UCL Université catholique de Louvain

École polytechnique de Louvain (EPL)

epl

Learning For Anomaly Detection in Industrial Vision

Dissertation presented by
Pierrick FICHEFET

for obtaining the Master's degree in
Computer Science and Engineering

Supervisor(s)
Christophe DE VLEESCHOUWER

Reader(s)
John LEE, Tahani MADMAD

Academic year 2017-2018

Contents

1 Introduction

1.1 Background

Anomaly detection is the process of identifying unexpected items or events in datasets, which differ from the norm. Contrary to standard classification tasks, anomaly detection is often applied on unlabeled data, taking only the internal structure of the dataset into account. This challenge is known as unsupervised anomaly detection and is addressed in many practical applications. For example: network intrusion detection (identifying strange patterns in network traffic that could signal a hack), system health monitoring (spotting a malignant tumor in an MRI scan), fraud detection in credit card transactions, production line defect detection. Generally speaking, image denoising is the operation of taking a corrupted image and estimating its original one.

Image anomaly detection is an extremely simple process from a human point of view, but complex for a computer. Give a child a bunch of images and ask him to find those with something anomalous and he would prevail. So why something so simple becomes so complex from a computer point of view? Because anomalies could be pretty much of any type, they might be anomalous shapes or colors, they might be a scratch or a crack. Moreover, clean images could represent anything: a cat, a car, a landscape... All of these unknown variables make the task of anomalies detection very difficult to generalize.

1.2 Related Work

The topic of automatic defect detection have been broadly covered by several works during this last decade. There are not a single way to tackle this problem, although several methods using image processing techniques have been proposed in recent years. As it is not practical to propose an exhaustive survey of all the methods that have been used for automatic defect detection, this thesis will focus on some techniques widely used and that have demonstrated good performance at it. These approaches can be categorized in three main groups: statistical, spectral and Artificial Neural Network (ANN) classifier.

Statistical approaches measure the spatial distribution of pixel values. More specifically, they use gray-level texture features derived from first order statistics to higher order statistics. Amongst many, co-occurrence matrices, local mean, standard deviation, auto correlation and local binary patterns have been applied for detecting fabric defects. For instance, these papers use various statistical approaches for detecting defects [13], [14], [15], [16], [17].

Spectral approaches consist mainly on locating defects in the spectral domain. [18], [19].

ANN classifiers rely on machine learning processing technique to identify and classify surface defects. This method needs labeled data for the training phase. During this phase the network tries to learn a mapping between every image and its correct class. The number of classes depends on how many type of defects the neural network could encounter. Afterwards, the ANN classifier should be able to correctly classify an unseen image. [20], [21].

1.3 Aim of the thesis

Instead of trying to develop new architectures for image denoising, this thesis will be based on the study of different existing models. It will focus on Convolutional Neural Network (CNN) for three reasons. First, CNN with very deep architecture are able to increase the capacity and flexibility for exploiting image characteristics. Second, significant progress has been made on regularization and learning methods for training CNN, such as Rectifier Linear Unit (ReLu), batch normalization and residual learning. These methods help to speed up the training process. Third, CNN is well-suited for parallel computation on modern powerful GPU, which can be exploited to improve the run time performance.

Two different patterns will be developed as accurately as possible. Both of them were first designed to be an effective Gaussian denoiser, but their creators have extended their utilization to a single image super-

resolution and to the JPEG image deblocking problem. The reproduction of these two models is tested on height data folds: they are based on the occurrence of any defect detection.

Three steps are necessary in order to detect defects on images. First the two reproduced Convolutional Neural Networks are trained over images that have been made artificially noisy. Structured noises like scratch, stain or unstructured noise (Gaussian noise for example) are added on clean images. Secondly CNN try to find a mapping, during the learning phase, between the artificially noisy images and their clean versions. In other words the CNN learns to remove the artificial noise from images. After this training phase a test dataset which contains a mix of real noisy images and clean images is given as input to these CNNs. If the input image is noisy the CNN should output its clean version, and if the input image is clean the CNN should output the exactly same image. Thirdly we compare each input with its corresponding output. If the input and the output are the same the input image shouldn't be noisy, and will be labeled as "clean". But if the input and output are different the input image should contain noise/defects, and will be labeled as "noisy". So far, CNN has not been used in this way to detect defects yet.

1.4 Structure

The thesis is organized as follows. Chapter 2 provides theoretical basis on which this thesis relies on. Chapter 3 presents the various architectures that I use. Chapter 4 is about the methodology. How datasets are preset and how they are used during the training and testing phases. The end of this chapter presents the way of anomaly detection after the denoising works. Results obtained from these architectures with the different datasets are presented in chapter 5 and are discussed afterwards. Finally, Chapter 6 gives the conclusions drawn on the basis of the presented work.

2 Preliminary information

2.1 What is an artificial neural network?

Neurons are part of artificial neural networks which are inspired by the brain neural system. Our brain contains billion of neurons inter-connected to each other. These neurons are the most basic structures of this system.

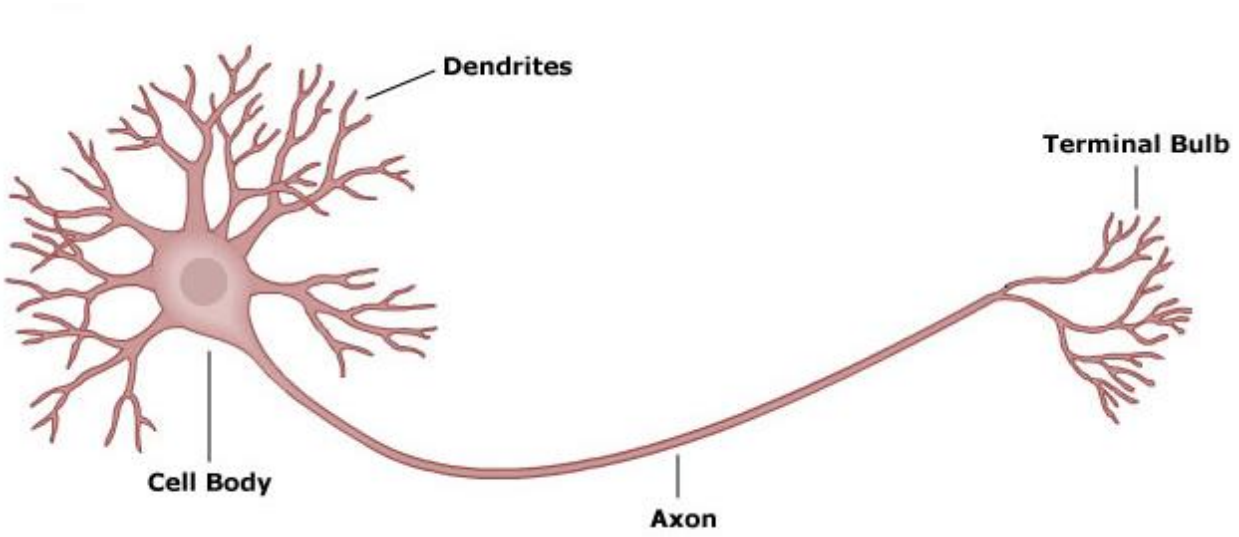

Figure 1: The neuron, image from : $http://news.unchealthcare.org/images/science-images/neuron-illustration-1/image_viewfullscreen$, May 2018

A neuron is an electrically excitable cell that takes up, processes and transmits information. The information in the form of an electrical impulse comes in from the dendrites that are connected to other neurons, is processed into the cell body and then moves along the axon towards other neurons.

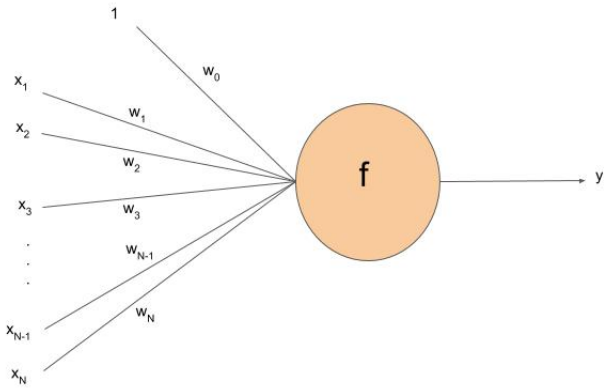

Figure 2: The artificial neuron

An artificial neuron follows the same principle, as show in Figure 2. Instead of an electrical impulse this artificial neuron takes a vector as input and sends a real number as output. In Figure 2, our artificial neuron take the vector $\boldsymbol{x}$ as input and processes it into y.

$$y = f(w_0 * x_0 + w_1 * x_1 + ... + w_n * x_n) \tag{1}$$

$$y = f(\boldsymbol{w}^T * \boldsymbol{x}) \tag{2}$$

Where w_0 ... w_n are the neuron's weights, and f is an activation function, see Section 2.2.

The process taking place into the cell body is then replaced by a mathematical function that gives us the result y that travels along the axon to other neurons. It is also possible to use more than one artificial neuron, this set of neurons is called a layer. In this case the output of the k^{th} neuron is:

$$y_k = f(\boldsymbol{w}_k^T * \boldsymbol{x}) \tag{3}$$

Now that we know how works a single layer into a neural network we can connect many layers together in order to form a more complex network, as shown in Figure 3. The network takes $\boldsymbol{x}$ as input which is composed of p signals (x_1 through x_p). Usually, the x_0 input is assigned to the value $+1$, which makes it a bias. This network is composed of N layers, and each of them could have an arbitrary number of neurons.

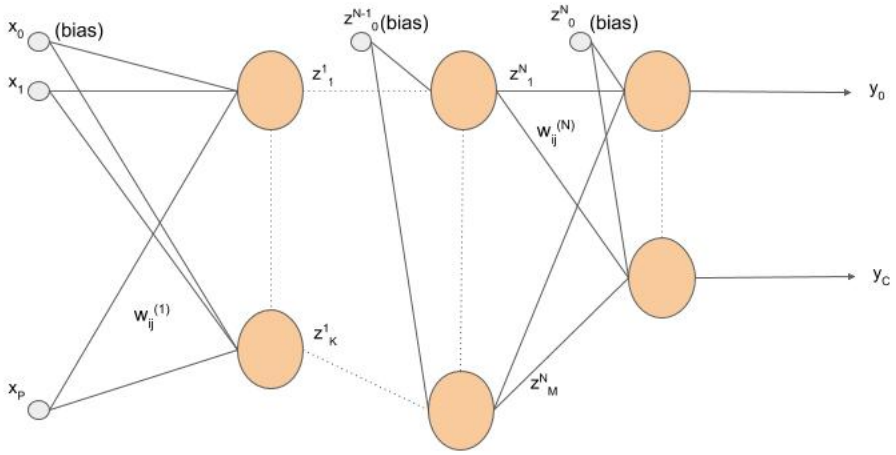

Figure 3: A more complex neural network

Of course we can also express this network by a mathematical expression. This expression is written down below for a neural network of 2 layers with g denoting the activation function of the first layer, and h denoting the activation function of the second.

$$y_k(\boldsymbol{x}) = h(\sum_{i=0}^{M} w_{ki}^{(2)} g(\sum_{j=0}^{P} w_{ij}^{(1)} * x_j)) \tag{4}$$

$$y(\boldsymbol{x}) = h(\boldsymbol{w}^{(2)} g(\boldsymbol{w}^{(1)} \boldsymbol{x})) \tag{5}$$

It is important to note that the number of neurons in the last layer defines the dimensionality of our output. In Figure 3 the output has a dimensionality of C, (y_1 through y_C).

2.2 Activation functions

An activation function of an artificial neuron defines the output of that neuron according to its input. In the literature we find that several activation functions are used, depending on the context and the network's architecture.

The only activation function used in this paper is the Rectifier Linear Unit (ReLU), which is a simple function:

$$f(z) = \begin{cases} 0 & \text{if } z < 0 \\ z & \text{if } z \geq 0 \end{cases}$$

A wide variety of sigmoid functions have been used as the activation function of artificial neurons, before being gradually replaced by the ReLU function that seems to be more beneficial. The sigmoid function:

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

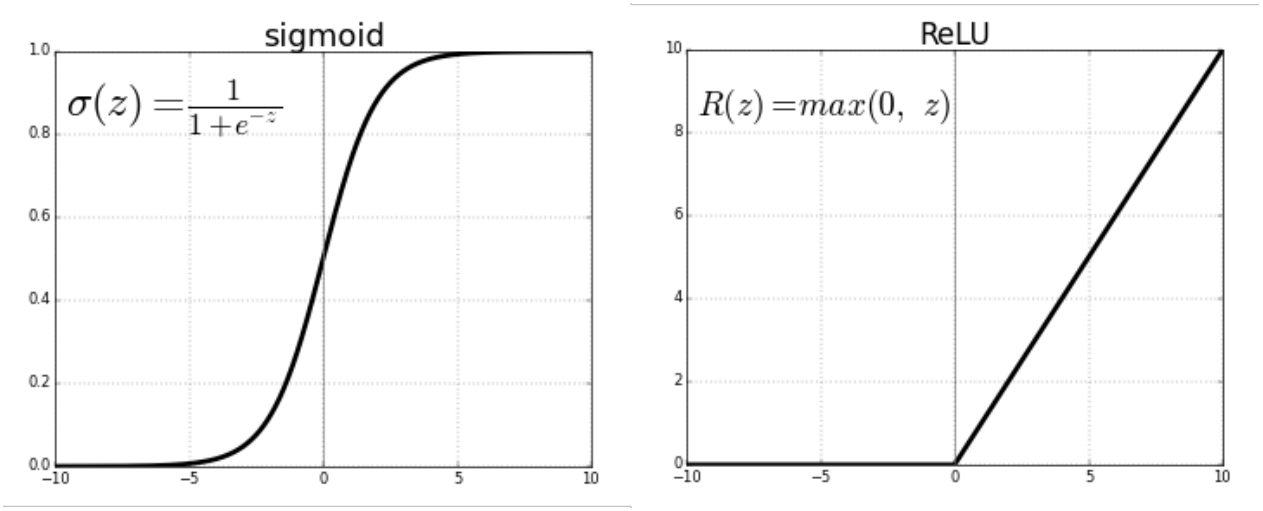

Figure 4: RelU VS Sigmoid, image from "https://towardsdatascience.com/activation-functions-neural-networks-1cbd9f8d91d6", May 2018

2.3 Training a neural network

Since we have now an artificial complex neural network, it is time to train it. How an artificial neural network learns is closely related to how we learn in our daily life. We perform an action and according on how good we were, we are congratulated, corrected or reprehended. Most of the time, hints are given to us in order to improve yourself. Similarly a trainer describes to the neural network what should have been produced as a response to the input. This artificial trainer is a cost function.

For example, if that we want to predict the noise (in decibels) produced by an airplane. We could collect as input a lot of data like the air pressure, the weight of the plane, the manufacturer and the wind direction. The information is saved into a vector $\boldsymbol{x}$, this vector $\boldsymbol{x}$ has then 4 components. For each vector $\boldsymbol{x}$, we also need to measure the noise produced in such conditions. If we do this for a while, we will have a lots of data couples ($\boldsymbol{x}$,r) where r (the label) is the noise produced in conditions $\boldsymbol{x}$.

For each input data $\boldsymbol{x}$ a prediction y is made by the network. We can now use a cost function to tell to the network whether it is doing a good job. This is done by comparing the prediction y (the predicted noise) with the target value r (the real noise produced). This cost function could be any function but we take as example one of the most used in the literature, the mean squared error:

$$E = \frac{1}{n}\sum_{i=1}^{n}(r_i - y_i)^2$$

The error between the prediction and the target value is then propagated through the network in order to help it improve its prediction, this error propagation tells the network how large is the error between the prediction and the target value and in which direction the prediction should be updated in order to decrease this gap. This is done multiple times for each couple ($\boldsymbol{x}$,r) until a stopping criterion is met, we can then hope that our neural network will be able to predict accurately the noise produced by a airplane for an unseen vector $\boldsymbol{x}$.

More informations can be found in [22].

2.4 Gradient descent

The gradient descent is a first order iterative method for finding the minimum of a function. This is perfect for us since in the previous section we show that what a neural network wants, it is finding the minimum of the cost function E. In other words we want to minimize the difference between the target value r and its corresponding prediction y.

This is the gradient descent algorithm:

$$\theta_i(j+1) := \theta_i(j) - \alpha * \frac{\partial}{\partial \theta_i} J(\theta_i) \tag{6}$$

In this equation, the term α is the learning rate and it controls how big is the step you take when updating the parameter θ_i. j represents the j^{th} iteration of the algorithm. The last term $\frac{\partial}{\partial \theta_i} J(\theta_i)$ is called the derivative term.

In order to give you a good intuition of what each of these two terms is doing, we will minimize, as example, a simple function of one parameter $J(\theta_1)$.

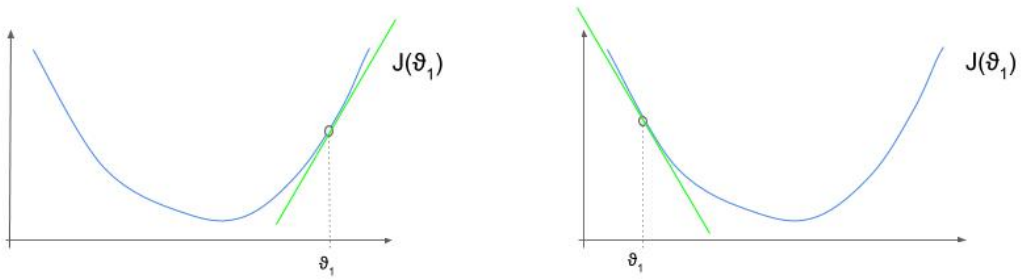

Figure 5: Function of one parameter.

The derivative term calculates a slope in a function point, in Figure 5, the slope is represented by the green line. We can see that the slope is positive on the left graph and negative on the right one. It basically means that on the left graph the term $-\alpha * \frac{\partial}{\partial \theta_1} J(\theta_1)$ is negative and positive on the right graph. Now lets have a look on how θ_1 will move over time. For the left graph, we have $\theta_1(j+1) := \theta_1(j)+$ "a negative term", meaning that θ_1 will move towards the left of the left graph at each step j and towards the right for the right graph. The learning rate α controls how far from the previous point the next point will be. At the minimum of the function, the slope is equal to zero, θ will then not move anymore.

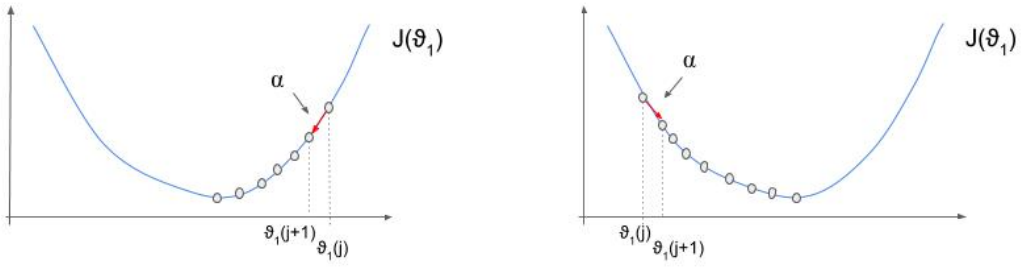

Figure 6: Example showing several steps of the gradient descent algorithm until convergence to a local minimum.

In our neural network we do not have only one parameter, every weight is a parameter. We then have to apply this algorithm for every weight of our artificial neural network to converge towards one minima of our

cost function. In practice we need to calculate every $\frac{\partial E}{\partial w_{ij}^{(l)}}$ in order to apply the gradient descent algorithm on every weight of our network.

$$w_{ij}^{(l)}(j+1) := w_{ij}^{(l)}(j) - \alpha * \frac{\partial E}{\partial w_{ij}^{(l)}} \tag{7}$$

See [9] and [23].

2.5 Error back-propagation

The error back-propagation consists of finding the expression $\frac{\partial E}{\partial \boldsymbol{w}}$ which represents how quickly the cost changes when we modify the weights and biases. It actually gives us detailed insights into how changing the weights and biases improve or deteriorate the value of the cost function.

We introduce some notations: z_j^l is the j^{th} output of the l^{th} layer, $w_{ij}^{(l)}$ is the weight between the j^{th} output of the layer $l-1$ and the i^{th} input of the layer l, $a_i^{(l)} = \sum_j w_{ij}^{(l)} z_j^{(l-1)}$, see Figure 7.

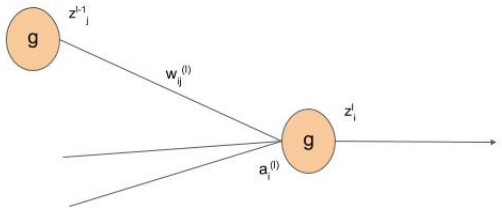

Figure 7: Some notations

First, we need to find an expression for every $\frac{\partial E}{\partial w_{ij}^{(l)}}$ in order to adjust every weight according to derivatives and a gradient descent scheme.

$$\frac{\partial E}{\partial w_{ij}^{(l)}} = \frac{\partial E}{\partial a_i^{(l)}} \frac{\partial a_i^{(l)}}{\partial w_{ij}^{(l)}} \tag{8}$$

$$z_j^{(l-1)} = \frac{\partial a_i^{(l)}}{\partial w_{ij}^{(l)}} \tag{9}$$

$$\delta_i^{(l)} = \frac{\partial E}{\partial a_i^{(l)}} \tag{10}$$

$$\frac{\partial E}{\partial w_{ij}^{(l)}} = \delta_i^{(l)} z_j^{(l-1)} \tag{11}$$

Second, we need to evaluate every $\delta_i^{(l)}$, which is easy for the output units which produce the prediction y:

$$\delta_i^{(l)} = \frac{\partial E}{\partial a_i^{(l)}} = \frac{\partial E}{\partial y_i^{(l)}} \frac{\partial y_i^{(l)}}{\partial a_i^{(l)}} \tag{12}$$

$$\delta_i^{(l)} = \frac{\partial E}{\partial y_i^{(l)}} g'(a_i^{(l)}) \tag{13}$$

In this last expression, the derivative g' and every $a_i^{(l)}$ are known. We still have the expression $\frac{\partial E}{\partial y_i^{(l)}}$ to evaluate which is trivial since $E = \frac{1}{n}\sum_{i=1}^{n}(r_i - y_i)^2$.

$$\frac{\partial E}{\partial y_i^{(l)}} = \frac{-2}{n} * (r_i - y_i) \tag{14}$$

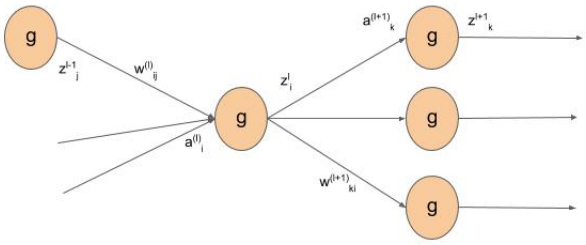

Figure 8: Some more notations

The evaluation of the $\delta_i^{(l)}$ is a little bit more complex for hidden units, see Figure 8:

$$\delta_i^{(l)} = \frac{\partial E}{\partial a_i^{(l)}} = \sum_k \frac{\partial E}{\partial a_k^{(l+1)}} \frac{\partial a_k^{(l+1)}}{\partial a_i^{(l)}} \tag{15}$$

$$\delta_i^{(l)} = \sum_k (\delta_k^{(l+1)}) \frac{\partial (\sum_{j\epsilon(l)} w_{kj}^{(l+1)} z_j^{(l)})}{\partial a_i^{(l)}} \tag{16}$$

$$\delta_i^{(l)} = \sum_k (\delta_k^{l+1} w_{ki}^{l+1} g'(a_i^{(l)})) = g'(a_i^{(l)}) \sum_k (\delta_k^{(l+1)} w_{ki}^{(l+1)}) \tag{17}$$

As you can see for the hidden units, the error term δ is expressed as a combination of errors in the next layer.

After all this mathematical work, we have all the tools we need for building an algorithm which will update weights in order to improve the cost function and then the overall network. The algorithm will follow these 5 steps:

- Apply an input vector $\boldsymbol{x}^k$ and propagate it through the network to evaluate all activations z_i^l and neuron outputs $a_i^{(l)}$.
- Evaluate error terms $\delta_i^{(o)}$ in output layer.
- Back-propagate error terms $\delta_i^{(l)}$ to find error terms $\delta_i^{(l-1)}$.
- Evaluate all derivatives $\frac{\partial E}{\partial w_{ij}^{(l)}} = \delta_i^{(l)} z_j^{(l-1)}$.
- Adjust weights according to derivatives and a gradient descent scheme.

Equations and Algorithm come from [4].
See also [25].

2.6 The gradient vanishing problem

Intuitively more a network has layers, more accurate it will be since they make the network able to learn more complex classification functions. Unfortunately it is not what is observed in experiments. At the beginning, when we add layers, the network has a tendency to perform a better job until we reach a threshold. Beyond this threshold, adding layers will only decrease performances. How is that possible?

Extra layers should help the network to learn, the problem does not really comes from these extra layers, it comes from our learning algorithm which does not find the appropriate weights and biases. This problem is known as the gradient vanishing problem.

To get more insight about the reason this problem occurs, we recall the mathematical expression for the gradient $\frac{\partial E}{\partial w_{ij}^{(l)}}$, see Figure 7. Remember that this expression is the rate of change of the cost function regarding the neuron's weights.

$$\frac{\partial E}{\partial w_{ij}^{(l)}} = \frac{\partial E}{\partial a_i^{(l)}} \frac{\partial a_i^{(l)}}{\partial w_{ij}^{(l)}}$$

Where

$$a_i^{(l)} = \sum_j w_{ij}^{(l)} z_j^{(l-1)}$$

For our example we consider a simple deep neural network:

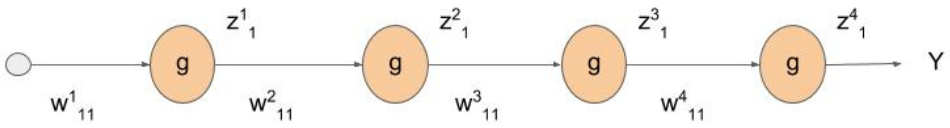

Figure 9: The simplest DNN

We write down the mathematical expression for the gradient $\frac{\partial E}{\partial w_{11}^1}$ in order to show why the vanishing gradient problem occurs.

$$\frac{\partial E}{\partial w_{11}^1} = g'(a_1^1) * z_1^1 * w_{11}^2 * g'(a_1^2) * w_{11}^3 * g'(a_1^3) * w_{11}^4 * g'(a_1^4) * \frac{\partial E}{\partial y} \tag{18}$$

$$\frac{\partial E}{\partial y} = -2(r - y) \tag{19}$$

Where $a_i^l = w_{ii}^l * z_i^{l-1}$, r is the target value and g the sigmoid function that is used as activation function. Then excepting the very last term, the expression of $\frac{\partial E}{\partial w_{11}^1}$ is a product of terms of the form $w^l * g'(a^l)$. In order to understand how each of these terms behave, take a look at the plot of the function g', shown in Figure 10. We can see that this function takes a maximum value of 0.25 when a = 0. Since usually the weights are initialized with a value lower than 1, we have that $|w^l * g'(a^l)| < 1/4$. Now we begin to see what is going on, when we do the product of many of such term the result will exponentially decrease. In other words, the more of these terms are in the product chain the smaller the result will be.

Conceptually it means that the earlier layers of a very deep neural network will learn slower than the last layers since their gradients are exponentially lower. If the network is deep enough it could be possible that

layers learn nothing at all which could lead to worser performances.

Of course this is not a rigorous proof, there are several escape clauses. But at least, this could give an hint of what could happen and leads to the gradient vanishing problem.

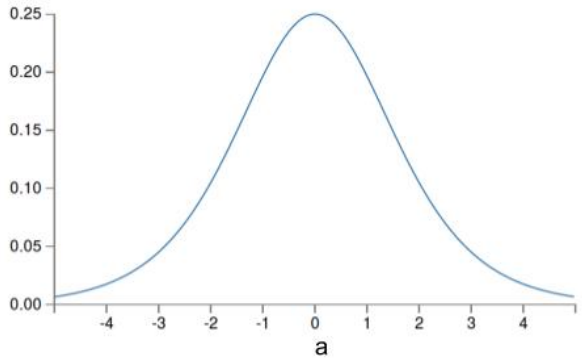

Figure 10: Derivative of sigmoid function, image from: "http://neuralnetworksanddeeplearning.com/chap5.html", May 2018

More informations about the gradient vanishing problem can be found in [10], [26], [27] and [28].

2.7 Epoch Vs Batch

The number of epochs is the number of time we run the entire dataset through the network. It is necessary since, as we have seen, gradient descent is an iterative process. Running the entire dataset only once will not be sufficient. The function that we try to learn will fit even more relevantly out dataset - going from under-fitting to optimal, and then, from optimal to over-fitting, as the number of epochs increases and as the number of time weights are updated increases.

The data are split in several batches, the number of batches is then the number of iterations that we need to complete one epoch.

This split in batches is necessary because we do not want to apply Gradient Descent algorithm over the entire training dataset. Applying such an algorithm over the whole dataset would mostly lead towards a local minimum, from which we would rather escape. On the other hand, the Mini-batch Gradient Descent algorithm updates weights after each batch. These updates may go off in a direction far from the Gradient Descent, since batch contains only a fraction of the training dataset and could be be very noisy. These occasionally updates in the "wrong" direction help you to escape from saddle points.

Moreover, in machine learning data are most of the time too big to be given to the computer all at once. So, to overcome this problem, splitting the data into batches is also a good solution.

See [6], [7], [8], [29] and [30].

2.8 The sparsity of a network

In numerical analysis, a sparse matrix is a matrix in which most of its elements are zero. In the same way a sparse neural network is a network in which most of its weights are zero.

In Convolutional Neural Network, it often is observed that sparse networks have better efficiency than dense Convolutional networks with the same number of parameters. The "why" is still an open question but papers like [11] have tried to explain this phenomenon. Their main intuition is that sparse Convolutional

networks promote diversity. This paper formalizes this with a simple observation that any dense network is in fact, a part of an exponentially large equivalence class, which is guaranteed to produce the same output for every input.

But as said, it is still an open question, in some context the activation function Maxout beats ReLU or perform more or less the same and it is not sparse at all, shown in paper [12]. ReLU allows a network to easily obtain sparse representations, since hidden units output are real zero for any input equal or under zero.

2.9 What is a Convolutional Neural Network?

Convolutional neural networks are very similar to artificial neural networks that we introduced in previous sections: they are made up of neurons that have learnable weights and biases. The main difference between these two kinds of network is that Convolutional architectures assume explicitly that the inputs are images. It is a kind of architecture specifically developed for the field of computer vision. This thesis does not intend to be a new lecture on Convolutional architectures and introduces only the basics on layers used.

2.9.1 The Convolutional layer

The most used layer in a CNN is the Convolutional layer, it takes an image as input which is seen as an array of pixel values by a computer, as shown in Figure 11. Depending on the resolution and the size of the image, the layer see a 32 x 32 x 3 array of numbers (The 3 refers to RGB values). The best way to explain a Convolutional layer is to imagine a flashlight that is shining over the top left of the image. We say that this flashlight covers a 5 x 5 area. And now, let's imagine this flashlight sliding across all the areas of the input image, shown in Figure 12.

What We See

What Computers See

Figure 11: Image seen by a computer, image from: [5]

In machine learning terms, this flashlight is called a filter and the receptive field is the region that it is shining over. It is important to note that this filter needs to have the same depth as the input image, the filter has then a dimension of 5 x 5 x 3. The filter contains weights that are multiplied by the original value of the image and then sum up together, that gives us one number. Remember, this number is just representative of when the filter is at the top left of the image. Now we repeat this process for every possible location of that filter, at each step the filter moves one pixel further. The reason you get a 28 x 28 array is that there are 784 different locations that a 5 x 5 filter can fit on a 32 x 32 input image. These 784 numbers are mapped to a 28 x 28 array.

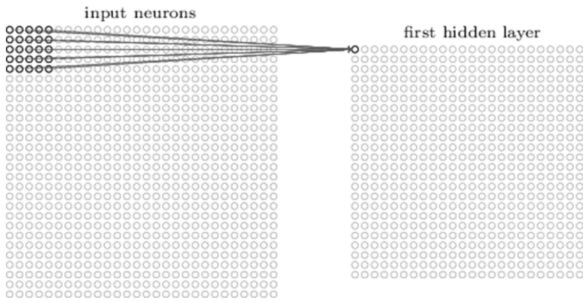

Figure 12: Visualization of a 5x5 filter convolving around a 32x32 input volume and producing a 28x28 activation map, image from: [5]

By using more filters, we are able to improve the preservation of the spatial dimensions. For example, if we use a second filter we would end up with an image of dimension 28 x 28 x 2. Mathematically, this is what happens in a Convolutional layer.

2.9.2 The stride and the padding parameters

At the moment we know how works a filter, but there are still two main parameters that we should cover, the stride and the padding. These two parameters control the behavior of the filter in the input volume.

Stride controls how the filter convolves around the input volume. In the example above, the filter convolves around the input volume by shifting one unit at a time. The amount by which the filter shifts is the stride. In that case, the stride was implicitly set at 1. By increasing the stride, we will end up with a smaller image since the filter will have less positions to fit in.

Now, we take a look at padding. Before getting into that, let's think about a scenario. What happens when you apply three 5 x 5 x 3 filters to a 32 x 32 x 3 input volume? The output volume would be 28 x 28 x 3. Notice that the spatial dimension decreases. As we keep applying Convolutional layers, the size of the volume will decrease faster than we would like. In the early layers of our network, we want to preserve as much information about the original input volume so that we can extract those low level features. Padding allows us to keep the same spatial dimension before and after a Convolutional layer by padding the border of the image with enough zeros. For example, we take a 32 x 32 x 3 input volume, pads every border of it with two zeros, the input volume would then be 36 x 36 x 3. This new input volume has 1024 possible locations for a filter of size 5 x 5 which can be mapped to a 32 x 32 volume. We then just have to apply 3 filters of size 5 x 5 in order to obtain a 32 x 32 x 3 output volume. With this little trick we are able to apply as much Convolutional layer as we want without decreasing the spatial dimension.

2.9.3 The Upsampling layer

Since Convolutional layers without padding decrease the dimension of the input volume, it could be interesting to have a layer that increases the dimension of the input volume. This is what the Upsampling layer is made for. The Upsampling layer takes a vector named *size* as input, and repeats the rows and columns of the input volume by $size[0]$ and $size[1]$ respectively, see Figure 13.

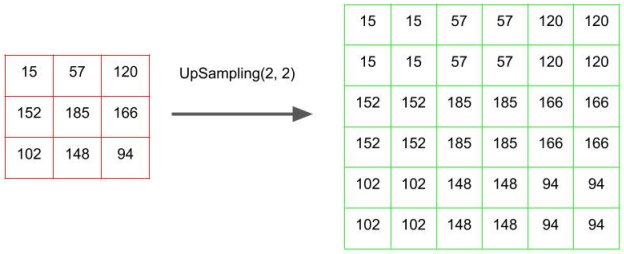

Figure 13: An Upsampling layer applied on an input volume 3 x 3 (with size[0] = 2 = size[1] = 2)

2.9.4 The residual layer

The residual layer - also called the skip layers -, takes a list of tensors in input and adds them up together. This layer could be seen as a bridge allowing the information to flow directly from one layer to another.

Figure 14: A Residual layer applied on 2 tensors

In practice, we generally use this layer to make a bridge between to Convolutional layers, which can be illustrated as follow:

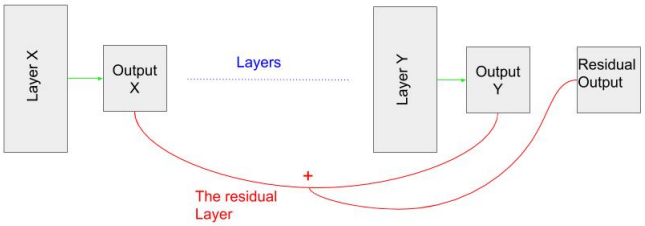

Figure 15: A Residual layer between two Convolutional layers, the residual layers is represented by red lines.

The output of the layer X and the layer Y must have the same dimensionality. The residual layer which is settled after the Y layer sums the two outputs together. This way, the information from X directly travels to the Y's output. The weights associated to the layers between X and Y will be influenced by this bridge. In the rest of the paper, a residual layer will be represented by a single line between two Convolutional layers.

2.9.5 The Batch Normalization layer

In machine learning, it is a good practice to normalize the input data in order to have each feature on the same scale. Normalizing avoids that one feature influences more the model than others. Since the input layer is benefiting from normalization, why do not we do it for every layer. It is what the batch normalization layer allows.

Technically batch normalization normalizes the output of a previous activation layer by subtracting the batch mean and dividing by the batch standard deviation. Thanks to this process, we reduce the phenomenon of internal covariate shift which allows us to use higher learning rates.

The phenomenon called covariate shift complicates the training of Deep Neural Networks by the fact that the distribution of each layer's input changes over time during training as the parameters of the previous layer change. Because of this problem we have to slow down the training by using low learning rates and choose carefully parameters initialization. For more information about Batch Normalization see [6] and [7].

2.9.6 Patch size

Image patch is a sort of pixels container. For example an image of 100 pixels high and 100 pixels wide (100 x 100) could be cropped into 100 patches of size 10 x 10.

Instead of giving the entire image as input to a CNN, we could split the image into patches and give those to the network as input. This way, we force the neural network to focus on one part at a time of the images instead of focusing on the whole picture.

See [31] and [32].

3 The proposed approach for denoising

Recently, deep neural networks (DNNs) have shown their superior performance in image processing and computer vision task. One of the early deep learning models which has been used for image denoising is the Stacked Denoising Auto-encoders. Denoising auto-encoders can be stacked to form a deep network by feeding output of the previous layer to the current layer as input. Since then different learning models have been used for image denoising up to nowadays where Convolutional Neural Networks (CNN) show their superior performance to denoise natural images.

3.1 Architectures used

As mentioned in the introduction, this work does not pretend to develop a new architecture. We have chosen to select 2 models of interest [1], [2]. The two architectures proposed below are a reproduction of these two selected models. They are as faithful as possible to the original ones. No cross validation where made for selecting the meta-parameters. Those used where the ones selected by the creators of these models. It is possible then that a better performance could have been achieved.

3.1.1 Convolutional Auto-encoders with Symmetric Skip Connections (CAeSSC)

This proposed architecture is directly inspired from the network proposed by Xiao-Jiao Mao, Chunhua Shen and Yu-Bin Yang [2]. This framework mainly contains a chain of 6 Convolutional layers and 6 symmetric Deconvolutional layers, shown in Figure 16, . Each of them used as activation function the Rectifier Linear Unit (ReLU). Every two layers, symmetric connections are made from Convolutional layer to Deconvolutional layer. No batch Normalization layers are used.

The framework is fully Convolutional and Deconvolutional. The down-sampling in Convolutional layers is made by using a stride = 2. Deconvolutional layers are an upsampling convolution (one Convolutional layer + one Upsampling layer). It is important to note that nor pooling nor unpooling are used in the network as pooling discards useful image details that are essential for denoising tasks. Each layer applied 64 filters of size 3 x 3, except the last one which uses only 1 filter in order to obtain the same output and input dimensionality.

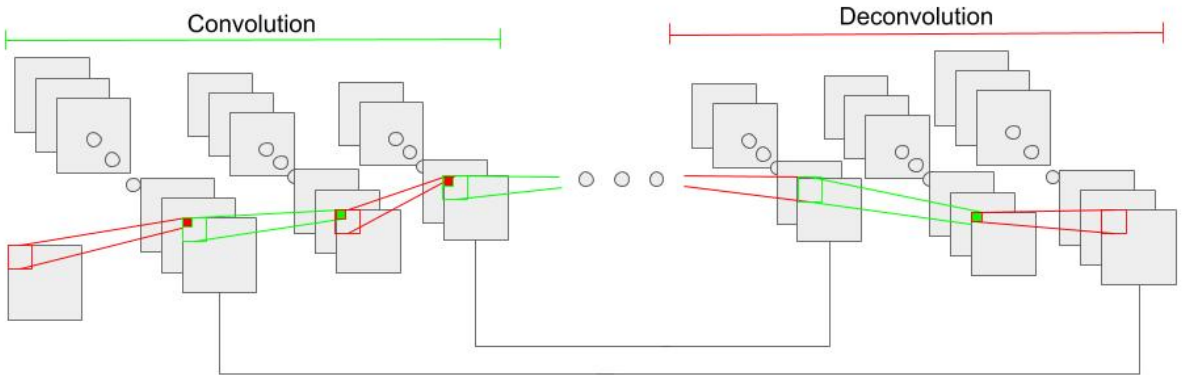

Figure 16: The architecture of this network. The network contains layers of symmetric convolution (encoder) and deconvolution (decoder). Skip shortcuts are connected every two layers (represented by black line between layers on this figure) from Convolutional feature maps to their mirrored Deconvolutional feature maps.

As shown by Matthew D. Zeiler and Rob Fergus [3] Convolutional layers act like features extractor. Each layer captures a little more useful information than the previous one. That is why combining more and more layers could be interesting in a Convolutional network. Indeed capturing more complex features allows

the network to understand more precisely what depicts the original image. Unfortunately, deep learning scientists agree on the fact that deeper network tends to easily suffer from performance loss, the reason is two folds. First, with more layers a significant amount of details could be lost, recovering a full image only from its abstraction is an under-determined problem. Second, deep networks generally suffer from gradients vanishing.

To counter these two problems, the network uses two major tools. First, as said before, after each Convolutional or Deconvolutional layer we add a rectification layer. The rectifier activation function allows a network to easily obtain sparse representations, see Section 2.8. Second, skip connections carry much image details, which helps Deconvolutional layer to reconstruct the images from its abstraction and it also achieves benefits on back-propagating the gradient to the first layer.

3.1.2 Normalized Convolutional Neural Network (NDNN)

This proposed architecture is directly inspired from the network proposed by Kai Zhang, Wangmeng Zuo, Yunjin Chen, Deyu Meng and Lei Zhang [1]. Creators of this network originally designed it to learn the residual mapping, meanings that the network tries to reconstruct the noise, which is the difference between the noisy observation and the latent clean image. In this thesis, I used this network not only for predicting the noise but also for predicting the denoised image.

As you can see on Figure 17, the first layer (CONV + ReLU) uses 64 filters of size 3 x 3 x c, c being the number of image channels, c = 1 for gray image and 3 for color image. These filters are used to generate 64 feature maps and then Rectifier Linear units are utilized for nonlinearity. The second layer up to the 19^{th} also uses 64 filters of size 3 x 3 combined with ReLU and batch Normalization. Batch Normalization speeds up the training and also boosts the denoising performance. The last layer uses only c filters in order to reconstruct the output which must have the same dimension as the input. It is important to note that we use padding at each layer to make sure that every feature map has the same dimension.

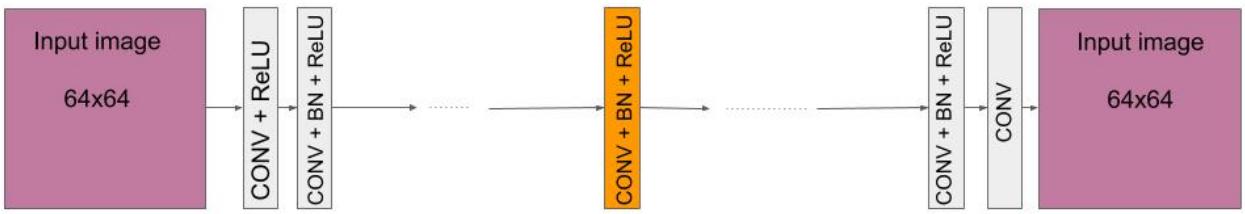

Figure 17: The architecture of this network. The network contains only Convolutional and Batch Normalized layers.

By combining convolution with ReLU, the network is able to gradually extract the structure of the image from its noisy observation through the hidden layers. It has been shown on the original paper [1], that residual learning and batch normalization can benefit from each other for Gaussian denoising, since both noise and batch normalization are associated with the Gaussian distribution. This claim is supported by the experiments made in this paper.

4 Validation and Methodology

Architectures presented in Section 3.1 are used to denoise images. In practice during the training phase, the neural network receives an image with an artificial noise input and tries to remove this noise from it. This artificial noise could be one of two kinds: a structured noise like a stain, a scratch or an unstructured noise (a Gaussian noise, for example). A Gaussian noise is a statistical noise having a probability density function equal to the Gaussian distribution. We can adapt the noise level of such a noise by changing the parameter σ. In this paper, a random noise level between 0 and 55 is used.
The probability density function p of Gaussian random variable z is given by:

$$p_G(z) = \frac{1}{\sigma\sqrt{2*\pi}} e^{-\frac{(z-\mu)^2}{2\sigma^2}}$$

We hope that by trying to denoise the input noisy image, the neural network will learn what is a noise and what is part of the inherent structure of the image. This way, the model should be able to recognize an unseen defect when it encounters one. More specifically, a defect that is not artificial (meaning a defect that have not been artificially added to the image).

4.1 The datasets

In this section, the different datasets, that are used, are presented. There are height datasets, three are pills and the five others are more complex textures.

There are three different classes of images for the pills: the ginseng pills, the magnesium pills and the mint pills. Each of these classes have two kinds of defaults that we want to detect: a contamination or a crack, as shown in Figure 18.

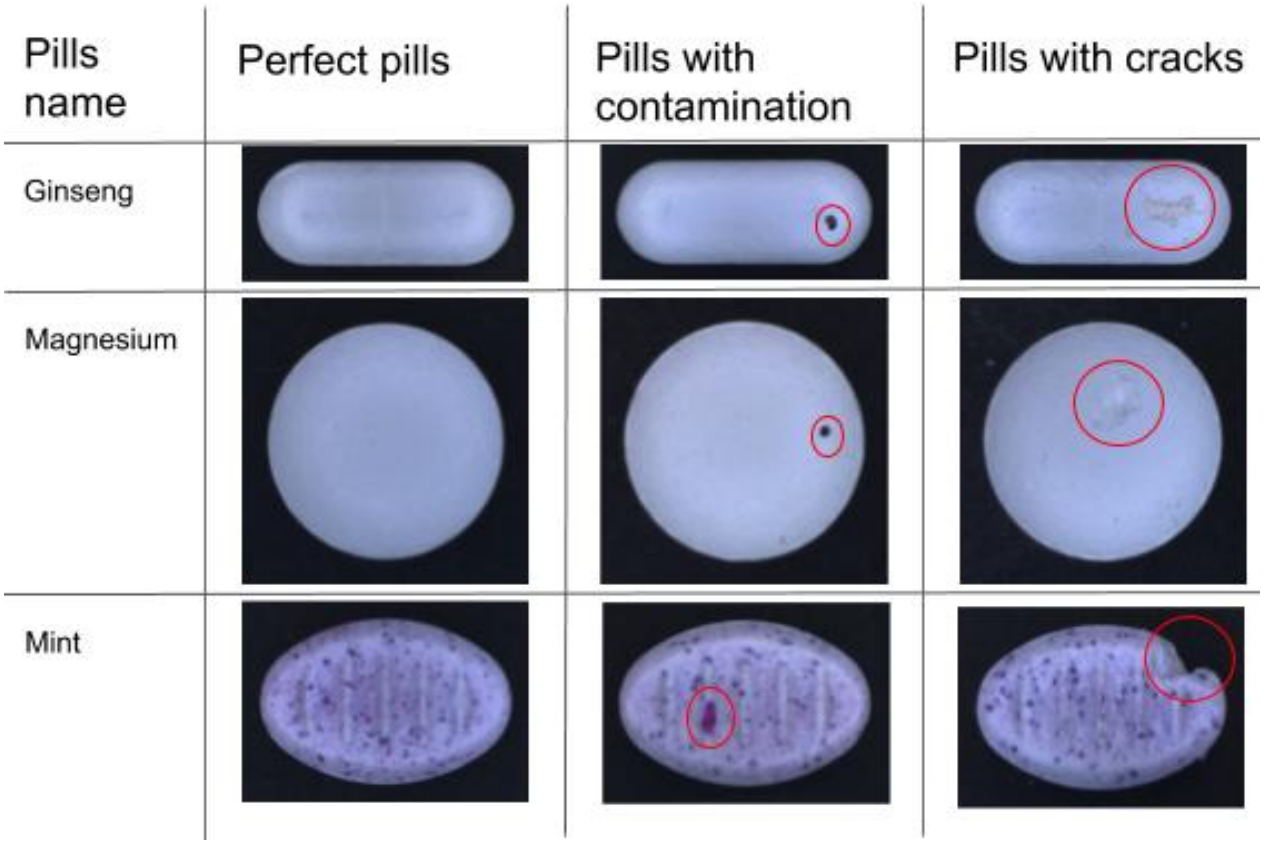

Figure 18: The 3 different pills classes

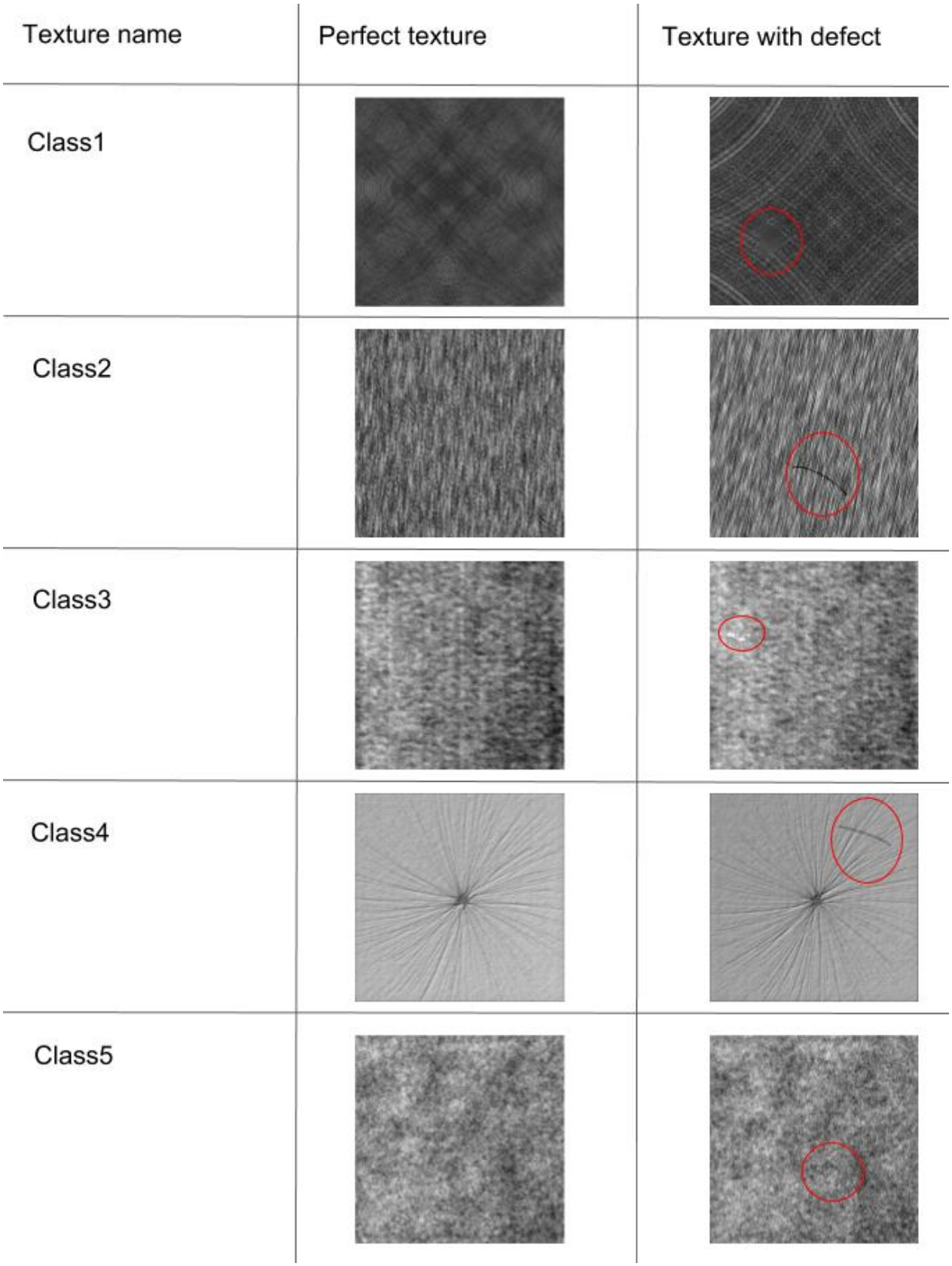

Figure 19: The five different texture classes

In practice each pills class is split into three datasets. For example, the Ginseng class is split into: Ginseng_good, Ginseng_contamination and Ginseng_cracks. Those 3 datasets contain respectively all the Ginseng pills flawless, contaminated and cracked respectively. The same principle is applied for the Magnesium and the Mint classes. We have then: Magnesium_good, Magnesium_contamination, Magnesium_cracks, Mint_good, Mint_contamination and Mint_cracks.

There are five different classes for the complex textures called Class1, Class2, Class3, Class4 and Class5. Each class has its own kind of defect and is then split into two datasets, shown in Figure 19. For example for the set of images called Class1 we have: Class1_good and Class1_defect.

4.2 CNN Training/Testing

4.2.1 Training

The training phase is when we run an algorithm to adapt the weights and biases of our network in order to find a correct mapping between the input and the output, as explained in Section 2.3. Since we want to do an unsupervised noise detection, we cannot do any guess about what would be the noise/the defect. We are not supposed to know what we are looking for.

What we do, is adding artificial noise on our clean datasets and ask our neural network to learn how to remove it. Two kinds of artificial noises have been added: a structured noise or an unstructured noise, shown in Figure 20.

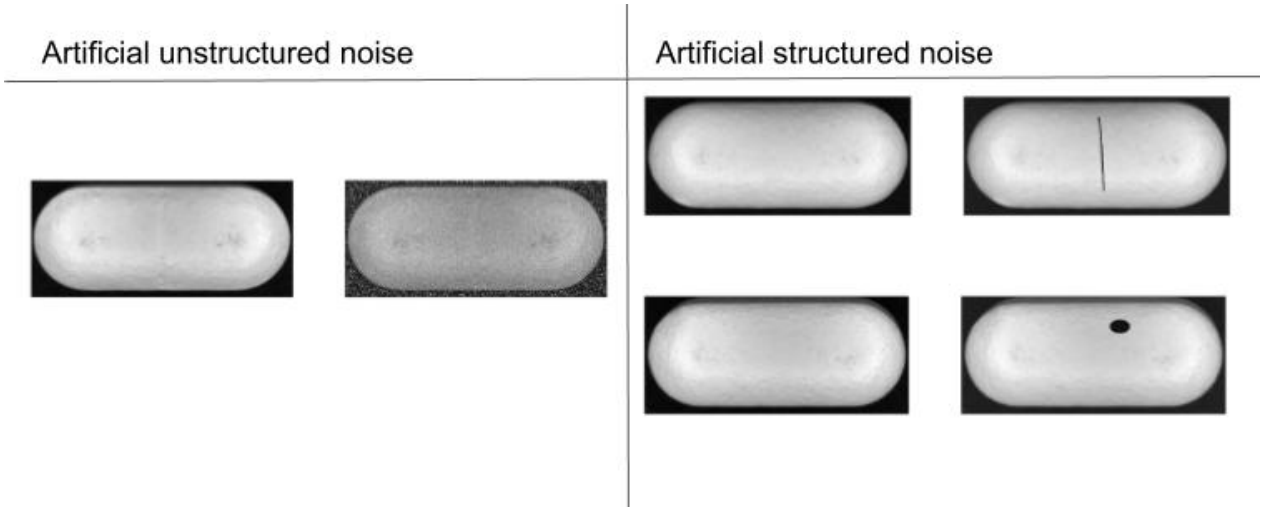

Figure 20: At the left the original image, at the right the original image with noise added.

For each dataset <name of the class>_good two copies are created. On these copies artificial noises are added. For example, the dataset Ginseng_good has two copies: Ginseng_noise and Ginseng_gaussian. Ginseng_noise are ginseng pills with artificial structured noise added. Ginseng_gaussian are ginseng pills with artificial unstructured noise. The artificial unstructured noise is a Gaussian noise with a random noise level between [0, 55].

During the training phase, our CNNs receive as input a noisy image and are asked to recover its denoised version or to isolate the noise. The CNNs try to find a mapping between an image and its targeted output. If we have x a clean image, and if our network is in denoised mode, it will try to learn a mapping such as $f(y) = x$ where $y = x + v$, v being an added noise. But if it is in residual mode, it will try to learn a mapping such as $R(y) = v$, shown in Figure 21.

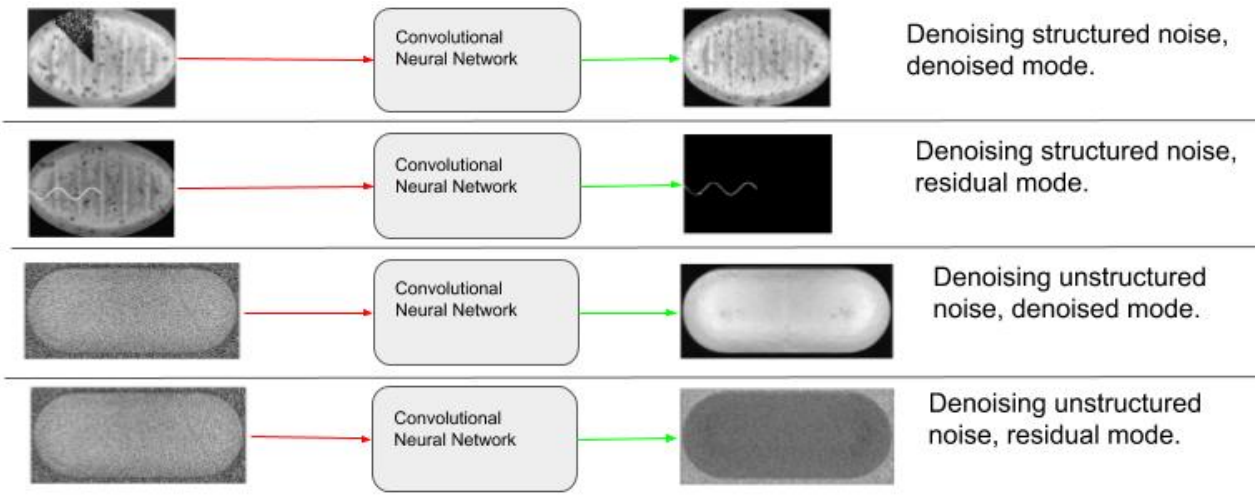

Figure 21: The CNN tries to find a mapping between the input (at the left) and its target (at the right).

Our two architectures are trained in four different ways for each class of images. For example for the class called "Class4":

Input		CNN model		output	mode
Class4_noise	⇒	CNN	⇒	Class4_good	Denoised mode
Class4_noise	⇒	CNN	⇒	Class4_noise - Class4_good	Residual mode
Class4_gaussian	⇒	CNN	⇒	Class4_good	Denoised mode
Class4_gaussian	⇒	CNN	⇒	Class4_gaussian - Class4_good	Residual mode

Figure 22: The six different training done for the five classes of texture images

4.2.2 Training with patches

When patches are used the CNN, do not focus on the whole image. Instead, it tries to find a mapping between patches given as input and those given as target. For example, imagine that in Figure 21, images on the right and on the left are cropped into smaller pictures. As said before in section 2.9.6, these smaller pictures are called patches. Therefore patches on the right will have a corresponding patch on the left. The CNN will try to find a mapping between right patches and their corresponding left patches.

If the defect is small compared to the image size, this trick should help the CNN to detect it. It is more or less like if you were using two magnifying glasses as a tool to compare the right images with the left ones. Moving both of them up and down and from the right to the left. Comparing right and left images pieces by pieces.

4.2.3 Training analysis

In all figures that follow, solid lines are learning rates obtained with residual learning and broken line are learning rates obtained with denoised learning. These figures represent the Peak Signal-to-Noise Ratio (PSNR) improvement through epochs. This ratio is often used as a quality measurement between the original and a compressed image. The higher the PSNR is, the better is the quality of the compressed, or reconstructed image. In the following equations, M and N are the number of rows and columns in the input images, respectively. R is the maximum fluctuation in the input image data type.

$$MSE = \frac{\sum_{M,N}(I_1(m,n)-I_2(m,n))^2}{M*N}$$
$$PSNR = 10log_{10}(\frac{R^2}{MSE})$$

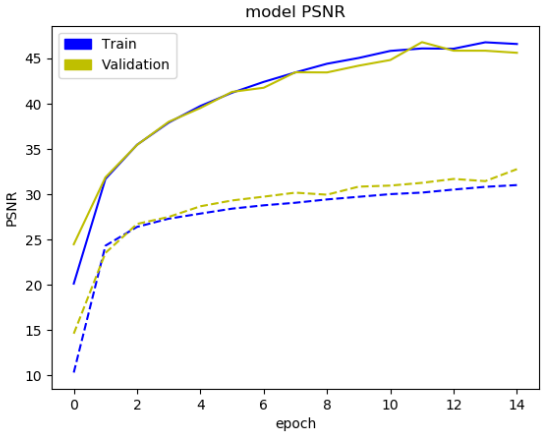

Figure 23: NDNN training history over images with unstructured noise.
Input: Class3_gaussian.

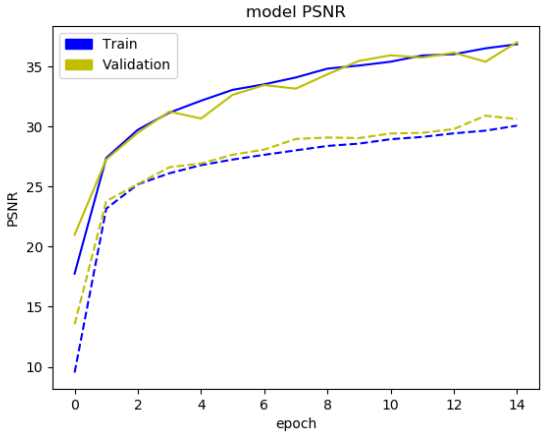

Figure 24: NDNN training history over images with structured noise.
Input: Class3_noise.

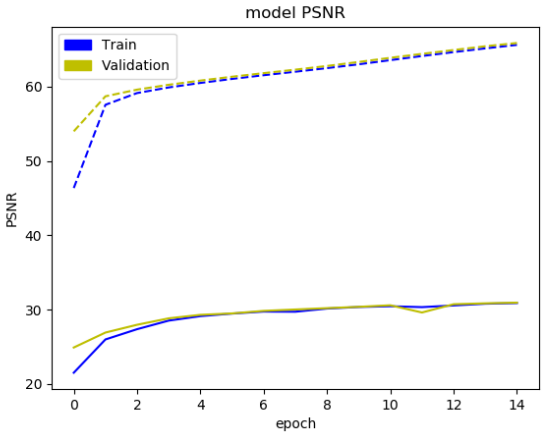

Figure 25: CAeSSC training history over images with unstructured noise.
Input: Class3_gaussian.

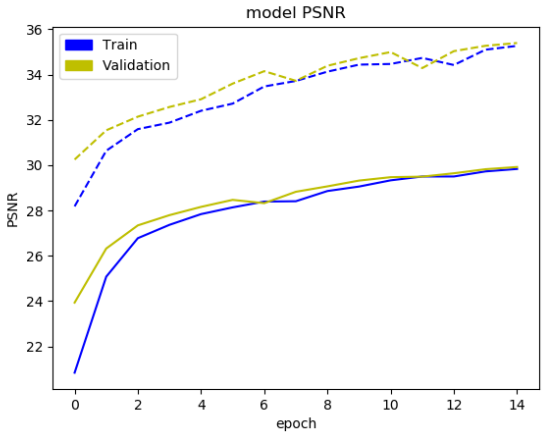

Figure 26: CAeSSC training history over images with structured noise.
Input: Class3_noise.

Figures 112 and 113 show that the NDNN architecture converges towards a higher PSNR value with a training in residual mode. And figures 114 and 115 show the inverse tendency for the CAeSSC model; the convergence reach a higher PSNR value when the training is in denoised mode. These examples of training are made over the class "Class3". Every texture class has similar training histories. We are then expecting better results, for them, regarding the NDNN model with a training in residual mode. On the contrary, better performance are expected with a denoised learning for the CAeSSC model.

Training histories for the pills classes are also all alike. We can see in Figures 100 and 101 that the NDNN model does not have any preferences between the residual or the denoised learning. On the other hand, the CAeSSC architecture is well suited for the denoised learning, shown in Figures 102 and 103. We are then expecting, for the pills classes similar performances with the NDNN model in both training mode. And the CAeSSC architecture should perform better after a training in denoised mode.

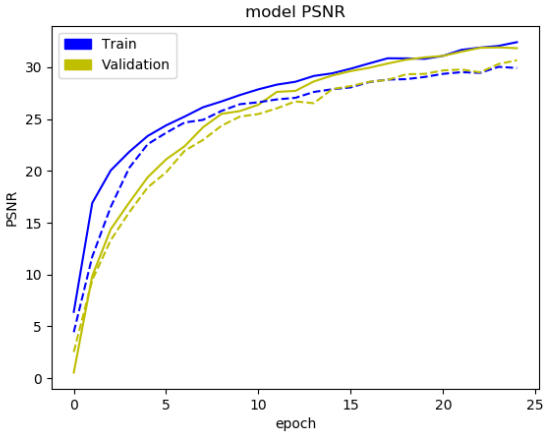

Figure 27: NDNN training history over images with unstructured noise.
Input: Mint_gaussian.

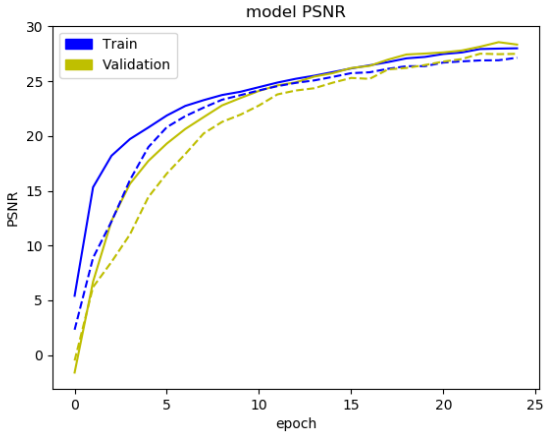

Figure 28: NDNN training history over images with structured noise.
Input: Mint_noise.

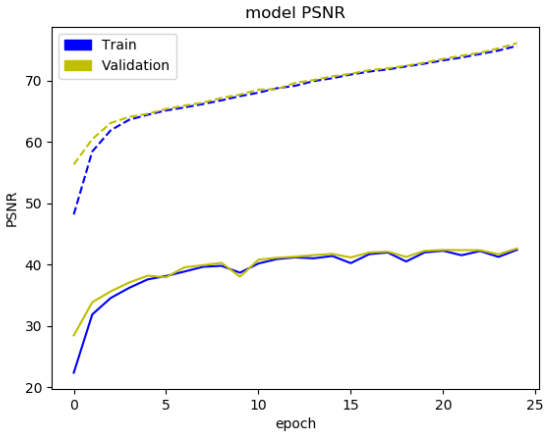

Figure 29: CAeSSC training history over images with unstructured noise.
Input: Mint_gaussian.

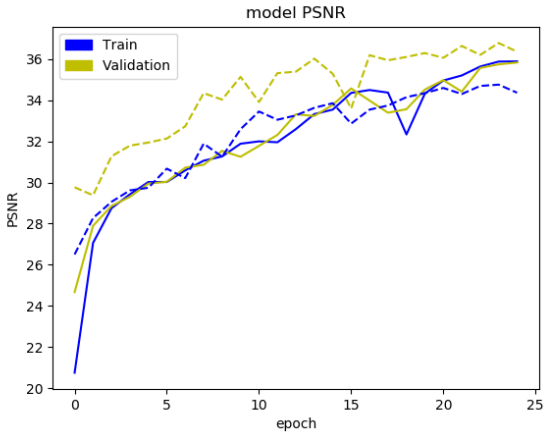

Figure 30: CAeSSC training history over images with structured noise.
Input: Mint_noise.

Training histories of other classes can be found in appendice A.

Another major difference between the training of the NDNN and the CAeSSC model is the learning speed. Since the CAeSSC model do a down-sampling of the input followed by an up-sampling, it learns way more faster than the NDNN model. For example, for the class "Class2" the CAeSSC architecture complete an epoch over 42000 patches of size 64 x 64 in about 15 seconds. While the NDNN model complete an epoch over these same patches in about 160 seconds (ten times slower!).

4.2.4 Testing

At the beginning of the training phase, an eighth of each <class name>_good has been put aside. These eighths have never been seen by our two CNNs, we call them <class name>_good_test. For the testing phase, random datasets are created for every classes of images. They contain a random number of images without defect (clean images) and a random number of images with defect (noisy images). These random datasets

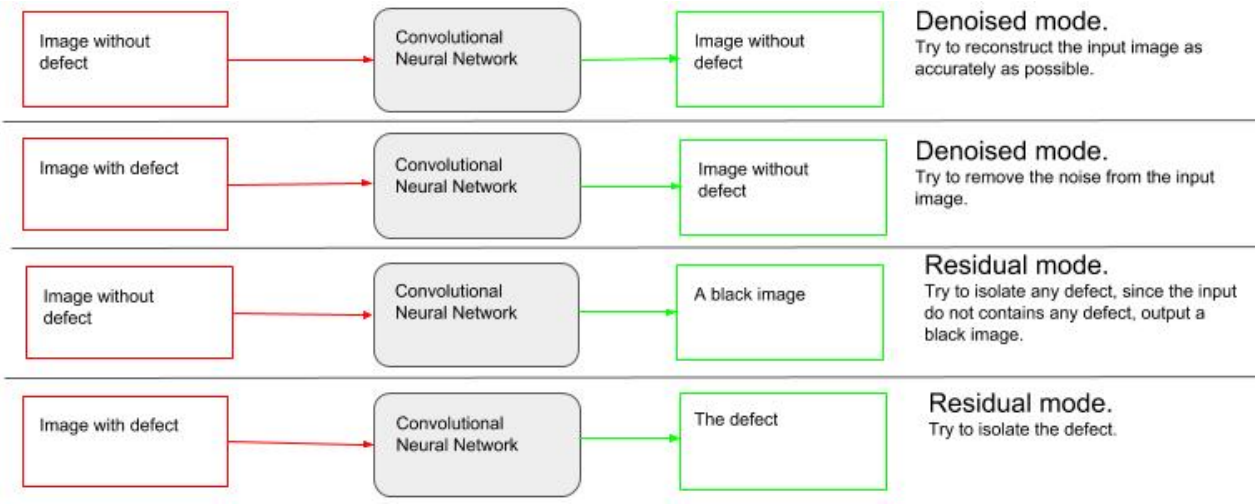

Figure 31: The tasks that we hope ours CNNS will achieve

have a percentage of clean images between [27% - 83%] and a percentage of noisy images between [17% - 73%].

For example, for the class ginseng two random test datasets are created: one with images pick up into Ginseng_good_test and Ginseng_cont (we call this new random dataset Ginseng_cont_test), and another with image pick up into Ginseng_good_test and Ginseng_crack (Ginseng_crack_test). These two random datasets contain images never seen by any of our models. These bunches of unseen images are then a mix of clean images and real noisy images (not artificial noisy images). They are later given to our CNNs as input and we hope that they will be able to complete at least one of the tasks described in Figure 31. Since classes of texture images only have their own kind of defect, we only have one random test dataset per class: Class1_test, Class2_test, Class3_test, Class4_test and Class5_test.

The next step is to decide if an input image actually contains a defect or not. It is indeed the true aim of this paper, trying to create an universal denoiser in order to decide at the end whether the input image is noisy. This is done by comparing the input image with its corresponding output image. On the one hand, if the input image contains a defect there should be some difference between the output and the input since our CNNs try to remove the noise from the input image. On the other hand, if the input image is not noisy the output image should be almost exactly the same as the input image. This is what is done in the next section, trying to score every couple of input and output in order to conclude which input image is noisy.

4.2.5 Anomaly detection

As explained at the end of the previous section, this section focus on scoring every couple of input and output image in order to establish whether the input originally contains a defect. Four different scoring functions are used in this aim : the Outliers score, the MAE (mean absolute error) score, the MSE (mean squared error) score and the MAX score. Each of these scores compare the input image with its corresponding output. The input and the output images should be similar and have a score close to zero, when the input image does not contain any defect. On the contrary, when the input is noisy its corresponding output should not contain any noise. The "bigger" the noise is, higher the score assign to this couple should be.

Here is a table resuming what each score function does, with X_{ij} being the pixel at position (i, j) in the input image, Y_{ij} being the pixel at position (i, j) in the output image, W the width of the image and H the height of the image.

$$\bar{err} = \frac{\sum_{i=0}^{H} \sum_{j=0}^{W} |X_{ij} - Y_{ij}|}{H * W}$$

$$sd = \sqrt{\frac{\sum_{i=0}^{H}\sum_{j=0}^{W}(|X_{ij}-Y_{ij}|-e\bar{r}r)^2}{(H*W)-1}}$$

Score function name	Mathematical function
Outliers	Score = Score + 1, $\forall \lvert X_{ij} - Y_{ij}\rvert \geq 2 * sd$
MAE	$e\bar{r}r$
MSE	$\frac{\sum_{i=0}^{H}\sum_{j=0}^{W}(X_{ij}-Y_{ij})^2}{H*W}$
MAX	$Max(\lvert X_{ij} - Y_{ij}\rvert), \forall i, j$

Figure 32: The four different score functions

If we compute these scores for every couple (input, output) that are tested, we can sort these couples in descending order. Then we end up with something like Figure 33. Then we have to choose a threshold for every score functions. The threshold decides from which score the image becomes clean. It gives us the ability to split the noisy couples from the clean ones. The problem is that the threshold will change for every score function and for every class of images, since the order of magnitude of the scores will change. For example, the dataset Ginseng_cont_test and the dataset Class4_test will not have the same threshold for a same score function. We need to find a way to choose the correct threshold for every score functions and for every class of images. The first thing that we can do is plotting the Receiver Operating Characteristic curve (ROC curve in short) for each of them. An example of such a curve is shown in Figure 34.

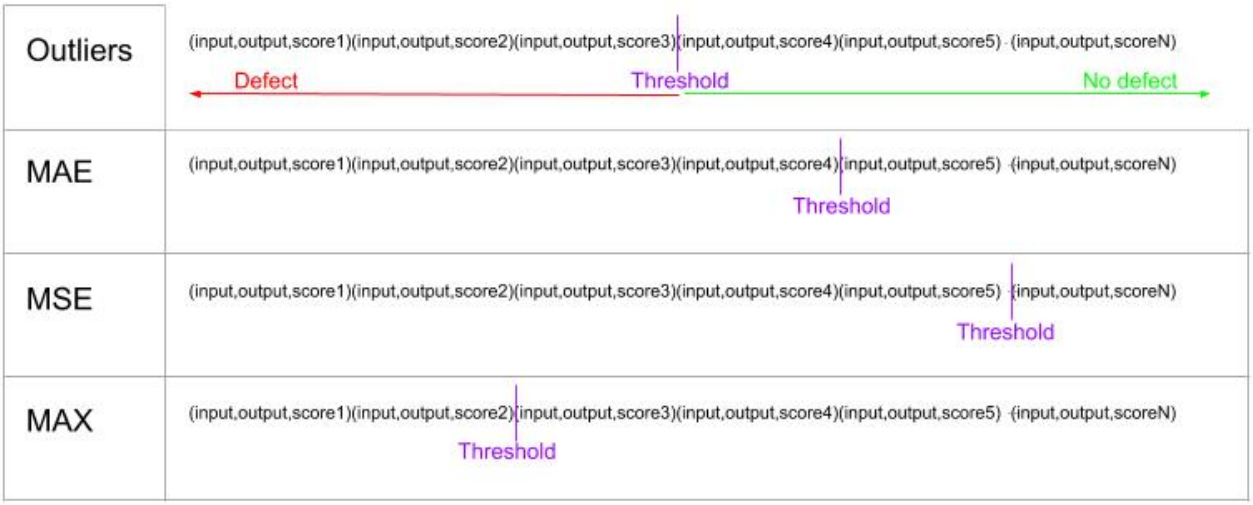

Figure 33: Scoring each couple, with score1 > score 2 > ... > scoreN.

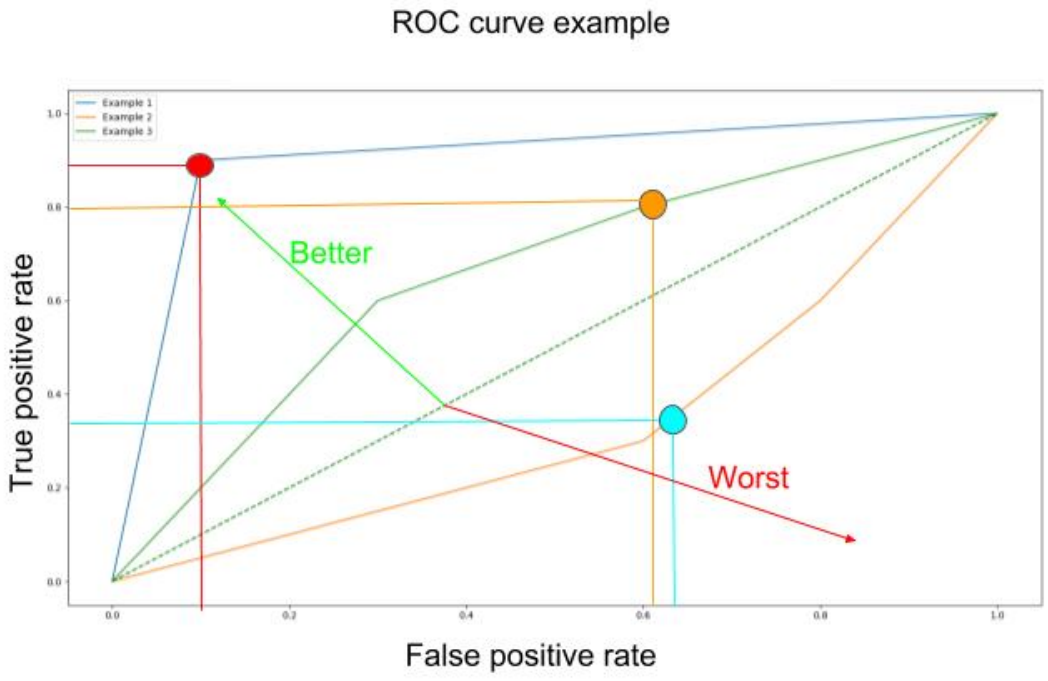

Figure 34: Roc curve example, the more the hollow of the curve moves toward the green arrow, the better the curve is. The more it move toward the red arrow, the worse it is. The three different curves correspond to a different score function. In this example the blue curve gives the better result. The broken line represents the performances of the random classifier, which labels each image randomly as clean or noisy.

A ROC curve plots the true positive rate (Sensitivity) in function of the false positive rate (Specificity). Each point in a ROC curve represents a Sensitivity/Specificity pair corresponding to a particular decision threshold. For example, the red point in Figure 34 corresponds to a Sensitivity of 0.9, for a Specificity of 0.1. This means that for a threshold at that point 90% of the noisy images are correctly labeled as noisy and 10% of the clean images are wrongly labeled as noisy. On the contrary, for the blue point only 35% of the noisy images are correctly labeled as noisy and 65% of clean images are wrongly labeled as noisy. In other words, closer the curve's hollow is to the upper left corner better is the curve. Therefore a ROC curve is effectively a good way to find the best threshold. All we need to do is recording the threshold at the best location of every curve. The curve that has a threshold that gives the highest Sensitivity with the lowest Specificity is the best. Its corresponding score function will then be selected among the others.

5 Results and Discussions

5.1 Visual results

Some visual results are shown in Figures below. These examples are the most visually meaningful. Each of them depicts three images: the left-most image is the input image, the middle image is the output image, and the right-most image is the input image minus the output image. Depending on the training mode, the middle image represents either the defect (residual mode) or the denoised image (denoised mode), and it is the contrary for the right-most image.

In Figures 35, 37, 38, 43, 44 and 45, which are all the examples trained in residual mode, the middle image highlights the defect. Inversely, the right-most image depicts the denoised image, the defect tends to be erased from this image.

The attentive reader will have noticed that in Figure 45 the middle image is filled of squares. This is because for this class of images significant better results are obtained when patches of size 64 x 64 are used. During the training phase, the input and targeted images, which have a size of 512 x 512, are cropped into 64 x 64 images. The architecture learns then a mapping between these input sub-images and their corresponding targeted sub-images. For the test phase the 512 x 512 input image is also cropped into 64 x 64 images, the model makes a prediction for each of these cropped images. We later use these predictions to reconstruct the output. This is why you see squares of size 64 x 64.

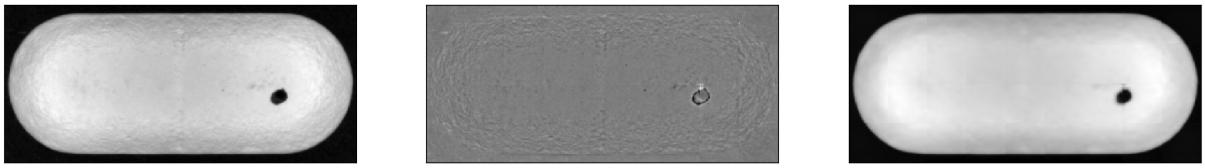

Figure 35: Visual result for contaminated Ginseng pills, this result is obtained after a residual training.

Figure 36: Visual result for cracked Ginseng pills, this result is obtained after a denoised training.

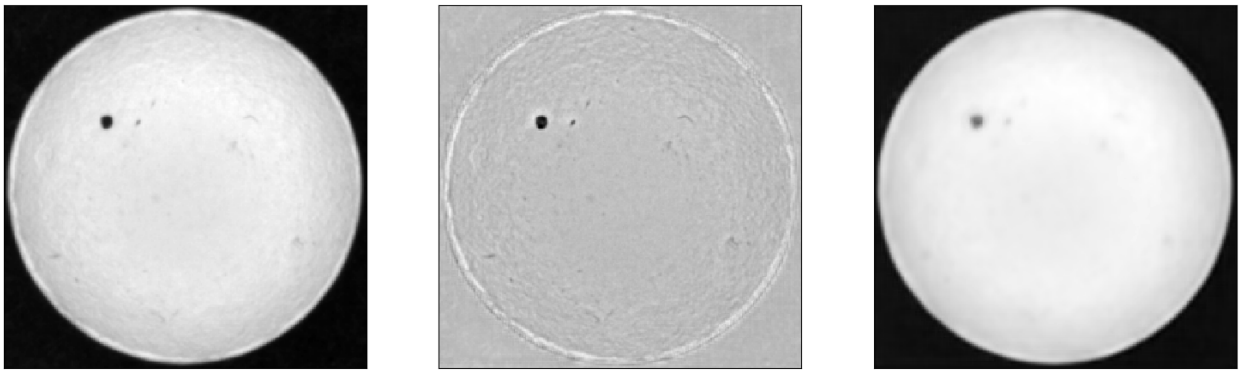

Figure 37: Visual result for contaminated Magnesium pills, this result is obtained after a residual training.

Figure 38: Visual result for cracked Magnesium pills, this result is obtained after a residual training.

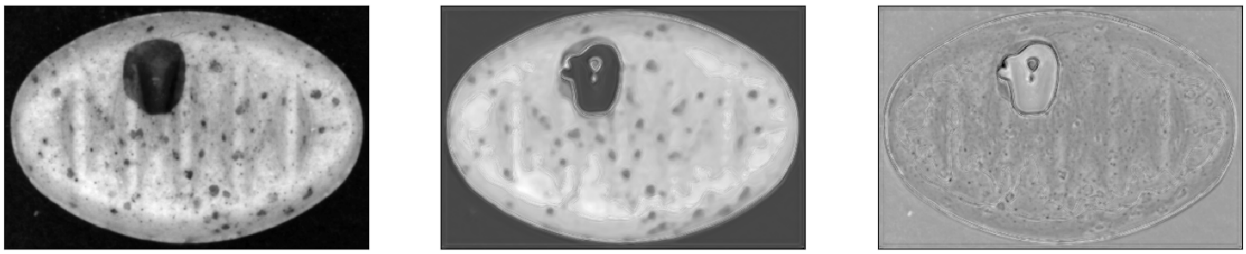

Figure 39: Visual result for contaminated Mint pills, this result is obtained after a denoised training.

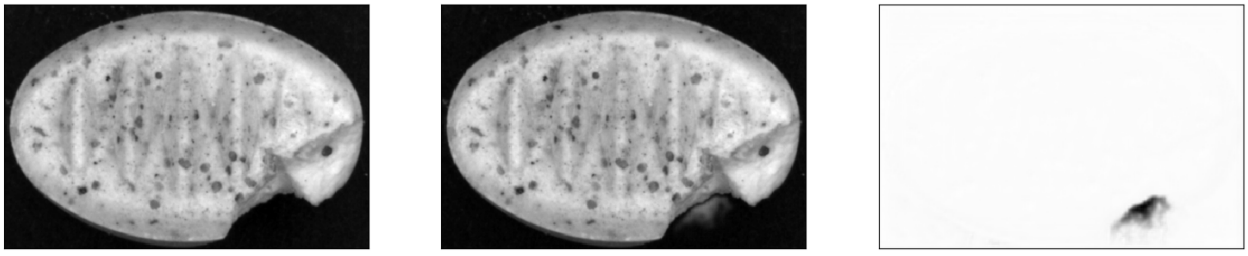

Figure 40: Visual result for cracked Mint pills, this result is obtained after a denoised training.

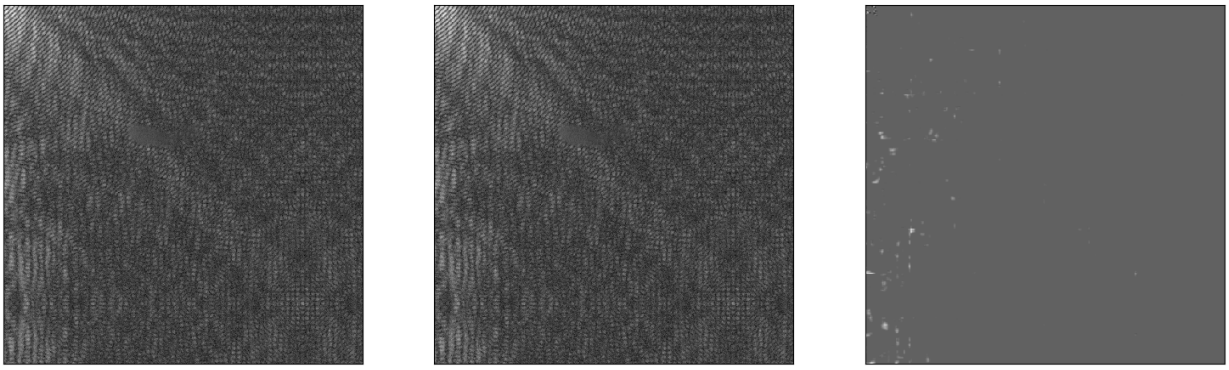

Figure 41: Visual result for Class1 textures, this result is obtained after a denoised training.

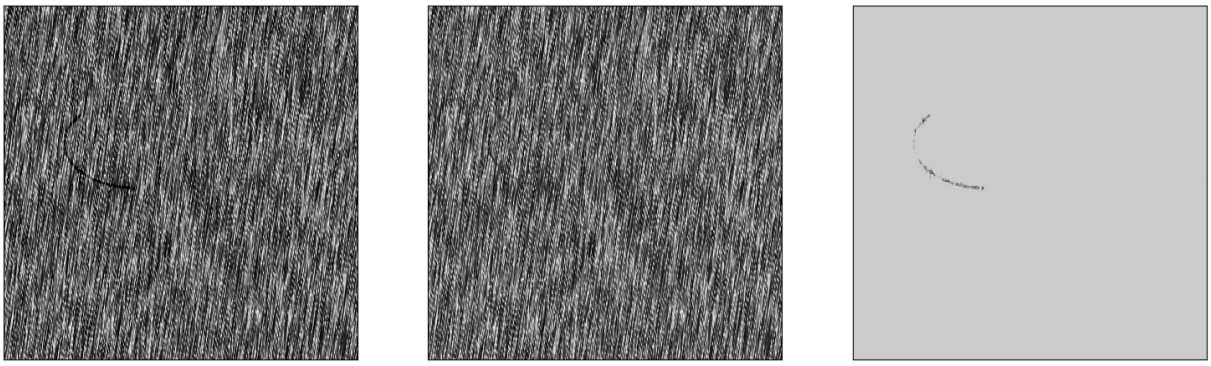

Figure 42: Visual result for Class2 textures, this result is obtained after a denoised training.

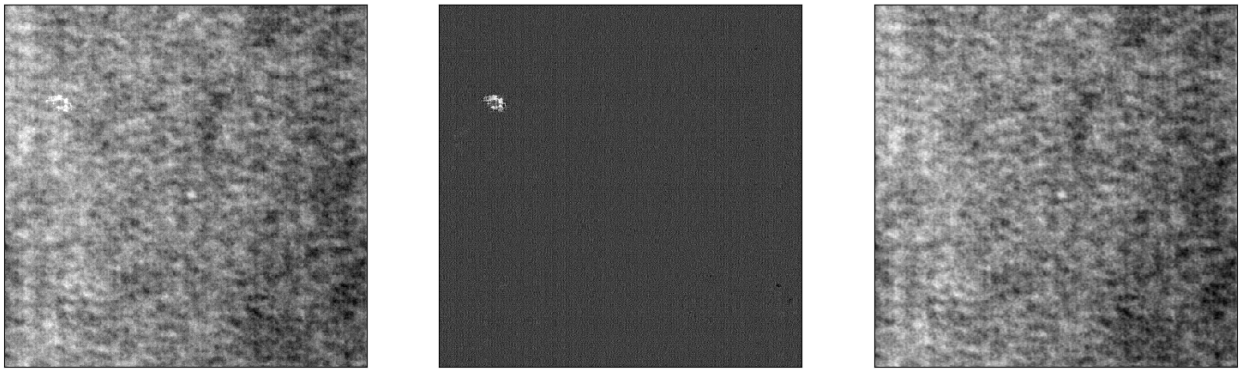

Figure 43: Visual result for Class3 textures, this result is obtained after a residual training.

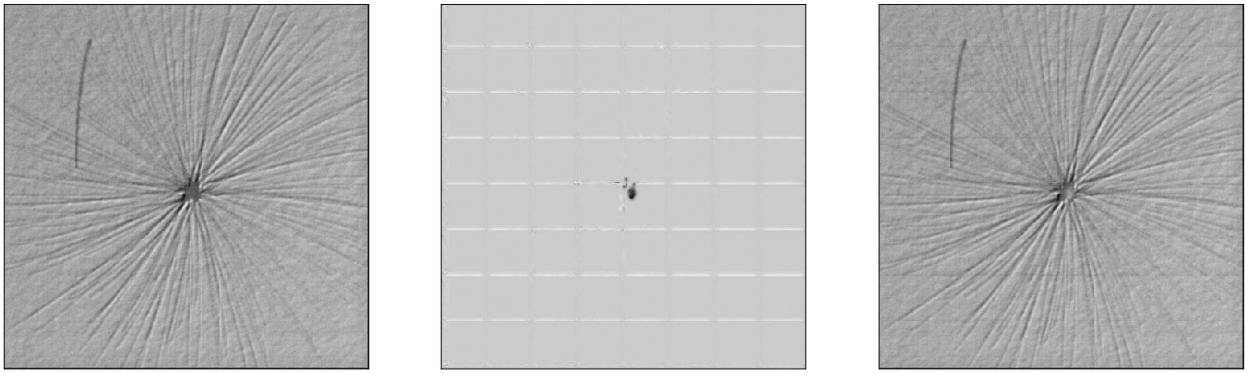

Figure 44: Visual result for Class4 textures, this result is obtained after a residual training.

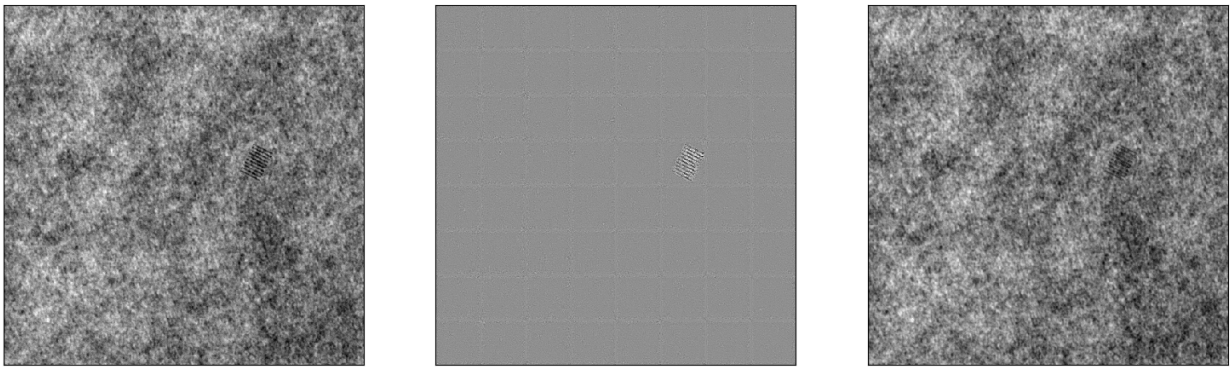

Figure 45: Visual result for Class5 textures, this result is obtained after a residual training.

Figures 36, 39, 40, 41 and 42 are all the examples trained in denoised mode. In these examples the middle image is the denoised image and the right-most image isolates the defect. The Figure 42 is probably the one that is the most visually meaningful.

As shown further in the paper, good classification performances are obtained for every class of images. Meaning that images which are given to the models during the test phase are most of the time correctly labeled as noisy or clean. Except for two classes: Class1 and Class4. And it is comprehensible considring that we can visually observe that models do not make a good denoising job at all. Figures 41 and 44 show that the defect is neither isolated nor erased.

5.2 Classification performances

Classification performances are evaluated in terms of ROC curve. It is usefull to recall that a point at the top left means that 100% of noisy images are correctly labeled as noisy. Conversely, a point at bottom right means that 100% of clean images are incorrectly classified as noisy.

In all Figures that follow solid lines are results obtained with the CAeSSC architecture and broken lines are performances obtained with the NDNN architecture. We can recall that the True Positive Rate is the percentage of noisy images that are correctly labeled as noisy. The False Positive Rate is the percentage of clean images that are incorrectly labeled as noisy.

5.2.1 Contaminated Ginseng pills

Figures 46, 47, 48 and 49 show classification results over the contaminated Ginseng pills with the different architectures and the different training modes. Good results are obtained with all configurations. Nevertheless, the training over structured noises seems to deliver slightly better results and especially for the CAeSSC model. The residual training does not give any improvement on that class of image. The anomaly detection on that class of images is able to achieve a True Positive Rate of 100% for a False Positive Rate of 0%.

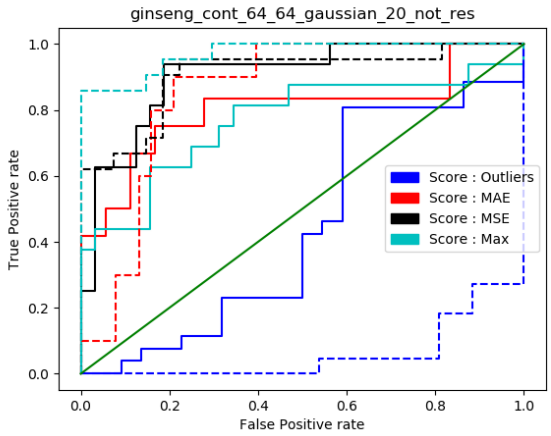

Figure 46: Performance obtained for both architectures trained in denoised mode over Ginseng pills with unstructured noise.

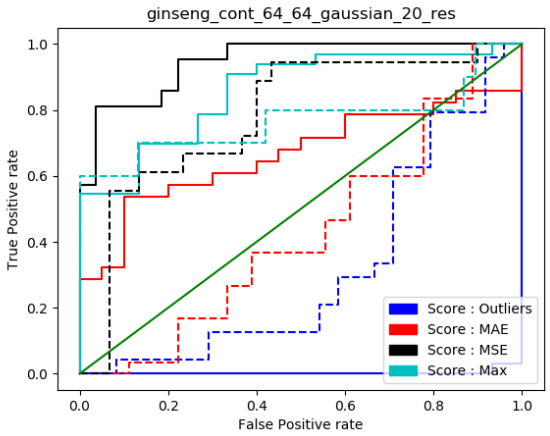

Figure 47: Performance obtained for both architectures trained in residual mode over Ginseng pills with unstructured noise.

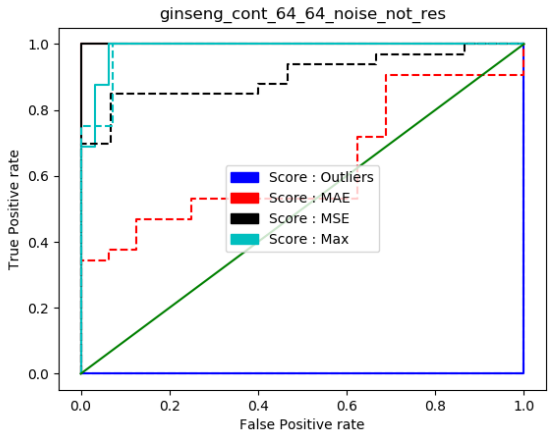

Figure 48: Performance obtained for both architectures trained in denoised mode over Ginseng pills with structured noise.

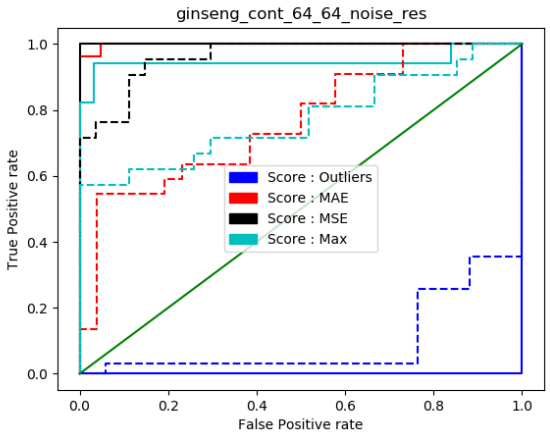

Figure 49: Performance obtained for both architectures trained in residual mode over Ginseng pills with structured noise.

5.2.2 Cracked Ginseng pills

The results obtained for the cracked Ginseng pills support the observation made on the contaminated Ginseng pills. The training over structured noises seems to give slightly better performance, but this time it is the NDNN architecture that benefits the more from it. This model is able to obtain a True Positive Rate of 100% for a False Positive Rate of 0%, shown in Figure 52.

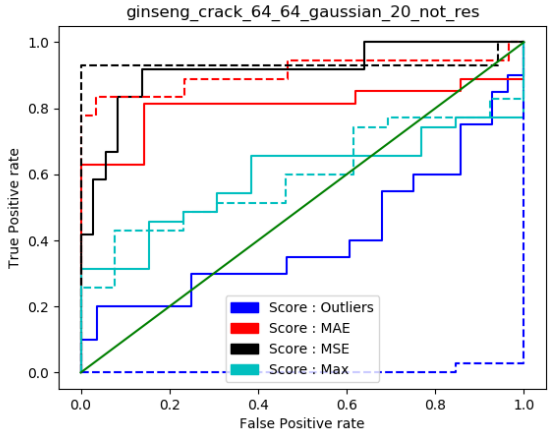

Figure 50: Performance obtained for both architectures trained in denoised mode over Ginseng pills with unstructured noise.

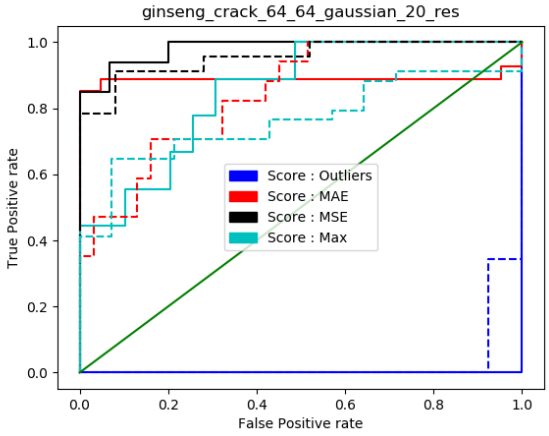

Figure 51: Performance obtained for both architectures trained in residual mode over Ginseng pills with unstructured noise.

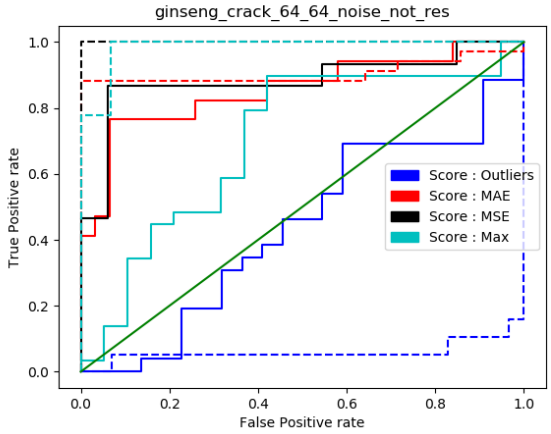

Figure 52: Performance obtained for both architectures trained in denoised mode over Ginseng pills with structured noise.

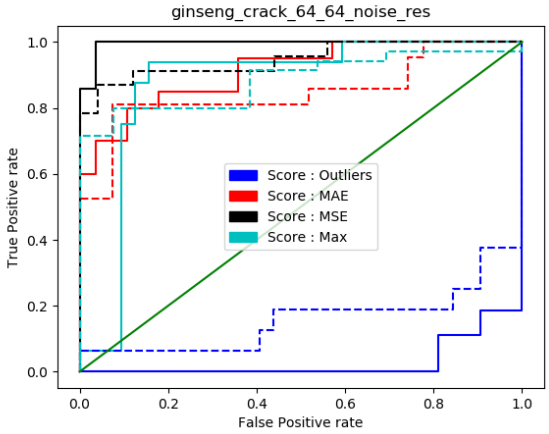

Figure 53: Performance obtained for both architectures trained in residual mode over Ginseng pills with structured noise.

5.2.3 Contaminated Mint pills

On that class of images the residual learning over structured noise induces better performance. This learning method seems also to be more appropriate for the NDNN model, in contrast with CAeSSC which seems to be penalized by it. Surprisingly Figure 55 shows that the residual learning over unstructured noise gives really bad performances though this same kind of learning gives good results on other kinds of pills.Again, anomaly detection achieves a True Positive Rate of 100% for a False Positive Rate of 0%, shown in Figure 57.

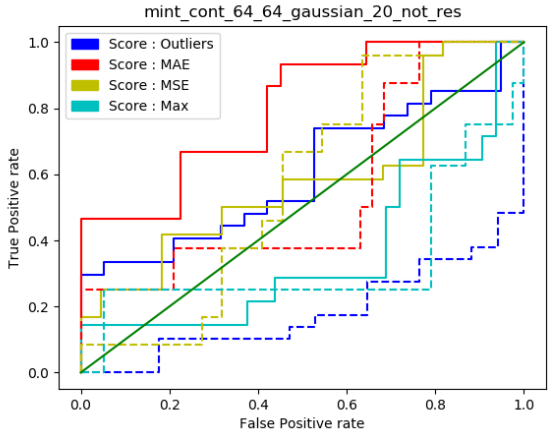

Figure 54: Performance obtained for both architectures trained in denoised mode over Mint pills with unstructured noise.

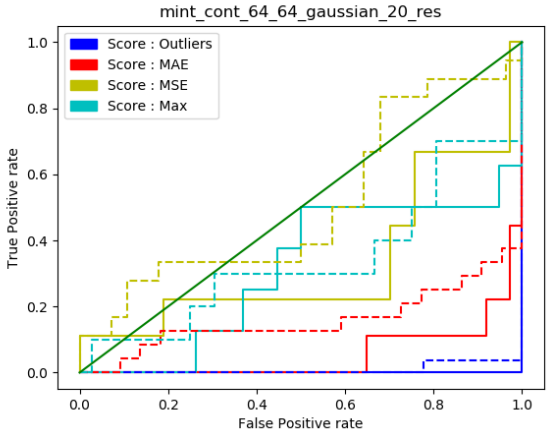

Figure 55: Performance obtained for both architectures trained in residual mode over Mint pills with unstructured noise.

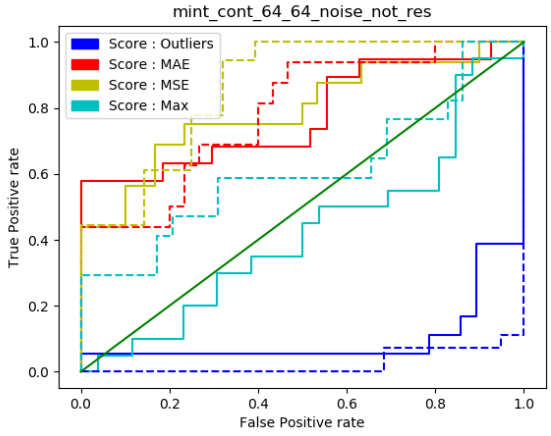

Figure 56: Performance obtained for both architectures trained in denoised mode over Mint pills with structured noise.

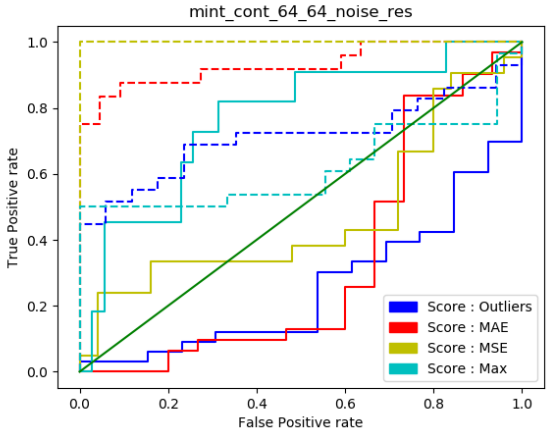

Figure 57: Performance obtained for both architectures trained in residual mode over Mint pills with structured noise.

5.2.4 Cracked Mint pills

For this dataset, the learning over structured noise seems more appropriate. We also observe that the residual learning benefits the NDNN architecture but penalizes the CAeSSC model. Nevertheless, every configuration has at least one score function giving really good performances.

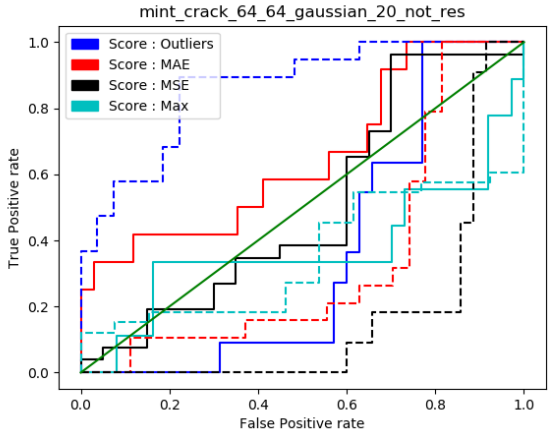

Figure 58: Performance obtained for both architectures trained in denoised mode over Mint pills with unstructured noise.

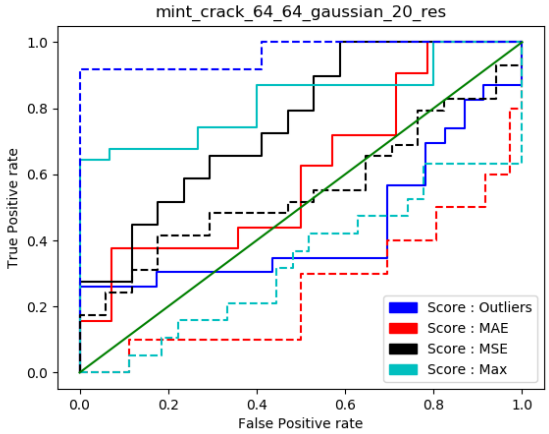

Figure 59: Performance obtained for both architectures trained in residual mode over Mint pills with unstructured noise.

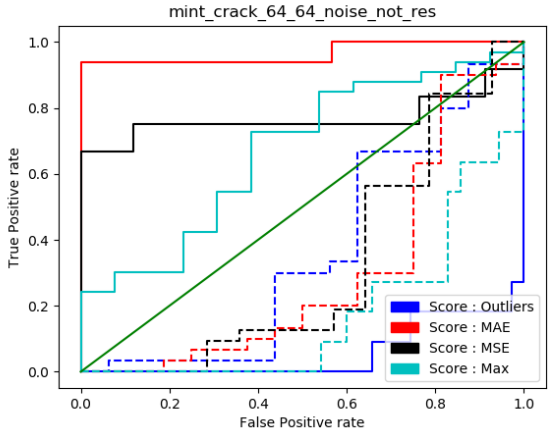

Figure 60: Performance obtained for both architectures trained in denoised mode over Mint pills with structured noise.

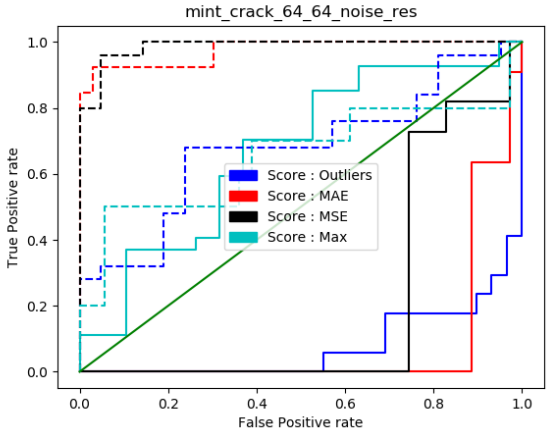

Figure 61: Performance obtained for both architectures trained in residual mode over Mint pills with structured noise.

5.2.5 Contaminated Magnesium pills

Good classification performances are obtained with every configuration. In contrast with the observation made on the Mint pills, the residual learning gains the CAeSSC model and penalizes the NDNN model. The next section supports this observation.

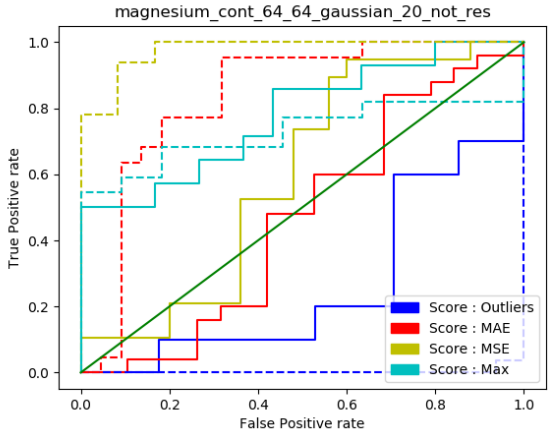

Figure 62: Performance obtained for both architectures trained in denoised mode over Magnesium pills with unstructured noise.

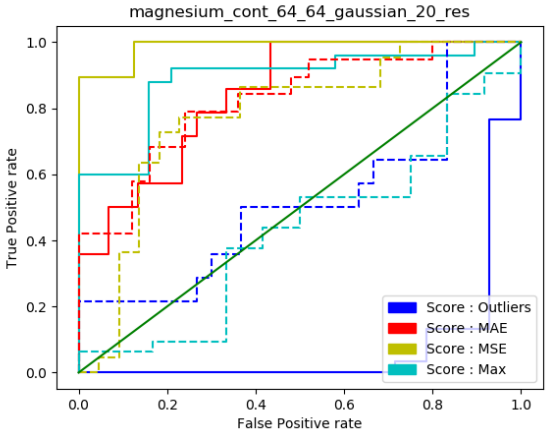

Figure 63: Performance obtained for both architectures trained in residual mode over Magnesium pills with unstructured noise.

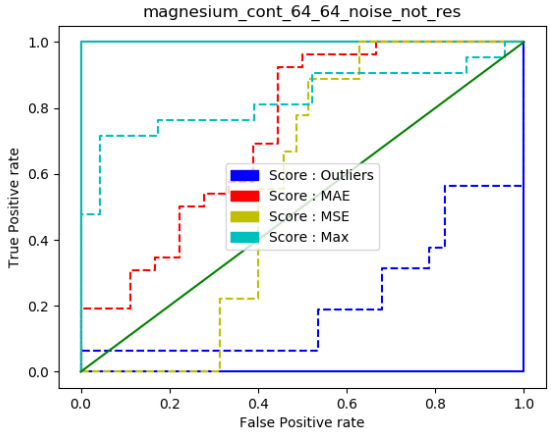

Figure 64: Performance obtained for both architectures trained in denoised mode over Magnesium pills with structured noise.

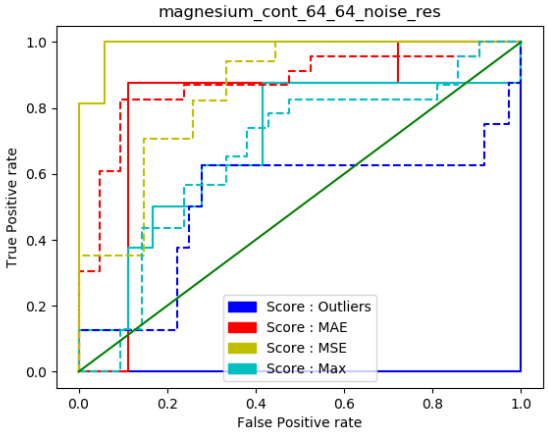

Figure 65: Performance obtained for both architectures trained in residual mode over Magnesium pills with structured noise.

5.2.6 Cracked Magnesium pills

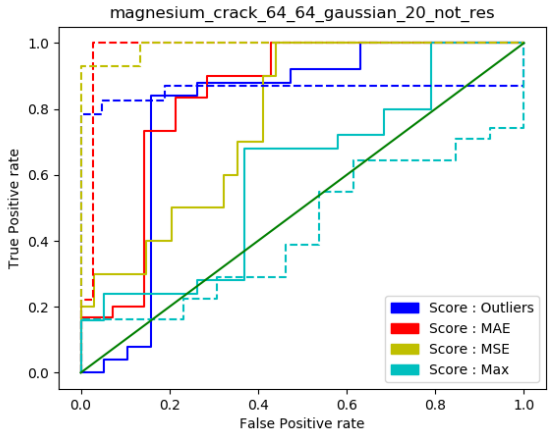

Figure 66: Performance obtained for both architectures trained in denoised mode over Magnesium pills with unstructured noise.

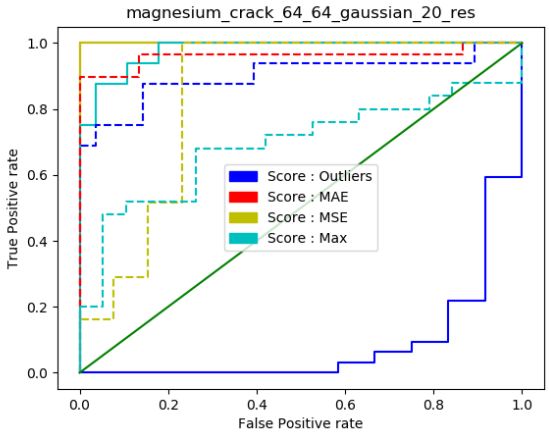

Figure 67: Performance obtained for both architectures trained in residual mode over Magnesium pills with unstructured noise.

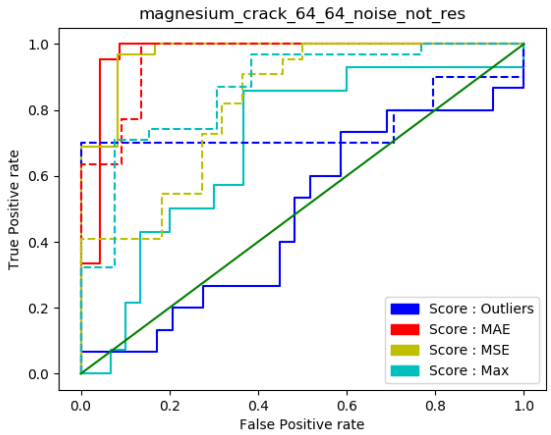

Figure 68: Performance obtained for both architectures trained in denoised mode over Magnesium pills with structured noise.

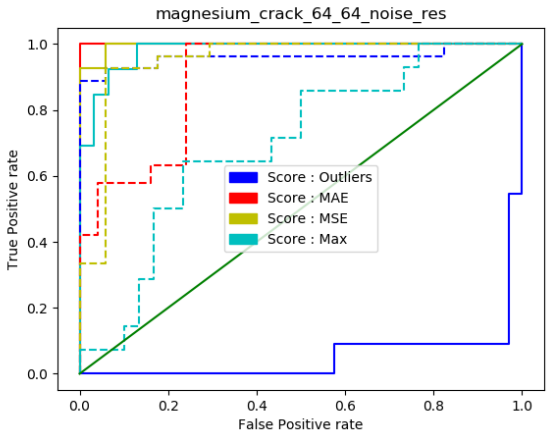

Figure 69: Performance obtained for both architectures trained in residual mode over Magnesium pills with structured noise.

5.2.7 Class1 texture images

For this class of images the classification performances are really bad. Every configuration is as bad as the random classification. It is not astonishing since the denoising job was really bad at the beginning. Furthermore the defect is not detected at all, as observed in Figure 41. Then, the anomaly detection cannot achieve good performances.

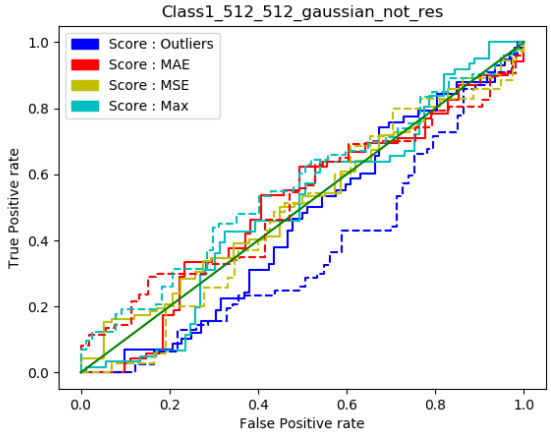

Figure 70: Performance obtained for both architectures trained in denoised mode over Class1 texture images with unstructured noise.

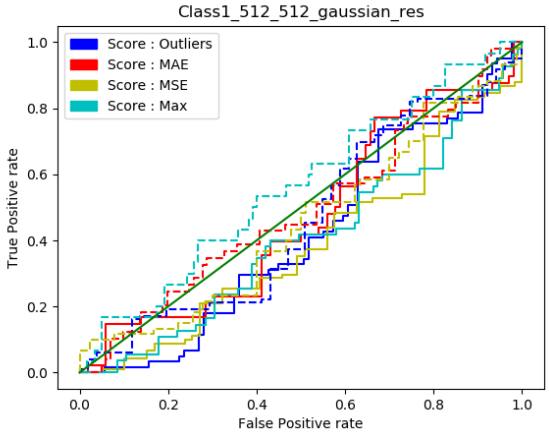

Figure 71: Performance obtained for both architectures trained in residual mode over Class1 texture images with unstructured noise.

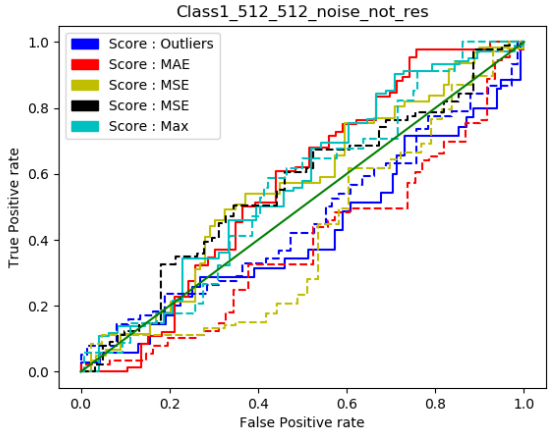

Figure 72: Performance obtained for both architectures trained in denoised mode over Class1 texture images with structured noise.

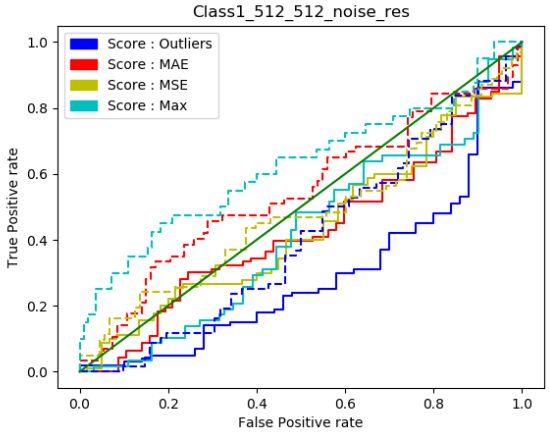

Figure 73: Performance obtained for both architectures trained in residual mode over Class1 texture images with structured noise.

5.2.8 Class2 texture images

The training over unstructured noise does not work out for this class of images. Only the max score in Figure 62 gets its head out of the water. Residual learning seems to be penalizing for the CAeSSC architecture. This architecture is clearly well suited for a denoised learning over images with structured noise, Figure 76.

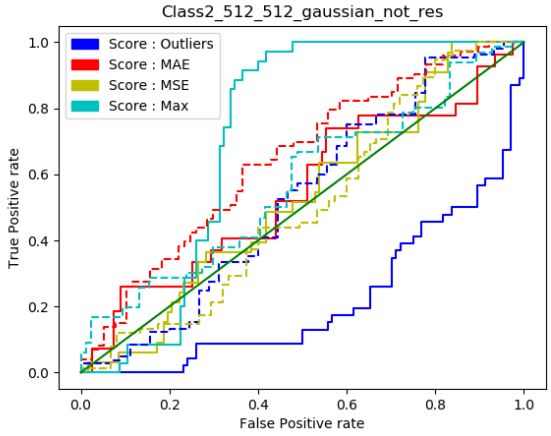

Figure 74: Performance obtained for both architectures trained in denoised mode over Class2 texture images with unstructured noise.

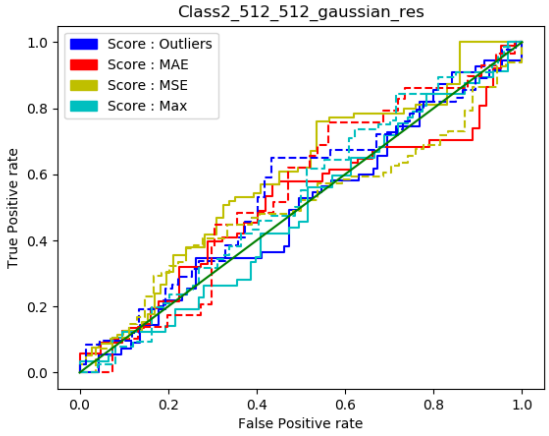

Figure 75: Performance obtained for both architectures trained in residual mode over Class2 texture images with unstructured noise.

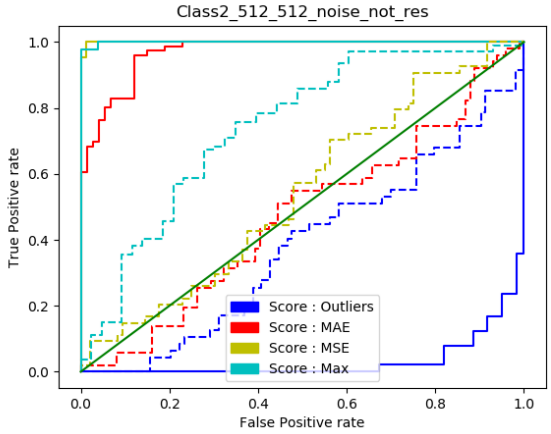

Figure 76: Performance obtained for both architectures trained in denoised mode over Class2 texture images with structured noise.

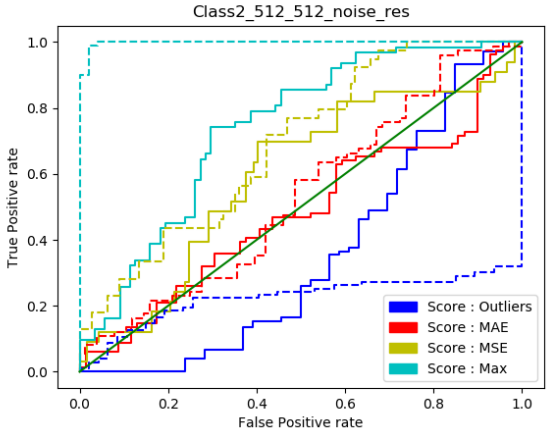

Figure 77: Performance obtained for both architectures trained in residual mode over Class2 texture images with structured noise.

5.2.9 Class3 texture images

Only the max score works well for the learning over images with unstructured noise. It was already the case for the class "Class2". But it is even stronger in this one. Again, The denoised learning over images with structured noise is well suited for the CAeSSC architecture. Figure 81 also shows that the residual learning over images with structured noise seems to improve the classification performance for the NDNN model but penalizes the CAeSSC architecture.

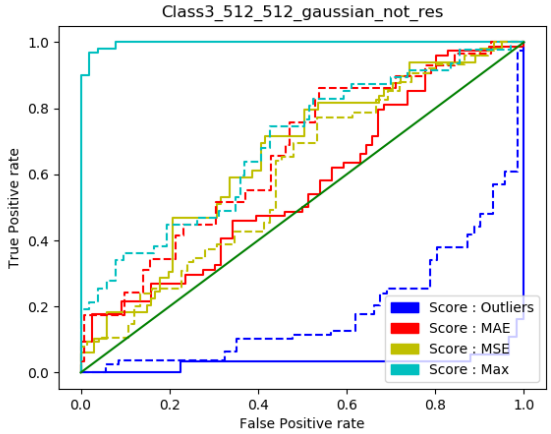

Figure 78: Performance obtained for both architectures trained in denoised mode over Class3 texture images with unstructured noise.

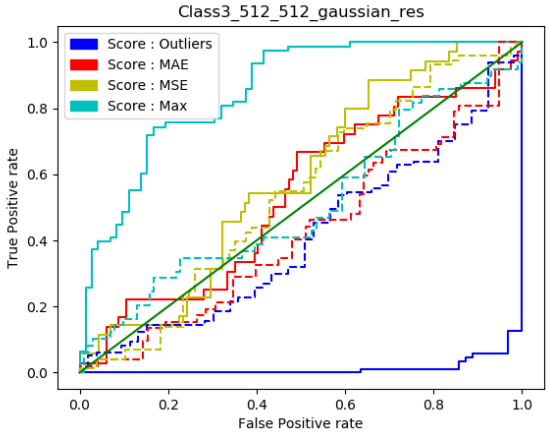

Figure 79: Performance obtained for both architectures trained in residual mode over Class3 texture images with unstructured noise.

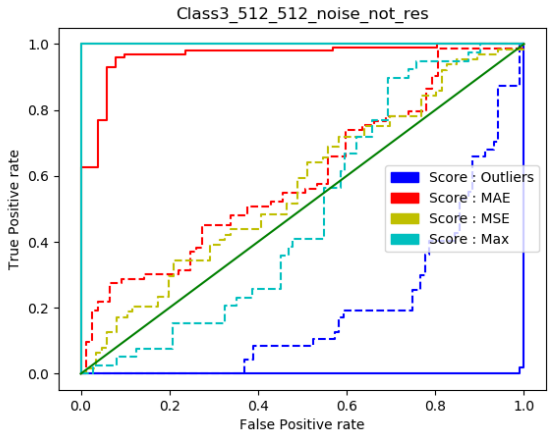

Figure 80: Performance obtained for both architectures trained in denoised mode over Class3 texture images with structured noise.

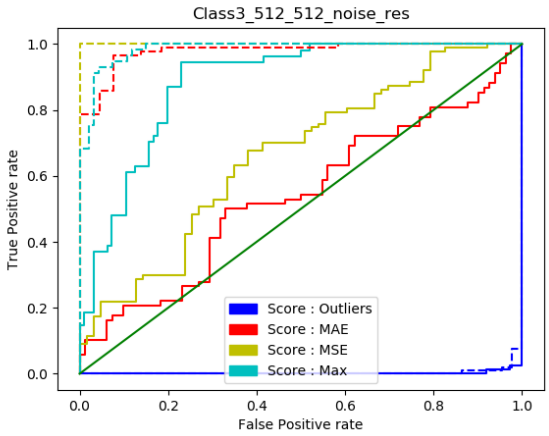

Figure 81: Performance obtained for both architectures trained in residual mode over Class3 texture images with structured noise.

5.2.10 Class4 texture images

On this class of images classification performances are as bad as the ones obtained in the class "Class1". That was predictable considering that these two classes have the worst denoising performances. Defects are not detected at all from the inherent structure of the image, as shown in Figures 41 and 44. Only the training in denoised mode over images with structured noise does better than the random classification. With this type of learning, the Max and the MSE score are able to obtain 60% of the noisy images correctly labeled as noisy, for 20% of the clean images incorrectly classified as noisy, as shown in Figure 84.

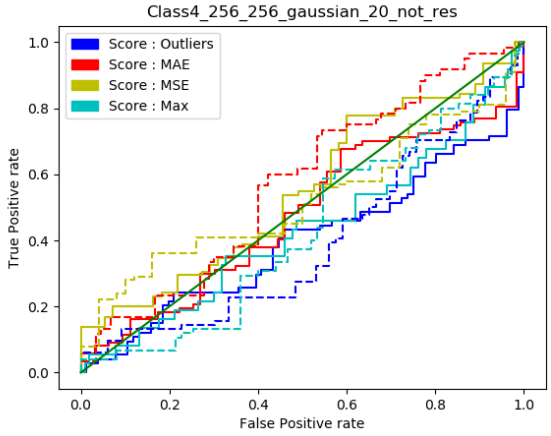

Figure 82: Performance obtained for both architectures trained in denoised mode over Class4 texture images with unstructured noise.

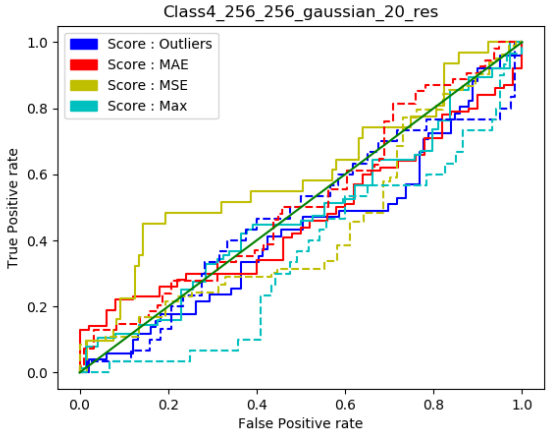

Figure 83: Performance obtained for both architectures trained in residual mode over Class4 texture images with unstructured noise.

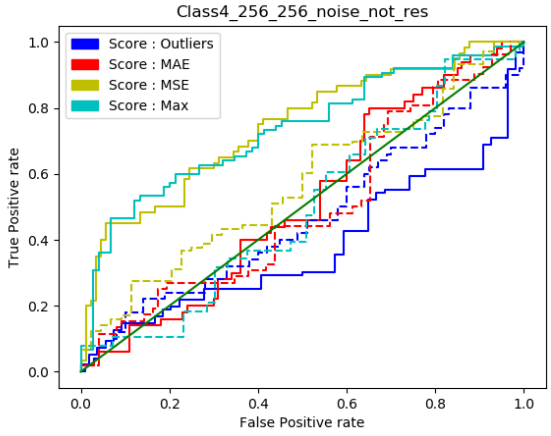

Figure 84: Performance obtained for both architectures trained in denoised mode over Class4 texture images with structured noise.

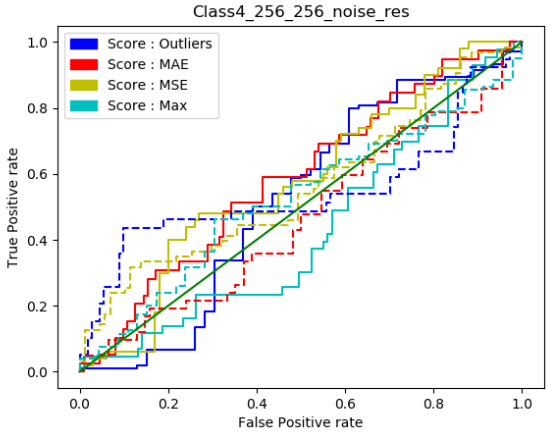

Figure 85: Performance obtained for both architectures trained in residual mode over Class4 texture images with structured noise.

5.2.11 Class5 texture images

Figures 86 and 87 show that the training in denoised mode benefits the CAeSSC model. Furthermore Figures 88 and 89 confirm this tendency. Conversely, residual learning seems to deliver slightly better results for the NDNN architecture. Good results are obtained over all configurations. The Max score labels almost a 100% of the noisy images as noisy, as shown in Figure 87.

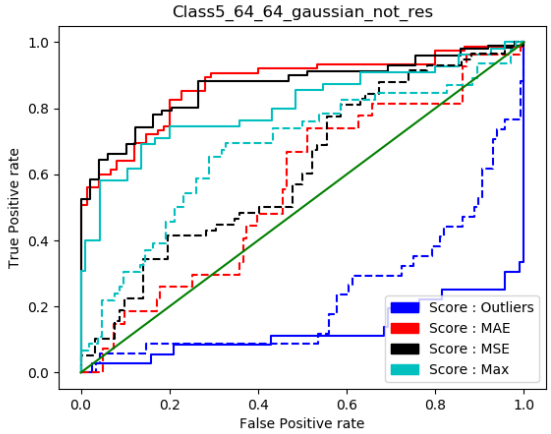

Figure 86: Performance obtained for both architectures trained in denoised mode over Class5 texture images with unstructured noise.

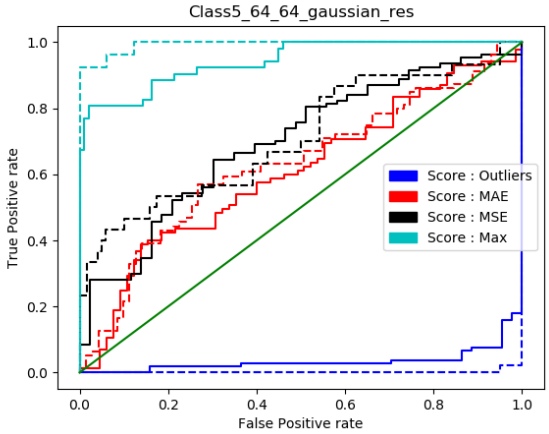

Figure 87: Performance obtained for both architectures trained in residual mode over Class5 texture images with unstructured noise.

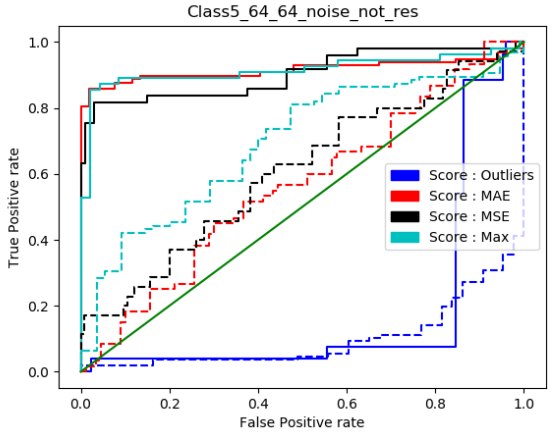

Figure 88: Performance obtained for both architectures trained in denoised mode over Class5 texture images with structured noise.

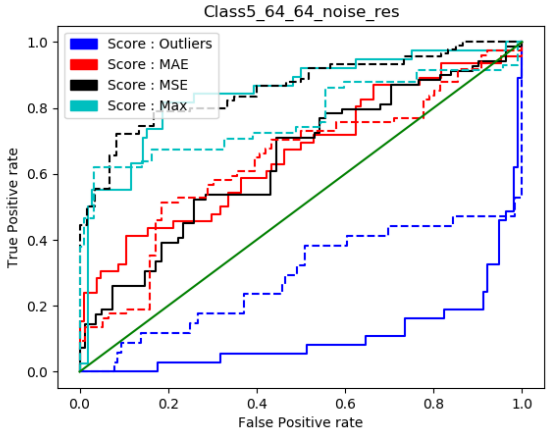

Figure 89: Performance obtained for both architectures trained in residual mode over Class5 texture images with structured noise.

For the pills classes, we do not observe a strong general tendency. The learning over structured noises gives, in general, slightly better results. But most of the time, every configuration has at least one score function giving really good performances. Amongst the score functions, it is the MSE which is the most reliable. Claims made in Section 4.2.3 are not confirmed. The CAeSSC model does not always perform better after a training in denoised mode.

For the texture classes, two tendencies pop-up. First, the architecture CAeSSC gives really good classification performance when it is trained in denoised mode over images with structured noise. In this configuration, the MSE, the MAE and the Max score always output good results. Secondly, the residual learning improves the performance of the NDNN model but penalizes the CAeSSC architecture. This last observation supports what we observed in Section 4.2.3, that the PSNR score blows-up for the CAeSSC model with a training in denoised mode. And the contrary is true for the NDNN architecture.

As we saw, learning over images with structured noise gives, in general, better performances than the

one over images with unstructured noise. And it is not astonishing since the artificial structured noises are alike real noises that we try to remove during the test phase. In contrast, an unstructured noise does not resemble at all to the noises that we want to detect. Nevertheless, the learning over images with unstructured noise performs really well on almost every class. In fact, bad results are obtained only over the following classes: Class1, Class2 and Class4. Class1 and Class4 do not give any good results in any configuration. Then, the learning over images with unstructured noise cannot perform as well as the training over images with structured noise for only one class. It means that unstructured noises are a good way to make the architecture know what is part of the inherent structure of an image and what is not. This could be a good lead to create an universal denoiser.

5.3 Patch influences

The patch size seems to have a quite strong influence on performance. Every class of images improves their results when the patch size decrease. Figures 208 and 209 demonstrate this tendency. The left figure uses one patch of size 512 x 512 which is the size of the input image. The right Figure uses patches of size 64 x 64. It is possible that better results were obtained with patches of size 64 x 64, because creators of both networks were using patches of this size. The meta-parameters may be calibrated for this size. Therefore, it is a question to explore.

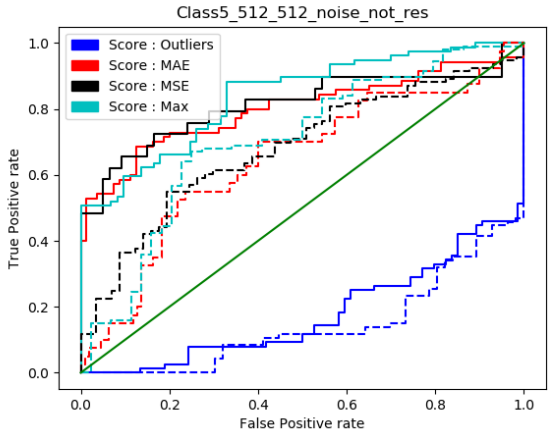

Figure 90: Performance obtained for both architectures trained in denoised mode over Class5 texture images with structured noise.

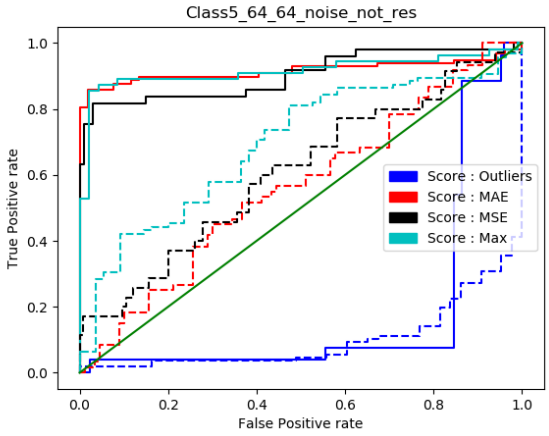

Figure 91: Performance obtained for both architectures trained in denoised mode over cropped Class5 texture images with structured noise.

More Figures supporting this claim can be found in appendice B.

6 Conclusion

In this thesis, two Convolutional Neural Networks were proposed for image denoising. Both of them were trained in two different modes. On the one hand, the residual learning tries to isolate the noise from the noisy observation. On the other hand, the denoised learning tries to remove the noise from the noisy observation. We showed that the model, which adopts batch normalization, benefits from the residual learning. These two types of learning were made over two type of dataset for each class of images. One dataset was made of images containing artificial structured noises, whereas the other one was made of images with artificial unstructured noises.

After the denoising task, we have performed anomalies detection. This detection is made by comparing the denoised image which was output by the CNNs with its corresponding input image. If the input image was noisy its corresponding output should be different since the noise should have been removed. If the input image was not noisy, its corresponding output should be exactly the same. It is this comparison between the input and the output that allows us to label each input image as noisy or clean.

The denoised task was made over different class of images. Good classification performances were obtained over almost all of them except for the classes "Class1" and "Class4". The bad results over these two classes come from the denoising task, which does not correctly detect the defects.

Results obtained show that a mircale configuration does not exist. One configuration which works well over a class of images does not perform always as well on another class. Nevertheless some tendencies appear. Perfomances obtained after a training over images with structured noises are generarally better: even if CNNs trained over unstructured noises are not far behind, it is possible that further investigations over the meta-parameters fill this gap. We also showed that batch normalization layers and residual learning profit from each other. Higher PSNR values are reached faster during the training phase.

Further investigation over the patch size should be made in the future. This parameter have demonstrated its importance.

To conclude, this paper has demonstrated a new promising way to make anomaly detection. There are still a lot of parameters to investigate on. No CNN optimization was done. Then it is possible that both architectures could improve their results.

7 References

[1] Kai Zhang, Wangmeng Zuo, Yunjin Chen, Deyu Meng, Lei Zhang, *"Beyond a Gaussian Denoiser: Residual Learning of Deep CNN for Image Denoising"*, IEEE Transaction on Image Processing, vol 26, pp. 3142-3155, 2017.

[2] Xiao-Jiao Mao, Chunhua Shen, Yu-Bin Yang, *"Image Restoration Using Convolutional Auto-encoders with Symmetric Skip Connections"*, Cornell University Library, arXiv:1606.08921, 2016.

[3] Matthew D. Zeiler, Rob Fergus, *"Visualizing and Understanding Convolutional Networks"*, Cornell University Library, arXiv:1311.2901, 2013.

[4] Michel Verleysen, *"Nonlinear regression with Multi-Layer Perceptrons"*, ELEC2870 - Machine learning: regression and dimensionality reduction, slides 23-29, 2018.

[5] Adit Deshpande, (July 20, 2016), A Beginner's Guide To Understanding Convolutional Neural Networks, article retrieve from: *"https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner's-Guide-To-Understanding-Convolutional-Neural-Networks/"*

[6] Firdaouss Doukkali, (Oct 20, 2017), Batch normalization in Neural Networks, article retrieve from: *"https://towardsdatascience.com/batch-normalization-in-neural-networks-1ac91516821c"*

[7] Sergey Ioffe, Christian Szegedy, *"Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift"*, Google Inc., 2015.

[8] Sagar Sharma, (Sep 22, 2017), Epoch vs Batch Size vs Iterations, article retrieve from: *"https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9"*

[9] Video presented by Andrew Ng (Co-founder, Coursera; Adjunct Professor, Stanford University; formerly head of Baidu AI Group/Google Brain), *"https://www.coursera.org/learn/machine-learning/lecture/GFFPB/gradient-descent-intuition"*, Videao by coursera

[10] Michael Nielsen, (Dec, 2017), Why are deep neural networks hard to train?, article retrieve from: *"http://neuralnetworksanddeeplearning.com/chap5.html"*

[11] Soravit Changpinyo, Mark Sandler and Andrey Zhmoginov, *"The Power of Sparsity in Convolutional Neural Networks"*, Cornell University Library, arXiv:1702.06257, 21 Feb 2017.

[12] Ian J. Goodfellow, David Warde-Farley, Mehdi Mirza, Aaron Courville and Yoshua Bengio, *"Maxout Networks"*, Cornell University Library, arXiv:1302.4389, 20 Sep 2013.

[13] Cho,C.,Chung,B.,and Park,M., *"Development of real-time vision-based fabric inspection system"*, IEEE Trans.on Industrial Electronics, 52(4), 1073–1079, 2005.

[14] Tajeripour,F.,Kabir,E.,and Sheikhi,A., *"Fabric defect detection using modified local binary patterns."*, EURASIPJ. on Advances in Signal Processing, ID 783898, 2008.

[15] R. Conners, C. McMillan, K. Lin, and R. Vasquez-Espinosa, *Identifying and locating surface defects in wood: Part of an automated timber processing system.* IEEE Transactions on Pattern Analysis and Machine Intelligence, 5:573–583, 1983.

[16] Y. Huang and K. Chan, *Texture decomposition by harmonic s extraction from higher order statistics.* IEEE Transactions on Image Processing, 13(1):1–14, 2004.

[17] A. Monadjemi, *Towards Efficient Texture Classification and Abnormality Detection.* PhD thesis, University of Bristol, UK, 2004.

[18] Kumar, A., *Computer-vision-based fabric defect detection: A survey.* IEEE Trans. on Industrial Electronics, 55(1), 348–363, 2008.

[19] Chan, C. and Pang, G.K., *Fabric defect detection by fourier analysis.* IEEE Trans. on Industry Applications, 36(5), 1267–1276, 2000.

[20] Ehab A. Kholief, Samy H. Darwish, Nashat Fors, *Detection of Steel Surface Defect Based on Machine Learning Using Deep Auto-encoder Network* 2017.

[21] S. Faghih-Roohi, S. Hajizadeh, A. Nùñez, R. Babuska, and B. De Schutter *Deep convolutional neural networks for detection of rail surface defects* Proceedings of the 2016 International Joint Conference on Neural Networks (IJCNN 2016), Vancouver, Canada, pp. 2584–2589, July 2016.

[22] Christopher M. Bishop, *Pattern Recognition and Machine Learning*, Springer-Verlag Berlin, Heidelberg, pp. 232-240, 2006.

[23] Christopher M. Bishop, *Pattern Recognition and Machine Learning*, Springer-Verlag Berlin, Heidelberg, p. 240, 2006.

[24] Christopher M. Bishop, *Pattern Recognition and Machine Learning*, Springer-Verlag Berlin, Heidelberg, pp. 241-248, 2006.

[25] Christopher M. Bishop, *Pattern Recognition and Machine Learning*, Springer-Verlag Berlin, Heidelberg, pp. 241-248, 2006.

[26] Yoshua Bengio, Patrice Simard, and Paolo Frasconi, *Learning long-term dependencies with 289 gradient descent is difficult*, IEEE transactions on neural networks, 5(2): 157-166, 1994.

[27] Xavier Glorot and Yoshua Bengio, *Understanding the difficulty of training deep feedforward 302 neural networks*, In AISTATS, pages 249-256, 2010.

[28] Sepp Hochreiter, *The vanishing gradient problem during learning reccurent neural nets and problem solutions*, IJUFKS, 6(2): 1-10, 1998.

[29] Jakub Konecny, Jie Liu, Peter Richtarik and Martin Takac, *Mini-Batch Semi-Stochastic Gradient Descent in the Proximal Setting*, IEEE Journal of Selected Topics in Signal Processing, 10(2), pp. 242-255, 2015.

[30] Mu Li, Tong Zhang, Yugiang Chen and Alexander J. Smola, *Efficient mini-batch training for stochastic optimization*, KDD '14 Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 661-670, 2014

[31] A. Levin, B. Nadler, F. Durand, and W. T. Freeman, *Patch complexity, finite pixel correlations and optimal denoising*, in European Conference on Computer Vision, 2012, pp. 73–86.

[32] D. Zoran and Y. Weiss, *From learning models of natural image patches to whole image restoration*, in IEEE International Conference on Computer Vision, 2011, pp. 479–486.

Appendices

A Training histories

In all Figures that follow solid lines are learning rates obtained with residual learning and broken line are learning rates obtained with denoised learning. These Figures represent the Peak Signal-to-Noise Ratio (PSNR) improvement through epochs.

A.1 Ginseng pills

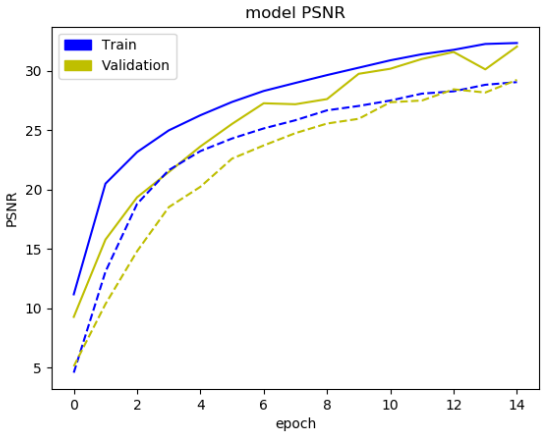

Figure 92: NDNN training history over images with unstructured noise.
Input: Ginseng_gaussian.

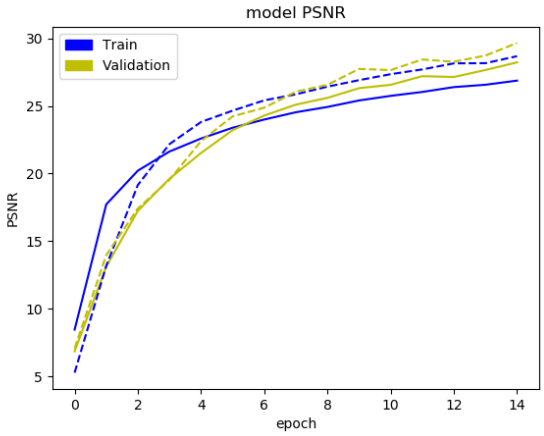

Figure 93: NDNN training history over images with structured noise.
Input: Ginseng_noise.

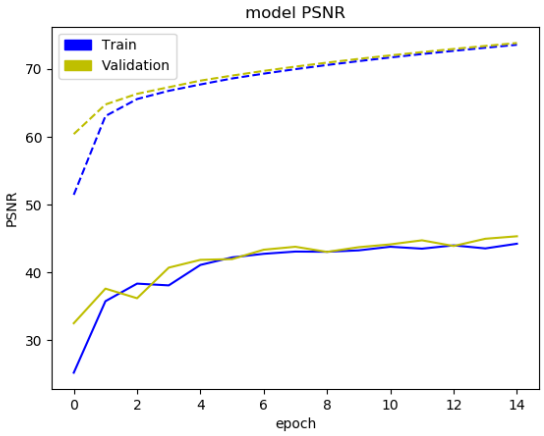

Figure 94: CAeSSC training history over images with unstructured noise.
Input: Ginseng_gaussian.

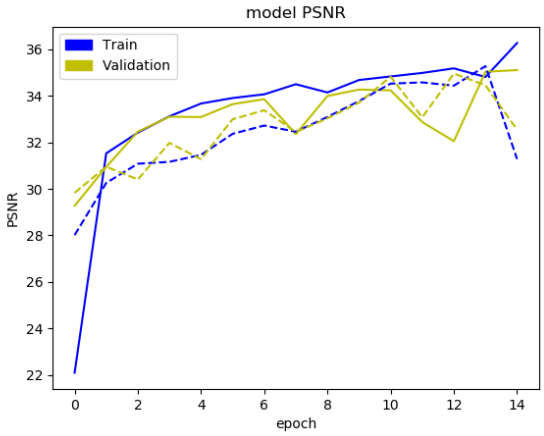

Figure 95: CAeSSC training history over images with structured noise.
Input: Ginseng_noise.

A.2 Magnesium pills

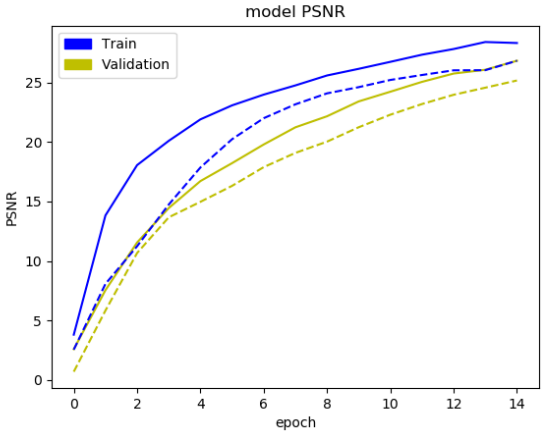

Figure 96: NDNN training history over images with unstructured noise.
Input: Magnesium_gaussian.

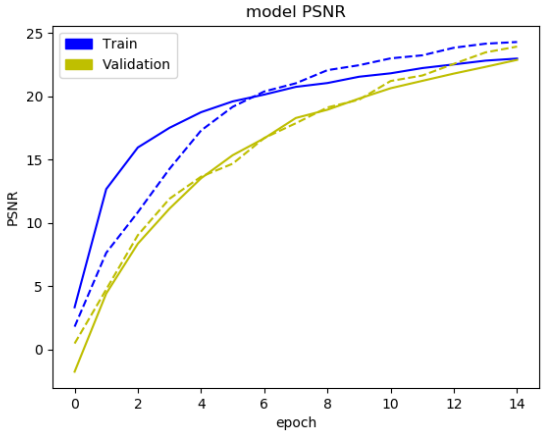

Figure 97: NDNN training history over images with structured noise.
Input: Magnesium_noise.

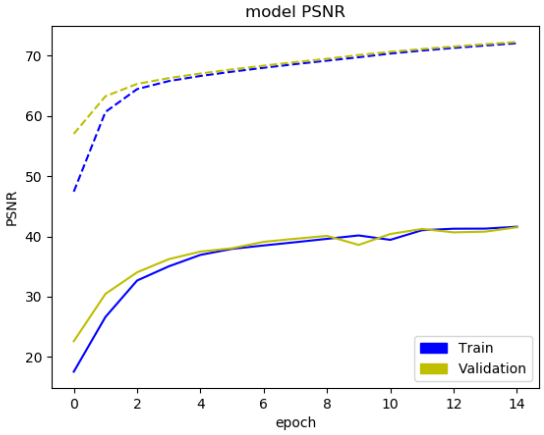

Figure 98: CAeSSC training history over images with unstructured noise.
Input: Magnesium_gaussian.

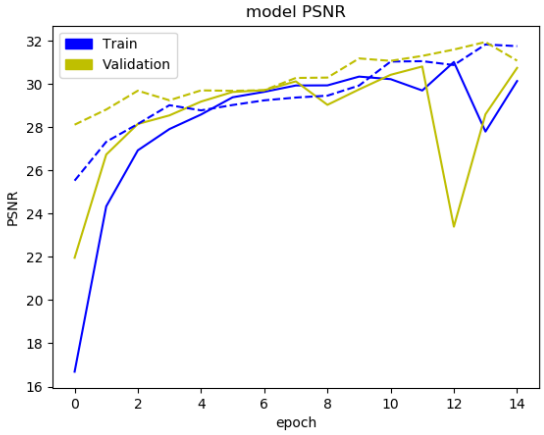

Figure 99: CAeSSC training history over images with structured noise.
Input: Magnesium_noise.

A.3 Mint pills

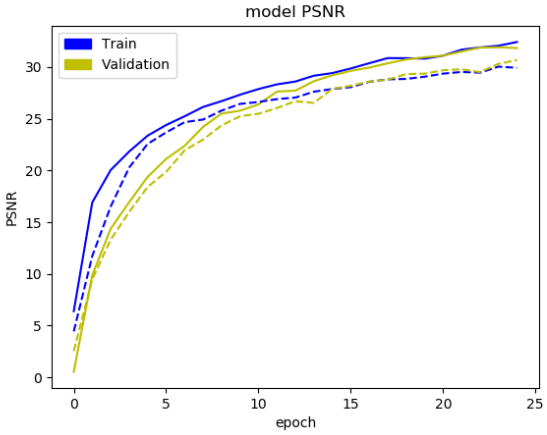

Figure 100: NDNN training history over images with unstructured noise.
Input: Mint_gaussian.

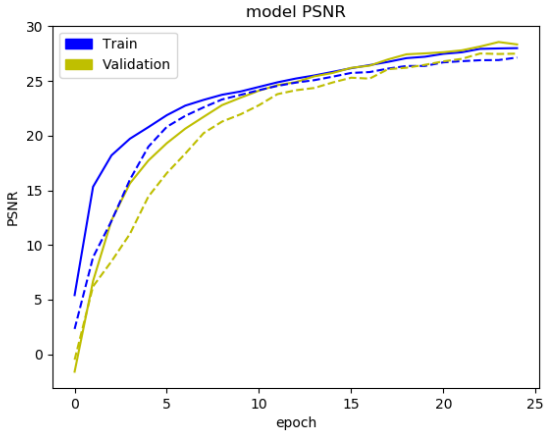

Figure 101: NDNN training history over images with structured noise.
Input: Mint_noise.

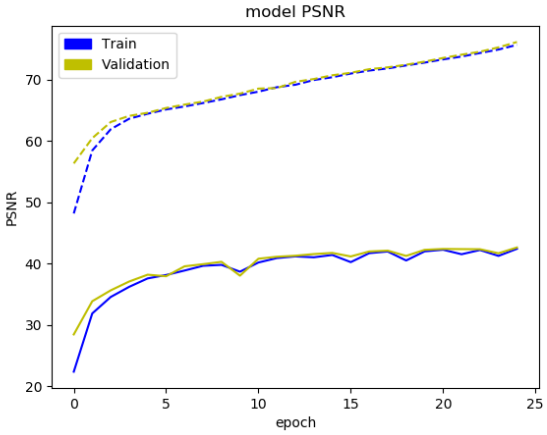

Figure 102: CAeSSC training history over images with unstructured noise.
Input: Mint_gaussian.

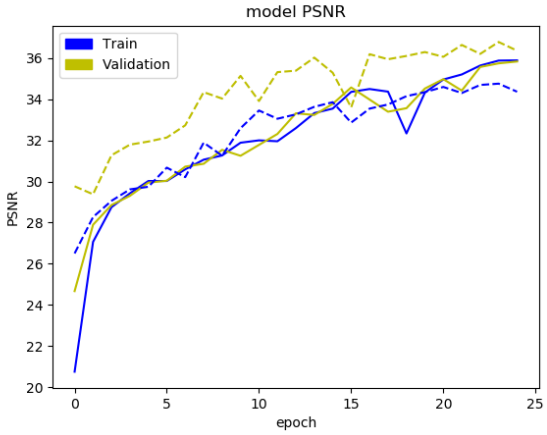

Figure 103: CAeSSC training history over images with structured noise.
Input: Mint_noise.

A.4 Class1

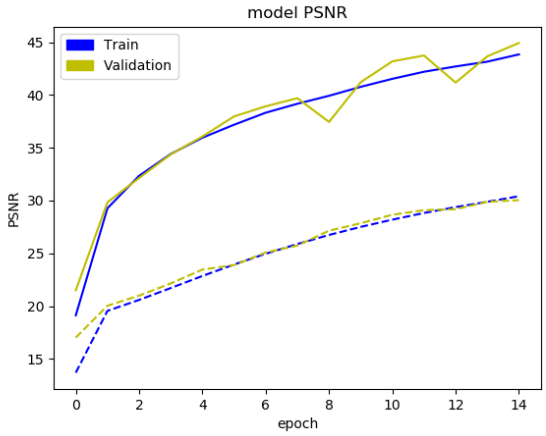

Figure 104: NDNN training history over images with unstructured noise.
Input: Class1_gaussian.

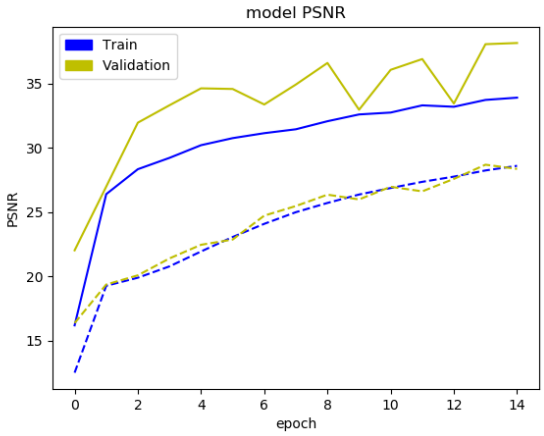

Figure 105: NDNN training history over images with structured noise.
Input: Class1_noise.

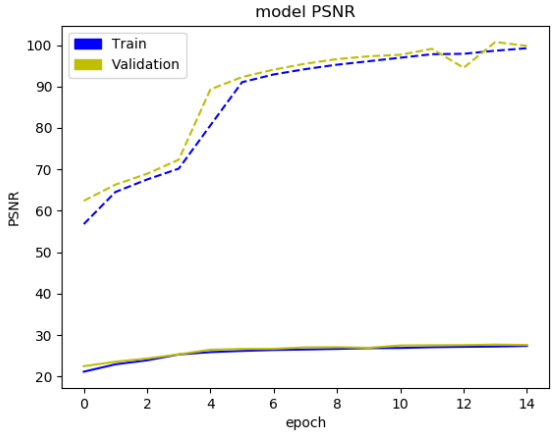

Figure 106: CAeSSC training history over images with unstructured noise.
Input: Class1_gaussian.

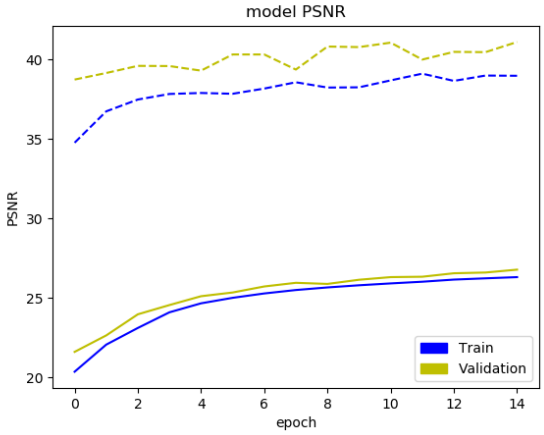

Figure 107: CAeSSC training history over images with structured noise.
Input: Class1_noise.

A.5 Class2

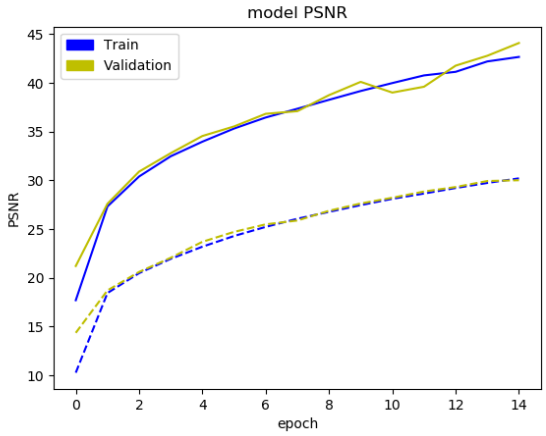

Figure 108: NDNN training history over images with unstructured noise.
Input: Class2_gaussian.

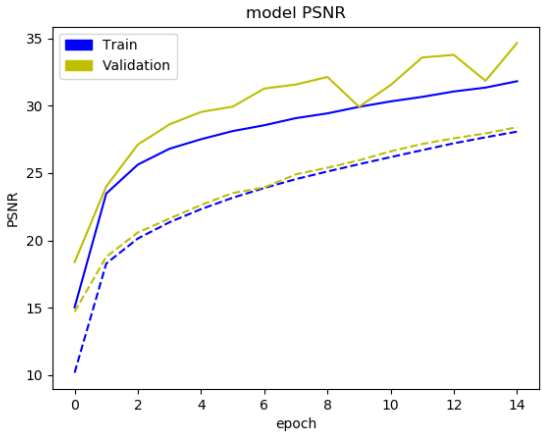

Figure 109: NDNN training history over images with structured noise.
Input: Class2_noise.

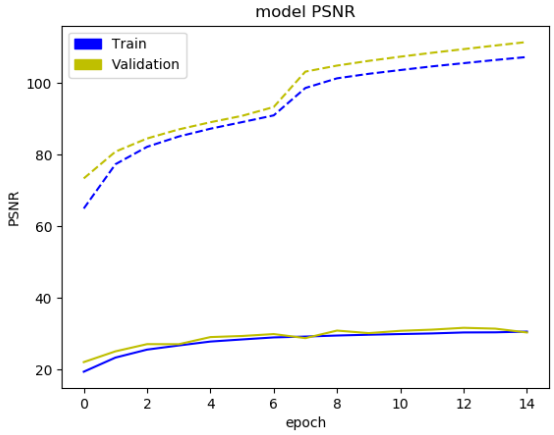

Figure 110: CAeSSC training history over images with unstructured noise.
Input: Class2_gaussian.

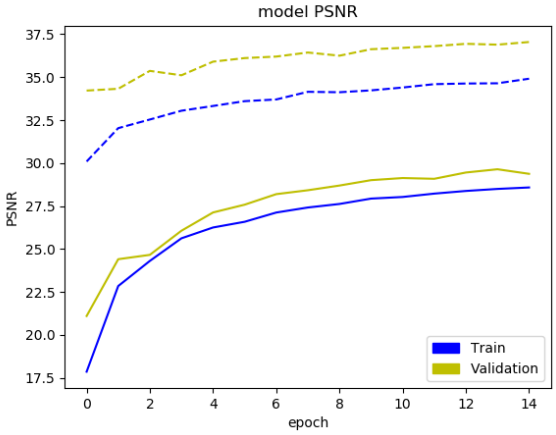

Figure 111: CAeSSC training history over images with structured noise.
Input: Class2_noise.

A.6 Class3

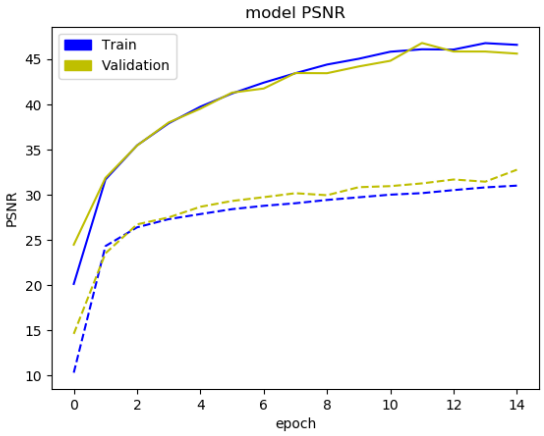

Figure 112: NDNN training history over images with unstructured noise.
Input: Class3_gaussian.

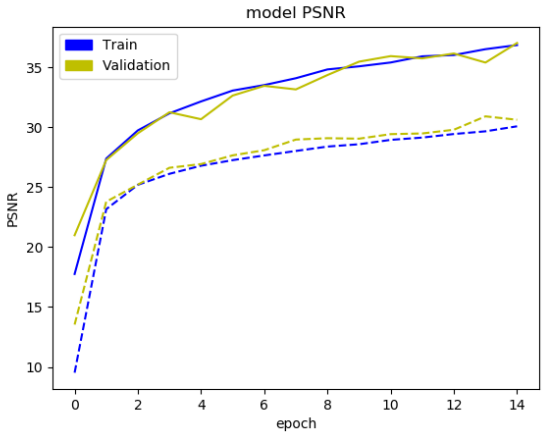

Figure 113: NDNN training history over images with structured noise.
Input: Class3_noise.

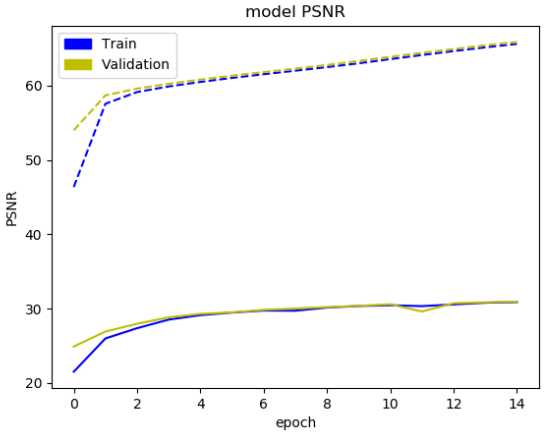

Figure 114: CAeSSC training history over images with unstructured noise.
Input: Class3_gaussian.

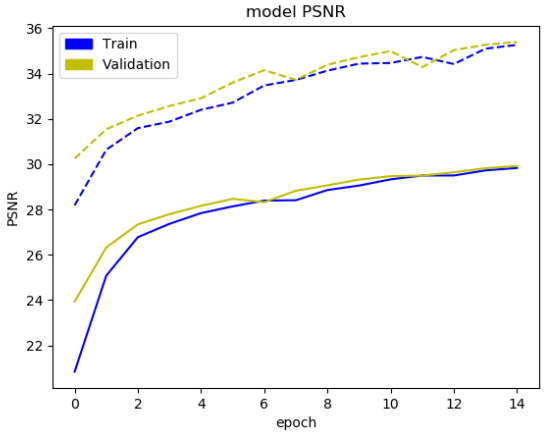

Figure 115: CAeSSC training history over images with structured noise.
Input: Class3_noise.

A.7 Class4

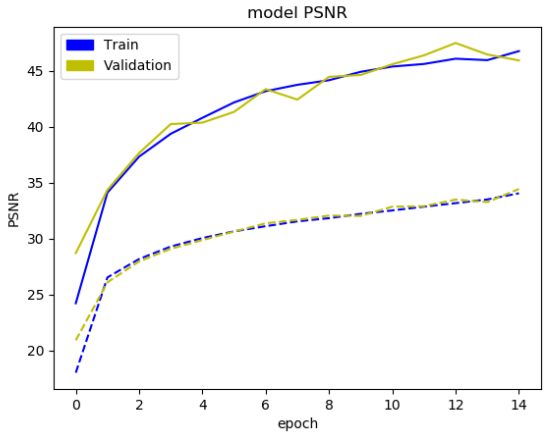

Figure 116: NDNN training history over images with unstructured noise.
Input: Class4_gaussian.

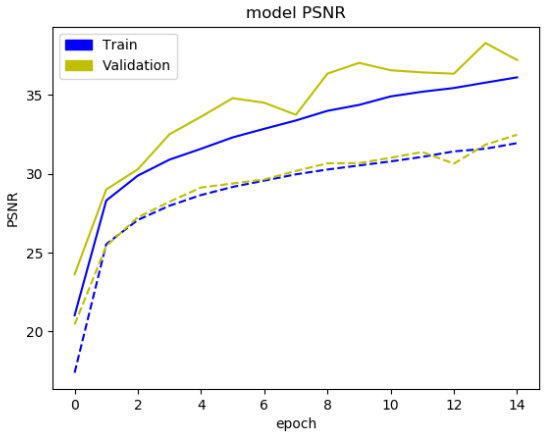

Figure 117: NDNN training history over images with structured noise.
Input: Class4_noise.

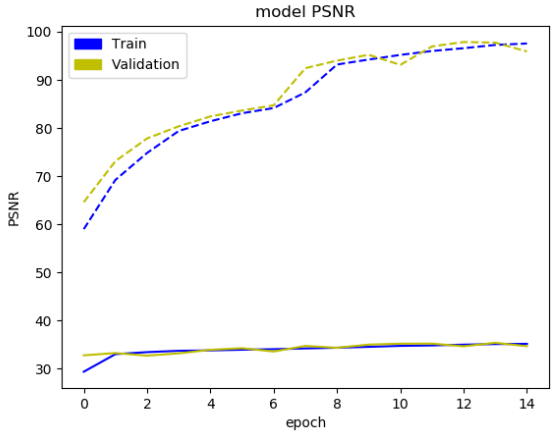

Figure 118: CAeSSC training history over images with unstructured noise.
Input: Class4_gaussian.

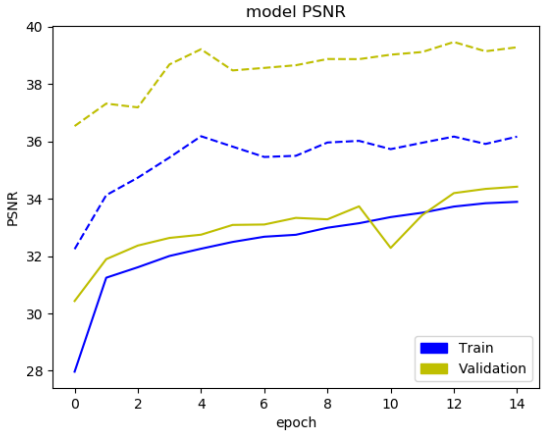

Figure 119: CAeSSC training history over images with structured noise.
Input: Class4_noise.

A.8 Class5

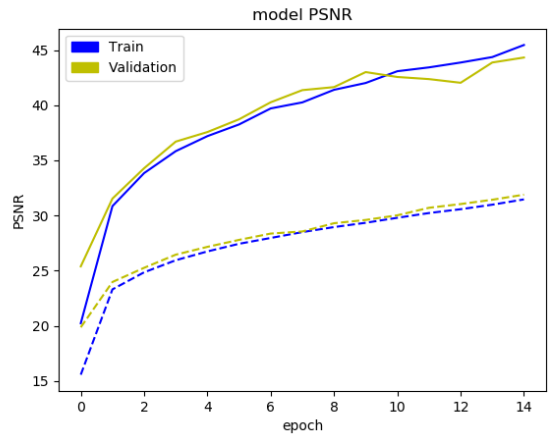

Figure 120: NDNN training history over images with unstructured noise.
Input: Class5_gaussian.

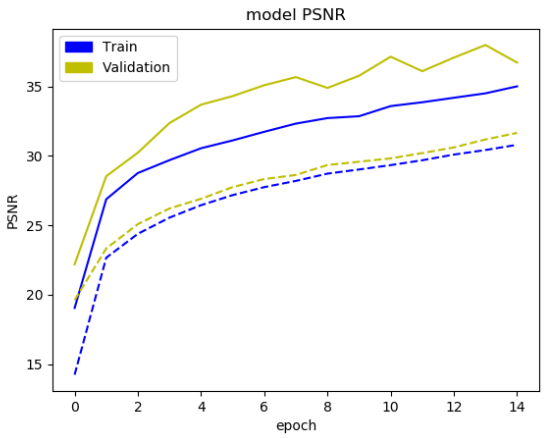

Figure 121: NDNN training history over images with structured noise.
Input: Class5_noise.

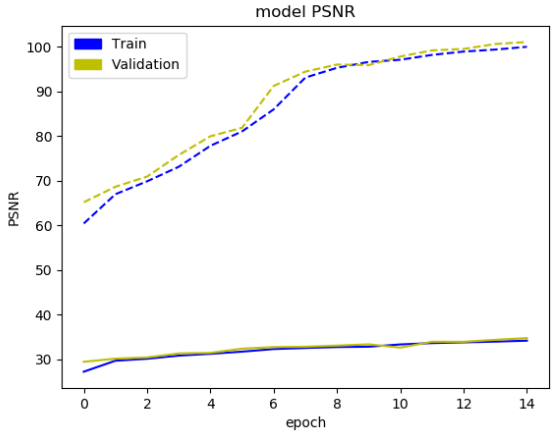

Figure 122: CAeSSC training history over images with unstructured noise.
Input: Class5_gaussian.

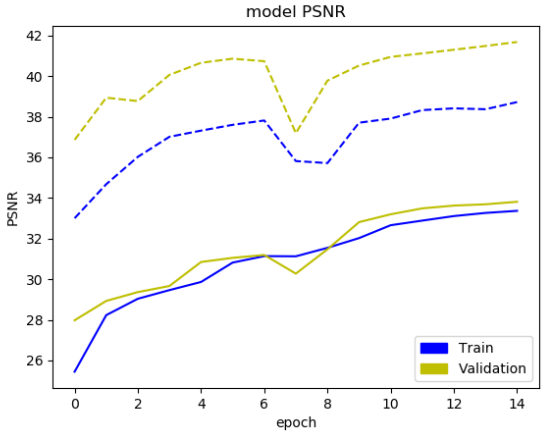

Figure 123: CAeSSC training history over images with structured noise.
Input: Class5_noise.

B Patch influences

In all Figures that follow solid lines are results obtained with the CAeSSC architecture and broken line are performances obtained with the NDNN architecture.

B.1 Pills classes

Left Figures use one patch of the input's size. Right Figures use several patches of size 64 x 64. Ginseng pill images have a size of 256 x 576, right Figures use then 36 patches. Magnesium pill images have size of 256 x 256 making 16 patches. And mint pill images have a size of 256 x 384 making 24 patches.

B.1.1 Contaminated Ginseng pills

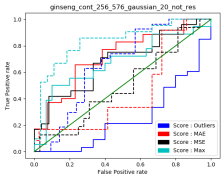

Figure 124: Performance obtained for both architectures trained in denoised mode over Ginseng pills with unstructured noise.

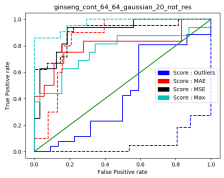

Figure 125: Performance obtained for both architectures trained in denoised mode over cropped Ginseng pills with unstructured noise.

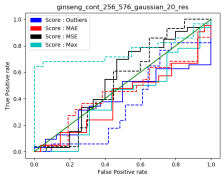

Figure 126: Performance obtained for both architectures trained in residual mode over Ginseng pills with unstructured noise.

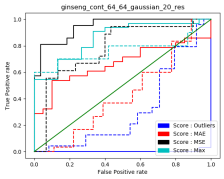

Figure 127: Performance obtained for both architectures trained in residual mode over cropped Ginseng pills with unstructured noise.

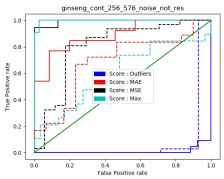

Figure 128: Performance obtained for both architectures trained in denoised mode over Ginseng pills with structured noise.

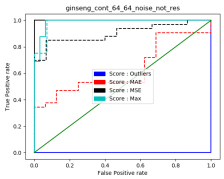

Figure 129: Performance obtained for both architectures trained in denoised mode over cropped Ginseng pills with structured noise.

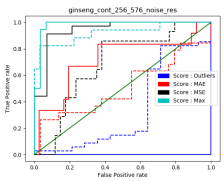

Figure 130: Performance obtained for both architectures trained in residual mode over Ginseng pills with structured noise.

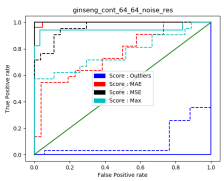

Figure 131: Performance obtained for both architectures trained in residual mode over cropped Ginseng pills with structured noise.

B.1.2 Cracked Ginseng pills

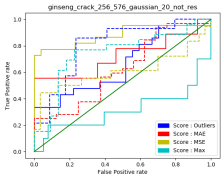

Figure 132: Performance obtained for both architectures trained in denoised mode over Ginseng pills with unstructured noise.

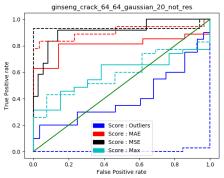

Figure 133: Performance obtained for both architectures trained in denoised mode over cropped Ginseng pills with unstructured noise.

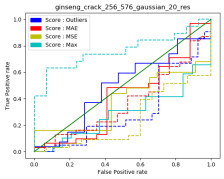

Figure 134: Performance obtained for both architectures trained in residual mode over Ginseng pills with unstructured noise.

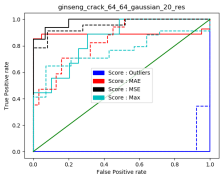

Figure 135: Performance obtained for both architectures trained in residual mode over cropped Ginseng pills with unstructured noise.

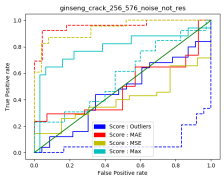

Figure 136: Performance obtained for both architectures trained in denoised mode over Ginseng pills with structured noise.

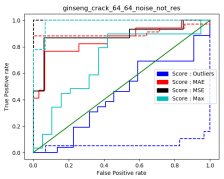

Figure 137: Performance obtained for both architectures trained in denoised mode over cropped Ginseng pills with structured noise.

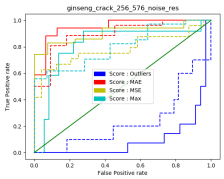

Figure 138: Performance obtained for both architectures trained in residual mode over Ginseng pills with structured noise.

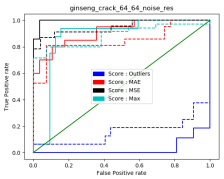

Figure 139: Performance obtained for both architectures trained in residual mode over cropped Ginseng pills with structured noise.

B.1.3 Contaminated magnesium pills

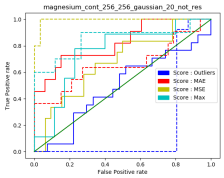

Figure 140: Performance obtained for both architectures trained in denoised mode over magnesium pills with unstructured noise.

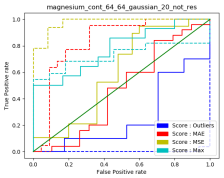

Figure 141: Performance obtained for both architectures trained in denoised mode over cropped magnesium pills with unstructured noise.

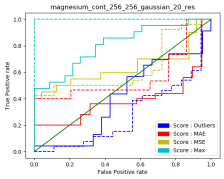

Figure 142: Performance obtained for both architectures trained in residual mode over magnesium pills with unstructured noise.

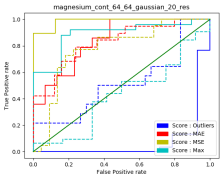

Figure 143: Performance obtained for both architectures trained in residual mode over cropped magnesium pills with unstructured noise.

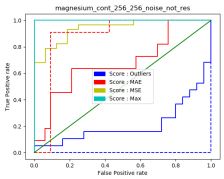

Figure 144: Performance obtained for both architectures trained in denoised mode over magnesium pills with structured noise.

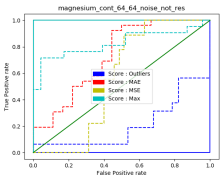

Figure 145: Performance obtained for both architectures trained in denoised mode over cropped magnesium pills with structured noise.

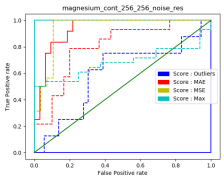

Figure 146: Performance obtained for both architectures trained in residual mode over magnesium pills with structured noise.

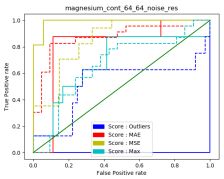

Figure 147: Performance obtained for both architectures trained in residual mode over cropped magnesium pills with structured noise.

B.1.4 Cracked magnesium pills

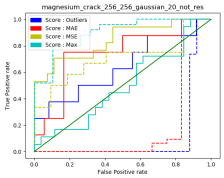

Figure 148: Performance obtained for both architectures trained in denoised mode over magnesium pills with unstructured noise.

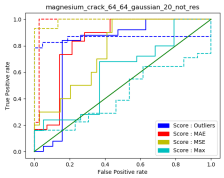

Figure 149: Performance obtained for both architectures trained in denoised mode over cropped magnesium pills with unstructured noise.

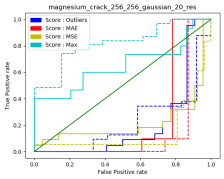

Figure 150: Performance obtained for both architectures trained in residual mode over magnesium pills with unstructured noise.

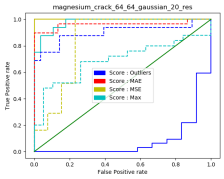

Figure 151: Performance obtained for both architectures trained in residual mode over cropped magnesium pills with unstructured noise.

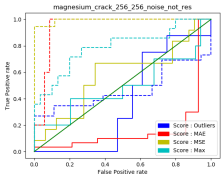

Figure 152: Performance obtained for both architectures trained in denoised mode over magnesium pills with structured noise.

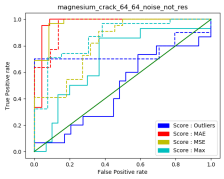

Figure 153: Performance obtained for both architectures trained in denoised mode over cropped magnesium pills with structured noise.

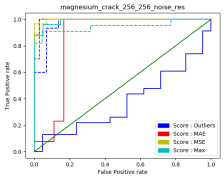

Figure 154: Performance obtained for both architectures trained in residual mode over magnesium pills with structured noise.

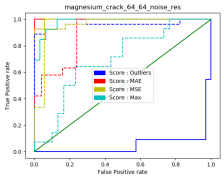

Figure 155: Performance obtained for both architectures trained in residual mode over cropped magnesium pills with structured noise.

B.1.5 Contaminated mint pills

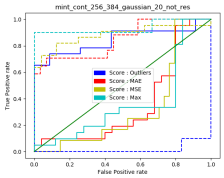

Figure 156: Performance obtained for both architectures trained in denoised mode over mint pills with unstructured noise.

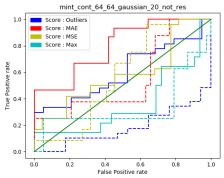

Figure 157: Performance obtained for both architectures trained in denoised mode over cropped mint pills with unstructured noise.

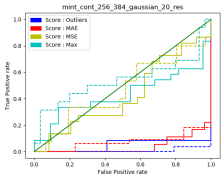

Figure 158: Performance obtained for both architectures trained in residual mode over mint pills with unstructured noise.

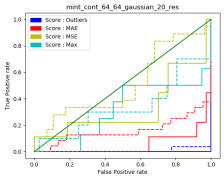

Figure 159: Performance obtained for both architectures trained in residual mode over cropped mint pills with unstructured noise.

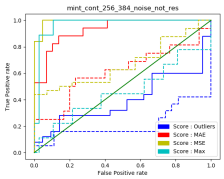

Figure 160: Performance obtained for both architectures trained in denoised mode over mint pills with structured noise.

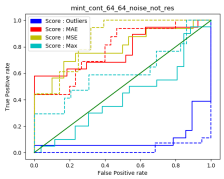

Figure 161: Performance obtained for both architectures trained in denoised mode over cropped mint pills with structured noise.

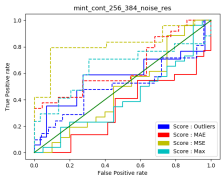

Figure 162: Performance obtained for both architectures trained in residual mode over mint pills with structured noise.

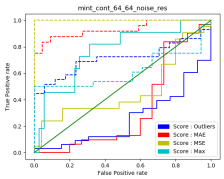

Figure 163: Performance obtained for both architectures trained in residual mode over cropped mint pills with structured noise.

B.1.6 Cracked mint pills

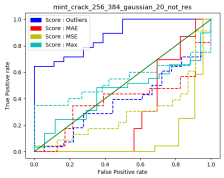

Figure 164: Performance obtained for both architectures trained in denoised mode over mint pills with unstructured noise.

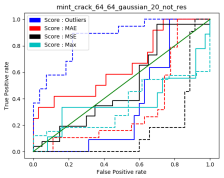

Figure 165: Performance obtained for both architectures trained in denoised mode over cropped mint pills with unstructured noise.

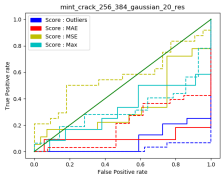

Figure 166: Performance obtained for both architectures trained in residual mode over mint pills with unstructured noise.

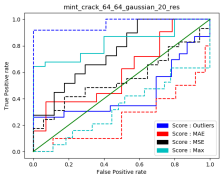

Figure 167: Performance obtained for both architectures trained in residual mode over cropped mint pills with unstructured noise.

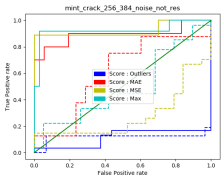

Figure 168: Performance obtained for both architectures trained in denoised mode over mint pills with structured noise.

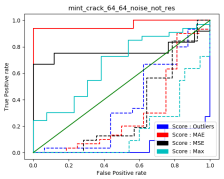

Figure 169: Performance obtained for both architectures trained in denoised mode over cropped mint pills with structured noise.

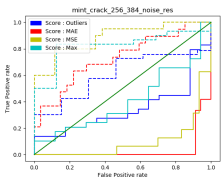

Figure 170: Performance obtained for both architectures trained in residual mode over mint pills with structured noise.

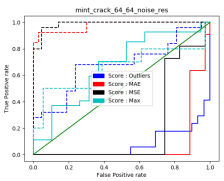

Figure 171: Performance obtained for both architectures trained in residual mode over cropped mint pills with structured noise.

B.2 Complex texture classes

Left Figures use one patch of size 512 x 512 which is the size of the input image. Right Figures use 64 patches of size 64 x 64.

B.2.1 Class1

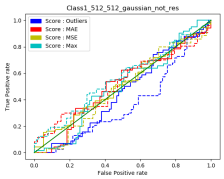

Figure 172: Performance obtained for both architectures trained in denoised mode over Class1 texture images with unstructured noise.

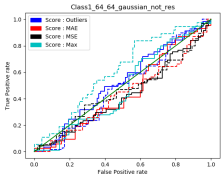

Figure 173: Performance obtained for both architectures trained in denoised mode over cropped Class1 texture images with unstructured noise.

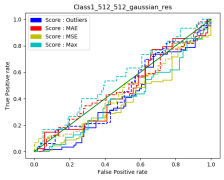

Figure 174: Performance obtained for both architectures trained in residual mode over Class1 texture images with unstructured noise.

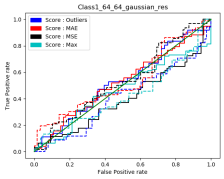

Figure 175: Performance obtained for both architectures trained in residual mode over cropped Class1 texture images with unstructured noise.

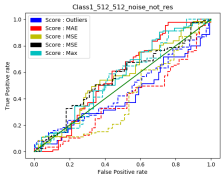

Figure 176: Performance obtained for both architectures trained in denoised mode over Class1 texture images with structured noise.

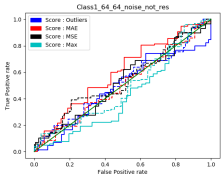

Figure 177: Performance obtained for both architectures trained in denoised mode over cropped Class1 texture images with structured noise.

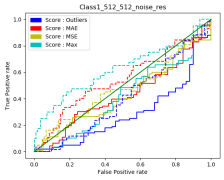

Figure 178: Performance obtained for both architectures trained in residual mode over Class1 texture images with structured noise.

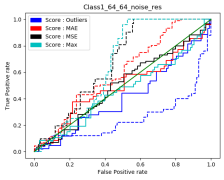

Figure 179: Performance obtained for both architectures trained in residual mode over cropped Class1 texture images with structured noise.

B.2.2 Class2

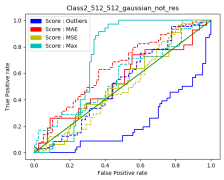

Figure 180: Performance obtained for both architectures trained in denoised mode over Class2 texture images with unstructured noise.

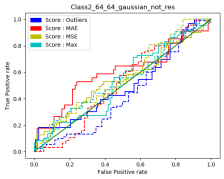

Figure 181: Performance obtained for both architectures trained in denoised mode over cropped Class2 texture images with unstructured noise.

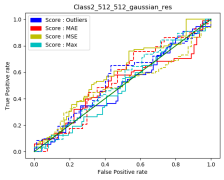

Figure 182: Performance obtained for both architectures trained in residual mode over Class2 texture images with unstructured noise.

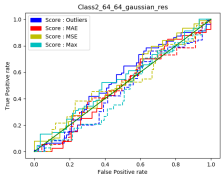

Figure 183: Performance obtained for both architectures trained in residual mode over cropped Class2 texture images with unstructured noise.

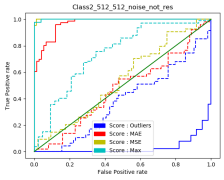

Figure 184: Performance obtained for both architectures trained in denoised mode over Class2 texture images with structured noise.

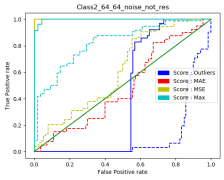

Figure 185: Performance obtained for both architectures trained in denoised mode over cropped Class2 texture images with structured noise.

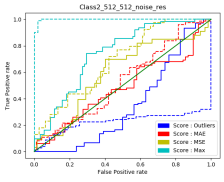

Figure 186: Performance obtained for both architectures trained in residual mode over Class2 texture images with structured noise.

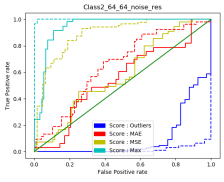

Figure 187: Performance obtained for both architectures trained in residual mode over cropped Class2 texture images with structured noise.

B.2.3 Class3

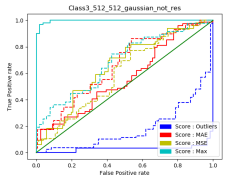

Figure 188: Performance obtained for both architectures trained in denoised mode over Class3 texture images with unstructured noise.

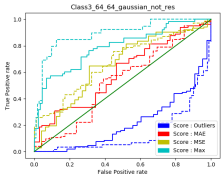

Figure 189: Performance obtained for both architectures trained in denoised mode over cropped Class3 texture images with unstructured noise.

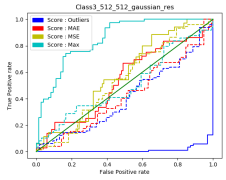

Figure 190: Performance obtained for both architectures trained in residual mode over Class3 texture images with unstructured noise.

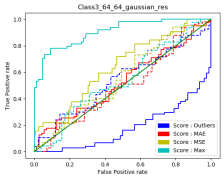

Figure 191: Performance obtained for both architectures trained in residual mode over cropped Class3 texture images with unstructured noise.

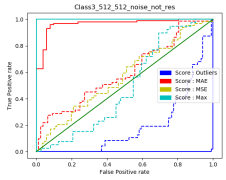

Figure 192: Performance obtained for both architectures trained in denoised mode over Class3 texture images with structured noise.

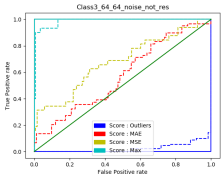

Figure 193: Performance obtained for both architectures trained in denoised mode over cropped Class3 texture images with structured noise.

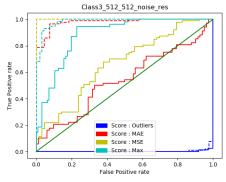

Figure 194: Performance obtained for both architectures trained in residual mode over Class3 texture images with structured noise.

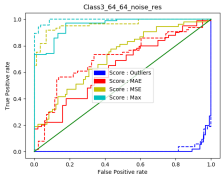

Figure 195: Performance obtained for both architectures trained in residual mode over cropped Class3 texture images with structured noise.

B.2.4 Class4

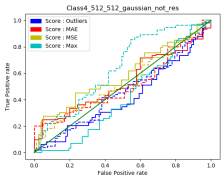

Figure 196: Performance obtained for both architectures trained in denoised mode over Class4 texture images with unstructured noise.

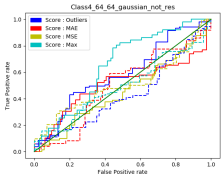

Figure 197: Performance obtained for both architectures trained in denoised mode over cropped Class4 texture images with unstructured noise.

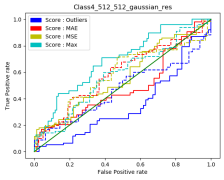

Figure 198: Performance obtained for both architectures trained in residual mode over Class4 texture images with unstructured noise.

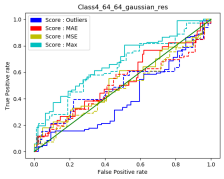

Figure 199: Performance obtained for both architectures trained in residual mode over cropped Class4 texture images with unstructured noise.

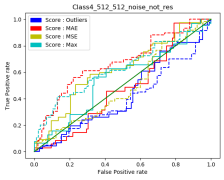

Figure 200: Performance obtained for both architectures trained in denoised mode over Class4 texture images with structured noise.

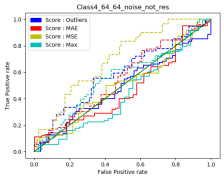

Figure 201: Performance obtained for both architectures trained in denoised mode over cropped Class4 texture images with structured noise.

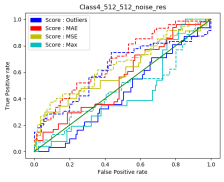

Figure 202: Performance obtained for both architectures trained in residual mode over Class4 texture images with structured noise.

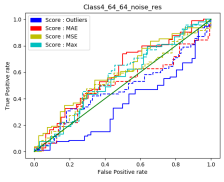

Figure 203: Performance obtained for both architectures trained in residual mode over cropped Class4 texture images with structured noise.

B.2.5 Class5

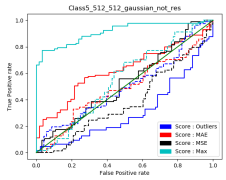

Figure 204: Performance obtained for both architectures trained in denoised mode over Class5 texture images with unstructured noise.

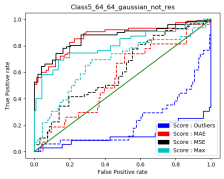

Figure 205: Performance obtained for both architectures trained in denoised mode over cropped Class5 texture images with unstructured noise.

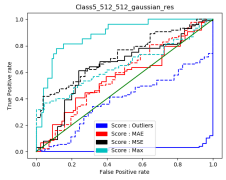

Figure 206: Performance obtained for both architectures trained in residual mode over Class5 texture images with unstructured noise.

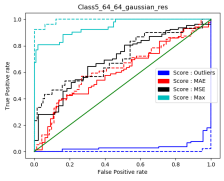

Figure 207: Performance obtained for both architectures trained in residual mode over cropped Class5 texture images with unstructured noise.

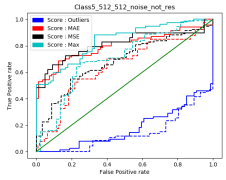

Figure 208: Performance obtained for both architectures trained in denoised mode over Class5 texture images with structured noise.

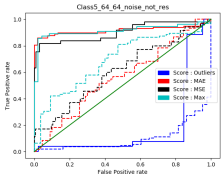

Figure 209: Performance obtained for both architectures trained in denoised mode over cropped Class5 texture images with structured noise.

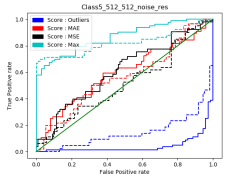

Figure 210: Performance obtained for both architectures trained in residual mode over Class5 texture images with structured noise.

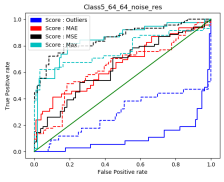

Figure 211: Performance obtained for both architectures trained in residual mode over cropped Class5 texture images with structured noise.