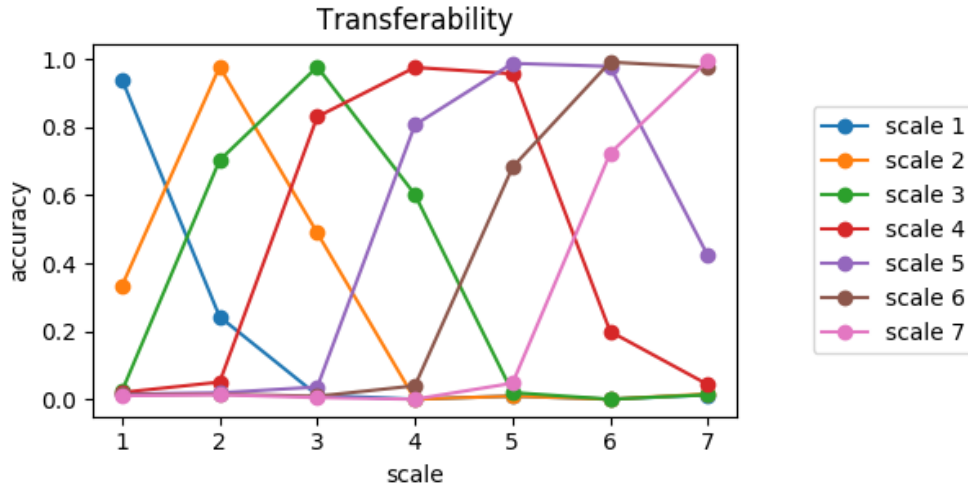


Figure 4.5: Median transferability of the network.

We can observe in this figure that the network always learns the foreground. Indeed, they are always scales on which the network is unable to generalise. That means that it does not learn exclusively the background and consider that there is a foreground when the image is too far from its model of the background, it must learn features from the foreground.

Moreover, it appears that the network is quite robust to foregrounds that are like the ones it has learned. Indeed, each line in Fig. 4.5 presents a plateau around the scale used for learning. The higher scales have also a bigger plateau and are thus more robust when they are able to generalise. Indeed, Fig. 4.3 illustrates a test where the network is not able to generalise at the end of learning. This network has logically poor results on all scales.

To verify if the learned networks are perfectly learning the foreground or not, we computed their transferability on images containing only a foreground and thus without background. Indeed, if the CNN is learning the foreground, it must be able to classify these images as images with foreground. The median transferability is presented in Fig. 4.6.

We can observe that for scales 1 to 5, the accuracy of the network trained on images with background, and a foreground at scale i , is almost always 100% on images without background, but with a foreground at the same scale. For these scales, the CNN learns thus perfectly the foreground. Moreover, it has slightly better performances on images without background than on images on which it has been trained. That shows that the network sees the background has a perturbation in this case.

Nevertheless, except for scale 1, each network has lower performances on images without background for at least one scale. This suggests that the background is also learned by the network. Indeed, adding the background helps the CNN in some cases. This sounds quite natural; the CNN is using all the information provided but focus more on the features of the foreground.

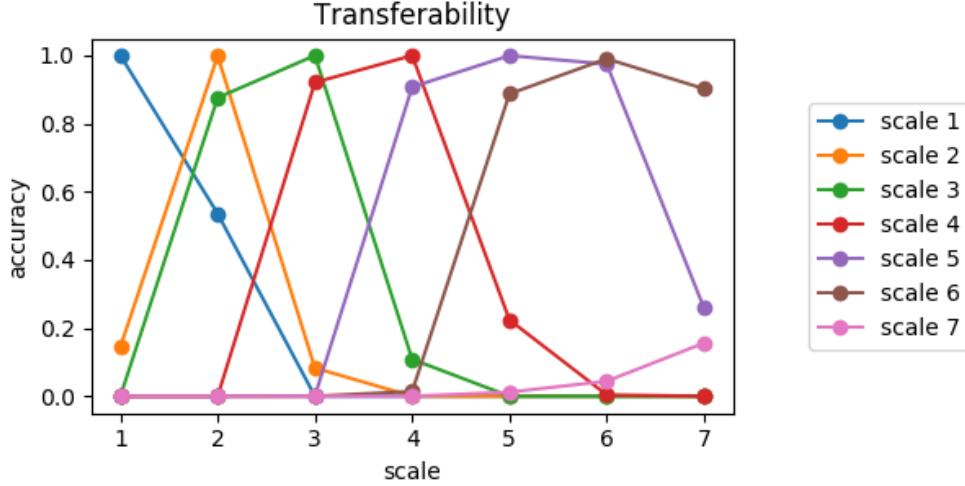


Figure 4.6: Median transferability of the network on images without background.

The behaviour of the CNN trained on images with foreground at scales 6 and especially 7 is very different from the others. Indeed, at scale 6, the test accuracy of the network on images without background but with a foreground at the same scale is not always 100% anymore. Sometimes it goes down to 85%. For scale 7, the performances are around 20% on average. In these two cases, the role of the background appears to be important to classify images. This is even crucial for scale 7.

Besides, on the 5 tests that we have done, one had a poor accuracy on the test set at scale 7. This network had poor results on images with background in general, but was among the best for images without background. Actually, it was the only network with higher results on images without background but with a foreground at scale 7.

This suggests that learning too well the foreground is not a good idea at such large scale. Indeed, the foregrounds at this scale are very similar to the backgrounds, due to the high coherence between the basis used to create them. Consequently, being able to recognise the foreground is not enough to distinguish it from the background, the network must learn both.

We constructed the foreground and the background as a combination of a few atoms from two dictionaries. We wonder if the network is also able to classify correctly the atoms from the DCT and the Haar wavelet dictionary at a fixed scale as well. Of course, atoms from the DCT dictionary should be classified as images containing only a background and atoms from the Haar wavelet dictionary should be classified in the other class. Surprisingly, only the networks trained on images with foreground built using Haar wavelet elements at scale 1 and sometimes 2 are able to classify these elements. On five tests, these two networks obtained a mean accuracy of 100% on atoms from the DCT dictionary and an accuracy of 92.915% and 40.7% respectively on atoms from the Haar wavelet one.

This is surprising as we just demonstrated that the networks trained at scale 1 and 2 are mainly learning discriminant features from the foreground. It appears thus that these networks are learning the combination of Haar wavelet atoms as well as the atoms themselves but in a less significant way. Moreover, the DCT atoms are always classified correctly. But the images in the training dataset containing only background were not always classified in the good class. A plausible explanation is that the combination of 16 elements from the DCT dictionary produces sometimes images with features that the network associate to the foreground, but that is not the case with only one atom.

For the other scales, the network is either classifying all the atoms as images with foreground or without foreground and this classification changes from one test to the other. Consequently, in these cases the network is learning the combination of the basic atoms. As the Haar wavelet contains only a non-zero pattern with variable size, that means that the network is looking for three of these patterns to classify an image as containing a foreground.

Thus, to determine the presence of foreground on images at scale between 1 and 5, the network mainly learns discriminant features from the foreground. The features of the background are also learned by the CNN (except maybe for scale 1), but seem to be less important. On the other hand, for scale 6 and especially 7, the network gives more importance to the learning of the background's features. In Sec. 4.1.1, we pointed that these 2 cases were more unstable in their generalisation than the others. This instability is maybe due to this difference in the discriminant features learned.

4.2 Impact of the frequency of occurrence

It is well known that the frequency of occurrence of each class in the training data influences the CNN. In general, it is recommended to use a balanced training set. However, in previous section, we pointed that when the foreground was constructed using a low-scale Haar wavelet basis, the network was mainly learning its features. Consequently, we can suppose that providing more images containing a foreground helps the training of the CNN. Even for the higher scales, the images with foreground contains in some sense all the information of the dataset, while the images containing only a background have only part of it. Of course, to be able to distinguish them the CNN needs training data from both classes, but we can wonder if providing more data from the hardest class to specify can help the CNN. We suppose thus that providing more images with foreground to the CNN always helps it to learn.

To verify this supposition, we constructed new training sets. These new sets have the same size that in previous section (20000 images), but the probability of an image to have or not a foreground is not anymore 50%. We consider the cases where 20%, 40%, 60% and 80% of the images in the set have a foreground. We trained again our algorithm during 30 epochs on 5 tests for each scale and each frequency of occurrence. Note that we used the same test set for all frequencies at a fixed scale. We did not change the loss function of the CNN even though it is recommended when we are working with unbalanced datasets. The goal is to analyse the influence of the input data, without changing the algorithm. The results are illustrated in Appendix B.2, but the main results are showed below.

4.2.1 When the foreground and the background have incoherent dictionaries

For scales 1 to 5, when the dictionaries used to build the foreground and the background are quite incoherent, our supposition appears to be quite accurate. Indeed, we can observe in figure 4.7 that having 40% or 50% of images with foreground makes the CNN significantly slower than when 60% or 80% of images have a foreground.

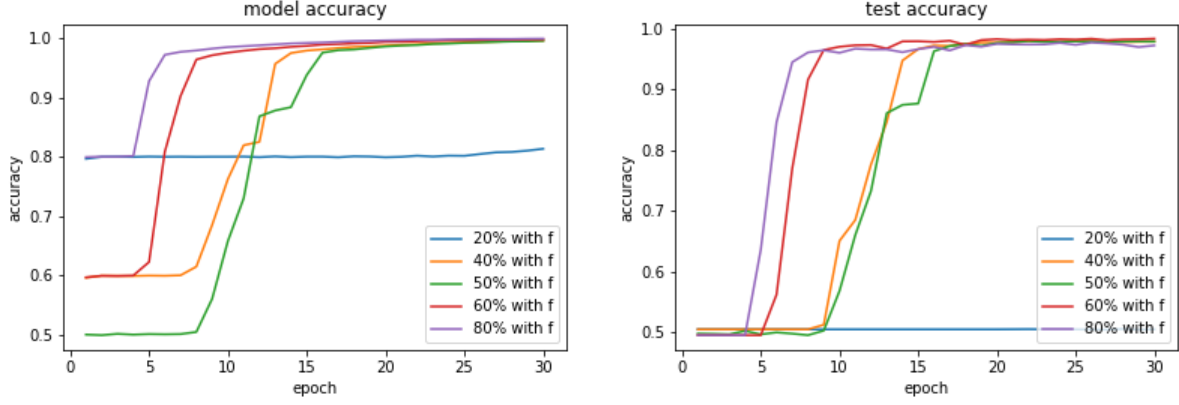


Figure 4.7: Performances of the CNN on images at scale 4 with different percentage of images containing a foreground in the training dataset.

Moreover, when only 20% of the training images contains a foreground, the network does not show signs of learning even on the training set. Note though that it is not the case for scale 5: in this case, the network is sometimes increasing the accuracy both on the training and the test set after 30 epochs, but it is very slow (see figure 4.8). Besides, the CNN has been able to learn on only 2 tests over 5. That means nevertheless that the network is not unable to learn when they are not much images with foreground available, but that it is very slow in this case. We can conclude the same for the other scales as we can see that the accuracy on the training set is slightly increasing during the very last epochs. We can suppose that as for scale 5, the network would have been able to learn if we had trained it for more epochs.

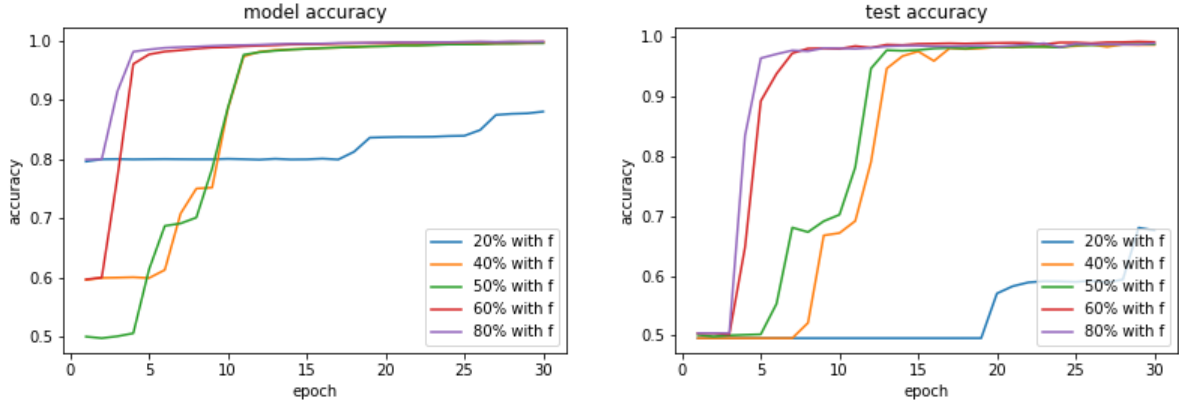


Figure 4.8: Performances of the CNN on images at scale 4 with different percentage of images containing a foreground in the training dataset.

We supposed that the network is able to learn when 4000 images in the training dataset contains a foreground, but that this learning is very slow when the training set contains a total of 20 000 images. To verify this hypothesis, we looked at the performances of the CNN when 4000 training images contains a foreground but the dataset has different size. Consequently, the percentage of images containing a foreground varies. We did this test at scale 1 as the network is quite slow at this scale. We can observe in Fig. 4.9 the network's performances when this percentage is between 25% and 80%.

We can see that decreasing the size of the training set (and thus increasing the percentage of images with foreground) until 8000 images (50%) speeds up the learning. The network is only able to generalise in this last case, but as it has not reached an accuracy around 100 % after the training phase in the other cases, it is not so surprising. Indeed, we could notice that the beginning of the generalisation of our CNN happens once it has learned the training set almost perfectly.

However, when the training set contains 6000 or 5000 images (66% - 80%), the network is overfitting. Indeed, it reaches an accuracy of 100 % almost immediately, but does not show any sign of generalisation.

This small test confirms that the network is able to learn when we provide it only 4000 images with a foreground. Nevertheless, we could observe cases where the CNN is overfitting. We can thus suppose that having too few training images leads the network to do memorization instead of learning.

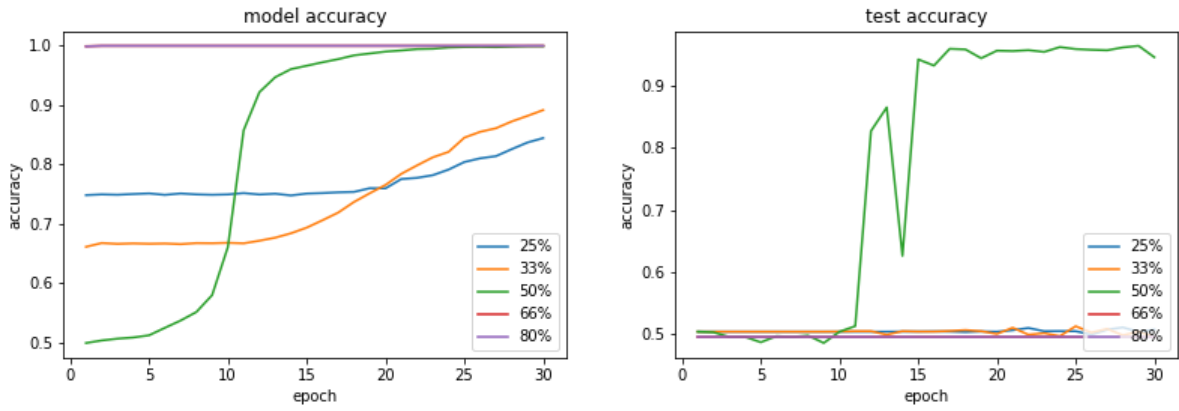


Figure 4.9: Performances of the CNN on images at scale 1 when 4000 images in the dataset contains a foreground.

On the other hand, we observed in Fig. 4.7 that having a lot of images with a foreground speeds up the CNN. But we know that having 0% of images containing only the background will certainly not allow the network to generalise. Indeed, the CNN will just classify systematically all the images as images with foreground with a certainty of 100% as this will provide him the best loss. It is thus interesting to look at the highest frequency of occurrence of images with foreground in the training set at which the network has a fast learning with a good generalisation on the test set.

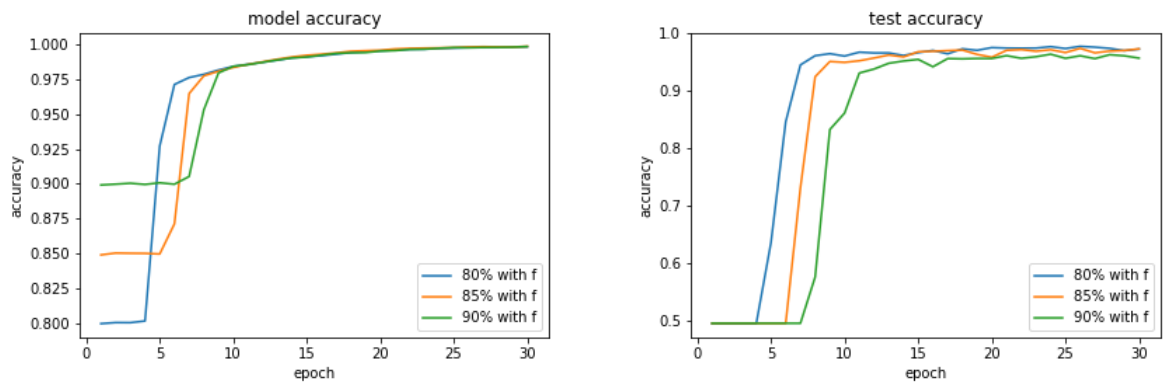


Figure 4.10: Performances of the CNN on images at scale 3 with high frequencies of occurrence of images with foreground in the training set.

Our tests allow us to conclude that for images with a foreground constructed using Haar wavelet elements at low scale, having 80% of training images with a foreground produces actually the best result. Indeed, when 85% or 90% of the images contains a foreground, the CNN is still able to learn quite fast during the training phase, but its generalisation is less stable and has lower value, as we can observe in Fig. 4.10. Moreover, the network is also slightly slower in these two cases.

For these small scales, if the data set has a fixed size of 20000 images, using 16000 images containing a foreground and 4000 without foreground is consequently the most efficient. This configuration has the fastest learning and is more reliable. This proportion can change if the size of the dataset changes, but in general our observations indicate that it is better to have a high proportion of images containing a foreground.

On the other hand, if we look closely at the results obtained at scale 1, we can observe that changing the frequency may avoid converging to an accuracy around 95% and go to 98-99% instead. Consequently, providing more images with foreground is doubly interesting in this case as it improves both the accuracy and the rate of convergence. This is illustrated in Fig. 4.11

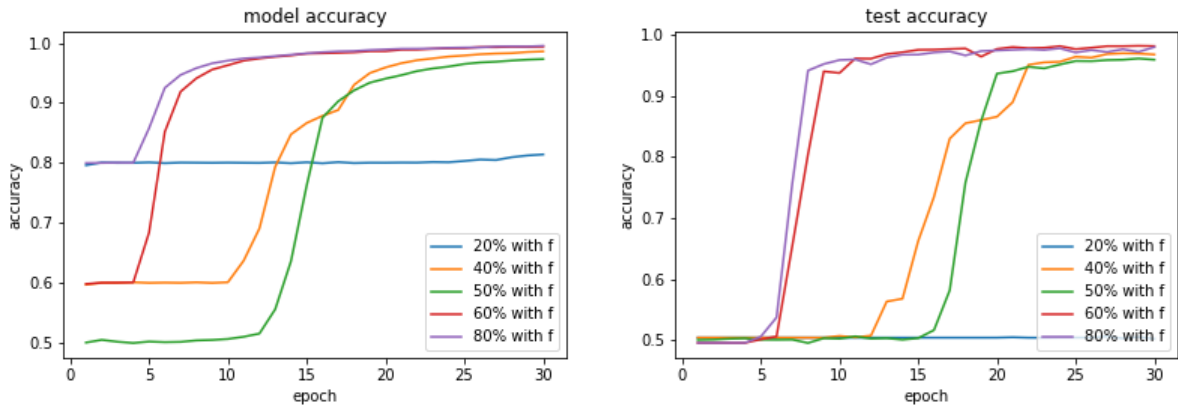


Figure 4.11: Performances of the CNN on images at scale 1 with different percentage of images containing a foreground in the training dataset.

4.2.2 When the foreground and the background have coherent dictionaries

For scales 6 and especially 7, the frequency does not appear to have a lot of impact, as illustrated by Fig. 4.12. The learning is fast anyway, and the generalisation is still unstable. Actually, at scale 6, having only 20% of images with a foreground makes the algorithm a bit slower, especially to generalise, and it is also slightly observable at scale 7. Our supposition make thus still sense in this case but has a drastically lower impact.

Fig.4.12 shows that the learning is similar for frequencies of occurrence of the foreground varying between 20% and 80 %. We wonder until which frequency the network is able to learn, even if it is in an unstable way. We can observe in Fig. 4.13 that increasing the frequency of occurrence of the foreground above 95 % increases also the stagnation time and slows then down the learning. Nevertheless, we can say that until 99% of foreground images, the final performances of the algorithm are comparable after 30 epochs. Even when only 20 of the 20000 training data does only have a background (99,9% of the images have a foreground), the CNN shows sign of learning at scale 7. It seems thus that it is able to generalise in a reasonable time in this case also. Of course, the generalisation stays unstable, but it is the case for all the tests at high scales.

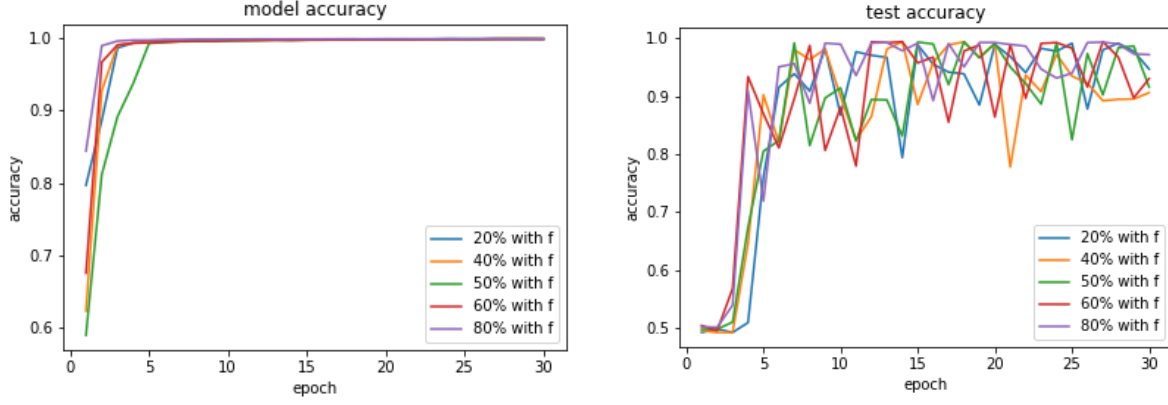


Figure 4.12: Performances of the CNN on images at scale 7 with different percentage of images containing a foreground in the training dataset.

We could observe similar results when the percentage of images with foreground was very low in the training dataset, expect that the network’s performances were very low at 1.25% already. It appears thus that the minimal required number of images containing a foreground is slightly higher than the minimal number of images containing only the background.

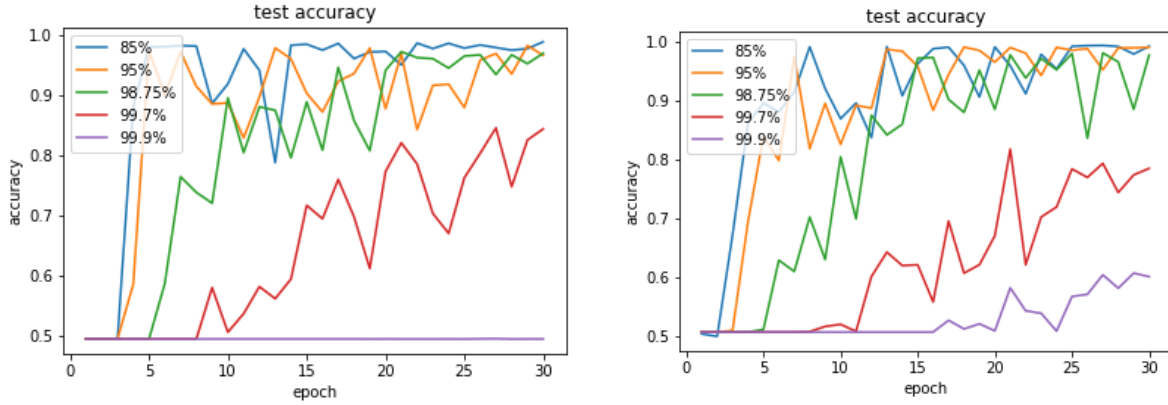


Figure 4.13: Performances of the CNN on images at scale 6 (left-image) and 7 (right-image) with high frequencies of occurrence of the foreground in the training dataset.

To conclude, when we try to classify images with or without foreground, but with similar backgrounds, it is interesting to provide more images with foreground to the CNN, as these images contain more information. This is even more important when the images are composed of foregrounds and backgrounds build using incoherent basis. Indeed, in this case, the network learns mainly features from the foreground and is thus faster when it sees more images containing this information. Of course, these results were obtained on one fixed network architecture and are thus not necessarily correct for other architectures.

4.3 Observations with a limited range of possible foreground

In this section we observe the behaviour of our CNN on images whose potential foreground is constructed using a restricted number of atoms at a fixed scale from the Haar wavelet dictionary.

Actually, as the atoms of this dictionary are translations of three distinct patterns, we decided to restrict the location of the foreground in the image. To do so, we defined a foreground containing each of the three patterns that we named "elementary foreground". The position of a pattern is defined as the position of its lower-left non-zero pixel.

Each pattern can be translated into a square of 32×32 pixels, or in other words its position is limited to a square of 32×32 pixels. We restricted then the dictionary of the foreground to the Haar wavelet atoms at scale j that produces such patterns. Figure 4.14 illustrates the construction of the foreground.

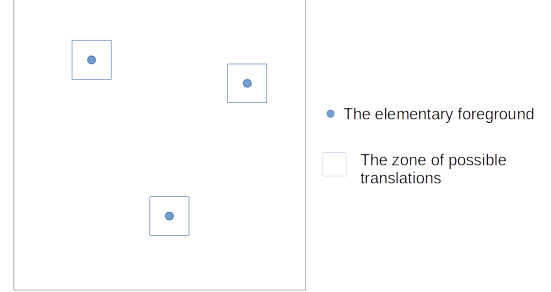


Figure 4.14: Construction of the foreground

During the training, the images are potentially flipped horizontally or vertically. Nevertheless, this construction of the dataset produces images whose foreground has limited possible positions.

Intuitively, this problem seems to be easier to solve for the CNN, especially for the smallest scales. Indeed, we pointed in section 4.1.2 that when the foreground is built using Haar wavelets coefficient at scales 1 to 5, the network is mainly learning the foreground to classify the images. But as the foreground is limited in this case, it should be easier to learn.

For scales 1, 2 and 3 this hypothesis seems to be confirmed by the experiments. Indeed, we can observe in Fig. 4.15 that the stagnation phase almost disappears in this case. Moreover, scale 1 reaches an accuracy around 99% after the training, while it has only an accuracy around 95% when the foreground was random (see Sec. 4.1.1). Nevertheless, the network is still able to generalise on the test set one epoch after than on the training.

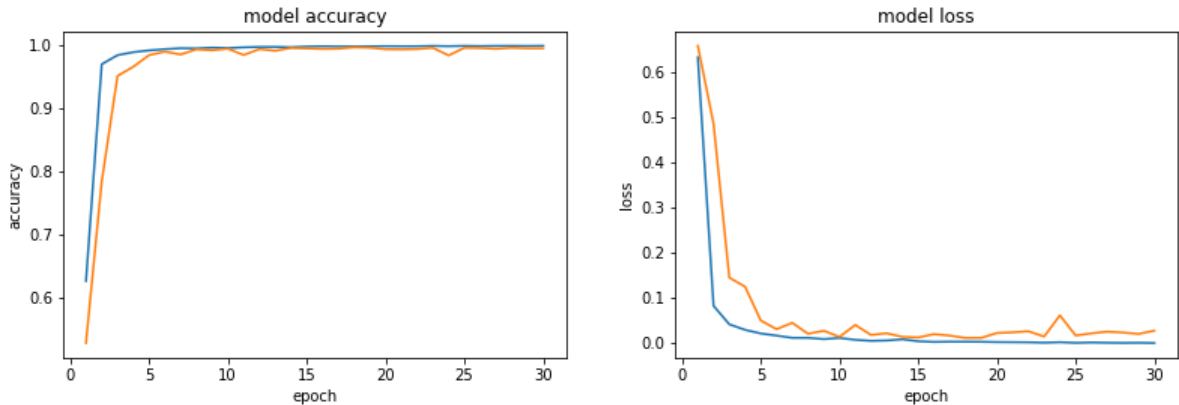


Figure 4.15: Model accuracy and loss on images with foreground at scale 2 on one test.

When we look at the performances of the networks trained on datasets with foregrounds at scale 4 or 5, we can still see that the training is faster than when the foregrounds are random, but the generalisation is a bit slower than before for the smaller scales. Indeed, as illustrated in figure 4.16, the CNN needs more or less 5 epochs before being able to generalise on the test set at scale 5 while it needed 10 epochs in the random case. The improvement due to the restriction of the foreground is thus less important in this case, even though it is still present.

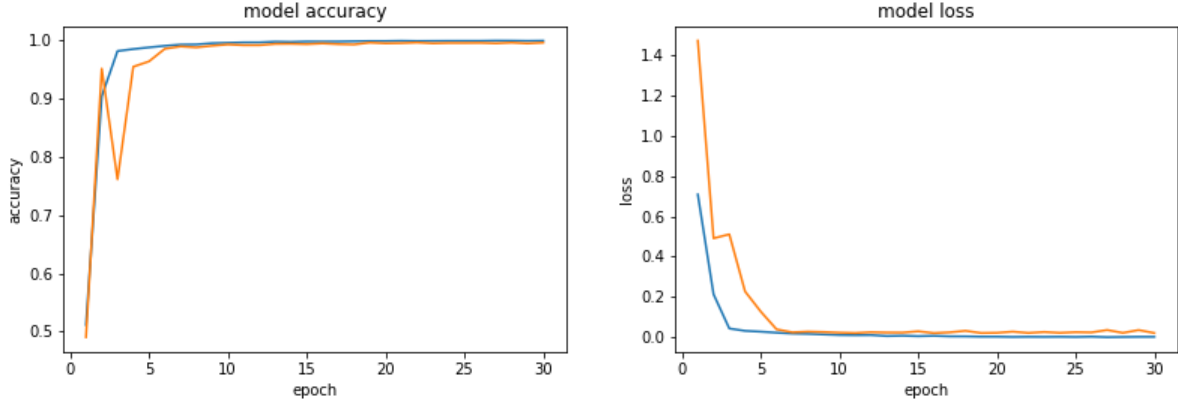


Figure 4.16: Model accuracy and loss on images with foreground at scale 5 on one test.

When we look at the performances of the algorithm at scales 6 and 7, we can observe that the network is not anymore faster when the foreground is limited. Indeed, figure 4.17 illustrates that the network needs more or less 4 epochs to generalise on the test set while it needed 2 epochs previously. However, the generalisation of the CNN is much better when the foreground is limited. Even though the accuracy is not always equal to 100% on the test set the generalisation of the network is much less unstable than in the random case. Consequently, we can say that reducing the range of the foreground is also benefit when it is built at large scale.

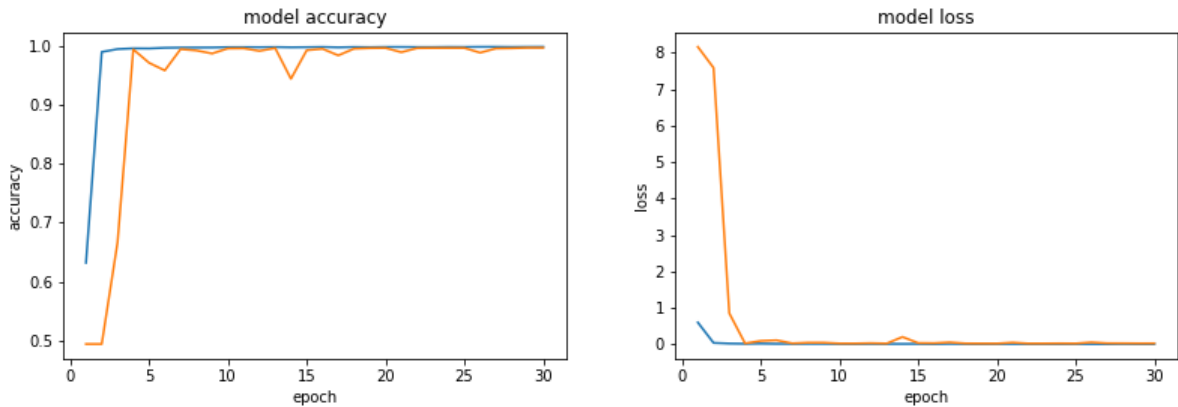


Figure 4.17: Model accuracy and loss on images with foreground at scale 7 on one test.

In this section we could observe that when the foreground is limited, the CNN learns faster than in the random case for the small scales. When the foreground has been built using high-scale wavelets, the network is slower when the range of possible foreground is reduced but has a better generalisation. This variation must be due to a difference in the features learned by the algorithm.

Chapter 5

Dissecting the CNN training

In this chapter, we are going to analyse the evolution of our convolutional neural network during the learning. For that, we will look at the weights learned at each layer, but also at the gradients back-propagated during the training. Then, we will observe the contents of each filter, and try to see if we can observe characteristic features from the background or the foreground of our constructed images.

5.1 The evolution of the learned weights

We could observe in previous chapter and especially in Sec. 4.1.2 that our CNN is not always learning the same features when the foreground is created using Haar wavelet elements at different scales. However, our CNN contains several layers, as illustrated by Fig. 2.7. It is thus interesting to look for the role of each layer in the learning, the features that they learn, and their importance.

For that, we computed first the mean value of the absolute difference between the weights of two consecutive epochs for each layer. As the weights are tensors, we flatten them before to compute the mean value. Fig. 5.1 shows the evolution of this value when the foreground is built using the Haar wavelet dictionary at scale 1 and 7. When we compare these graphs with the observations made in section 4.1.1, we can observe that the peak in the evolution of the mean corresponds to the beginning of learning. Moreover, this peak is more important for layers GL1 and GL2 as well as the output layer in some cases. But layers GL1 and GL2 are the two first layers in the network. It seems thus that having good weights in the first layers is very important for the learning of the network.

On the other hand, we can observe for the highest scales that the weights are not varying much after the learning. Indeed, their variation is similar to the one observed for the smaller scales. The instability observed in Fig. 4.3 can thus not be explained by huge variations in the weights. Moreover, the height of the pick is similar for all scales. Actually, it is slightly lower for the highest scales. Nevertheless, it seems that the distance between the initial and the final weights is similar in all cases.

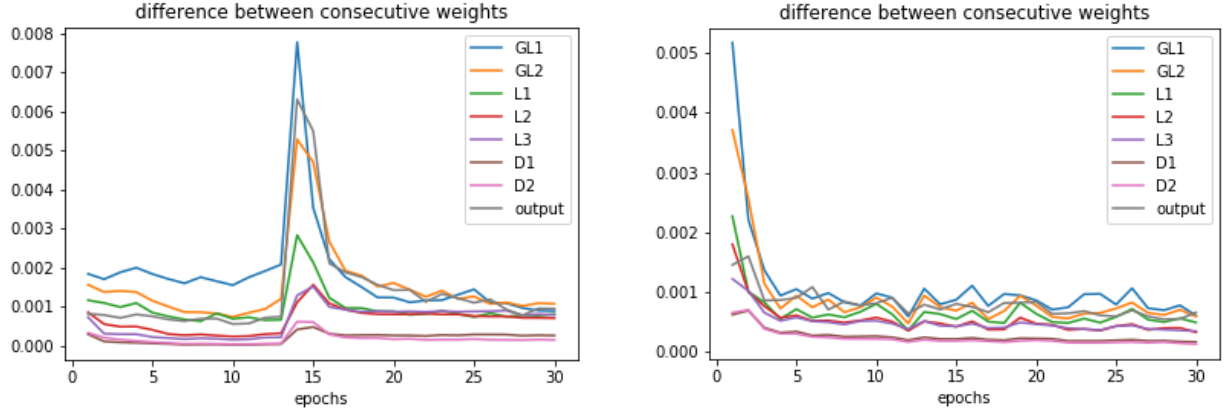


Figure 5.1: Evolution of the mean weight difference between two consecutive epochs for scale 3 and 7.

To verify this hypothesis, we computed the mean absolute difference of the weights at each epoch with respect to the final weights for each layer. Fig. 5.2 shows the evolution of this value for scales 1 and 7. We can see in this figure that the final weights of GL1 and GL2 are almost twice closer to their final value for scale 7. The random initialisation for layers GL1 and GL2 seems thus more appropriate when the foreground is build using high-scale Haar wavelets transform. This is maybe one of the reasons why the network is faster for scales 6 and 7. For the other layers, the difference between the two scales is smaller.

Nevertheless, we can observe that the two dense layers do not change much during the learning, and their evolution is similar. Layers L1, L2 and L3 (the 3 last convolutional layers) also change more or less in the same way during the training. For scale 7, the two first layers also change together while for the smaller scales, the layer GL2 is slower than the others to reach its final value. This is observed until scale 5.

Moreover, it is interesting to notice that the weights do never move away from their final values. This is probably the sign that the weights that change significantly does it in a monotone way.

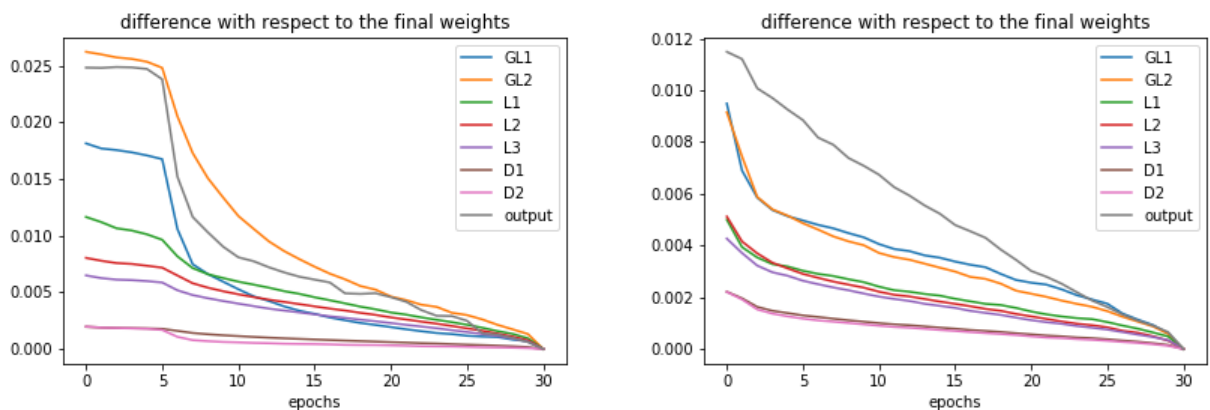


Figure 5.2: Evolution of the mean weight difference with respect to the final weights for scale 1 and 7.

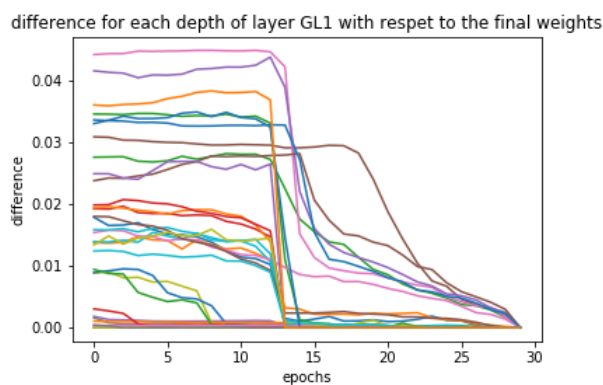
Remember that each convolutional layer has a fixed depth d . It is thus composed of d different filters. We wonder if all the filters have the same importance in the learning. In other words, does the network uses all the filter to learn, or are there filters more important than others? To answer this question, we looked again at the mean absolute difference of each layer weights with respect to its final value but this time filter by filter. We could observe that the evolution of each layer is more or less the same for each scale. The only differences are the length of the stagnation time in the beginning that is longer for the smaller scales, and the value of the mean absolute difference that is lower for the higher scales. But that has already been observed in the previous tests.

Fig. 5.3 illustrates the evolution of each layer at different scales. We can observe that the filters of a given layer does not have the same importance. Indeed, some filters change much more than others. Unfortunately, the proportion of important filters is hard to determine. The filters form a block more or less thick that close up progressively. They are rarely filters that set themselves apart from the others, especially in the three last convolutional layers. Actually, in some tests we can see some layers that change significantly more than the others, but this behaviour disappears in the next test. Consequently, it cannot explain the general behaviour of the CNN that stays very similar at each test.

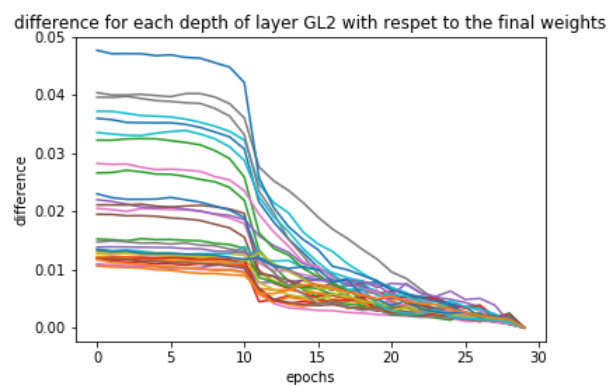
Let us now examine more precisely the behaviour of the filters in the two first convolutional layers, GL1 and GL2. These two layers have both a depth equal to 32, but their filters have quite different behaviours. In GL1, the mean change of the weights is more spread than in GL2 even though the most changing weights are more or less at the same distance from their final value. Actually, in GL2, the filters form kind of clusters, or groups. Indeed, in the example figure we can class the filters in 3 or 4 distinct groups of filters that evolve in the same way. Are the filters in a same group learning similar features? It is hard to say at this stage since the filters are initialised randomly, but it is interesting to keep this possibility in mind during the next tests.

On the other hand, it appears that in GL1 some filters are not changing, or nearly not changing, during the training. This may be the sign that the CNN does not need all the filters to learn. The filters were initialised randomly and contain thus non-zero weights, but the fact that they do not change is the sign that the back-propagated gradients are zero or close to zero at these filters. Indeed, we update them using the AdaDelta method that is an improvement of the gradient descend method. Basically, the weights are update with the following rule $x^{t+1} = x^t - \alpha g_t$ where g_t is the back-propagated gradient at weight x and α is a coefficient determined by the AdaDelta method. A null back-propagated gradient during all the training can be the sign that the activation produced by the filters is null as well.

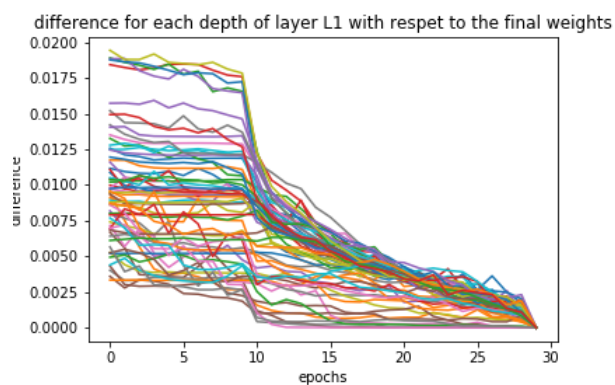
We did not look at the activation pattern of the first filter, but we verified if the number of filters in the first layer is appropriate for the learning. We trained thus the CNN on the same datasets but with fewer layers in GL1. We could observe in these small tests that the stagnation phase was not reduced for the first layers but was sometimes increased instead. The number of filters in the first layer is thus not too large.



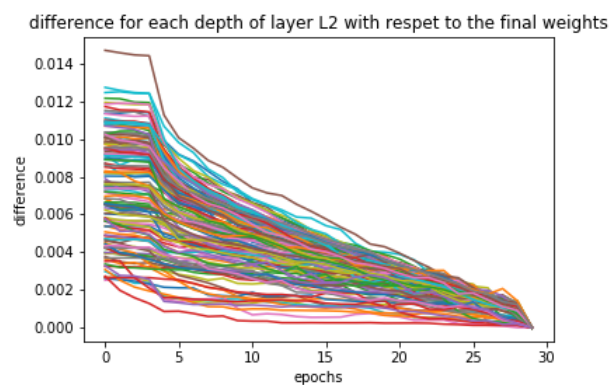
Evolution of the filters in GL1 at scale 1.



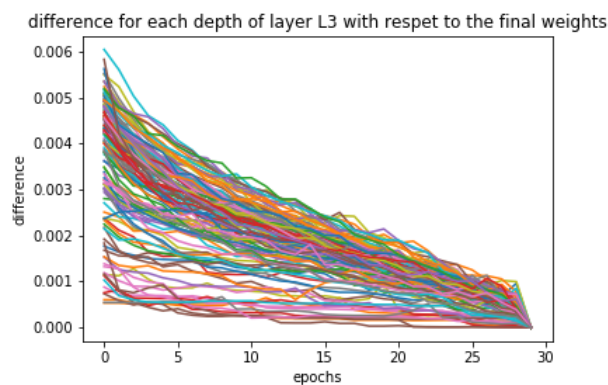
Evolution of the filters in GL2 at scale 2.



Evolution of the filters in L1 at scale 4.



Evolution of the filters in L2 at scale 5.



Evolution of the filters in L3 at scale 7.

Figure 5.3: Evolution of the filters in each layer at different scales

Comparison between weights trained on different datasets

In Sec. 4.1.2, we pointed that the CNN is often able to classify images with a foreground close to the one used for training. In our datasets, that means that a network trained with images having a foreground at scale i can classify images at scale $i-1$ and $i+1$, sometimes with high accuracy.

This behaviour can be due to similar values in the network’s weights. When the images are similar, the weights change maybe in a comparable way. To verify this hypothesis, we computed the mean absolute difference between the weights of networks trained at different scales, but with the same initialisation.

In general, we can observe an evolution like the one in Fig. 5.4. This figure is in the continuity of the observations made in this section. Indeed, we can observe that the two first layers, GL1 and GL2, as well as the output layer are the most changing layers.

Moreover, we can see two steps in the evolution, each time one of the networks begins its learning. But even though the network trained at scale 2 has an accuracy around 45% on images with foreground at scale 3, and the reverse case has an accuracy around 65 %, the weights of these networks are quite distant. Indeed, their mean absolute difference is like the one between the CNN trained at scale 1 and the one trained at scale 7. But these two networks have an accuracy of 0 % on images with a foreground at the other scale.

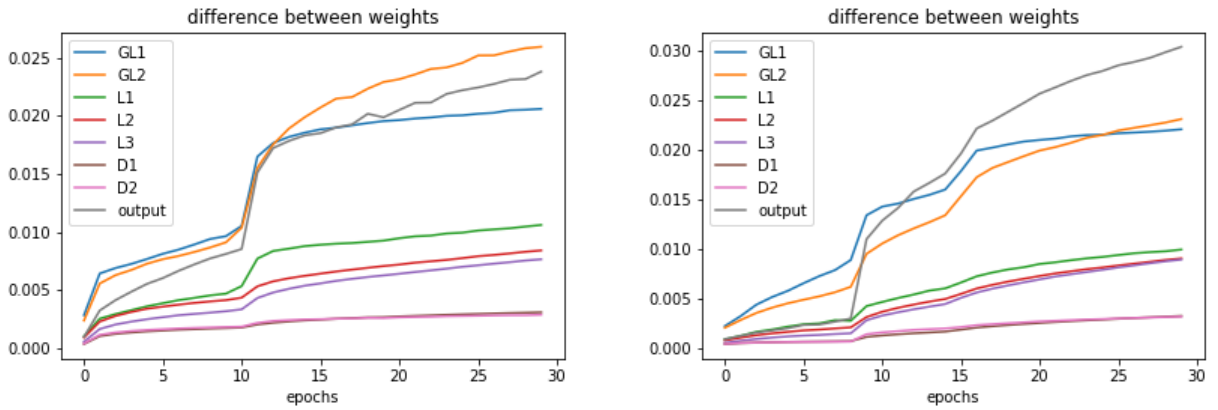


Figure 5.4: Evolution of the mean weight difference between scale 1 and 7 (left-image), and between scale 2 and 3 (right-image).

Nevertheless, we can observe that the highest scales have closer weights, as illustrated in Fig. 5.5. But this is probably not directly due to the fact that they are learning more similar weights. First, even though it is not illustrated here, the weights of networks trained at scale 5 and 7 are as similar as the weights of networks trained at scale 5 and 6. But these two networks have poor performances on images from the other training dataset.

Actually, if we remember the results presented in the beginning of this section and especially in Fig. 5.2, we can observe that the distance between the weights of two networks is similar to the difference between these weights and their initialisation value. That means that in general the weights of a network trained on images at a certain scale are as far from their initialisation than from the weights of the same network trained at another scale. Note that if the networks are at two different distances from the initialisation, they will be at the same distance from each other after the learning phase than the highest difference from initialisation.

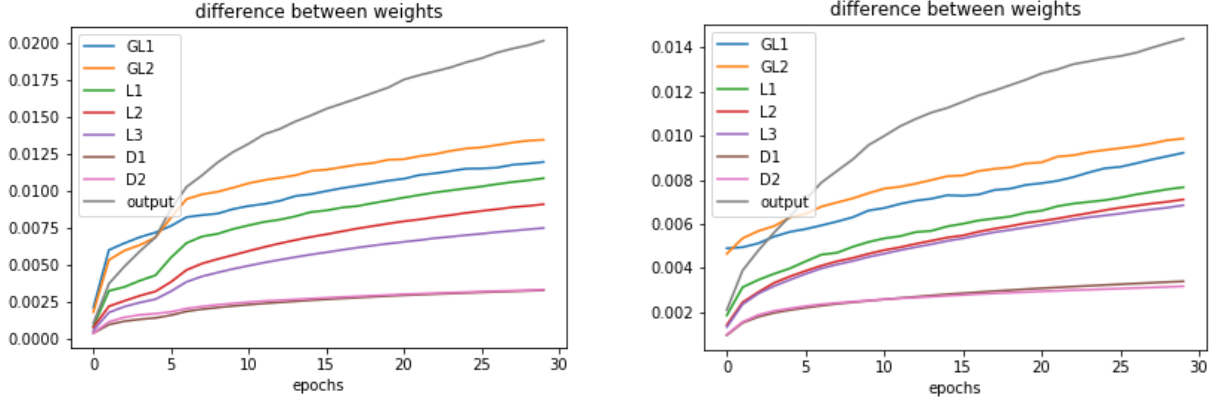


Figure 5.5: Evolution of the mean weight difference between scale 5 and 6 (left-image), and between scale 6 and 7 (right-image).

5.2 Evolution of the gradients back-propagated

We pointed in previous section that the depth of the convolutional layers does not explain the stagnation observed for the small scales in Sec. 4.1.1. Indeed, the phenomenon correspond to a stagnation of each filter in each layer.

To understand this stagnation, we looked at the gradients that are back-propagated by the algorithm. Indeed, we are using a stochastic gradient method with AdaDelta to update the weights of the network. The update of the weights depends thus mainly on the value of the gradients, but also on the previous values of the gradients and the previous updates.

Indeed, let g_t be the gradient of the loss function that we want to minimise, with respect to parameter x at iteration t and u , G be two accumulative variables and p the decay rate (in this case 0.95). The AdaDelta method is written as

$$\begin{aligned} G_t &= pG_{t-1} + (1-p)g_t^2 & u_t &= pu_{t-1} + (1-p)\Delta x_t^2 \\ \Delta x_t &= \frac{\sqrt{u_{t-1}}}{\sqrt{G_t}} g_t & x^{(t+1)} &= x^{(t)} - \Delta x_t. \end{aligned}$$

Let us first have a look to the values of the gradients back-propagated in the beginning of the learning. Since we are using stochastic gradient descend, the gradients are computed for batches of data, that are provided one after the other to the algorithm, in order to browse all the training data set. Moreover, the batches are built randomly. The gradients propagated are thus not the same during each tests. Nevertheless, to approximate them, we computed the gradients on several batches of 128 images that form a set of 5000 images were 50 percent contains a foreground. Moreover, we decided to put images from only one class on each batch, to can see if they are differences between the gradients that they back-propagate.

It is intriguing to see that the gradients back-propagated in the network at the initialisation are very similar for all scales. Indeed, even though the gradients are different for different weights initialisation, they are more or less the same for different foregrounds, as illustrated by figure 5.6.

It is natural that images without foreground have the same gradients in all cases as they are constructed in the same way. This is not the case of images with a foreground. The fact that the mean gradient on these images is the same for each kind of foreground means that they produce statistically the same activation in the network. Indeed, as the architecture of the network and its loss function is the same in all cases, the back-propagated gradients will be different if and

only if the activation values are different (see section 2 for an explanation of the backpropagation, the algorithm used).

This can be a sign that the backpropagated signals are dominated by the background and that the foreground has a negligible influence on values backpropagated by a random initialisation of the network. However, the gradients are not exactly opposed. In Fig. 5.6 the gradients propagated by the images without foreground have clearly a lower amplitude than the ones propagated by images containing both a background and a foreground. We can think that this phenomenon is linked to the batches we used or more generally to the training data set. Indeed, the backgrounds of the images containing also a foreground are constructed in the same way that the images without foreground, but as the DCT dictionary contains 65 536 different atoms, we can create $(65536)^{16}$ different backgrounds. We did thus not build all the possible backgrounds in a dataset of 20 000 images only. Consequently, the difference observed could be simply a coincidence.

But this explanation is not convincing. Indeed, the databases at different scales were constructed in an independent way, we should thus observe differences between the different scales. Therefore, it appears that the presence of the foreground has an influence on the gradient back-propagated in the network, but it seems to be lower than the one of the background. Moreover, the influence is the same for all the datasets that we built.

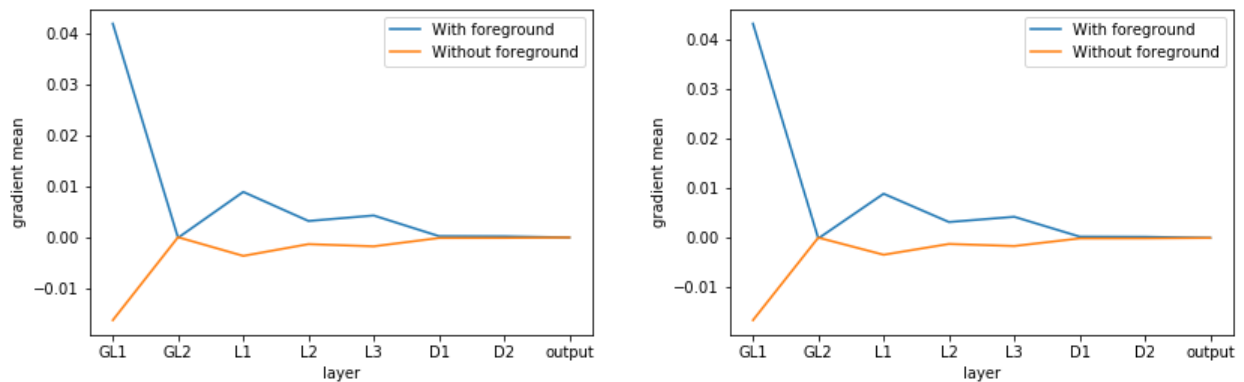


Figure 5.6: Mean value of the gradient at initialisation for scales 2 and 6.

Consequently, a network with random weights does not make any difference between the datasets that we constructed. The very first update(s) are apparently the same for all scales. Their different behaviour occurs then later in the learning. This conclusion seems to be in contradiction with the observations made in section 4.1.1, as it would imply that the different algorithms have the same learning behaviour at the first epoch. But it is not the case, the seventh scale has clearly a different behaviour than the others.

The images in section 4.1.1 illustrate the evolution of the accuracy and the loss during learning after each epoch. But the CNN updates its weights after each batch. Moreover, the results on the initial weights are not represented on these figures, but they were very similar for each scales. Therefore, the fact that the very first updates are the same for all scales is not in contradiction with the general results, as illustrated by figure 5.7 that shows the evolution of the accuracy after each batch during the first epoch for scale 1 and 7. The two CNNs do not have the same progression in the beginning as they do not see the same batches at the same time, but they both have an accuracy between 40% and 60% during at least 50 batches.

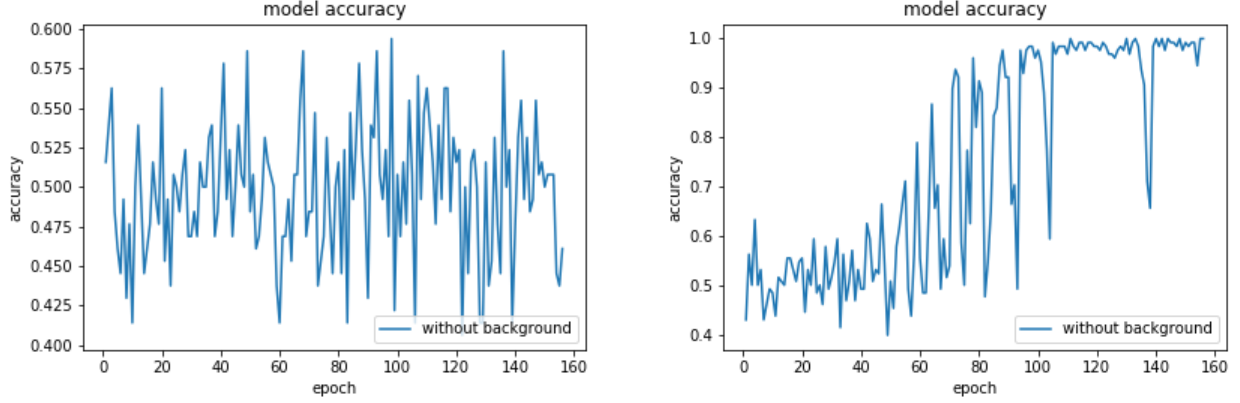


Figure 5.7: Evolution of the accuracy after each batch for scales 1 and 7 during the first epoch.

On the other hand, we can observe in figure 5.7 that the gradients of the images with or without foreground are opposite. This is quite natural as they must be classified in two different classes. Moreover, it seems that the mean gradient value in the first layer is more important than in the others and that they are several layers with a zero-mean value of the gradient. To see if this zero-mean value is due to low gradients values or not, we looked at the absolute mean of the gradients. We obtained again results very similar for all scales, that are illustrated on two different initialisation in figure 5.8.

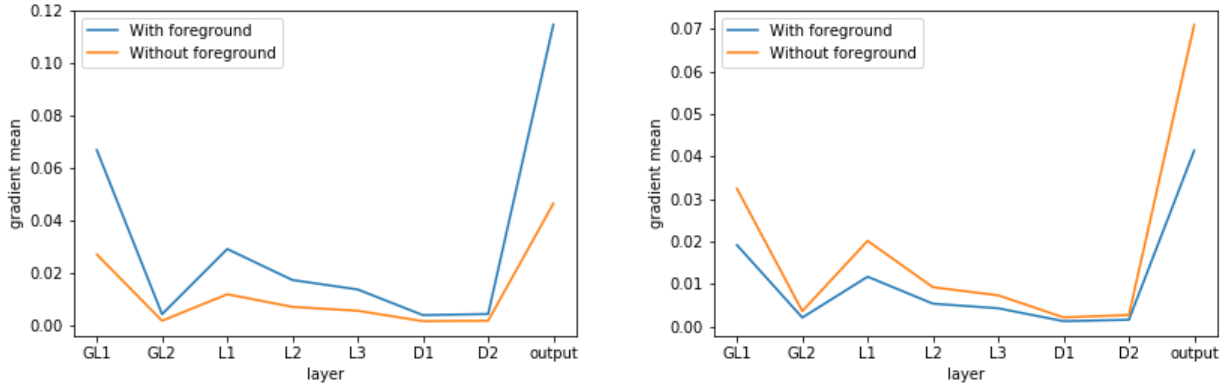


Figure 5.8: Absolute mean value of the gradients at two initialisations for scale 1.

We can see in this figure that layers GL2, D1 and D2 back-propagate gradients with a low mean absolute value. The weights in these layers will thus not be much updated in the beginning of the learning. On the contrary, the first layer GL1 and the last layer, the output layer, have gradients with a high mean value in comparison to the others. The weights of these two layers will thus change more than the others in the beginning of the training.

We looked then at the evolution of the gradients in each layer during the training. For that, we computed the mean absolute value of the gradients after each epoch. Again, we distinguish gradients coming from images with foreground from gradients of images without foreground. We can observe that even if the mean gradients are very different in these two cases, they do not present similar behaviour from test to test. Nevertheless, we can see interesting behaviours in general.

Images in Fig. 5.9 show the evolution of the gradients in one test at scale 3. We can observe that at a certain point, the mean value of the gradients at layer GL1 increases suddenly, and stays large until the end of the training phase. This point corresponds to the beginning of learning. This behaviour is observed for all the small scales, until scale 5.

We observed in Sec. 5.1 that the values of the weights at GL1 change a lot when the network begins to learn, but they do not change much after that. It is thus surprising that the gradient stays high and even increases during the last epochs. The update method that we used, the AdaDelta method is recalled in the beginning of this section. This method updates the weights as $x^{(t+1)} = x^{(t)} - \Delta x_t$. The fact that the weights does not change much means that Δx_t is low, and thus is $\frac{\sqrt{u_{t-1}}}{\sqrt{G_t}} g_t$. As g_t is the gradient value that we know is high, either $\sqrt{u_{t+1}}$ is small or $\sqrt{G_t}$ is high.

We know that $G_t = pG_{t-1} + (1-p)g_t^2$, with $p = 0.95$. When the network begins to learn, the value of G_t is not very high as the gradients are low in the beginning of learning, except at the initialisation, but the influence of the initialisation is minimal at the 10th epoch. Moreover, after a while the value of G_t will be more or less equal to $g^2 t$, as the gradients remain around the same value, and the gradients will thus be normalised.

On the other hand, u_t is equal to $pu_{t+1} + (1-p)\Delta x_t^2$. When the network begins to learn this value is low as the weights do not change much before and the update coefficient is only 0.05. Consequently, u_t smooths the influence of the huge increasing of the gradients in the beginning.

Note that when the gradients begin to be significantly larger, when the network begins to learn, the value of u_{t-1} and G_t do not change much due to the small update coefficient. Consequently, the weights change much more than before. This must last during a few batches. Indeed, we observe the evolution of the weights after each epoch while they are updated after each batch. The fact that we observe a peak in the weights evolution on each test is thus a sign that the weights are higher during several batches. Then, once the influence of the first epochs with higher gradient is lower in the two accumulative variables, G_t is high enough to normalise the influence of g_t and the evolution of the weights is thus lower.

We can also observe that the gradient is very high at the initialisation. Nevertheless, it does not seems to have much influence in the learning. The gradient must decrease very fast in the first batches and the influence of this high gradient is smoothed and can thus not be observe when we look at the gradients after each epoch.

Note that we did an analyse using the gradients at layer GL1 because its back-propagated gradients have a significantly higher value than the gradients of the other layers. However, we explained in Sec. 2.2 that our CNN contains a batch normalisation of the first convolutional layer only. Consequently, we can suppose that this difference between the backpropagated gradients is due to this normalisation. The conclusions that we made can thus probably be extends to the other layers.

The peak observed in the evolution of the weights in Sec. 5.1 can thus logically be explained by the evolution of the back-propagated gradients and the update method used.

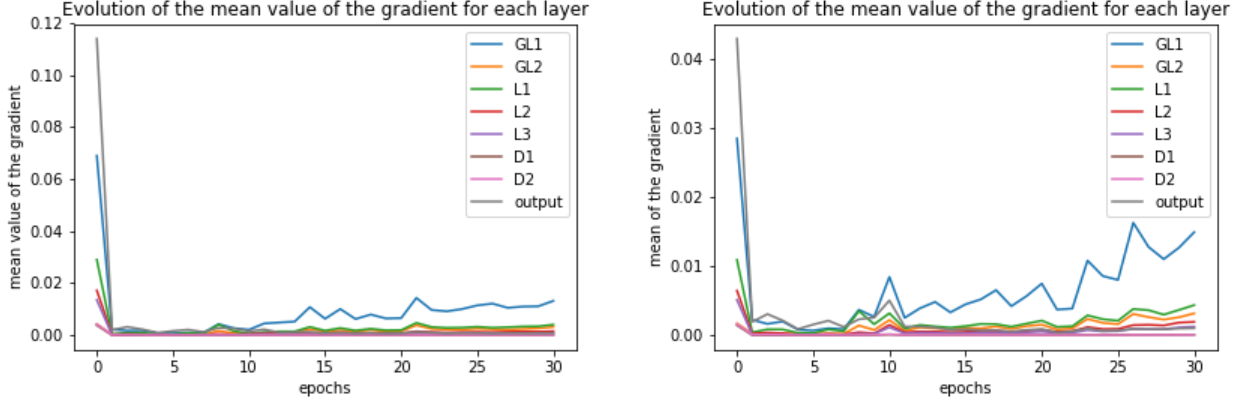


Figure 5.9: Evolution of the gradient mean value at each layer during the training one images with (left-image) or without (right-image) foreground at scale 3.

When the foreground is built using the Haar wavelet dictionary at scales 6 or 7, the mean absolute value of the gradient backpropagated is comparable to the one observed for the smaller scales, except that the gradient in GL1 is always high, even though it is lower than at the initialisation, as illustrates on Fig. 5.10. This is quite natural as the network obtains good results on the training after the first epoch. Consequently, we cannot distinguish the beginning of its learning on a graph with the gradients after each epoch. To be able to distinguish it, it would be interesting to look at the gradient after each batch. Unfortunately, we did not have the occasion to do it, as compute the values of the gradients is quite time-consuming.

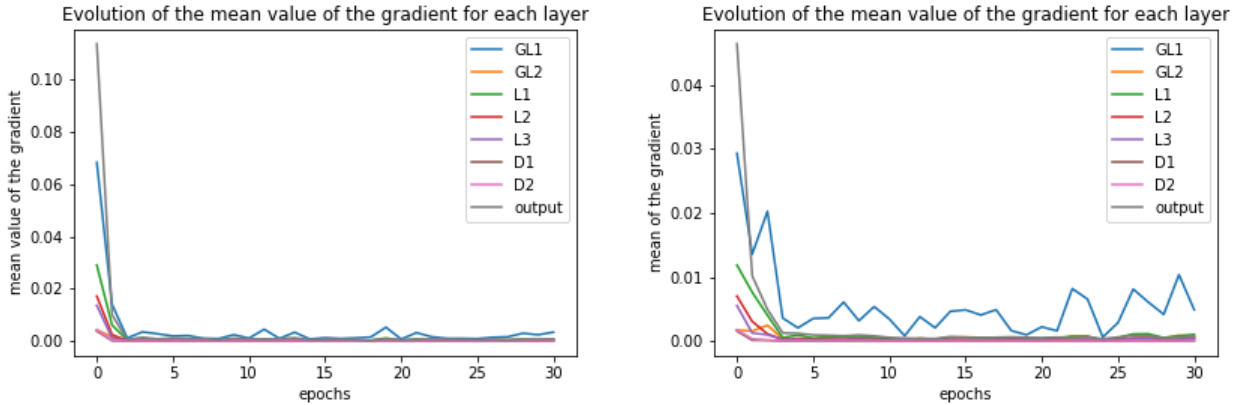


Figure 5.10: Evolution of the gradient mean value at each layer during the training one images with (left-image) or without (right-image) foreground at scale 7.

To conclude, in this section we pointed that the gradients backpropagated in the network at the initialisation depend on the initial weights but not on the dataset. The divergences must then occur later. But even though the evolution of the gradients during the learning phase is very distinct from one test to another, this evolution does not seem to be linked to the training set. Nevertheless, we could observe that the gradient of the first layer GL1 has a higher absolute mean value than the others once the network obtains good results on the training set. However, this huge difference can probably be explained by the fact that GL1 is the only layer containing a batch normalisation. Nevertheless, we observe that the beginning of the learning of the CNN on the training set corresponds to an increasing of the mean absolute value of the backpropagated gradients.

5.3 The filters contents

We pointed in Sec. 5.1 that the two first convolutional layers are primordial in the learning as their weights change a lot at the beginning of learning. Therefore, we want to analyse their contents and verify what they are actually learning.

To analyse these two filters, we must first find a way to visualise them. For GL1 it is quite easy. Indeed, this filter has size $7 \times 7 \times 32 \times 1$, since its depth size is equal to 32. We can thus look at each depth the 7×7 corresponding filter.

For GL2, it is more complicated. Indeed, this filter has size $3 \times 3 \times 32 \times 32$. This filter has depth 32, and must deal with images with depth 32, the outputs of GL1. A possibility to visualise them is to look at each $32 \cdot 32 = 1024$ 3×3 filters.

When looking at the filters in the first convolutional layer (GL1), we cannot observe significant patterns for any scale. Even if we observe sometimes filters like in figure 5.11 where we can observe patterns similar to the Haar wavelets ones, a lot of filter are more like 5.12 where is hard to distinguish anything.

Looking at the filters in GL2 does not provide more information. Even though we can again find patterns like the Haar wavelet ones, the huge number of filter and their small size is maybe the reason of these observations.

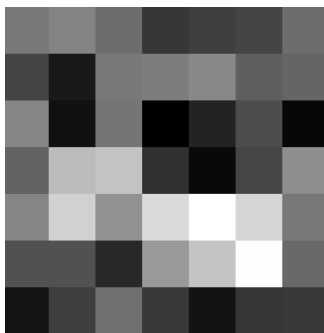


Figure 5.11: GL1 filter with patterns similar to the wavelets.

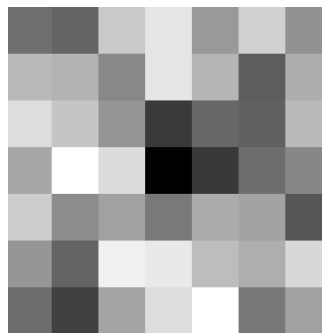


Figure 5.12: GL1 filter without observable pattern.

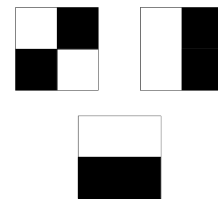


Figure 5.13: The 3 Haar wavelet patterns

Looking directly at the filters in each layer is consequently not really useful to understand the behaviour of the algorithm. That does not mean that these filters do not contain information of course, but due to their number and their small size it is hard to extract this information visually. This problem is common when trying to analyse the behaviour of CNNs.

We look then for other ways to visualise the specificities of the filters. A common strategy is to look to the input image that maximises the activation of each neuron, for dense layers, or each filter, for convolutional layers. This is named the Activation Maximization [12]. We used the implementation provided by Keras that uses a gradient descend approach to estimate the input maximising the activation.

The first thing that we can point is that this method is more appropriate for real images. Indeed, it allows to see easily if a filter is learning eyes or edges, for example. In our case, it is often hard to determine if the patterns that we see in the images are linked to the background that comes from a DCT dictionary, or are just random data. It is slightly easier for the foreground that comes from a Haar wavelet dictionary. Indeed, in a Haar wavelet dictionary, they are three

main patterns that are easier to distinguish. But for the smaller scales, these patterns are very small and can be hard to see in the visualisation of the filter. Nevertheless, we could observe some interesting things in this visualisation.

For the two smallest scales, the images are difficult to interpret. Indeed, as exposed before, the foreground is very small (2×2 or 4×4 pixel) and as the visualisation method is maybe not the more appropriate to be able to see it. Nevertheless, some filters seem visually contain Haar wavelet patterns. It is the case for all layers, but the patterns seem to be more precise in the last layers. This could indicate that the foreground is not only learned on the first layers, but again the images are too hard to interpret to make any conclusion. Figure 5.14 shows the maximum activation input for a filter in layer GL2 and layer L3 (the last convolutional layer).

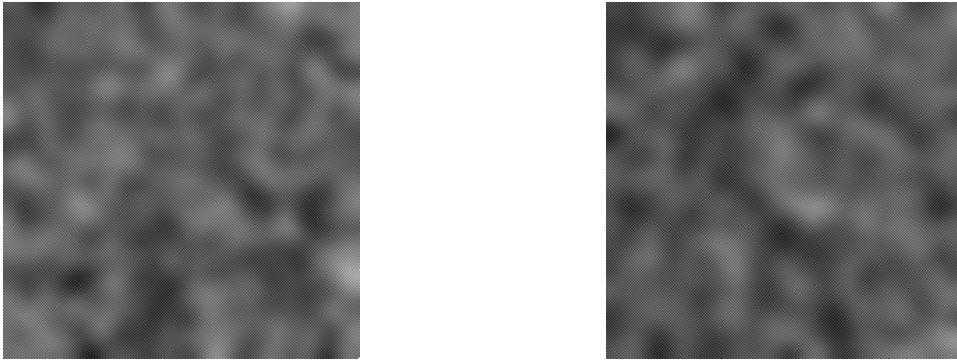


Figure 5.14: Maximum activation input for a filter in layer GL2 (right-image) and in layer L3 (left-image) at scale 1.

Scales 3,4 and 5 do not provide much more information than the first two ones. However, at scale 5, we can observe very interesting maximum activation inputs. First, we can observe some layers in GL2 that have a maximum activation input very similar to the possible foregrounds. Indeed, at this scale the non-zero patterns are big ($1/8$ of the image) but with small intensity. This illustrates that the CNN is learning well the foreground.

Then, when we have a look at the next layers, we can observe in layers L1 and L2 that some filters have a maximum activation input that consists of a lot of small haar wavelet elements. This time, they cannot be directly linked to the foreground as they are much smaller. Nevertheless, as illustrated in Fig. 5.15, the haar wavelets elements are present everywhere in the input figure. This illustrates that the CNN is invariant to the translations of the foreground on the image. Moreover, it suggests that the network is putting a lot of attention to the change of pixel value in the foreground patterns.

Finally, the maximum activation inputs observed in layer L3 are quite surprising. Indeed, some of them contain black holes. This behaviour is intriguing and hard to explain. Maybe it is also a sign that the networks put a lot of attention on the change of pixel value that occurs in the foreground.

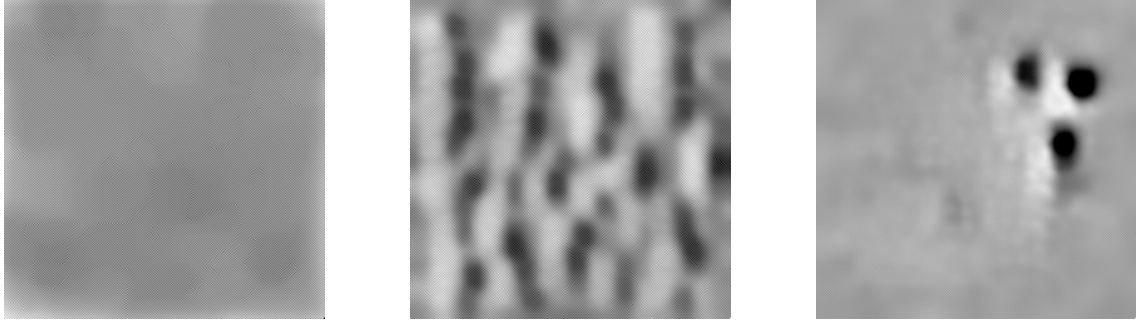


Figure 5.15: Maximum activation input for a filter in layer GL2, L1 and one in layer L3 at scale 5.

At scales 5 and 6, some maximal activation inputs in layer GL2 have also all their pixels close to zero. But it is hard to determine if they contain Haar wavelet patterns. Indeed, at these scales, the patterns have a really low intensity and are consequently hard to distinguish. Consequently, we looked at the Haar wavelet transform of these inputs and could observe that they have only a few non-zero coefficients. That is the case only for these filters with pixels close to zero, the other filters of GL2 contain more non-zero coefficients in the Haar wavelet transform. We consider then that these filters contain Haar wavelets elements and are thus learning the foreground.

In layers L1 and L2, some inputs can be compared to a grid of black points, as illustrated in figure 5.16. This can be a sign that the network is putting a lot of attention to the change of pixel intensity in the foreground or on the contrary a sign that the networks try to capture the regularity of the image. Indeed, the background is regular while the foreground is singular. These two interpretations are quite opposite, but we can at least notice that contrarily at the previous case, we cannot affirm that we encounter Haar wavelet patterns, which tends to reveal that the foreground is less well learned in this case.

Finally, in layer L3, we observe again black holes, but they are often close to white strips. Their association forms often a kind of wavelet pattern. It is interesting to see here that the network is dealing in a complete different way the black and the white pixels.

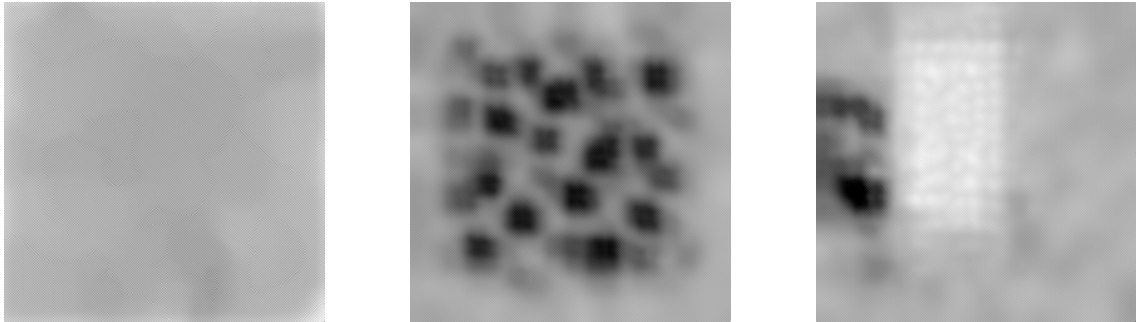


Figure 5.16: Maximum activation input for a filter in layer GL2, L1 and one in layer L3 at scale 5.

The analyse of the maximal activation input for each filter shows that some filters seem to be activated mainly by the foreground. For the small scales, this is observed on all the convolutional layers while for scales 6 and 7, the last convolutional layers do not present explicitly signs of Haar wavelet patterns.

In general, the analyse of the filters did not provide much information. Nevertheless, the observation of the filters combined to the observations made in section 4.1.2 make us think that when the foreground is at scale 1 to 5, the network contains filters that are learning the foreground at each layer while for scales 6 and 7, the two first convolutional layers extract the foreground features from the input image while the three others convolutional layers seems to focus more on the background.

Chapter 6

Analyse of the generalised coherence inside the network

In the beginning of this thesis, we computed the mutual coherence between the background dictionary and the foreground one to try to explain the performances of a CNN. But the main algorithms from image processing whose performances rely on the mutual coherence do not modify their input data. Indeed, the Basis Pursuit, for example, try to recover a sparse vector α such that $\mathbf{x} = \mathbf{D}\alpha$, where $\mathbf{x}$ is the input image and $\mathbf{D} = [\mathbf{D}_1, \mathbf{D}_2]$ is a dictionary. To do so, this algorithm iterates over α , but do not modify $\mathbf{x}$. The same can be observe for the MCA and the compressive sensing.

Actually, the goal of the compressive sensing is to compress the input images, and to be able to recover them as perfectly as possible after. This algorithm does thus modify the input data to do the compression, but in a linear way. The input data are not modified to recover the data.

On the contrary, the CNN modifies in a non-linear way the input data to be able to classify them. Actually, the classification is the result of the application of a non-linear function on the inputs. We can thus write the CNN as a function $\phi(\mathbf{x}) : \mathbb{R}^{n \times n} \rightarrow \{0, 1\}$. Cho and Saul demonstrated that this function can be kernelised at each layer when the network contains only dense layers [6]. That means that there exists functions $k_l(x, y)$ such that $k_l(\mathbf{x}, \mathbf{y}) = \langle \phi_l(\mathbf{x}), \phi_l(\mathbf{y}) \rangle$ where $\mathbf{x}$ and $\mathbf{y}$ are two inputs vectors, and $\phi_l(\mathbf{x})$ is the output of the network at the l^{th} layer.

Traditionally, the Kernels are used to simplify the computation of separation algorithms. Indeed, do a change of variable to consider them in higher space can make the data linearly separable. This is of course a very interesting property in classification and allows to use effective algorithms. The problem is of course to find an appropriate change of variables, but also to apply it. The kernel trick allows to avoid this issue. Indeed, the separation algorithms rely only on scalar products between input data. But the kernel function is able to compute this scalar product without having to compute the change of variable and in a much more efficient way.

Say that the CNN are kernelisable means that they are comparable to a non-linear function that makes the input data linearly separable. As the goal of these algorithms is to classify data, this comparison make sense. A CNN does not always map the input data to an higher dimension space, and it has not been proved that the convolutional layers are kernelisables as well, but we decided to deepen to comparison to see if it could highlight interesting properties of CNNs.

The use of the kernel trick impose that the input data are never used directly, but always through a scalar product. If we want to analyse how the CNN separates data, we should thus use a metric based on inner product. Fortunately, it is the case of the mutual coherence. Indeed, as explained in section 2.4.2, the mutual coherence between two normed dictionaries is the maximal absolute inner product between elements in each dictionary. Mathematically, if we have dictionaries $\Phi = \{\phi_i\}$ and $\Psi = \{\psi_j\}$ the mutual coherence will be $\mu = \max_{i,j} |\langle \phi_i, \psi_j \rangle|$.

6.1 The generalised coherence

We observed that the mutual coherence is only based on inner products and we would like to compute this metric after each layer, to see if its evolution can teach us more about the behaviour of our CNN on datasets constructed using background and foreground dictionaries with different mutual coherence.

Ideally, we would like to compute it in the following way: if the output of the l^{th} layer of the CNN can be written as $\phi_l(\mathbf{x})$ for an input image $\mathbf{x}$, we would compute the mutual coherence at each layer between the DCT basis ($\Gamma = \{\gamma_k\}$) and the Haar wavelet dictionary at a fixed scale ($H_j = \{h_{j,m}\}$) as follows :

$$\mu_l = \max_{k,m} \left| \left\langle \frac{\phi_l(\gamma_k)}{\|\phi_l(\gamma_k)\|}, \frac{\phi_l(h_{j,m})}{\|\phi_l(h_{j,m})\|} \right\rangle \right|.$$

Unfortunately, we saw in Sec. 4.1.2 that our CNN is most of the time unable to classify correctly atoms from the DCT and the Haar wavelet dictionary. The outputs of each network's layers computed with these atoms is thus not representative of the learning of the network.

Consequently, we decided to use the training images from each class instead of the atoms from the dictionaries. This leads us to a definition of the generalised coherence:

Definition 6.1. *The **generalised coherence** between two sets of signals in $\mathbb{R}^N$, $\mathcal{X} = \{\mathcal{X}_i\}_{i=1}^I$ and $\mathcal{Y} = \{\mathcal{Y}_j\}_{j=1}^J$ is defined as*

$$\mu^{\text{gen}}(\mathcal{X}, \mathcal{Y}) = \max_{i,j} \left| \left\langle \frac{\mathcal{X}_i}{\|\mathcal{X}_i\|}, \frac{\mathcal{Y}_j}{\|\mathcal{Y}_j\|} \right\rangle \right|$$

We will also use the concept of mean generalised coherence that is defined as follows:

Definition 6.2. *The **average generalised coherence** between two sets of signals in $\mathbb{R}^N$, $\mathcal{X} = \{\mathcal{X}_i\}_{i=1}^I$ and $\mathcal{Y} = \{\mathcal{Y}_j\}_{j=1}^J$ is defined as*

$$\mu^{\text{av}}(\mathcal{X}, \mathcal{Y}) = \frac{1}{I \cdot J} \sum_{i,j} \left| \left\langle \frac{\mathcal{X}_i}{\|\mathcal{X}_i\|}, \frac{\mathcal{Y}_j}{\|\mathcal{Y}_j\|} \right\rangle \right|$$

We computed thus the generalised coherence between images containing a foreground or not of each training set after each layer of the network. We designate the generalised coherence after layer l as $\mu_l^{\text{gen}} = \mu^{\text{gen}}(\phi_l(\mathcal{X}), \phi_l(\mathcal{Y}))$, and the mean generalised coherence as $\mu_l^{\text{av}} = \mu^{\text{av}}(\phi_l(\mathcal{X}), \phi_l(\mathcal{Y}))$

Note that we consider that the output of a convolutional layer is the output of the pooling layer that follows it (if it exists). Indeed, the pooling is not learned by the CNN and has always the same behaviour. It can be seen as part of the convolutional layer, as we did in figure 2.7. Moreover, this allows us to deal with smaller output vectors and to compute more easily the generalised coherence.

6.2 Experimental results

We want to compute the generalised coherence between the two classes of our training datasets after each layer of our network. Unfortunately, it would be very long to compute it exactly. Indeed, compute all the $\phi_l(\cdot)$ is not an issue as it means simply pass all the images into the CNN, but compute all the possible inner products is much longer. Since we have 10 000 images in each class, that would mean do 10^8 computation of the inner product. To have an idea of the value of the generalised coherence in a reasonable time, we decided thus to reduce the number of inner product tested. The results presented here after are thus approximations of the real value of each μ_l^{gen} .

We decided to take randomly 1000 images from each class and compute the generalised coherence between these subsets. During our tests we could observe that the value of the generalised coherence is similar when we use subsets of 500 images or subsets of 1000 images. This convinces us that the approximation that we computed is quite accurate.

We computed first the generalised coherence after each layer of a random CNN, to see its value at the initialisation.

We can observe in figure 6.1 that the generalised coherence increases with the layers. This is actually quite logical. Indeed, we can see in tabular 6.1 that the output size decreases with each layer. But the probability of two groups of vectors to be linearly (at thus easily) separable increases with the vector's dimension. Note also that the coherence of the output layer is always 1. As this layer's output is a 2×1 vector, it is not surprising that when the network is random it classifies two images from different class in the same way, knowing that we have 10 000 images in each class.

Moreover, we can observe that all the datasets have the same generalised coherence after each layer of a random network and that this value is very high. The same observation can be done on the average generalised coherence. This illustrates that the network does not make any difference between the two classes before its learning. Moreover, we already observed in Sec. 5.2, that our CNN has the same behaviour on all our datasets at the initialisation.

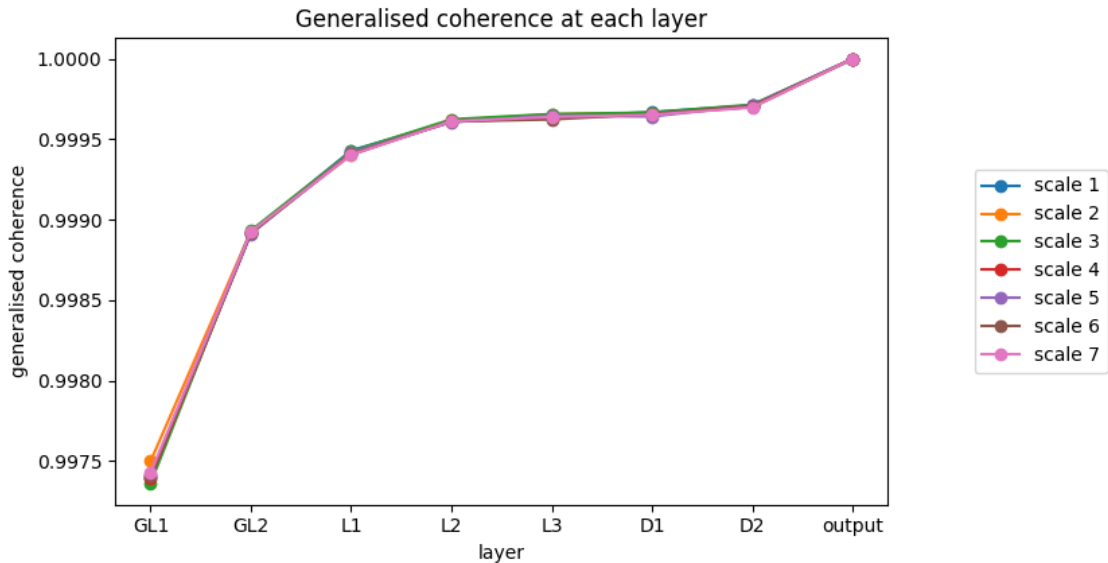


Figure 6.1: Evolution of the generalised coherence through each layer in a network with random weights.

Input	GL1	GL2	L1	L2	L3	D1	D2	output
65 536	127 008	30752	12 544	4608	2048	1024	1024	2

Table 6.1: Size of the output at each layer

We looked then at the value of the generalised coherence at the end of learning. But as some images are miss-classified by the CNN, even after the training phase, the generalised coherence is corrupted and we can not take much conclusions of its evolution except that its value is lower than at the initialisation. The evolution through the layers of the average generalised coherence is more interesting. We can observe in Fig. 6.2 that unlike the random case, the average generalised coherence tends to decrease after each layer at the end of the training phase.

We can observe that when the foregrounds have been constructed using Haar wavelets atoms at scale between 1 and 5, the average generalised coherence is very similar at the end of the training phase. This is not very surprising as the CNN has similar performances on these datasets at the end of the training. Moreover, in these cases, the coherence at the end of the third layer, L1, is much lower than the one at the end of layer GL2. We observed in Sec. 5.1 that the two first convolutional layers change more than the others when the network begins the learning. Moreover, the convolutional layers are supposed to extract the discriminant features from the images. This seems verified by our results. Indeed, if the two first convolutional layers extract the discriminant features from the images, some features extracted will be similar when the image contains or not a foreground, as the backgrounds are similar for all the images. The distinction between the two classes is done after, when the network analyses the combination of the discriminant features.

For the smallest scales, the features extracted by the three last convolutional layers allows already to separate the input images. We saw in Sec. 4.1.2 that the network was mainly learning features from the foreground in this case with some features of the background. Our results seem to indicate that the two first convolutional layers extract features from both the background and the foreground but that the three last convolutional layers put more attention on the features from the foreground. This would explain the good separability of the output after these last layers.

On the other hand, when the foreground has been built using Haar wavelet atoms quite incoherent with the DCT basis, at scale 6 or 7, the decreasing of the average generalising coherence is smoother. We observed in Sec. 4.1.1 that at these two scales, the network does not generalise well. However, we computed the coherence on a network with high accuracy on the test set for both scales. The higher value of the average coherence is maybe a sign of the instability of the network, but it also indicates that the network seems to need all the convolutional layers to produce incoherent outputs. This emphasises the observation that the network is learning both features from the background and the foreground. This observation has been made in Sec. 4.1.2

We observed that the CNN decreases significantly the average generalised coherence between images with or without foreground during its training phase. We wonder thus what is the evolution of the average coherence during the learning. We computed first this evolution for the first test on scale 7. Indeed, we know that during this test the generalisation of the CNN is not good on the test set. Consequently, we want to see if the average generalised coherence highlights this behaviour.

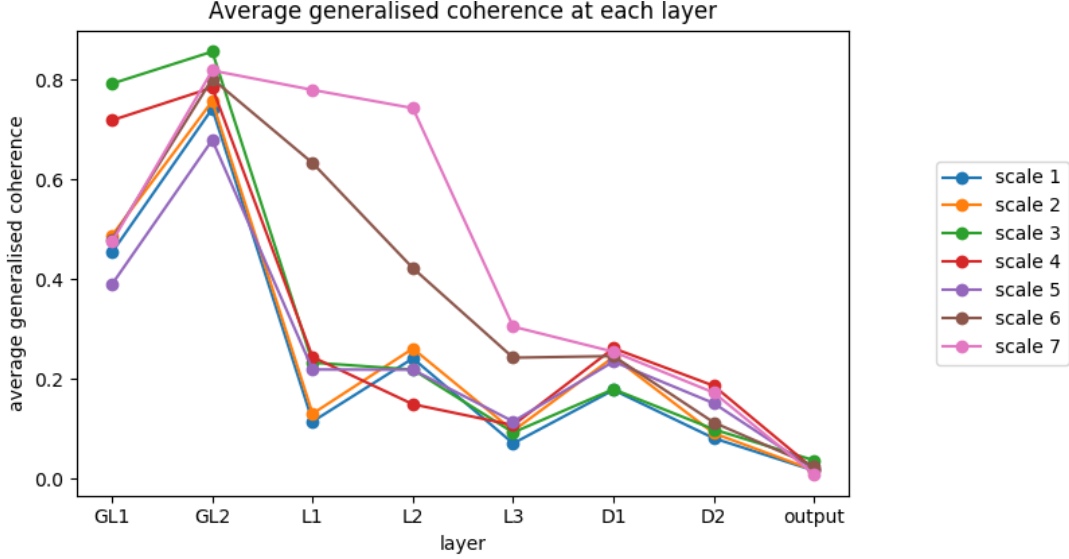


Figure 6.2: Evolution of the average generalised coherence through each layer at the end of the training.

We can see in Fig. 6.3 the evolution of the average generalised coherence during the training. Moreover, we also printed the evolution of the accuracy on the test set. Actually, we modified it in the following way: $\text{acc} = 2 \cdot (1 - \text{acc})$. This means that if the accuracy is around 100% the curve that we printed is around 0 and if the accuracy is around 50%, the curve will be around 1.

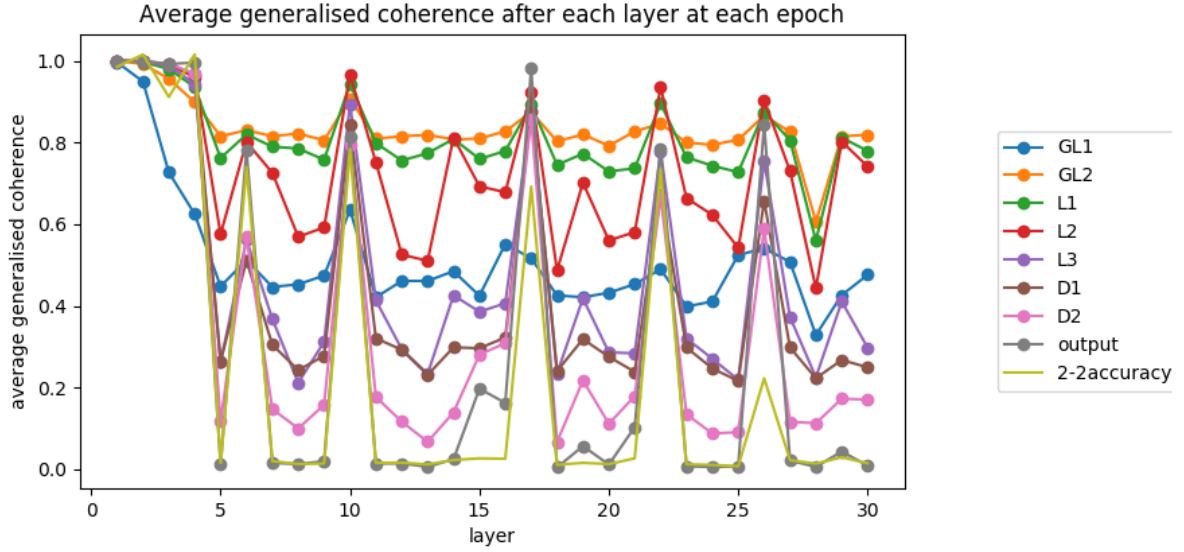


Figure 6.3: Evolution of the average generalised coherence during the training at scale 7.

We can observe that the average generalised coherence after the dense layers (D1, D2 and output) is very close to the curve representing the accuracy on the test set except around the 26th epoch. But the coherence on the dense layers is always higher than the curve representing the accuracy. Consequently, we can observe that the average generalised coherence on the training set after the dense layers allows to determine the performances of the network on the test set. Indeed, an average coherence around 0 indicates that the network generalises well while a higher coherence is the sign of a bad generalisation.

This property is very interesting. Indeed, if it is verified in all cases, it would mean that we do not need a test set to compute the generalisation level of the trained network but that this

can be done through the training set. The average generalised coherence after the dense layers is thus in this case a metric that evaluates the network.

In Fig. 6.4 we can observe the evolution of the average generalised coherence at scale 1. We can observe as previously that the evolution of the accuracy on the test set is proportional to the evolution of the coherence at the output layer. We cannot observe much specificities when the network is unable to generalise except that the coherence is above 0.5 for all layers.

We observed again in this second test that the average generalised coherence can be used as a metric of the CNN performances. Note that we pointed in Sec 4.1.1 that the certainty level of the CNN on the test set can also be used to evaluate the network in a quasi-certain way. Note that these two metrics are linked. Indeed, a high certainty level means that all the images produce an output vector close to $(1, 0)$ or $(0, 1)$ depending of their class. But the generalised coherence on such vectors will be close to $1 \cdot 0 + 0 \cdot 1 = 0$. On the other hand, when the certainty level is low, the outputs vectors will be close to $(0.5, 0.5)$ and the coherence close to $0.5^2 + 0.5^2 = 0.5$. Nevertheless, these two values are not proportional and the average generalised coherence is more accurate to evaluate the performances of the CNN.

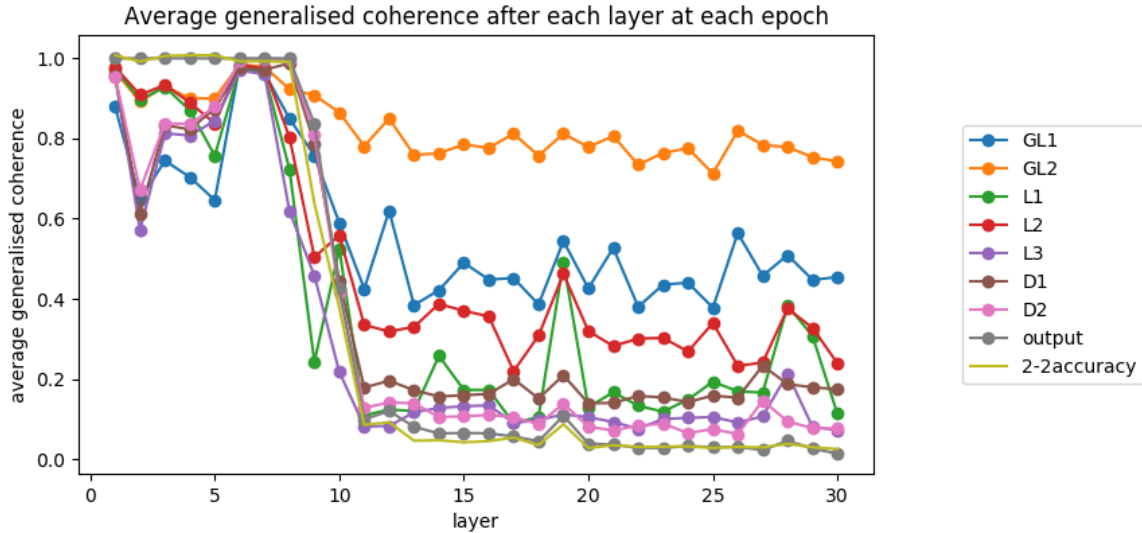


Figure 6.4: Evolution of the average generalised coherence through each layer during training at scale 1.

6.3 Comments on the choice to use mutual coherence to characterise the datasets

In previous section we pointed that it is more appropriate to use the average generalised coherence to analyse the behaviour of the network, as the generalised coherence is corrupted by the miss-classified images. Like the generalised coherence, the mutual coherence relies on a maximum. We wonder thus what is the average mutual coherence between the dictionaries used to build the background and the foreground of our images.

Formally, we compute the average coherence between the DCT dictionary $\mathbf{\Gamma} = \{\gamma_i\}_{i=1}^I$ and the Haar wavelet dictionary at a fixed scale j , $\mathbf{H}_j = \{h_{j,m}\}_{m=1}^M$ as $\mu^{\text{average}} = \frac{1}{M \cdot I} \sum_{i,m} |\langle \gamma_i, h_{j,m} \rangle|$. We can observe in Tab. 6.2 that even if the mutual coherence increases with the scale of the Haar wavelet, the average coherence decreases instead. This means that for the highest scales, the Haar wavelets atoms are most of the time locally incoherent with respect to the DCT dictionary, but some atoms are very coherent with this dictionary. This high variability of the local coherence

inside the dataset can explain the behaviour of the algorithm. Indeed, as most of the time the local coherence is low, the produced images will often be easy to distinguish and the network will learn fast. But the high variability of the local coherence causes difficulties to the network for the generalisation.

Scale	1	2	3	4	5	6	7
Mutual coherence	0.016	0.024	0.046	0.090	0.177	0.326	0.512
Average coherence	0.0038	0.0032	0.0024	0.0018	0.0012	0.0008	0.0005

Table 6.2: Comparison of the mutual and the average coherence between the DCT dictionary and the Haar wavelet one at given scale.

To verify if the images produced by the wavelets at high scale have a low average coherence, we computed the average generalised coherence on these images. We could observe that this value was similar for all datasets. The average coherence of any training dataset is around 98.4%. However, if we look at the average generalised coherence between the Haar wavelet dictionary and the DCT one inside the network in Fig. 6.5, we can observe that it is globally higher for the higher scales, except at the output.

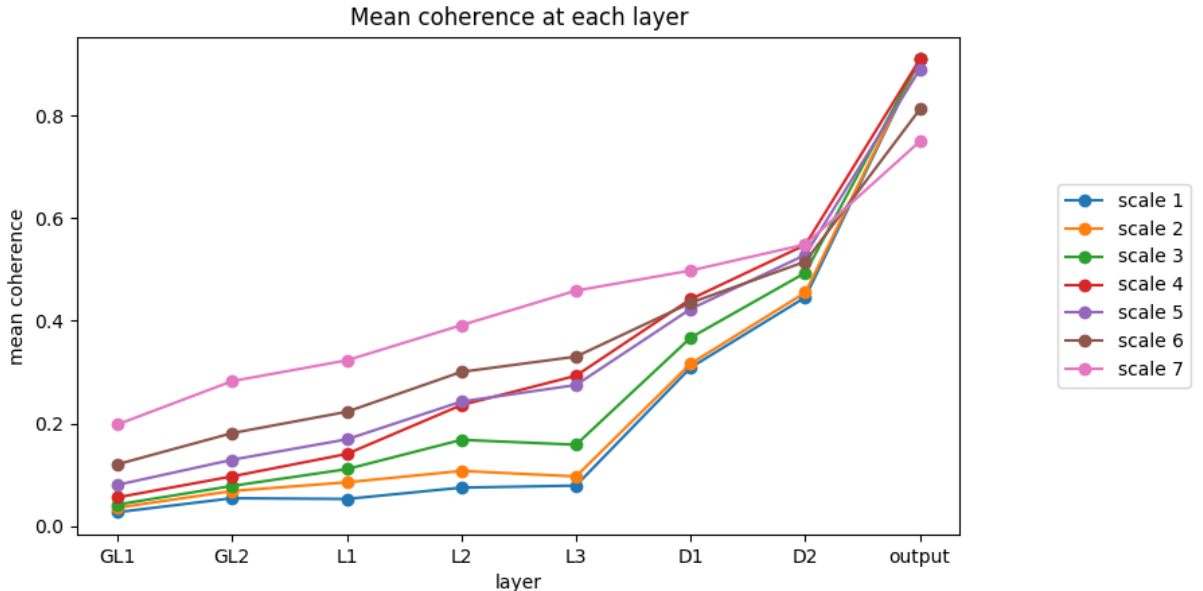


Figure 6.5: Evolution of the mean coherence between the atoms of the Haar wavelet dictionary and the DCT one after each layer of a network with random weights.

Consequently, even though the average coherence is very distinct from the mutual coherence, the use of the mutual coherence between the dictionaries to describe the dataset seems to be quite accurate for our problem. Indeed, the average generalised coherence through the network changes in the same way than the mutual coherence.

Nevertheless, in practice it can be hard to determine dictionaries that generate the background and the foreground of the images. However, we observed that the generalised coherence between the images of the training set allows to characterise the learning of the CNN for our problem. Of course, it is impossible to generalise this conclusion based only on the results that we presented here. However, we hope that our observations will encourage further researches about the utility of the average generalised coherence to characterise the behaviour of CNNs.

Chapter 7

Conclusion

In this work, we analysed the behaviour of a Convolutional Neural Network containing five convolutional layers and two dense layers on different input datasets. The goal of the network was always to determine the presence or the absence of a foreground, or anomaly, in the provided images. All the backgrounds of the images were sparse in the DCT-II dictionary, while the foregrounds were sparse in a Haar wavelet dictionary, such that the coherence between these two dictionaries could be adjusted in each dataset by tuning the scale of the wavelet dictionary.

Even though we did not observe a direct link between the performances of the CNN and the coherence between the foreground and background dictionaries, we have seen that the data created with the smallest coherence slowed down the performances of the algorithm, but provided more reliable results and were more able to generalise for unknown data. Moreover, the network had not only different performances, but was also learning different features. An analyse of the network's performances on images containing only a foreground allows to highlight the features learned by the CNN (Sec. 4.1.2). When the coherence was low (at small wavelet scale), and the foreground and background were thus a priori easier to discriminate, the network was mainly learning the foreground. Whereas when the coherence was high (at high wavelet scale), it has to learn both the foreground and the background to be able to discriminate them. But if the foreground at a fixed scale could be seen as translations of three different patterns, it was not the case of the background. The difficulties of the network trained on high-scale images to classify unknown data could be related to the fact that the training data did not represent all the diversity of possible backgrounds.

The fact that the network is mainly learning the foreground of the images at low scales (and thus low coherence) means that the training images containing both foreground and background are more important for the learning of the network than images containing only background. This has been confirmed by the observation that the learning was much faster when 80% of the training images contained a foreground. On the contrary, when the network was learning both the foreground and the background, having between 10% and 95% of images with foreground produces more or less the same results.

On the other hand, we have observed that the fact to restrict the range of possible foregrounds was more useful for the smallest scales and the highest ones, but not the intermediate ones at which the network was already performing well. Indeed, at lowest scales, the CNN had a faster learning, while at highest scales, the restriction of the foreground made the network able to generalise well on unknown data. This last observation was quite surprising as we supposed that the generalisation problems of the network were linked to its difficulty to learn the background. Consequently, either this hypothesis was false, or the network changed its learning behaviour in this case. This last possibility does not seem improbable. Indeed, limiting the spatial variability of the foreground made perhaps the network focus more on the foreground and have a learning

behaviour more similar than the one at lower scales. But since our limitation of the foreground was equivalent to restricting it at some locations in the images, the better generalisation is perhaps due to a restriction of the background learned. Indeed, the network can focus on the positions where a foreground may appear and learn the background at these positions only.

When looking more precisely at the learning behaviour of the CNN we have observed that the two first convolutional layers were essentials for the learning. Indeed, these two layers were those that changed most during the learning, and when the images were built using incoherent dictionaries we saw that the features extracted by the two first layers allowed the other layers to discriminate images with or without foreground. Moreover, we observed that the network did not learn the basic patterns that constitute the foreground even though it was able to classify perfectly an image containing only a foreground. Nevertheless, once the scale was large enough, we have observed in some "maximum activation inputs" patterns similar to the Haar one at the end of the convolutional layers.

Even if it allowed to construct datasets on which the CNN had completely different behaviours, the mutual coherence between the foreground and the background dictionaries was perhaps not enough to explain our observations. To complete our investigation, we looked at the evolution of this coherence inside the network. For this purpose, we defined the concept of generalised coherence to compute the coherence between set of non-orthonormal images. We computed the generalised coherence of the training dataset after each layer in the network. We have observed that at initialisation, the network had the same behaviour for all our datasets. That was also observed in the gradient back-propagated at the beginning of learning: the network had the same behaviour at all scales in the beginning of learning, the differences occurred later.

Moreover, we saw that the concept of generalised coherence could be used to evaluate the performances of the trained CNN. Indeed, this coherence was proportional to the accuracy obtained by the network on unknown data.

To sum up, we observed in this master thesis the performances of a fixed Convolutional Neural Network trained to distinguish images containing a foreground from images having only a background. We have seen that this network had different behaviours on images with coherent or incoherent background/foreground. We observed the features learned by the network and could explain some differences. Nevertheless, as our tests were restricted to only two dictionaries, it is impossible to know if our observations can be extended to a general case. Indeed, the observed differences can be due to other reasons such as the choice of the CNN, or to the size/intensity/spatial support of the foreground, or the frequency of occurrence of discriminating features in the training dataset or maybe to something else harder to distinguish when looking at the dataset, like the generalised coherence inside the network.

Even though we could explain some aspects of the behaviour of our CNN, some questions stay open. Why is the network stagnating at low scales and what causes the sudden learning? Is the mutual coherence between background and foreground a good descriptor of an input dataset? How can we use the generalised coherence to analyse CNNs?

However, even if we cannot make general conclusions, this work offers an analyse of the behaviour of a CNN and highlights that the composition of the training dataset has a huge influence on the performances of CNNs. This influence can sometimes be counter-intuitive and needs a closer look to the learning made by the network. We hope that the results that we observed will motivate further researches on the influence of the input data on CNN, the role of the mutual coherence between foreground and background in foreground detection, and the interest of the generalised coherence as a metric to evaluate CNNs.

Acknowledgements

I would like to thank ...

My two supervisors Laurent Jacques and Christophe de Vleeschouwer for their availability, for the time they dedicated to me and for their valuable tips. Our regular meetings helped me a lot to develop this thesis. They also provided me access to appropriate resources to do my simulations, in particular to a server of the UCLouvain named Cassiopee.

François Hubin and Brigitte Dupont who were always ready to solve the problems with Cassiopee, and to answer my questions.

Cassiopee users who provided me their tricks for a better use of the server.

Simon Carbonnelle who has been a great help, especially at the beginning of my work. Indeed, he taught me how to manipulate Keras and helped me to connect to the server. Moreover, he was always available to answer my questions.

Amirafshar Moshtaghpour for his advices on the use of the coherence concept. He and Chunlei Xu are also very nice office-mate.

Michel Hautecoeur and Christine Farnetti who carefully read the different parts of my thesis for the spelling and sentence construction.

My family and some friends who encouraged me a lot during the construction of this thesis.

Bibliography

- [1] Devansh Arpit, Stanisław Jastrzębski, Nicolas Ballas, David Krueger, Emmanuel Bengio, Maxinder S Kanwal, Tegan Maharaj, Asja Fischer, Aaron Courville, Yoshua Bengio, et al. A closer look at memorization in deep networks. *arXiv preprint arXiv:1706.05394*, 2017.
- [2] Jérôme Bobin, Jean-Luc Starck, Jalal M Fadili, Yassir Moudden, and David L Donoho. Morphological component analysis: An adaptive thresholding strategy. *IEEE Transactions on Image Processing*, 16(11):2675–2681, 2007.
- [3] Joan Bruna and Stéphane Mallat. Invariant scattering convolution networks. *IEEE transactions on pattern analysis and machine intelligence*, 35(8):1872–1886, 2013.
- [4] Nitesh V Chawla. Data mining for imbalanced datasets: An overview. In *Data mining and knowledge discovery handbook*, pages 875–886. Springer, 2009.
- [5] Scott Shaobing Chen, David L Donoho, and Michael A Saunders. Atomic decomposition by basis pursuit. *SIAM review*, 43(1):129–159, 2001.
- [6] Youngmin Cho and Lawrence K Saul. Kernel methods for deep learning. In *Advances in neural information processing systems*, pages 342–350, 2009.
- [7] François Chollet et al. Keras. <https://keras.io>, 2015.
- [8] David L Donoho and Michael Elad. Optimally sparse representation in general (nonorthogonal) dictionaries via ℓ_1 minimization. *Proceedings of the National Academy of Sciences*, 100(5):2197–2202, 2003.
- [9] David L Donoho and Xiaoming Huo. Uncertainty principles and ideal atomic decomposition. 1999.
- [10] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.
- [11] Michael Elad, Mario AT Figueiredo, and Yi Ma. On the role of sparse and redundant representations in image processing. *Proceedings of the IEEE*, 98(6):972–982, 2010.
- [12] Dumitru Erhan, Yoshua Bengio, Aaron Courville, and Pascal Vincent. Visualizing higher-layer features of a deep network. *University of Montreal*, 1341(3):1, 2009.
- [13] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.
- [14] Alfred Haar. Zur theorie der orthogonalen funktionensysteme. *Mathematische Annalen*, 69(3):331–371, 1910.

- [15] Robert Hecht-Nielsen. Theory of the backpropagation neural network. In *Neural networks for perception*, pages 65–93. Elsevier, 1992.
- [16] Geoffrey Hinton, Li Deng, Dong Yu, George E Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Tara N Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.
- [17] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.
- [18] Jason Jo and Yoshua Bengio. Measuring the tendency of CNNs to learn surface statistical regularities. *arXiv preprint arXiv:1711.11561*, 2017.
- [19] Felix Krahmer and Rachel Ward. Stable and robust sampling strategies for compressive imaging. *IEEE Transactions on Image Processing*, 23(2):612–622, 2014.
- [20] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.
- [21] Miroslav Kubat, Stan Matwin, et al. Addressing the curse of imbalanced training sets: one-sided selection. In *ICML*, volume 97, pages 179–186. Nashville, USA, 1997.
- [22] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.
- [23] Steffen Limmer and Slawomir Stanczak. Optimal deep neural networks for sparse recovery via laplace techniques. *arXiv preprint arXiv:1709.01112*, 2017.
- [24] Stéphane Mallat. *A wavelet tour of signal processing*. Academic press, 1999.
- [25] Stéphane Mallat. Understanding deep convolutional networks. *Phil. Trans. R. Soc. A*, 374(2065):20150203, 2016.
- [26] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.
- [27] Omkar M Parkhi, Andrea Vedaldi, Andrew Zisserman, et al. Deep face recognition. In *BMVC*, volume 1, page 6, 2015.
- [28] Xueheng Qiu, Le Zhang, Ye Ren, Ponnuthurai N Suganthan, and Gehan Amaratunga. Ensemble deep learning for regression and time series forecasting. In *Computational Intelligence in Ensemble Learning (CIEL), 2014 IEEE Symposium on*, pages 1–6. IEEE, 2014.
- [29] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.
- [30] Hariharan Ravishankar, Prasad Sudhakar, Rahul Venkataramani, Sheshadri Thiruvankadam, Pavan Annangi, Narayanan Babu, and Vivek Vaidya. Understanding the mechanisms of deep transfer learning for medical images. *arXiv preprint arXiv:1704.06040*, 2017.
- [31] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

- [32] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.
- [33] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.
- [34] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016.
- [35] Xiaolong Wang and Abhinav Gupta. Unsupervised learning of visual representations using videos. *arXiv preprint arXiv:1505.00687*, 2015.
- [36] Daniel Weimer, Bernd Scholz-Reiter, and Moshe Shpitalni. Design of deep convolutional neural network architectures for automated feature extraction in industrial inspection. *CIRP Annals-Manufacturing Technology*, 65(1):417–420, 2016.
- [37] Bo Xin, Yizhou Wang, Wen Gao, David Wipf, and Baoyuan Wang. Maximal sparsity with deep networks? In *Advances in Neural Information Processing Systems*, pages 4340–4348, 2016.
- [38] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.
- [39] Matthew D Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.

Appendices

Appendix A

Theoretical bound on the coherence

In this section we will derive a theoretical bound of the local coherence of a DCT atom with respect to the Haar wavelet dictionary, using the same reasoning as in [19].

Theorem A.1. *Let $N = 2^p$ with $p \in \mathbb{N}$. In $\mathbb{R}^{N \times N}$, the local coherence of the bivariate Haar wavelet dictionary $\mathbf{H} = \{h_{j,l,v}\}$ with respect to the orthonormal two-dimensionnal DCT dictionary $\mathbf{\Gamma} = \{\gamma_{k_1,k_2}\}$ as defined respectively in definition 2.2 and 2.1, is bounded by*

$$\mu_{j,l,v}^{\text{loc}}(\mathbf{H}, \mathbf{\Gamma}) \leq \begin{cases} \pi^2 2^{j-p-2} & \text{When } j > 1 \\ \pi^2 2^{-p} & \text{When } j = 1. \end{cases}$$

Proof. We know that $h_{j,(l_1,l_2),(v_1,v_2)}[n_1, n_2] = h_{j,l_1,v_1}[n_1] \cdot h_{j,l_2,v_2}[n_2]$ and $\gamma_{k_1,k_2}[n_1, n_2] = \gamma_{k_1}[n_1] \cdot \gamma_{k_2}[n_2]$ (see definition 2.2 and 2.1). Using the notation $A(*, i)$ to design the i^{th} column of matrix A we can say that

$$\begin{aligned} \langle h_{j,(l_1,l_2),(v_1,v_2)}, \gamma_{k_1,k_2} \rangle &= \sum_i \langle h_{j,(l_1,l_2),(v_1,v_2)}(*, i), \gamma_{k_1,k_2}(*, i) \rangle \\ &= \sum_i \langle h_{j,l_1,v_1}, \gamma_{k_1} \rangle \cdot h_{j,l_2,v_2}[i] \cdot \gamma_{k_2}[i] \\ &= \langle h_{j,l_1,v_1}, \gamma_{k_1} \rangle \cdot \langle h_{j,l_2,v_2}, \gamma_{k_2} \rangle. \end{aligned}$$

For $0 \neq k_1, k_2$, we can use the bound found in lemma A.4

$$|\langle h_{j,(l_1,l_2),(v_1,v_2)}, \gamma_{k_1,k_2} \rangle| \leq \begin{cases} \pi^2 2^{j-p-2} & \text{When } j > 1 \\ \pi^2 2^{-p} & \text{When } j = 1. \end{cases}$$

If $k_1 = 0$ or $k_2 = 0$, in one dimension we have $|\langle h_{j,l,v}, \gamma_0 \rangle| \leq \sqrt{2^{-p}} 2^j 2^{-\frac{j}{2}} = 2^{\frac{j-p}{2}}$. Consequently, the bound if we suppose $k_1 = 0$ is

$$|\langle h_{j,(l_1,l_2),(v_1,v_2)}, \gamma_{0,k_2} \rangle| \leq \begin{cases} \pi 2^{j-p-1} < \pi^2 2^{j-p-2} & \text{When } j > 1 \\ \pi 2^{\frac{1-2p}{2}} < \pi^2 2^{-p} & \text{When } j = 1. \end{cases}$$

If $k_1 = k_2 = 0$ we have $|\langle h_{j,(l_1,l_2),(v_1,v_2)}, \gamma_{0,0} \rangle| \leq 2^{j-p}$.

To conclude, we must take the most restrictive bound as k_1 and k_2 cannot be limited. That leads to the bound :

$$\mu_{j,l,v}^{\text{loc}}(\mathbf{H}, \mathbf{\Gamma}) \leq \begin{cases} \pi^2 2^{j-p-2} & \text{When } j > 1 \\ \pi^2 2^{-p} & \text{When } j = 1. \end{cases}$$

□

Lemma A.1. $|1 - e^{i\pi 2^{-j}k}| \leq \min(2, \pi 2^{-j}|k|)$.

Proof.

$$\begin{aligned}
|1 - e^{i\pi 2^{-j}k}| &= \sqrt{1 + \cos^2(\pi 2^{-j}k) - 2\cos(\pi 2^{-j}k) + \sin^2(\pi 2^{-j}k)} \\
&= \sqrt{2(1 - \cos(\pi 2^{-j}k))} \\
&= \sqrt{2(\cos(0) - \cos(\pi 2^{-j}k))} \\
&= \sqrt{2(-2\sin(\pi 2^{-j-1}k)\sin(-\pi 2^{-j-1}k))} \\
&= 2|\sin(\pi 2^{-j-1}k)|.
\end{aligned}$$

To conclude, we recall that $|\sin(\alpha)| \leq |\alpha|$ and $|\sin(\alpha)| \leq 1 \forall \alpha$. This allows to build the bound

$$|1 - e^{i\pi 2^{-j}k}| \leq \min(2, \pi 2^{-j}|k|).$$

□

Lemma A.2. For $|k| \leq 2^{p-1}$, $|1 - e^{i\pi 2^{-p}k}| \geq 2^{-p+\frac{3}{2}}|k|$.

Proof. Following the same steps than in lemma A.1, we know that

$$|1 - e^{i\pi 2^{-p}k}| = 2|\sin(\pi 2^{-p-1}k)|.$$

As $|k| \leq 2^{p-1}$, $\pi 2^{-p-1}|k| \leq \pi 2^{-2}$. But the function $\sin(\alpha)$ is convex for $\alpha \in [0, \pi]$. That means that for $0 \leq \alpha \leq \beta \leq \pi$, $\sin(\alpha)$ is above the line passing through $\sin(0)$ and $\sin(\beta)$, or mathematically : $\sin(\alpha) \geq \alpha \sin(\beta)\beta^{-1}$.

In our case, we have $\beta = \pi 2^{-2}$ and we can thus bound $\sin(\pi 2^{-p-1}|k|) \geq 2^{-p}|k|\sqrt{2}$. As $\sin(|\alpha|) = |\sin(\alpha)|$ we can conclude that

$$|1 - e^{i\pi 2^{-p}k}| \geq 2^{-p+\frac{3}{2}}|k|.$$

□

Lemma A.3. For $|k| < 2^p$, $|1 - e^{i\pi 2^{-p}k}| \geq 2^{-p+1}|k|$.

Proof. Imposing $|k| \leq 2^p$ means that $\pi 2^{-p-1}|k| \leq \pi 2^{-1}$. We can thus follow the exactly same reasoning than in A.2 and conclude that $\sin(\pi 2^{-p-1}|k|) \geq 2^{-p}|k|$, and thus

$$|1 - e^{i\pi 2^{-p}k}| \geq 2^{-p+1}|k|.$$

□

Lemma A.4. Let $N = 2^p$ with $p \in \mathbb{N}$. In $\mathbb{R}^N$, the one-dimensional DCT basis vectors γ_k , $k = 1 \dots N-1$ ($k \neq 0$) and the one-dimensional Haar wavelet atom $h_{j,l,v}$, as defined in definition 2.1 and 2.2 respectively, satisfies the following relation

$$|\langle h_{j,l,v}, \gamma_k \rangle| \leq \begin{cases} \pi 2^{\frac{j-p}{2}-1} & \text{When } j > 1, \\ \pi 2^{\frac{-p}{2}} & \text{When } j = 1. \end{cases}$$

Proof.

$$\begin{aligned}
\langle h_{j,l,v}, \gamma_k \rangle &= \sum_{z=l}^{l+2^{j-1}-1} 2^{\frac{-j}{2}} \sqrt{\frac{2}{2^p}} \cos\left(\frac{k\pi(2z+1)}{2 \cdot 2^p}\right) + \sum_{z=l+2^{j-1}}^{l+2^j-1} (-1)^v 2^{\frac{-j}{2}} \sqrt{\frac{2}{2^p}} \cos\left(\frac{k\pi(2z+1)}{2 \cdot 2^p}\right) \\
&= 2^{\frac{1-p-j}{2}} \left[\sum_{z=l}^{l+2^{j-1}-1} \frac{e^{ik\pi(2z+1)2^{-p-1}} + e^{-ik\pi(2z+1)2^{-p-1}}}{2} \right. \\
&\quad \left. + \sum_{z=l+2^{j-1}}^{l+2^j-1} (-1)^v \frac{e^{ik\pi(2z+1)2^{-p-1}} + e^{-ik\pi(2z+1)2^{-p-1}}}{2} \right] \\
&= 2^{\frac{1-p-j}{2}-1} \left[e^{ik\pi(2^{-p-1}+2^{-p}l)} \left(\sum_{z=0}^{2^{j-1}-1} e^{ik\pi z 2^{-p}} \right) + e^{-ik\pi(2^{-p-1}+2^{-p}l)} \left(\sum_{z=0}^{2^{j-1}-1} e^{-ik\pi z 2^{-p}} \right) \right. \\
&\quad \left. + (-1)^v \left(e^{ik\pi(2^{-p-1}+2^{-p}l+2^{j-p-1})} \left(\sum_{z=0}^{2^{j-1}-1} e^{ik\pi z 2^{-p}} \right) + e^{-ik\pi(2^{-p-1}+2^{-p}l+2^{j-p-1})} \left(\sum_{z=0}^{2^{j-1}-1} e^{-ik\pi z 2^{-p}} \right) \right) \right] \\
&= 2^{\frac{-1-p-j}{2}} \left[e^{ik\pi(2^{-p-1}+2^{-p}l)} (1 + (-1)^v e^{ik\pi 2^{j-p-1}}) \left(\sum_{z=0}^{2^{j-1}-1} e^{ik\pi z 2^{-p}} \right) \right. \\
&\quad \left. + e^{-ik\pi(2^{-p-1}+2^{-p}l)} (1 + (-1)^v e^{-ik\pi 2^{j-p-1}}) \left(\sum_{z=0}^{2^{j-1}-1} e^{-ik\pi z 2^{-p}} \right) \right] \\
&= 2^{\frac{-1-p-j}{2}} \left[e^{ik\pi(2^{-p-1}+2^{-p}l)} \left(1 + (-1)^v e^{ik\pi 2^{j-p-1}} \right) \left(\frac{1 - e^{ik\pi 2^{j-p-1}}}{1 - e^{ik\pi 2^{-p}}} \right) \right. \\
&\quad \left. + e^{-ik\pi(2^{-p-1}+2^{-p}l)} \left(1 + (-1)^v e^{-ik\pi 2^{j-p-1}} \right) \left(\frac{1 - e^{-ik\pi 2^{j-p-1}}}{1 - e^{-ik\pi 2^{-p}}} \right) \right].
\end{aligned}$$

Lets now consider 2 cases :

For $0 < |k| \leq 2^{p-1}$: Using lemma A.1 and A.2, and the fact that $|e^{i|k|\pi(2^{-p-1}+2^{-p}l)}| \cdot |1 + (-1)^v e^{i|k|\pi 2^{j-p-1}}| \leq 2$ we can say that

$$\begin{aligned}
|\langle h_{j,l,v}, \gamma_k \rangle| &\leq 2^{\frac{-1-p-j}{2}} \left[2 \frac{\min(2, \pi 2^{j-p-1} |k|)}{2^{-p+\frac{3}{2}} |k|} + 2 \frac{\min(2, \pi 2^{j-p-1} |k|)}{2^{-p+\frac{3}{2}} |k|} \right] \\
&\leq 2^{\frac{p-j}{2}} \min(2|k|^{-1}, \pi 2^{j-p-1}) \\
&\leq \min\left(2^{\frac{p-j}{2}+1} |k|^{-1}, \pi 2^{\frac{j-p}{2}-1}\right).
\end{aligned}$$

When j is fixed, the second term in the minimum is fixed too, and the worst bound is $\min\left(\max_k \left(2^{\frac{p-j}{2}+1} |k|^{-1}\right), \pi 2^{\frac{j-p}{2}-1}\right) = \min\left(2^{\frac{p-j}{2}+1}, \pi 2^{\frac{j-p}{2}-1}\right)$.

As $p-j \geq 0$, $\frac{p-j+2}{2} \geq \log_2(\pi) \frac{j-p-2}{2} \Leftrightarrow p-j \geq -2 + \log_2(\pi) = -0.35$ is always true and we can say that

$$|\langle h_{j,l,v}, C_k \rangle| \leq \pi 2^{\frac{j-p}{2}-1}.$$

For $2^{p-1} < |k| < 2^p$: Using lemma A.1 and A.3, we can say that

$$\begin{aligned}
|\langle h_{j,l,v}, C_k \rangle| &\leq 2^{\frac{-1-p-j}{2}} \left[2 \frac{\min(2, \pi 2^{j-p-1} |k|)}{2^{-p+1} |k|} + 2 \frac{\min(2, \pi 2^{j-p-1} |k|)}{2^{-p+1} |k|} \right] \\
&\leq 2^{\frac{p-j+1}{2}} \min(2 |k|^{-1}, \pi 2^{j-p-1}) \\
&\leq \min \left(2^{\frac{p-j+3}{2}} |k|^{-1}, \pi 2^{\frac{j-p-1}{2}} \right) \\
&\leq \min \left(2^{\frac{p-j+3-2p+2}{2}}, \pi 2^{\frac{j-p-1}{2}} \right) \\
&\leq \min \left(2^{\frac{-p-j+5}{2}}, \pi 2^{\frac{j-p-1}{2}} \right).
\end{aligned}$$

As $\frac{-p-j+5}{2} \leq \log_2(\pi) + \frac{j-p-1}{2} \Leftrightarrow 1.35 \leq j$, we can conclude that

$$|\langle h_{j,l,v}, C_k \rangle| \leq \begin{cases} 2^{\frac{-p-j+5}{2}} & \text{When } j > 1, \\ \pi 2^{\frac{-p}{2}} & \text{When } j = 1. \end{cases}$$

Finally, noticing that $2 \log_2(\pi) + j - p - 2 \geq -p - j + 5$ for $j \geq 2$, we found that

$$|\langle h_{j,l,v}, C_k \rangle| \leq \begin{cases} \pi 2^{\frac{j-p}{2}-1} & \text{When } j > 1, \\ \pi 2^{\frac{-p}{2}} & \text{When } j = 1. \end{cases}$$

□

Appendix B

Complete description of the experimental steps

B.1 Detailed first tests

This section presents the five tests that we made on each dataset built using the DCT dictionary for the foreground and the Haar wavelet one at fixed scale for the foreground. Each test has been done during 30 epochs.

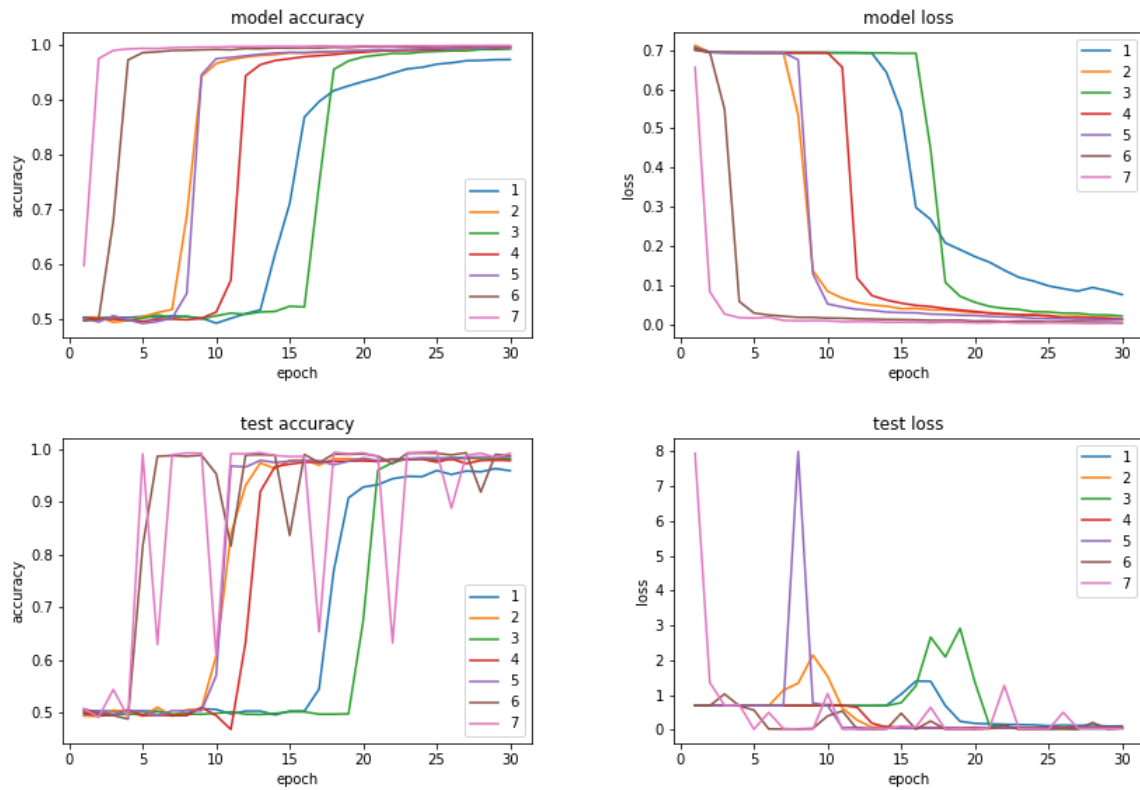


Figure B.1: Performances of the CNN on images with different scale, both on the training and the testing set. First test

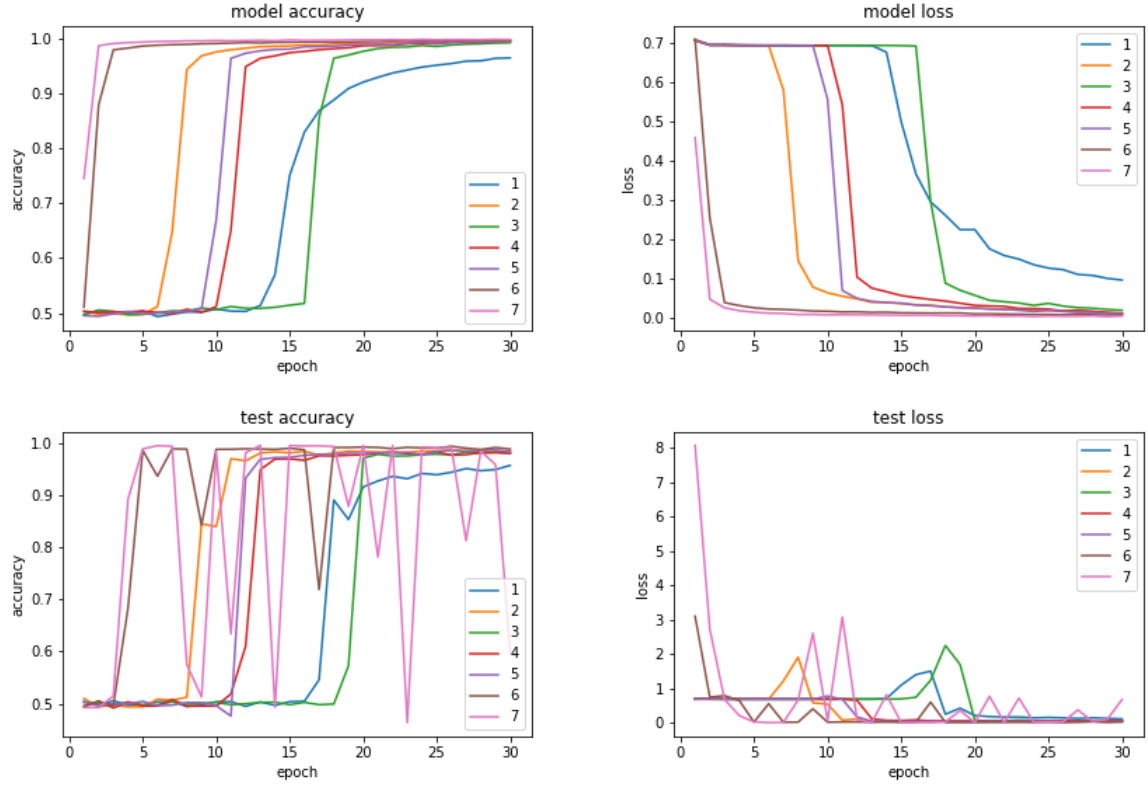


Figure B.2: Performances of the CNN on images with different scale, both on the training and the testing set. Second test

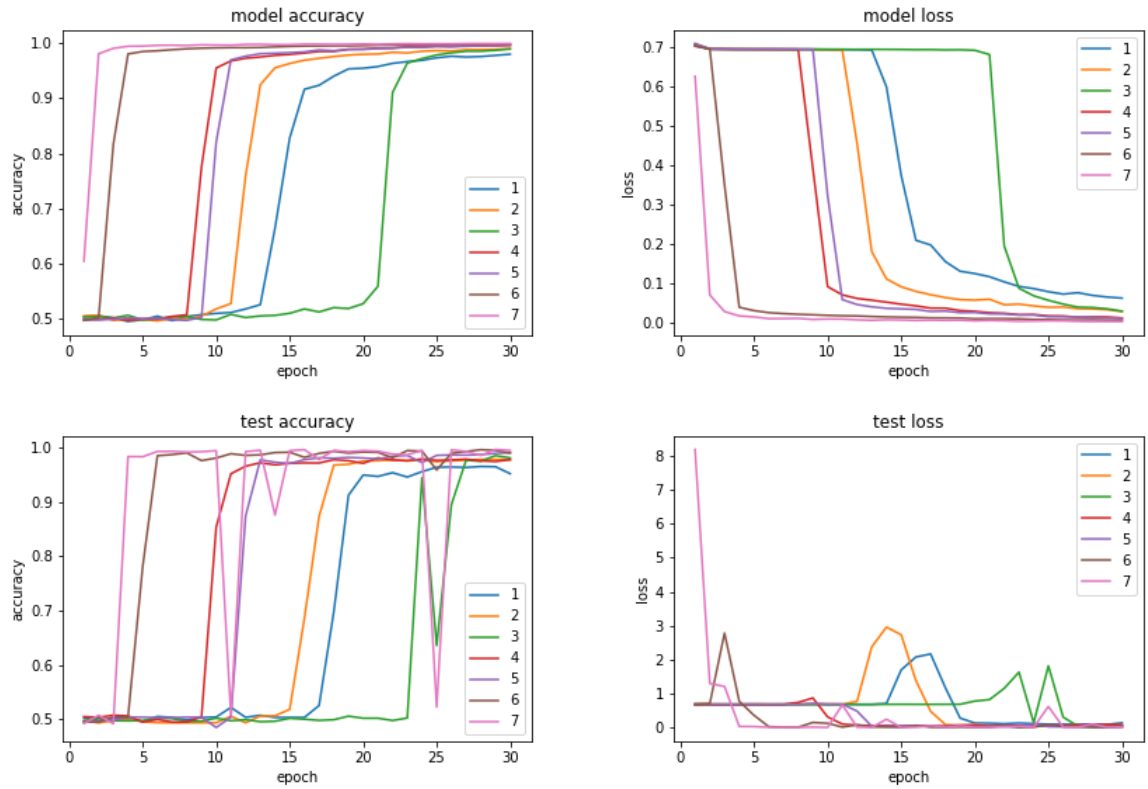


Figure B.3: Performances of the CNN on images with different scale, both on the training and the testing set. Third test

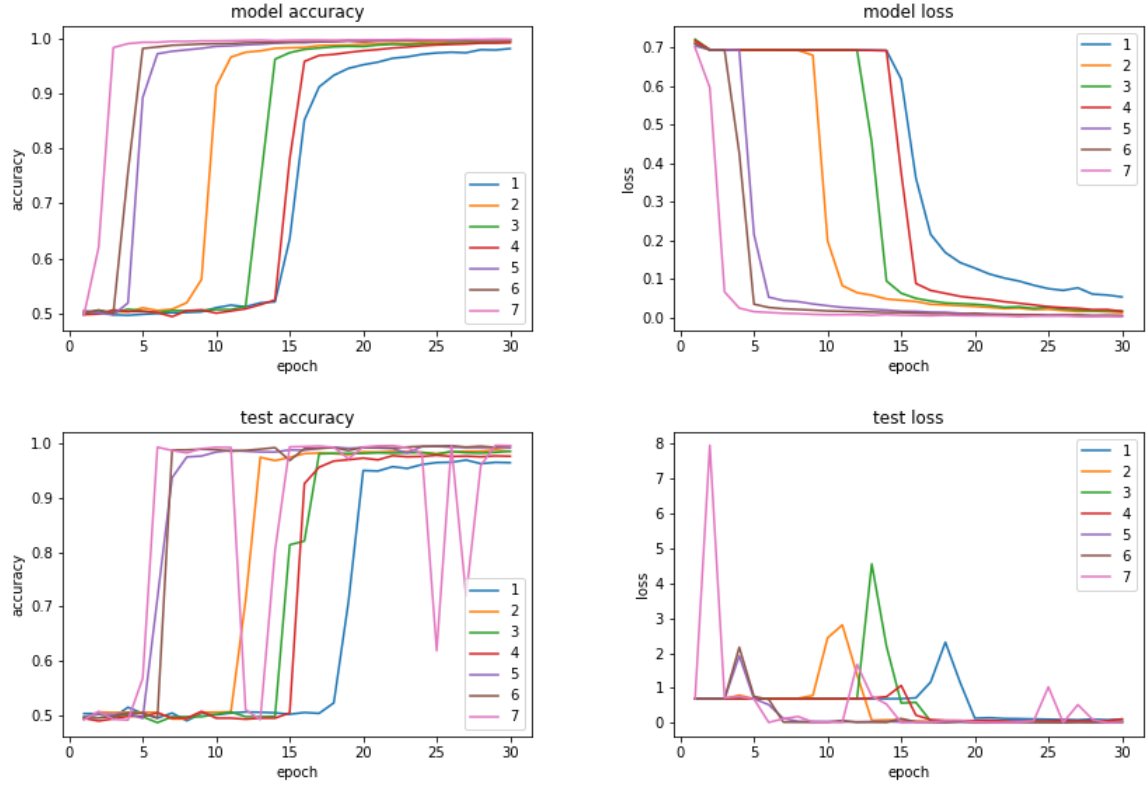


Figure B.4: Performances of the CNN on images with different scale, both on the training and the testing set. Fourth test

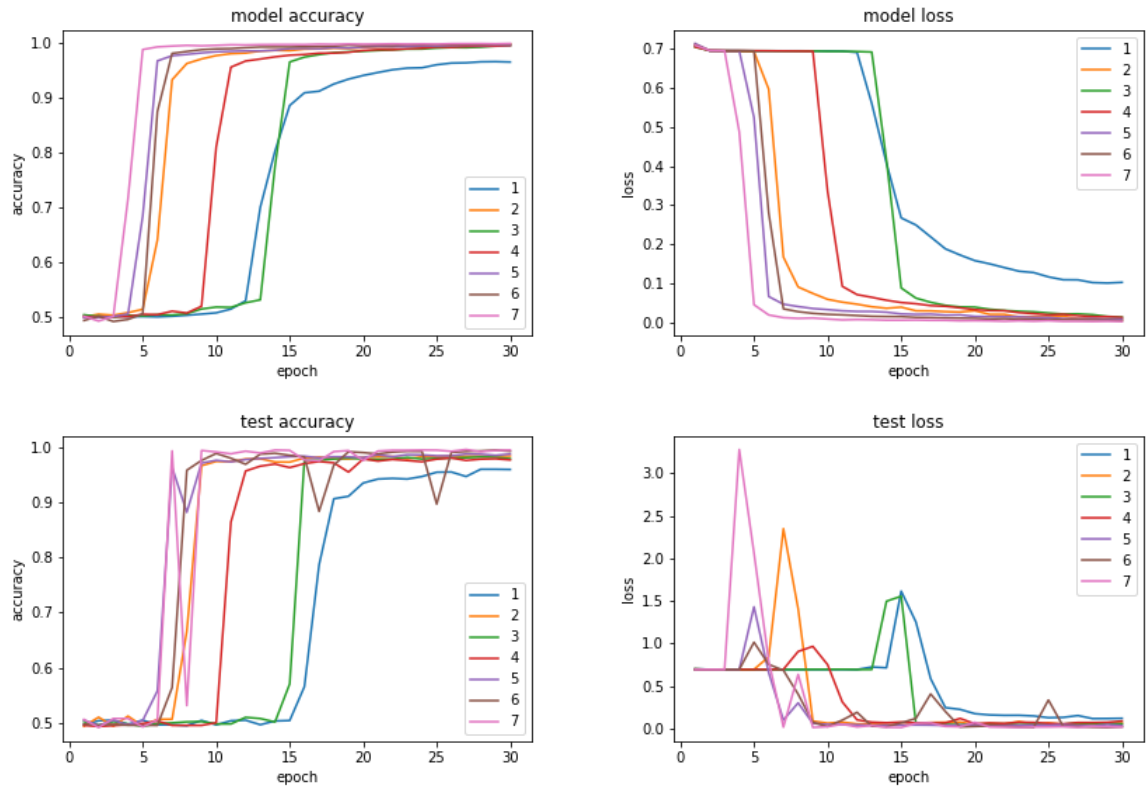


Figure B.5: Performances of the CNN on images with different scale, both on the training and the testing set. Fifth test

B.2 Frequency results

This section presents the results obtained on datasets containing different percentages of images with foreground at each scale. Each image is the mean result over five tests over 30 epochs.

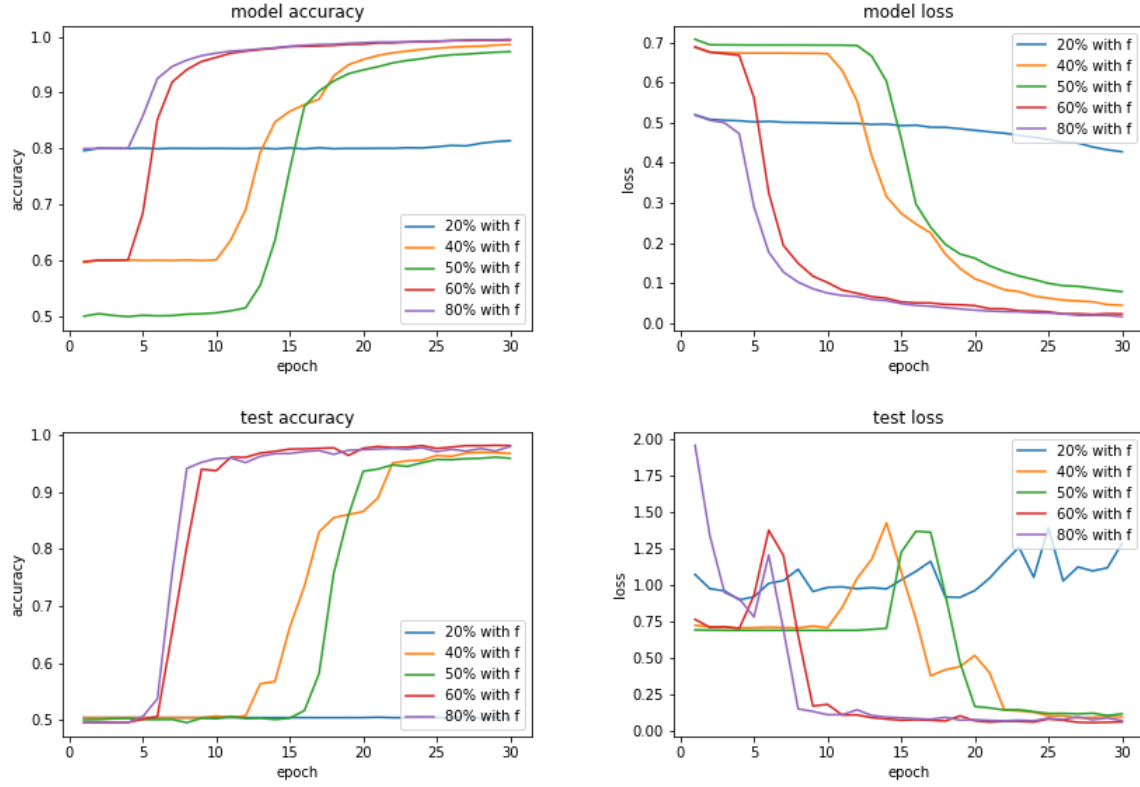


Figure B.6: Performances of the CNN on images at scale 1 with different percentage of training images containing a foreground.

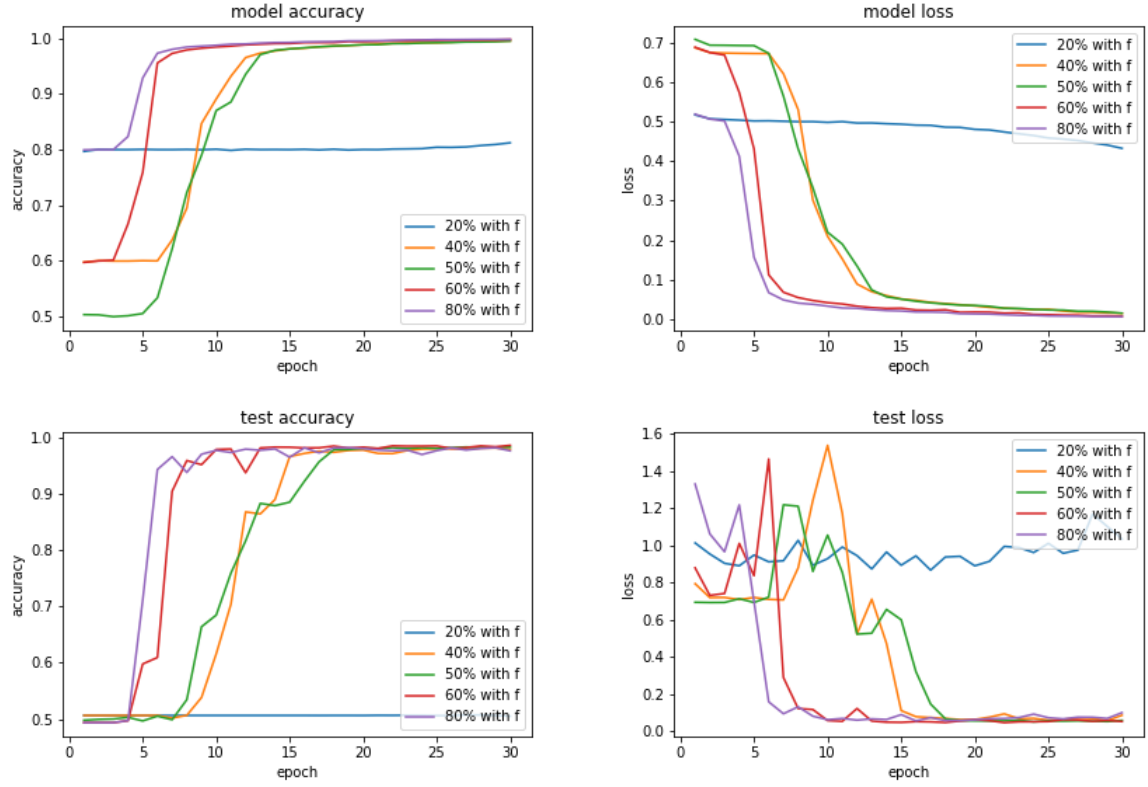


Figure B.7: Performances of the CNN on images at scale 2 with different percentage of training images containing a foreground.

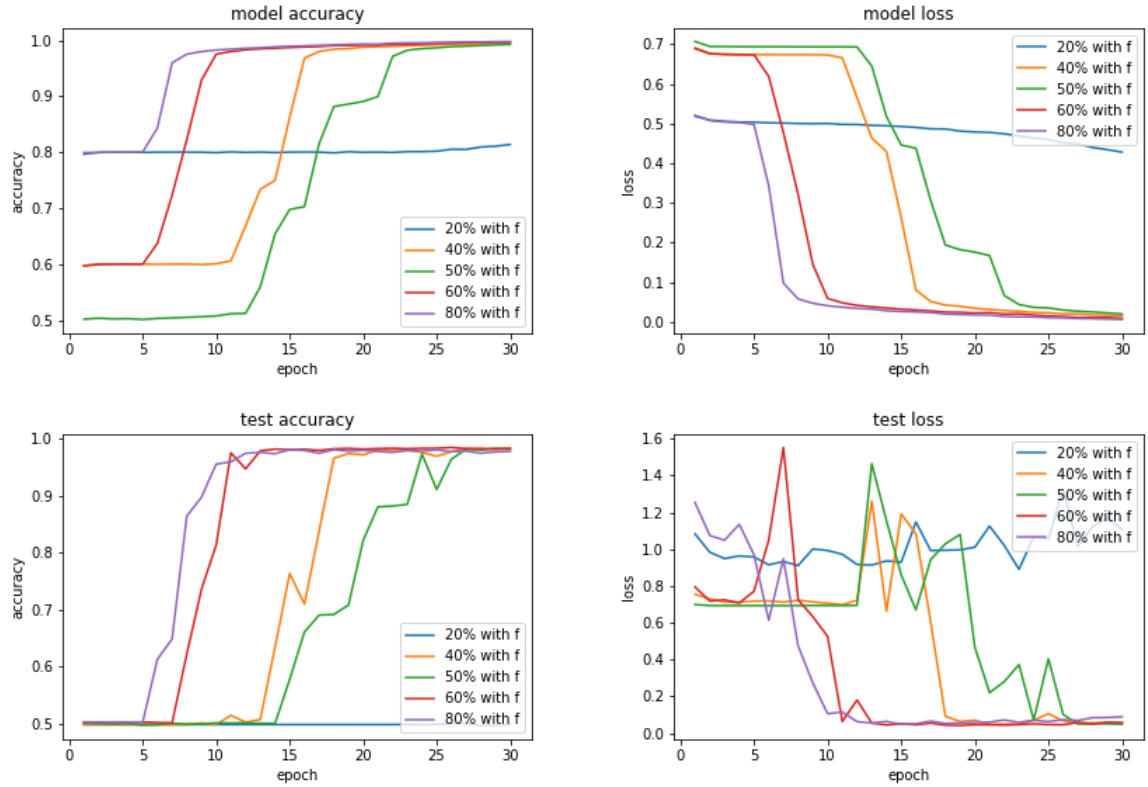


Figure B.8: Performances of the CNN on images at scale 3 with different percentage of training images containing a foreground.

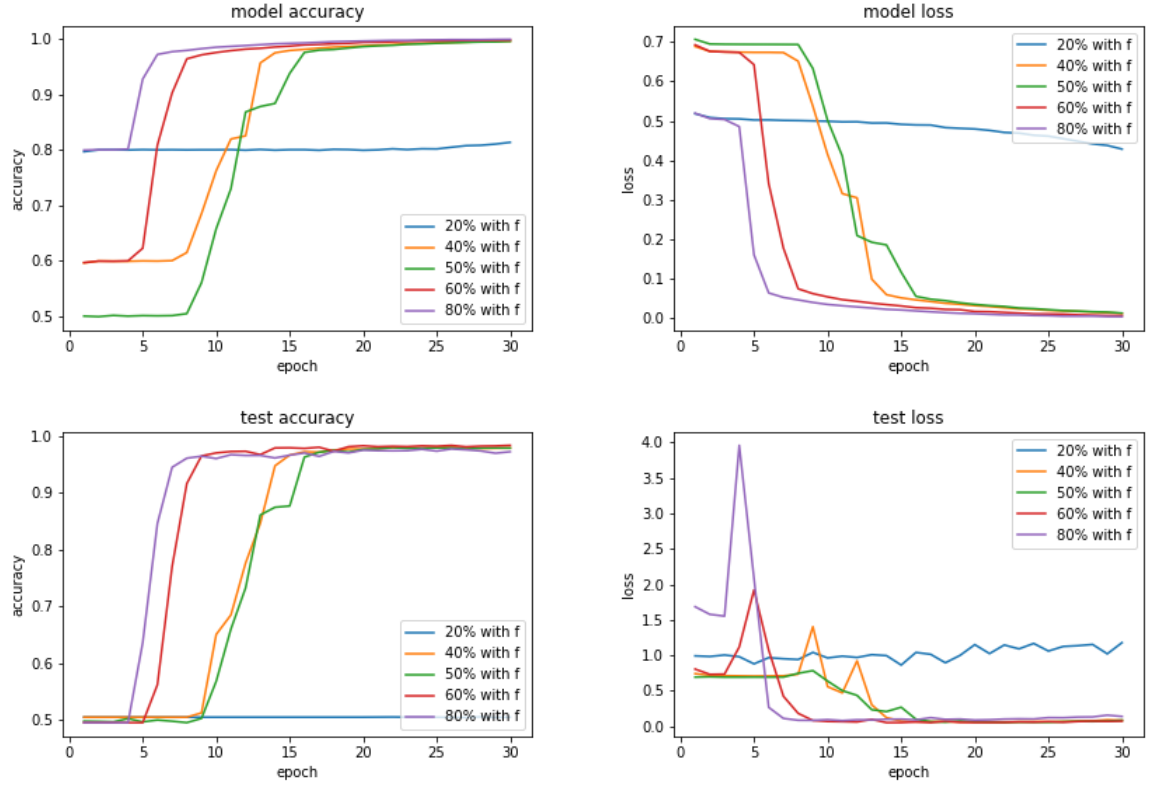


Figure B.9: Performances of the CNN on images at scale 4 with different percentage of training images containing a foreground.

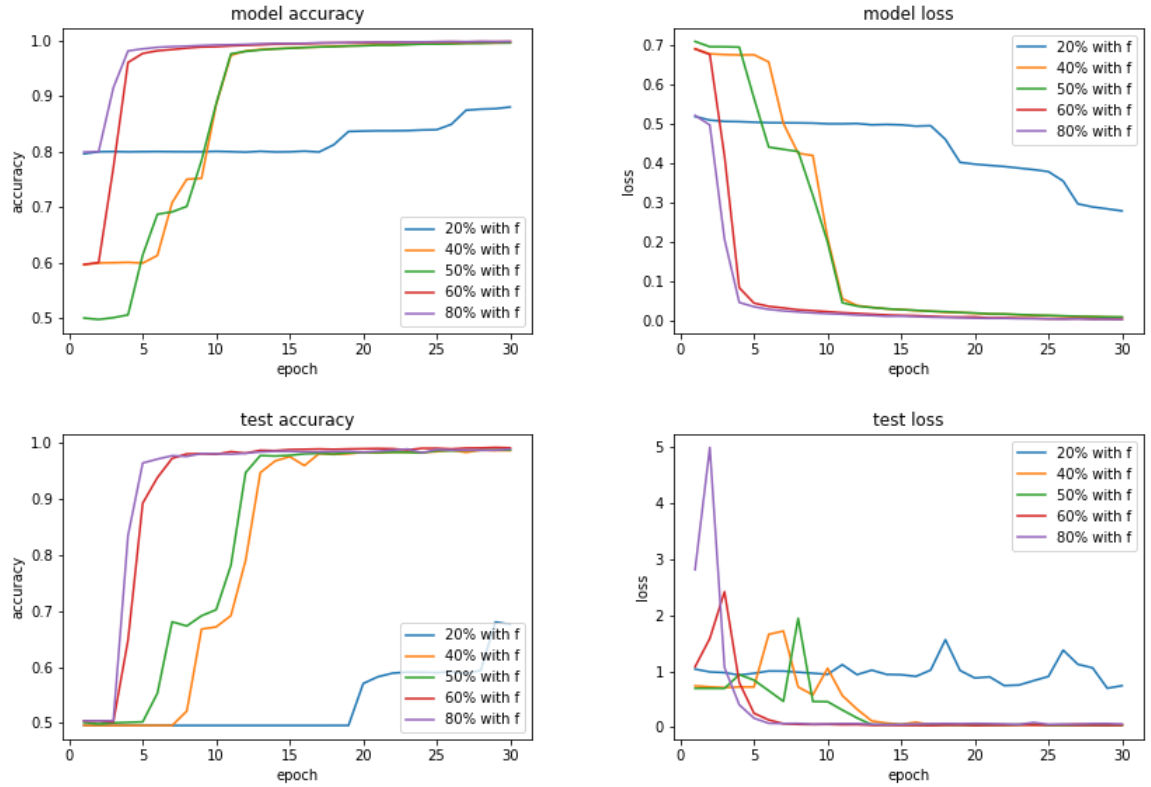


Figure B.10: Performances of the CNN on images at scale 5 with different percentage of training images containing a foreground.

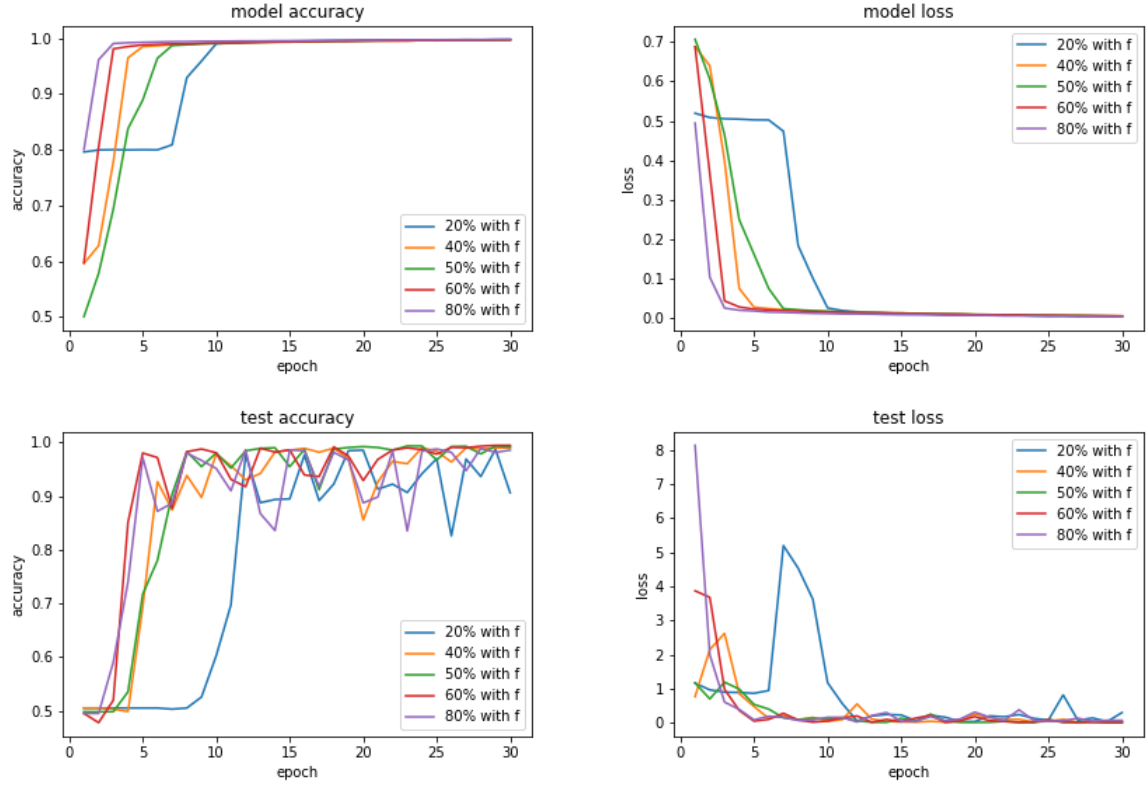


Figure B.11: Performances of the CNN on images at scale 6 with different percentage of training images containing a foreground.

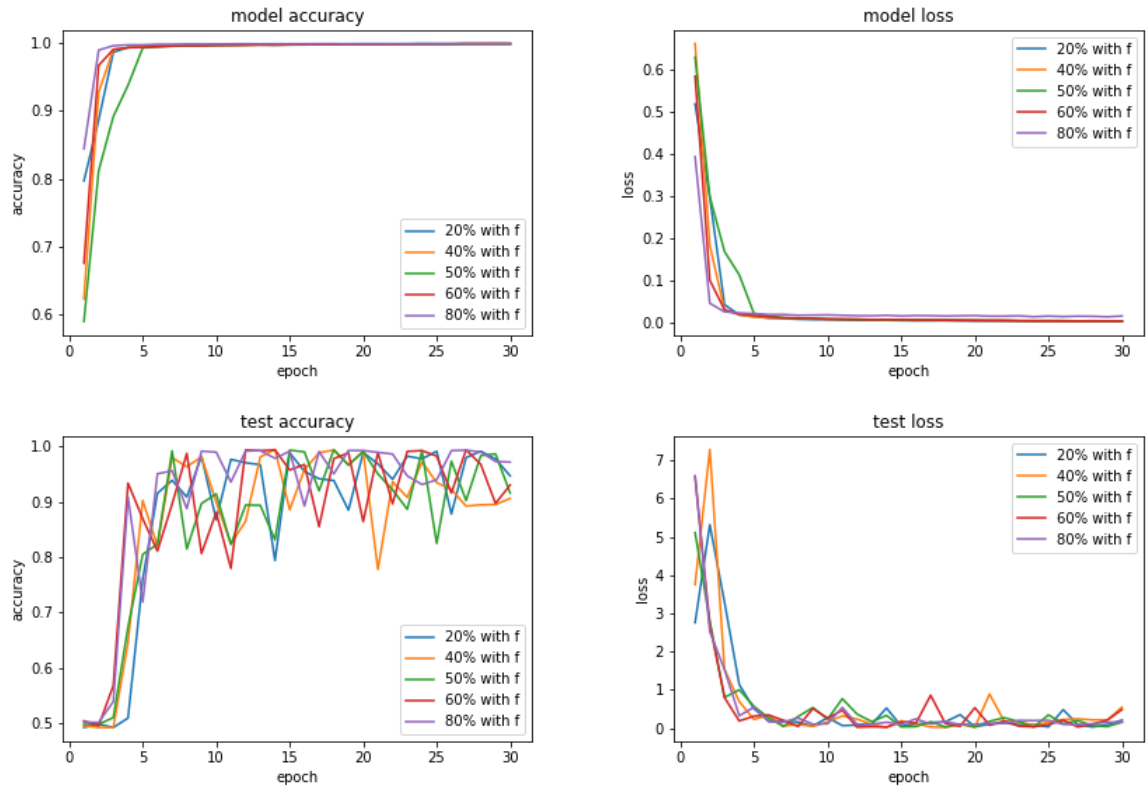


Figure B.12: Performances of the CNN on images at scale 7 with different percentage of training images containing a foreground.

