Technical University of Denmark

Special Course

From Low Resolution to High Resolution Sattelite images using deep learning methods

Submitted by:

Mélissande Machefer
s161362

Supervisors at DTU:

Ole Winther

Valentin Liévin

Project for Thales Alenia Space, representative:

Jean-Michel Gaucel

January 8, 2018

Contents

1 Introduction **2**

2 Tools and Technologies **3**
2.1 Python . . . 3
2.2 Tensorflow . . . 3
2.3 Using GPU's . . . 3
2.4 Jupyter Notebook . . . 4

3 Theory about Convolutional Neural Network **5**
3.1 Neural Network terminology . . . 5
3.1.1 Main Structure and Forward Propagation . . . 5
3.1.2 Activation functions . . . 6
3.2 Find a solution . . . 6
3.2.1 Minimization of the Loss . . . 7
3.3 Backward Propagation . . . 7
3.4 Optimization methods . . . 8
3.4.1 Mini batch . . . 8
3.4.2 Momentum and adaptive learning rate . . . 9
3.4.3 Algorithms with adaptive learning rates . . . 11
3.5 Convolutional Neural Network . . . 13

4 History of the project **15**
4.1 Description of the project . . . 15
4.2 History of the project . . . 16

5 Litterature involved **17**
5.1 SRCNN . . . 17
5.2 EnhanceNet . . . 18
5.3 Generative network . . . 18
5.4 Discriminative network . . . 19
5.5 Different loss functions . . . 20
5.5.1 Pixel-wise loss in the image-space . . . 20
5.5.2 Perceptual loss in feature space . . . 20
5.5.3 Texture matching loss . . . 21
5.5.4 Adversarial training loss . . . 21
5.6 Data and implementation details . . . 21
5.7 Results . . . 21

6 Method **24**
6.1 Pre-processing . . . 24
6.2 Check points . . . 24
6.3 Grey image baseline . . . 25
6.4 Regularization and optimization methods . . . 26

7 Results **28**
7.1 Data Augmentation 30
7.2 Network size 31
7.3 Batch Size Reduction 32
7.4 Dropout 33
7.5 Learning rate 33
7.6 Training over a long time with and without Data Augmentation 34
7.7 Early Stopping 38

8 Discussion and Conclusions **40**
8.1 Discussion 40
8.2 EnhanceNet 40

1 Introduction

Considering the expansion of artificial intelligence, many casual methods used in image processing (and in many other fields) are now challenged by deep learning methods. The algorithms involved in these methods are computationally demanding but the evolution of the hardware over the last decades made these methods possible to be applied to more and more problems.

At the opposite of traditional image processing methods, the heavy computation has to be done only once and the model obtained through this step, called training, will be re-used and quick to compute to predict any other sample considering the same problem.

The problem considered is firstly modeled as a neural network structure and as we have input data and a groundtruth for the output data of our problem, the neural network can learn how to solve this problem on the data and groundtruth shown. This is a learning process by success and failure as a human could do. For instance, when we learn the name of colors as a child, we are shown several times different objects of different colors, and after hearing a certain number of times the name of each color, we are able to determine the color of any object.

In this project, the goal was to challenge image processing algorithms which aim at recovering high resolution satellites images from low resolution images as input data.

We will first expose the tools and technologies involved in deep learning methods and the theory around this field. Then, we will describe in detail the project and right after, we will explain the content of the papers on which the method relies on and of course the method itself. To finish with, experiences and results on the neural network built will be described and a discussion about these results will be performed.

2 Tools and Technologies

Deep learning methods have an environment of tools that have to be mastered before implementing any algorithm. It goes from the programming language used through the acknowledged libraries to the hardware supported by the running of the algorithms. This section describes this technological environment I had to get to know and to be at ease with before and while implementing my algorithm.

2.1 Python

Python is not the fastest language for deep learning but it's a good trade off between complexity and efficiency. It's a pseudo readable pseudocode which actually relies on Fortran or C backend. Moreover, Python has good libraries to perform deep learning like Tensorflow, Theano or Kaffe and is easy to run codes on GPU's, rather than CPU, because more powerful. Python also has other libraries which are convenient for manipulation of images and arrays like numpy, PIL, opencv and h5py.

2.2 Tensorflow

tensorflow is merely an extensive library for performing numerical computations that are optimized towards running neural networks on GPU's as well as CPU's. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) that make the nodes communicating between each other. TensorFlow was originally developed by researchers and engineers working on the Google Brain Team within Google's Machine Intelligence research organization for the purposes of conducting machine learning and deep neural networks research, but the system is general enough to be applicable in a wide variety of other domains as well.

2.3 Using GPU's

The computationally intensive part of neural network is made up of multiple matrix multiplications. To make this task being realized faster we can do it simultaneously rather than doing it one after the other. This is why we use GPU rather than CPU for training a neural network.

To understand the difference, we take a classic analogy which explains the difference intuitively.

"Suppose you have to transfer goods from one place to the other. You have an option to choose between a Ferrari and a freight truck.

Ferrari would be extremely fast and would help you transfer a batch of goods in no time. But the amount of goods you can carry is small, and usage of fuel would be very high.

A freight truck would be slow and would take a lot of time to transfer goods. But the amount of goods it can carry is larger in comparison to Ferrari. Also, it is more fuel efficient so usage is lower.

So which would you chose for your work?

Obviously, you would first see what the task is; if you have to pick up your girlfriend urgently, you would definitely choose a Ferrari over a freight truck. But if you are moving your home, you would use a freight truck to transfer the furniture."

So a GPU is a hardware on which we run the deep learning algorithms. In 2006, Nvidia came out with a high level language CUDA which helped to easily launch deep learning algorithms. I first tried to learn it but then chose tensorflow which contains almost everything operation you need to launch on a GPU.

2.4 Jupyter Notebook

Jupyter Notebook is a client-server application that allows to edit and run a script via a web browser. The script can be stored on a remote server and is accessed through internet. It is really useful to get an easy edition of scripts and not go through **nano** or **pico** commands via terminal which lead to text editors very limited.

3 Theory about Convolutional Neural Network

3.1 Neural Network terminology

3.1.1 Main Structure and Forward Propagation

An ANN is based on a collection of connected units called artificial neurons. Each connection between neurons can transmit a signal to another neuron. The receiving neuron can process the signal and then signal downstream neurons connected to it. Neurons may have state, generally represented by real numbers, typically between 0 and 1. Neurons and synapses may also have a weight that varies as learning proceeds, which can increase or decrease the strength of the signal that it sends downstream. Further, they may have a threshold such that only if the aggregate signal is below (or above) that level is the downstream signal sent, represented by the activation functions ([2]).

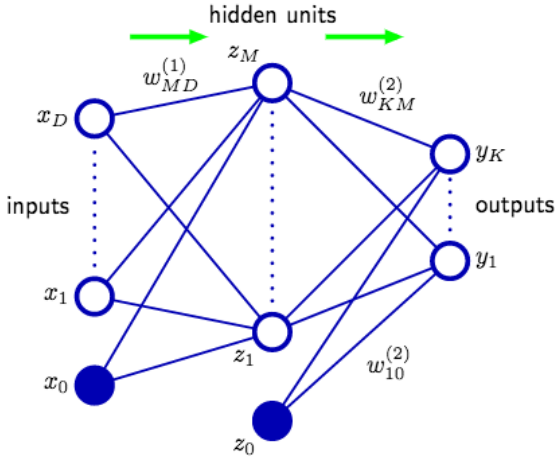


Figure 1: Network diagram for a two layers neuron network. The input, hidden, and output variables are represented by nodes, and the weight parameters are represented by links between the nodes, in which the bias parameters are denoted by links coming from additional input and hidden variables x_0 and z_0. Arrows denote the direction of information flow through the network during forward propagation [4]

Figure 1 shows the traditional structure of a fully connected neural network. The input x_i with i from 1 do D (where D is dimension of input data) fetch the neural network which has output y_j with j from 1 to K (where K is the number of outputs we want). Each node of the inputs is multiplied by a certain number of weights w and an activation function, which is a non linear function is then applied to obtain the node of the next layer (z). Therefore, the next layer (hidden layer) has nodes obtained thanks to the contributions to all nodes from the input data. The last layer which gives the output is controled by a linear function. If M is the number of nodes we have in the hidden layer (the one between the input and

output layer), we can re write the connection between the input layer and the hidden layer of the network described in Figure 1 as:

$$a_j = w_{j0}^{(1)} x_0 + \sum_{i=1}^{M} x_i w_{ji}^{(1)} \tag{1}$$

with $z_j = h(a_j)$ where h is the non linear activation function.

We can of course add hidden layers as much as we want to build a more complex model.

3.1.2 Activation functions

There are several non linear activation functions. These non linearities are important because, they are the keys of success for these neural networks. Indeed, the non linearity aims at not giving a shallow network and only let the information downstreaming if its worth to. The state of the art activation function is the ReLU defined as, for any a:

$$h(a) = \max(a, 0) \tag{2}$$

Other activation functions are the sigmoid, often used for small networks, defined as:

$$h(a) = \frac{1}{1 + \exp(-a)} \tag{3}$$

One last activation function is the hyperbolic tangent, also used for small networks, defined as:

$$h(a) = \frac{\exp(a) - \exp(-a)}{\exp(a) + \exp(-a)} \tag{4}$$

3.2 Find a solution

All the weights and biases parameters we talked about earlier are set randomly so the information in output is going to be pretty random after having been streamed through the network. The output produced can be compared with the groundtruth using a cost function E. If we face a regression problem, we will use the regular L^2 euclidean norm but if it is a classification problem, we use another cost. Indeed, the last layer of the network, in a classification problem, is obtained after been passed through a function called softmax defined as:

$$softmax(a_k) = \frac{\exp(a_k)}{\sum_j \exp(a_j)} \tag{5}$$

where k is a node of the last layer.

This function gives a probability value to each node of the output and the class chosen is the node with the highest probability.

The cost considered is then the cross entropy loss defined as:

$$E = -\sum_j t_j \log(z_m) \tag{6}$$

where t is the ground truth.

3.2.1 Minimization of the Loss

The question is now how to minimize this cost (loss) by changing the value of weights and biases, that it to say the model $\mathbf{w}$. The solution is to find a direction $\Delta\mathbf{w}^\tau$ which is going to decrease the cost E between the iteration τ and $\tau + 1$ such that

$$\mathbf{w}^{\tau+1} = \mathbf{w}^\tau + \Delta\mathbf{w}^\tau \tag{7}$$

This direction is given by the gradient of the cost and this why we finally update the model such as

$$\mathbf{w}^{\tau+1} = \mathbf{w}^\tau + \eta \times \mathbf{grad}(E(\mathbf{w}^\tau)) \tag{8}$$

where η is the learning rate that means how much the next model parmeters are going to follow this direction.

3.3 Backward Propagation

Computing the gradient requires to compute partial derivatives with respect to any weight parameter (or bias, which can be considered as weight in reality) of the model. However, all the layers are dependent of each other and to compute these derivatives we need to back propagate the information. Following the schema on Figure 1 and the chain rule:

$$\frac{\partial E}{\partial w_{ij}} = \frac{\partial E}{\partial a_k}\frac{\partial a_k}{\partial a_j}\frac{\partial a_j}{\partial w_{ij}} \tag{9}$$

and the corresponding backpropagation equations are, at layer l,

$$\delta_j^l = \frac{\partial E}{\partial a_j^l} = h'(a_j^l)\sum_k w_{kj}^{l+1}\delta_k^{l+1} \tag{10}$$

$$\frac{\partial E}{\partial w_{ji}^l} = \delta_j^l z_i^{l-1} \tag{11}$$

$$\frac{\partial E}{\partial b_j^l} = \delta_j^l \tag{12}$$

where b represents biases.

3.4 Optimization methods

As explained just before, the backward propagation aims at adjusting the weights so that the loss function becomes smaller and smaller. Some optimization methods exist to either decrease it quickly or avoid finding a local minimum. This optimization methods go through optimizing the algorithm of the neural network itself but also include finding strategies for initializing parameters or building adaptive learning rates.

One thing to clearly understand in artificial neuron network is that for each iteration, we train the network on a part of the input dataset (training set) and we take another part of the input data to validate this result (validation set). Indeed, there is a need to test the network on "new data" unseen by the network. Therefore, at each iteration, we can observe the validation output getting better and better (see Figure 2a). This implies that the network can work really well on the training set and that the loss can be super low (because we force the weights to "correspond" to this set) but once it is use to predict an output on the validation set, never seen by the network, the loss could be really high: this is situation is called overfitting. As there might be many samples in a training set, the loss is simply the averaged sum of the loss of each sample in the batch.

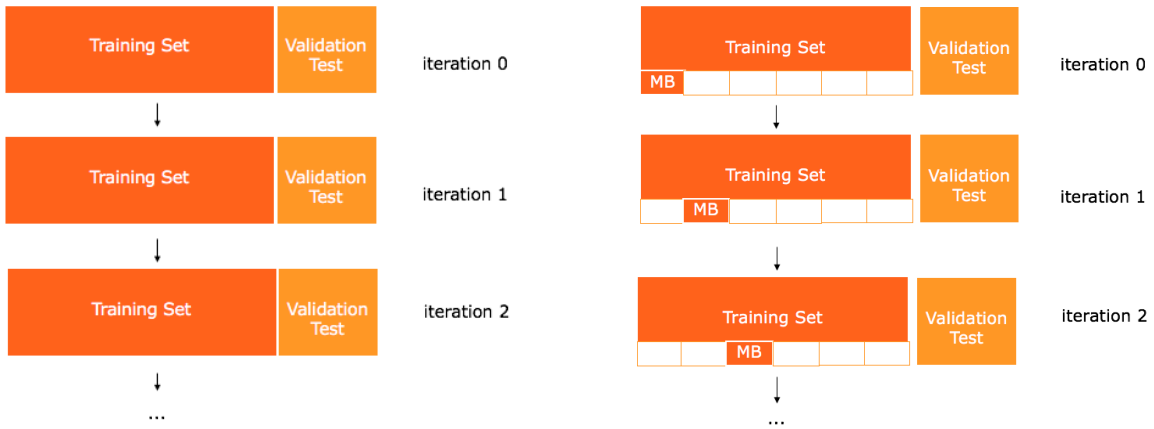


Figure 2: How the input dataset is gone through during the training. (a) At each iteration each sample of the training set (bactch) is plugged to the network and another part of the available dataset is kept for the validation, (b) At each iteration, only a few samples (minibatch MB) og the batch is plugged to the network and another part of the available dataset is kept for the validation

3.4.1 Mini batch

When we are lucky enough to get a huge amount of data, processing all the samples at each iteration can be "decades" consuming. Therefore, an optimization method, to reduce the time processing, is to only train the network on part of the training set that we will call batch (see Figure 2b). This technique is call minintach. Once all the training set has been gone through, after a certain number iterations, we call this an epoch. This method leads

to a rapid computation of approximate estimate of the gradient rather than slowly computing the exact gradient as it's done with using the full training set at each iteration. Larger batches provide a more accurate estimate of the gradient, but with less than linear returns. Small batches can offer a regularizing effect ([10]), perhaps due to the noise they add to the learning process. Generalization error is often best for a batch size of 1. Training with such a small batchsize might require a small learning rate to maintain stability due to the high variance in the estimate of the gradient. The total runtime can be very high due to the need to make more steps, both because of the reduced learning rate and because it takes more steps to observe the entire training set.

Using minibatches instead of the full training set is called Stochastic Gradient Descent (SGD) and signifies that it is possible to obtain an unbiased estimate of the gradient by taking the average gradient on a minibatch of m examples. Figure 3 shows the corresponding algorithm performs to obtain a model thanks to the neural network built. The stopping criterion is actually a certain number of epochs. θ only represents the parameters of the network, meaning all the possible weights, f is the model function we try to evaluate.

Algorithm 8.1 Stochastic gradient descent (SGD) update at training iteration k

Require: Learning rate ϵ_k.
Require: Initial parameter $\boldsymbol{\theta}$
while stopping criterion not met **do**
Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
Compute gradient estimate: $\hat{\boldsymbol{g}} \leftarrow +\frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \epsilon \hat{\boldsymbol{g}}$
end while

Figure 3: Stochastic gradient descend (SGD)[4]

3.4.2 Momentum and adaptive learning rate

The learning rate can be fixed with SGD for each iteration but it can also be interesting to change it over the time. Indeed, the SGD gradient estimator introduces source of noise (because of the random sampling of the elements present in the minibatch) that doesn't vanish even when we arrive at a minimum. Conversely, the true gradient of the total cost function becomes small and then 0 when we approach and reach a minimum using batch gradient descent, so batch gradient descent can use a fixed learning rate.

It is common to decay the learning rate linearly until iteration τ such as

$$\epsilon_k = (1 - \alpha)\epsilon_0 + \alpha\epsilon_\tau \tag{13}$$

with $\alpha = \frac{k}{\tau}$. After τ iterations, it is common to leave ϵ constant.

Choosing this learning rate is more "an art than a science" and the guidance over this topic should be regarded with some skepticism.

However, some advice might keep being reliable. The parameters that need to be fixed in this scheme are $\epsilon_0, \epsilon_\tau$ and τ. Usually τ may be set to the number of iterations required to make a few hundred passes through the training set. Usually, ϵ_τ should be set to roughly 1% the value of ϵ_0. The main question is how to set ϵ_0. If it is too large, the learning loss shows violent oscillations up to a certain point but the loss decreases a lot at the beginning. If the learning rate is too low, the learning rate is too slow and we might get a high loss even after a huge amount of iterations.

The learning with SGD can be pretty slow so Momentum method is often preferred. The momentum algorithm accumulates an exponentially decaying moving average of past gradients and continues to move in their direction. Figure 4 explains how works this momentum method.

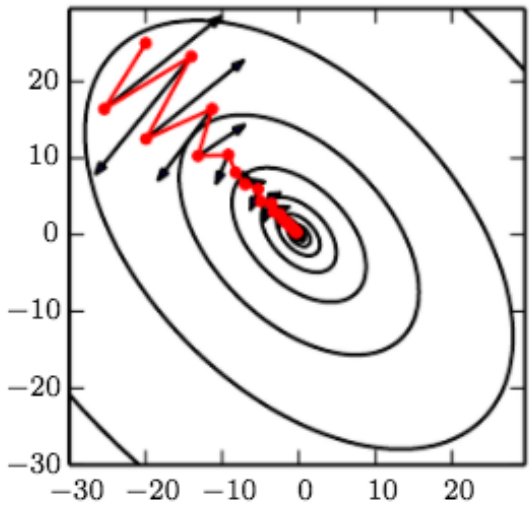


Figure 4: Momentum aims primarily to solve two problems: poor conditioning of the Hessian matrix and variance in the stochastic gradient. Here, we illustrate how momentum overcomes the first of these two problems. The contour lines depict a quadratic loss function with a poorly conditioned Hessian matrix. The red path cutting across the contours indicates the path followed by the momentum learning rule as it minimizes this function. At each step along the way, we draw an arrow indicating the step that gradient descent would take at that point. We can see that a poorly conditioned quadratic objective looks like a long, narrow valley or canyon with steep sides. Momentum correctly traverses the canyon lengthwise, while gradient steps waste time moving back and forth across the narrow axis of the canyon.[4]

Figure 5 shows the momentum algorithm where the velocity v accumulates the gradient elements. The larger α is relative to ϵ, the more previous gradients affect the current direction. The step size is largest when many successive gradients point in exactly the same direction.

Algorithm 8.2 Stochastic gradient descent (SGD) with momentum

Require: Learning rate ϵ, momentum parameter α.
Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient estimate: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \epsilon \boldsymbol{g}$
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$
  end while

Figure 5: Momentum algorithm [4]

3.4.3 Algorithms with adaptive learning rates

The cost is often highly sensitive to some directions in parameter space and insensitive to others and the momentum algorithm makes actually add another parameter which also can be concerned by this issue. A tunable learning rate throughout the course of learning can answer this sensitiveness issue. We present here some algorithms that include adaptive learning rate.

The **delta-bar-delta** algorithm is coarse approach of individual adaptive learning rate for each parameter. The simple idea is that if the partial derivative of the loss, with respect to given model parameter, remains the same sign, then the learning rate should increase. If the partial derivative with respect to that parameter changes sign,then the learning rate should decrease. This kind of rule applies to the batch optimization so there was a need of finding an equivalent for minibatch based algorithms.

The **AdaGrad** algorithm scales parameters inversely proportional to the square root of the sum of all of their historical squared values. Therefore, parameters with large partial derivative of the loss will have a rapid decrease in their learning rate whereas parameters with smaller partial derivative will have a small decrease in their learning rate. This method is not that suitable for deep networks because it had been shown that it leads to a too quick decrease in the learning rate in the beginning of the training.

The **RMSProp** algorithms modifies AdaGrad to perform better in the non-convex setting by changing the gradient accumulation into an exponentially weighted moving average. This method is really useful to discard history from the extreme past.

The **Adam** algorithm is a variant on the combination of RMSProp and momentum with a few important distinctions. In RMSProp, the momentum is applied to the rescaled gradients. Adam includes bias corrections to the estimates of both the first-order moments (the momentum term) and the (uncentered) second-order moments to account for their initialization at the origin. Adam is generally regarded as being fairly robust to the choice of hyper pa-

rameters, though the learning rate sometimes needs to be changed from the suggested default.

Figure 6 shows the RMSProp algorithm shown combined with the Nesterov momentum and Figure 7 shows the Adam algorithm.

Algorithm 8.6 RMSProp algorithm with Nesterov momentum

Require: Global learning rate ϵ, decay rate ρ, momentum coefficient α.

Require: Initial parameter $\boldsymbol{\theta}$, initial velocity $\boldsymbol{v}$.

Initialize accumulation variable $\boldsymbol{r} = \boldsymbol{0}$

while stopping criterion not met **do**

Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.

Compute interim update: $\tilde{\boldsymbol{\theta}} \leftarrow \boldsymbol{\theta} + \alpha \boldsymbol{v}$

Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\tilde{\boldsymbol{\theta}}} \sum_i L(f(\boldsymbol{x}^{(i)}; \tilde{\boldsymbol{\theta}}), \boldsymbol{y}^{(i)})$

Accumulate gradient: $\boldsymbol{r} \leftarrow \rho \boldsymbol{r} + (1 - \rho) \boldsymbol{g} \odot \boldsymbol{g}$

Compute velocity update: $\boldsymbol{v} \leftarrow \alpha \boldsymbol{v} - \frac{\epsilon}{\sqrt{\boldsymbol{r}}} \odot \boldsymbol{g}$. ($\frac{1}{\sqrt{\boldsymbol{r}}}$ applied element-wise)

Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \boldsymbol{v}$

end while

Figure 6: RMSProp combined with Nesterov momentum [4]

Algorithm 8.7 The Adam algorithm

Require: Step size ϵ (Suggested default: 0.001)
Require: Exponential decay rates for moment estimates, ρ_1 and ρ_2 in $[0, 1)$. (Suggested defaults: 0.9 and 0.999 respectively)
Require: Small constant δ used for numerical stabilization. (Suggested default: 10^{-8})
Require: Initial parameters $\boldsymbol{\theta}$
  Initialize 1st and 2nd moment variables $\boldsymbol{s} = \mathbf{0}$, $\boldsymbol{r} = \mathbf{0}$
  Initialize time step $t = 0$
  while stopping criterion not met **do**
    Sample a minibatch of m examples from the training set $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ with corresponding targets $\boldsymbol{y}^{(i)}$.
    Compute gradient: $\boldsymbol{g} \leftarrow \frac{1}{m} \nabla_{\boldsymbol{\theta}} \sum_i L(f(\boldsymbol{x}^{(i)}; \boldsymbol{\theta}), \boldsymbol{y}^{(i)})$
    $t \leftarrow t + 1$
    Update biased first moment estimate: $\boldsymbol{s} \leftarrow \rho_1 \boldsymbol{s} + (1 - \rho_1)\boldsymbol{g}$
    Update biased second moment estimate: $\boldsymbol{r} \leftarrow \rho_2 \boldsymbol{r} + (1 - \rho_2)\boldsymbol{g} \odot \boldsymbol{g}$
    Correct bias in first moment: $\hat{\boldsymbol{s}} \leftarrow \frac{\boldsymbol{s}}{1 - \rho_1^t}$
    Correct bias in second moment: $\hat{\boldsymbol{r}} \leftarrow \frac{\boldsymbol{r}}{1 - \rho_2^t}$
    Compute update: $\Delta\boldsymbol{\theta} = -\epsilon \frac{\hat{\boldsymbol{s}}}{\sqrt{\hat{\boldsymbol{r}}} + \delta}$ (operations applied element-wise)
    Apply update: $\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} + \Delta\boldsymbol{\theta}$
  end while

Figure 7: Adam algorithm [4]

3.5 Convolutional Neural Network

The neural network described above is commonly called fully connected neural network. As there are lots of matrix multiplications which are computationally demanding, we need to evaluate the size of a full neuron network in terms of parameters. If we work on medium images, for instance 200×200 in RGB, so 3 channels, and with 1000 hidden units we will have to evaluate in our model $200 \times 200 \times 3 \times 1000 = 120000000$ which is already completely crazy. To avoid this explosion, we use convolution instead of multiplication. We choose weights tensors of size $w \times w \times n_f$ where w is the spatial size of the tensor and n_f the number of filters and we convolve these filters with the input image. Therefore, each neuron is connected to all pixels but pixels which are far away are not correlated. The main idea is to look locally and share weights for all pixels. To connect pixel at longer distances, adding more layers to the network (more weight tensors to convolve) is the solution. Figure 8 illustrates the comparison between fully connected and convolutional neuron networks.

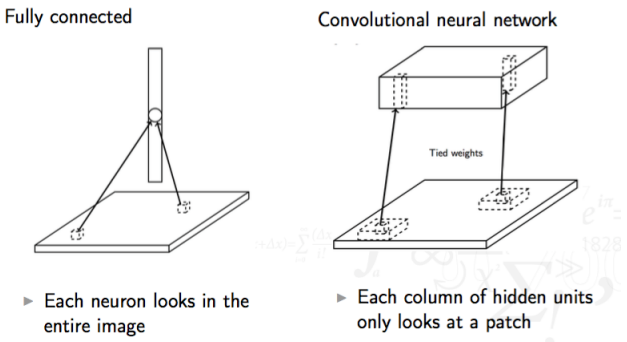


Figure 8: Comparison between fully connected and convolutional neuron networks [4]

Even if the multiplication of matrices is replaced by convolutions,there is still the detector stage with non linear activation functions but another step is added which is called the pooling stage. This pooling stage is made for discarding spatial information as the usual neural network (not the one we want to build for our problem!) only aims at retrieving the spectral information, meaning which output label or value is associated to the entire input image. To do so, the pooling stage reduces the number of features by taking the maximum, average or weighted average among a certain number of parameters. Figure 9 shows the sequence of the stages in a convolutional neural network.

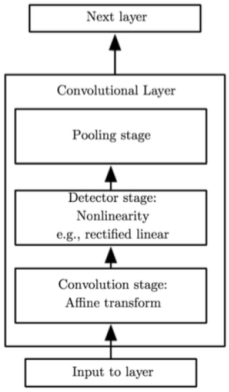


Figure 9: Stages in a convolutional neural network[4]

4 History of the project

4.1 Description of the project

Satellites usually acquire two types of images: a **Panchromatic** image of one channel captured over a wide spectrum covering all visible wavelength in High Resolution (HR) and 4 multi spectral images with disjoint spectral bands covering visible (blue, green and red) and near infra red but with a lower resolution of 1/4 the resolution of the panchromatic channel.

These products are provided in two different formats:

- the **Bundle** format containing the panchromatic image and another colored image (4 bands) but with a 4× worse resolution.
- the **Pansharpened** format containing a colored image (4 bands RGBA) with the spatial resolution of the panchromatic band (HR).

Going from **Bundle** format to **Pansharpened** format is usually realized through an algorithm of fusion of spatial and spectral data.

The goal of this project is to challenge this algorithm (chosen by Thales Alenia Space TAS-F) with deep learning methods.

Figure 10 shows a graph summarizing the goal of this project. Considering this view point, we therefore have an algorithm with an image of $nb_{ci} = 5$ channels in input (1 panchromatic and 4 narrow spectral bands) and a high resolution of $nb_{co} = 4$ channels in output.

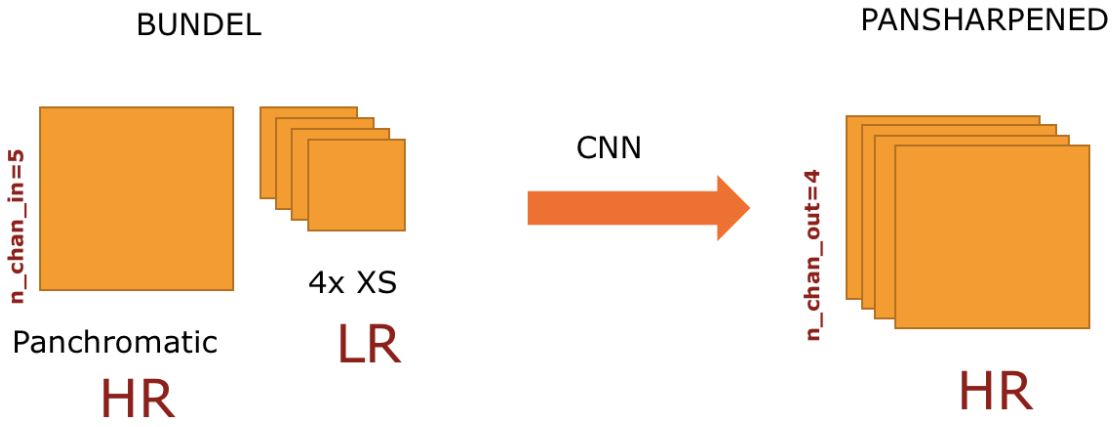


Figure 10: Project Description

TAS-F provided the input dataset composed of the bundel format and the corresponding ground truth image of high resolution.

4.2 History of the project

I have been working on this project, starting from scratch in deep learning, during the summer of 2017 for the course "30220 - Synthesis in Earth and Space Physics". The idea was to play around with the algorithm of UNET([6]) used as a regression algorithm and not as a multi class classification algorithm as it is originally used for. The results I had were uncertains in a way that the goal was a bit ambitious for the time allowed to the project. Therefore, it could be seen that "something" was happening in the neural network built, but results couldn't lead to any conclusion and the code was not deployable an reusable by Thales. To sum up, it was more a draft to a more mature and research oriented way to tackle this problem.

For this special course, I choose to use a basic method, named SRCNN, developed in [3], which I already had the opportunity to come across last summer but my implementation was wrong. I choose to re use this paper, because it aims at building RGB HR images from RGB LR images. Therefore, the goal is the same than our problem and the structure of the network (detailed in 5.1) is simple so it was a good baseline to begin with. Indeed, the only difference between SRCNN [3] and our problem is the number of channels and the resolution of the input data. The idea of this special course was to be able to provide an algorithm that works with a reasonable accuracy and it has been done through re-coding everything from the pre-processing through the network itself to the prediction of any input. The focus was particularly on providing clean, easy and deployable algorithm with highlights on optimization.

During the special course itself, Valentin, my supervisor, advised me to read the EnhanceNet paper ([8]) which has been published on 31/07/2017 and which is therefore very recent. This paper seems to me a great discovery and it could lead to way further improvements than the SRCNN algorithm, but I didn't have time to implement it. Details of EnhanceNet are provided in 5.2.

5 Litterature involved

5.1 SRCNN

The paper [3] suggets a mapping between low and high resolution image using a CNN that takes the low-resolution image as the input and outputs the high-resolution one.

The algorithm first upscales the low resolution image to the desire high resolution through an interpolation. Let's call the low resolution image $\mathbf{Y}$, the goundtruth high resolution image $\mathbf{X}$ and the the mapping F that we want to learn to recover from $\mathbf{Y}$ an image F($\mathbf{Y}$) as similar as possible from $\mathbf{X}$.

This network is divided in 3 parts: the patch extraction and representation, the non linear mapping and finally the reconstruction. These are basically 3 coarse layers of the network.

The patch extraction and representation simply is a usual sequence of convolution and ReLU so it can be written as, if we call F_1 this step:

$$F_1(Y) = \max(0, W_1 * Y + B_1) \tag{14}$$

where $*$ is the convolutional operator, W_1 and B_1 respectively represent the filters and biases. W_1 corresponds to n_1 filters of support $c \times f_1 \times f_1$ where c is the number of channels in the input image and f_1 is the spatial size of a filter.

The next step is to map each of these n_1-dimensional vectors into an n_2-dimensional one and therefore create a non linearity. We therefore apply another sequence of convolution-ReLU which follows, if we call this step F_2:

$$F_2(Y) = \max(0, W_2 * F_1(Y) + B_2) \tag{15}$$

where W_2 and B_2 respectively represent the filters and biases. W_2 corresponds to n_2 filters of support $n_1 \times f_2 \times f_2$ and f_2 is the spatial size of a filter. We basically can add more layers for this step to increase the non linearity but the authors prove in this paper that a deep network doesn't really improve the performance and, of course, takes more time to run.

The last step is to reconstruct the high resolution image with a linear step only using convolution which boils down to:

$$F(Y) = W_3 * F_2(Y) + B_3 \tag{16}$$

where W_3 and B_3 respectively represent the filters and biases. W_3 corresponds to c filters of support $n_2 \times f_3 \times f_3$ and f_3 is the spatial size of a filter.

The loss of this neural network is a Euclidean L_2 norm as this problem is seen as a regression problem.

5.2 EnhanceNet

The EnhanceNet paper [8], recently published in late July 2017, aims at high resolution reconstruction from low resolution images, as it's done in the SRCNN [3] paper. EnhanceNet paper highlights that the PSNR measure (Peak Signal to Noise Ratio, see (23)) coorelates poorly with human perception. Therefore, this paper introduces a new perceptual loss, especially considering texture perception of human eye after reconstruction. In parallel to a better evaluation of the goodness of the reconstruction, the improvement of image quality is done in this paper using automated texture synthesis, to be coupled to the new perceptual loss, and adversarial training.

Adversarial training is a combination of two parallel networks: a generative network which does the main job (reconstructing high resolution images from low resolution) and therefore generates synthetic images and a discriminative network which teaches the network to recognize a synthetic reconstruction from a real image.

5.3 Generative network

The generative network aims at reconstructing high resolution features. The strategy of this CNN is a bit different from SRCNN [3] as the upsampling of the image is done in the middle of the network to avoid redundancies and high computational costs. Figure 11 shows the architecture of the network, and due to the upsampling done later in the network (using two nearest neighbour upsampling), most of the computation is done in the low resolution image space. A deconvolution layer is added after upsampling to retrieve desired number of channels in output (3 for RGB). Another interesting point is that EnhanceNet works at estimating the residual image between the true high resolution image and the basic upsampled image using cubic interpolation. Residual blocks are beneficial for faster convergence compared to stack convolutional layers and it helps stabilizing training and reduce color shifts in the output during training.

Output size	Layer
$w \times h \times c$	Input I_{LR}
$w \times h \times 64$	Conv, ReLU
	Residual: Conv, ReLU, Conv
	...
$2w \times 2h \times 64$	2x nearest neighbor upsampling Conv, ReLU
$4w \times 4h \times 64$	2x nearest neighbor upsampling Conv, ReLU
	Conv, ReLU
$4w \times 4h \times c$	Conv
	Residual image I_{res}
	Output $I_{\text{est}} = I_{\text{bicubic}} + I_{\text{res}}$

Table 1. Our generative fully convolutional network architecture for 4x super-resolution which only learns the residual between the bicubic interpolation of the input and the ground truth. We use 3×3 convolution kernels, 10 residual blocks and RGB images ($c = 3$).

Figure 11: Generative network [8]

5.4 Discriminative network

The disciminative network is a classidication CNN which has two possible output: "yes" the image fed is a real high resolution image and "no" the image fed is not a real high resolution image and is simply reconstructed.

As seen in Figure 12, the goal of this network is to discard spatial information to only obtain fetatures describing the state of the image fed in input. This process is done by using successive convolution layers but not coupled to pooling layers to reduce spatial dimension (as done in VGG [9] from which this network takes inspiration from) but coupled to strided convolutions. The activation function is not traditional ReLU but leaky ReLU [5]. Of course the activation function of the last layer is a sigmoid to produce the label 0 or 1.

Output size	Layer
$128 \times 128 \times 3$	Input I_{est} or I_{HR}
$128 \times 128 \times 32$	Conv, lReLU
$64 \times 64 \times 32$	Conv stride 2, lReLU
$64 \times 64 \times 64$	Conv, lReLU
$32 \times 32 \times 64$	Conv stride 2, lReLU
$32 \times 32 \times 128$	Conv, lReLU
$16 \times 16 \times 128$	Conv stride 2, lReLU
$16 \times 16 \times 256$	Conv, lReLU
$8 \times 8 \times 256$	Conv stride 2, lReLU
$8 \times 8 \times 512$	Conv, lReLU
$4 \times 4 \times 512$	Conv stride 2, lReLU
8192	Flatten
1024	Fc, lReLU
1	Fc, sigmoid
1	Estimated label

Table 1. The network architecture of our adversarial discriminative network at 4x super-resolution. As in the generative network, we exclusively use 3×3 convolution kernels. The network design draws inspiration from VGG [17] but uses leaky ReLU activations [11] and strided convolutions instead of pooling layers [13].

Figure 12: Discriminative network [8]

5.5 Different loss functions

5.5.1 Pixel-wise loss in the image-space

This loss is the one used in [3] and is defined as

$$L_E = ||I_{est} - I_{HR}||_2^2 \tag{17}$$

where I_{HR} is the true high resolution image and I_{est} is the image estimated in outpout of the neural network and

$$||I||_2^2 = \frac{1}{whc} \sum_{w,h,c} (I_{w,h,c})^2 \tag{18}$$

5.5.2 Perceptual loss in feature space

Both I_{est} and I_{HR} are first mapped into a feature space by a differentiable function ϕ and then the loss is computed such as:

$$L_P = ||\phi(I_{est}) - \phi(I_{HR})||_2^2 \tag{19}$$

The function ϕ is obtained using VGG-19 network and consists of "stacked convolutions coupled with pooling layers to gradually decrease the spatial dimension of the image and to extract higher-level features in higher layers"([8]). Only the second and fifths pooling layers are used and the MSE is computed on their feature activation which allows to capture both low-level and high-level features.

5.5.3 Texture matching loss

Knowing the target texture image, the output of the texture generation network is generated using the loss:

$$L_P = ||G(|\phi(I_{est})) - G(\phi(I_{HR}))||_2^2 \tag{20}$$

where G is the gram matrix. ϕ is used as statistics because it is extracted from a pre-trained network to the target texture (VGG-19). The texture loss is only computed on small patches during the training to avoid high computational costs and to be sure that the texture is the same locally. Empirically, a size of 16x16 has been found to be a good trade off between "faithful texture generation and the overall perceptual quality of the images" ([8]).

5.5.4 Adversarial training loss

In adversarial training loss, the generative network G is trained to minimize the loss:

$$L_A = \log(D(G(z))) \tag{21}$$

while the discriminative network D is trained to minimize the loss:

$$L_D = \log(D(x)) - \log(1 - D(G(z))). \tag{22}$$

where x are the real images and $G(z)$ the images outputed by the generative network.

5.6 Data and implementation details

The input patches size for the generative network is 32x32 and the network is trained 24 hours maximum and around 200k images are used. All images are cropped centrally to a square and then downsampled to 256x256 to reduce noise and JPEG artifacts.

The issue with the metrics in this type of problem is that they are not able to capture correct perceptual quality of the models produced. Indeed, the results for EnhanceNet don't correspond between PSNR evaluation and human eye image quality assessment. This is also an issue using SRCNN in our problem. To deal with this problem, the authors of EnhanceNet conducted a survey over people to ask them to assess the quality of the high resolution images using EnhanceNet, with all different losses and other papers and EnhanceNet was always the winning one and from far.

5.7 Results

In the paper, different combinations of loss are tested as well as adversarial training against baseline MSE which would be the closest of SRCNN implementation [3]. Figure 13 details the different tests and give them relevant names.

Network	Loss	Description
ENet-E	$\mathcal{L}_E$	Baseline with MSE
ENet-P	$\mathcal{L}_P$	Perceptual loss
ENet-EA	$\mathcal{L}_E + \mathcal{L}_A$	ENet-E + adversarial
ENet-PA	$\mathcal{L}_P + \mathcal{L}_A$	ENet-P + adversarial
ENet-EAT	$\mathcal{L}_E + \mathcal{L}_A + \mathcal{L}_T$	ENet-EA + texture loss
ENet-PAT	$\mathcal{L}_P + \mathcal{L}_A + \mathcal{L}_T$	ENet-PA + texture loss

Table 2. The same network trained with varying loss functions.

Figure 13: Loss functions [8]

Figure 14 details the PSNR for the architecture of EnhanceNet trained with different combinations of losses at 4x super resolution. We observe that the baseline with MSE is actually the one obtaining the highest PSNR (in bold).

Dataset	Bicubic	ENet-E	ENet-P	ENet-EA	ENet-PA	ENet-EAT	ENet-PAT
Set5	28.42	**31.74**	28.28	28.15	27.20	29.26	28.56
Set14	26.00	**28.42**	25.64	25.94	24.93	26.53	25.77
BSD100	25.96	**27.50**	24.73	25.71	24.19	25.97	24.93
Urban100	23.14	**25.66**	23.75	23.56	22.51	24.16	23.54

Table 3. PSNR for our architecture trained with different combinations of losses at 4x super resolution. ENet-E yields the highest PSNR values since it is trained towards minimizing the per-pixel distance to the ground truth. The models trained with the perceptual loss all yield lower PSNRs as it allows for deviations in pixel intensities from the ground truth. It is those outliers that significantly lower the PSNR scores. The texture loss increases the PSNR values by reducing the artifacts from the adversarial loss term. Best results shown in bold.

Figure 14: Loss functions [8]

But having a look to the high resolution images in output, in Figure 15, we realize that the PSNR measure doesn't correlate with visual results. Indeed, the ENet-PAT gives the closest results from the true high resolution image and ENet-P and ENet-PA gives unrealistic artificats wheres ENet-E (MSE baseline) gives still really blurry image.

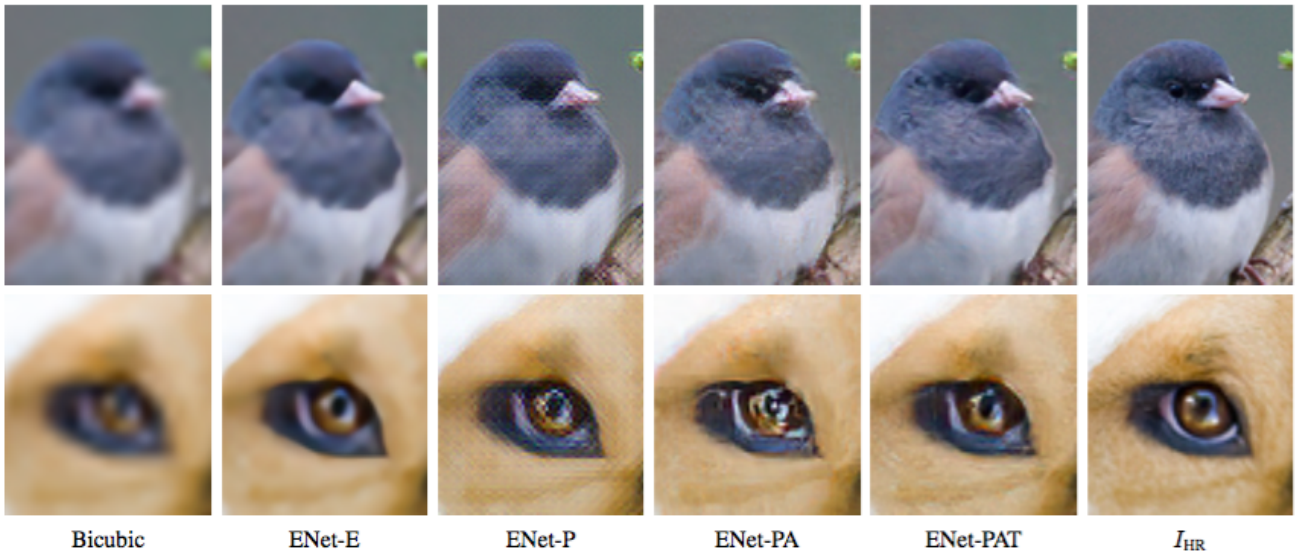


Figure 15: Comparison of the different loss in terms of output HR image [8]

To finish with, the paper shows a comparison of the output of their algorithms towards SRCNN, and we can see in Figure 16 that using EnhanceNet could lead to results way better than the SRCNN implementation we used until now for our problem.

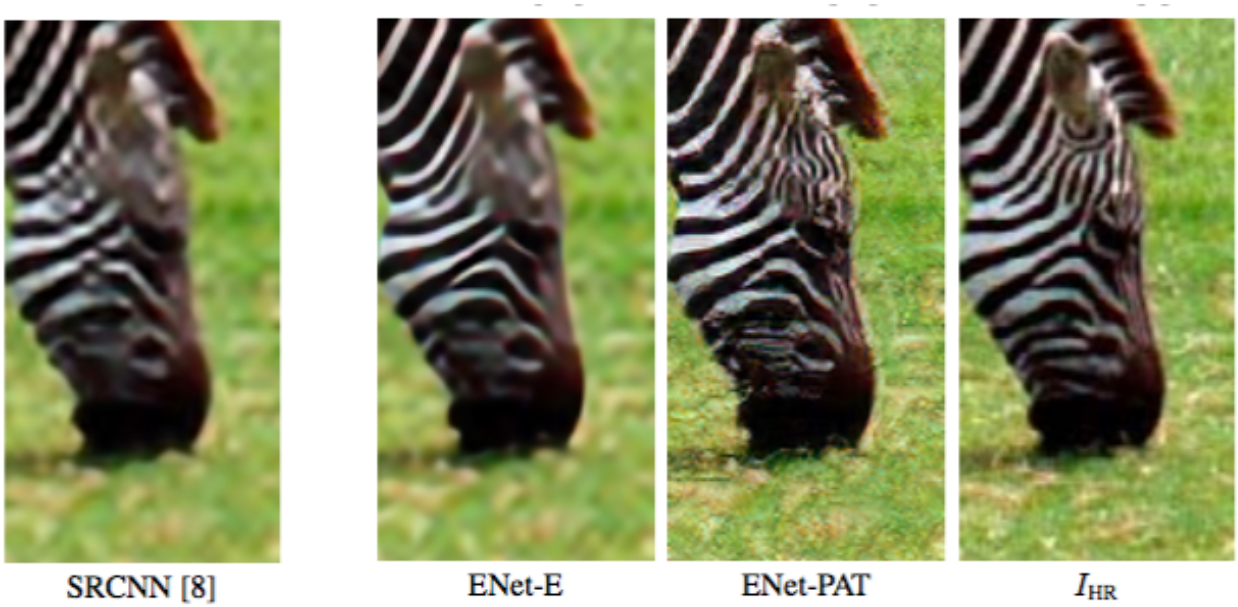


Figure 16: Comparison of EnhanceNet and SRCNN

6 Method

6.1 Pre-processing

One of the major problems of the dataset used in this problem is that it is really big. Indeed, a dataset of only one image is, after upsampling step of the 4 **XS** channels, at least, $12000 \times 12000 \times 9 = 1296000000$ as $nb_{ci} + nb_{co} = 9$. Therefore, storing these data in memory is really costly, and as the training set has to be separated from the validation and verification set, it requires at least 3 images of this size. Memory errors were therefore a major issue that was happening while trying to extract patches from these multi channel images. A choice has been done to put one or more distinct images in each of the type of set: TRAINING, VERIFICATION and VALIDATION so that no overfit occurs. The pre-processing is done in two steps. First step is to extract sub images of maximum 6000×6000 spatial resolution from any high spatial resolution input dataset, which is the maximal dimension the memory can hold considering how consuming is the following step. The next step is to extract small patches from all of these sub images and storing them under **.h5** format.

The SRCNN paper ([3]) already performed a data number requirement to obtain enough good results. They recommend, after experimentation, to use patches of 33×33 with stride number of 14 and a number of at least 30000 patches in the training set.

This way of storing all the patches requires a huge space on hard disk and it could be optimized with extracting random patches from an image (composed of actually 4 "images": pansharpened, panchromatic and 4 XS) inside each batch and realizing the pre processing inside of each batch (upsampling and concatenation of the necessary channels).

6.2 Check points

To avoid writing messy and non deployable code and which leads to implementation errors, some checkpoints have been settled, which can also be a veritable asset in non loosing time, as the deep learning algorithms often takes time to run.

The first step is to build a script having the role of dataset generator. This script contains an object which gives access to data inside a batch. The dataset generator built, in this case, stores all the paths of all patches but only opens and reads the data input and corresponding true output for the batch concerned which limits the size of the data stored on the heap.

Another important point is that this algorithm relies on the observation of the L2 Euclidean Norm (or mean root square error **MSE**) as a loss and this value is the only way to know if the network is performing good or not. For this type of decompression problem (high resolution retrieval), it is advised to use another measure called **PSNR** (Peak Signal to Noise Ratio) to determine the power of corrupting noise that affects the fidelity of representation of the true data. PSNR is defined such as:

$$\text{PSNR} = 10 \times \log_{10} \left(\frac{\text{MAX}_I}{\text{MSE}} \right) \tag{23}$$

where MAX_I is the maximum value of a pixel of the ground truth image (pansharpened image). The higher the PSNR, the closer the output of the SRCNN algorithm is from the pansharpened image. This new measure is a better one to follow the evolution of the algorithm during the training and the assessment of the goodness of the retrieval during the validation step.

In addition to PSNR evolution observation, displaying some samples of the validation set at the end of each epoch allows to see the visual evolution of the these patches and to confirm the evolution of the PSNR.

Running deep learning algorithms can take up to a really long time and it is important to implement a storing and restoring function to avoid retraining the model from scratch and loosing time. Tensorflow provides a really easy way to do and this has to be considered to implement from the early beginning.

6.3 Grey image baseline

To test the implementation of SRCNN [3], grey images downsampled have been fed to the algorithm so that the initial setting of the paper, with same kind of input, was tested before using our data. Results are seen in Figure 17 and as we can see, the output of SRCNN doesn't seem to do a way greater job that the bicubic upsampling (e.g the input). However, the network has been trained only for a little time, just to verify something was happening and our problem has been considered right after. Indeed, recall that in our problem we provide a high resolution channel so the problem is more about retrieving the RGB color.

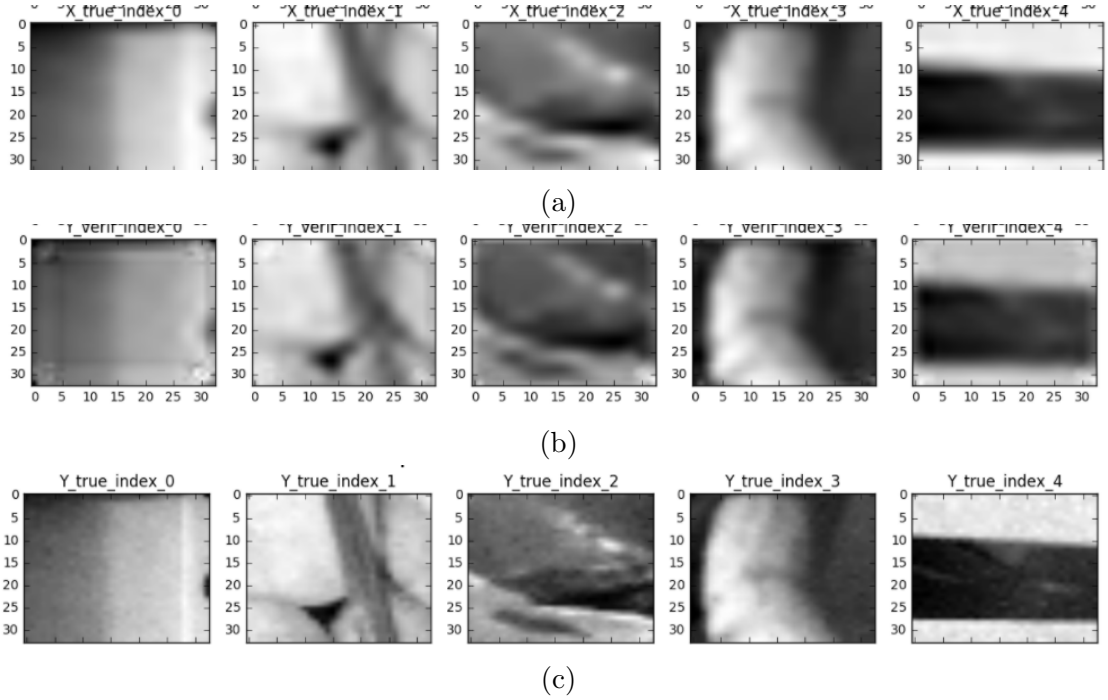


Figure 17: Up: input of SRCNN, Middle: output of SRCNN, Down: True high resolution patch

6.4 Regularization and optimization methods

Following [3] regularization and optimization methods considered and techniques advised in [7], some methods have been considered.

The first one to be tested was data augmentation as satellite images are really sensitive to light exposure and noise measurement during the acquisition. The python library **imgaug** has been used to implement data augmentation methods. The techniques selected were: adding and multiply pixels with random values to create overlay and change in the hue and adding gaussian noise.

The next step was to play with the network size of the SRCNN implementation. In the SRCNN paper [3], the authors prove by experiences that a deeper network than one internal layer (step called non linear mapping which corresponds to equation (15)) doesn't bring greater improvement, neither increasing the number of filters for each layer. However, increasing the kernel width and height of the filters could lead to greater improvements and this is what has been tried to investigate.

Regularization methods have then been considered with, firstly reducing the size of the batch (and of course increasing the number of iterations in each epoch) because it brings greater generalization of the network. Moreover, dropout method, which consists of "forgetting" some non output units in the network different at each iteration has also been implemented.

To finish with, different values of the initial learning rates have been tested to see how quicker the network could be trained.

7 Results

To test our setting, the initial choosen version of our model, based on reading the SRCNN paper and my own experimentation of our dataset, was: around 30 0000 images per epoch, 170 epochs, 50 images per batch, 600 iterations per epoch, a dropout probability of 0.8, an initial learning rate of 0.0001 and adam optimization. The initial architecture advised in SRCNN was to choose only one non linear mapping layer as deeper architectures were not bringing striking improvements, so three layers in total. The architecture advised was for filter widths $[9, 1, 5]$ and filter number $[5, 64, 32, 4]$ with of course the first and the last ones respectively equal to the dimension of the channels in input (bundle format) and output (pansharpened format).

For this first model, we can see in Figure 18 that the training PSNR is lower than the validation PSNR which is actually very weird. Dropout might be one of the reasons or maybe the validation set, as it comes from another image than the training set is more gentle to high resolution retrieval. We can see that the evolution to a resanobaly stable PSNR is rather quick and the validation PSNR even drops a little bit, which could make us think of an early stopping criterion but we will investigate firstly a long and stable training as early iterations also give higher PSNR but with bigger variance.

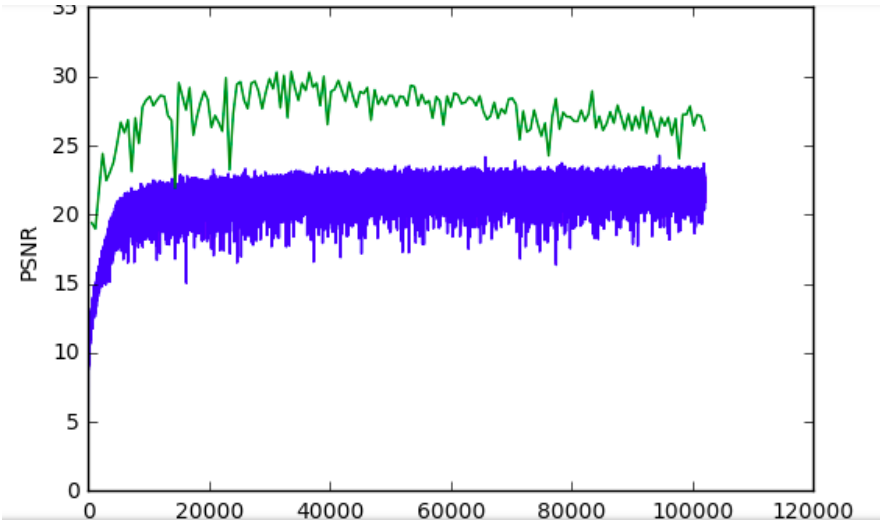
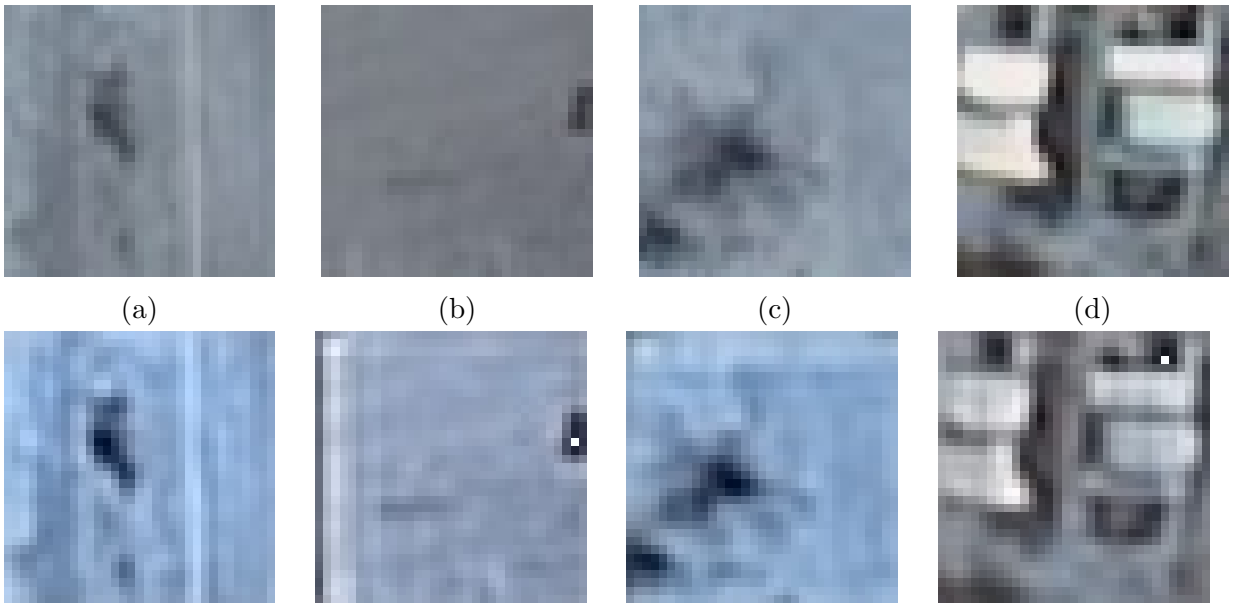


Figure 18: Basic Model PSNR in function of iterations. In blue, the training PSNR and in green the validation PSNR

Some test patches are displayed in Figure 19 after having trained the algorithm with Basic Model setting. We can see that patches with uniform background tend to have blue hue but the exposure and high resolution are retrieved which is already a good thing.



(a) (b) (c) (d)

(e) (f) (g) (h)

Figure 19: Test patches for Basic Model. First row are the True patches and second row are the Estimated patches

Figure 20 shows a test subimage of 6000x6000 estimated through this model and we can see that the retrieval is rather good but the colors not completely exact.

(a) True (b) Estimated

Figure 20: Test subimage of 6000x6000 for Basic Model. Left is the True image and Right is the Estimated image

7.1 Data Augmentation

Data augmentation allows to add data in the available dataset which are distorted but with the same corresponding groundtruth to compare the output to. The techniques chosen and implemented depend on the problem tackled. In our case, we want to retrieve color high resolution images from combination, in input, of low resolution narrow band channels and high resolution black and white image.Therefore, this kind of input is very dependent of hue shift, exposure, saturation and noise of measurement. I chose to implement a shift added on the channels in input or a multiplication as well as adding gaussian noise. This is a really simple process done through the python library **imgaug**.

In Figure 21, from the PSNR of the verification set, we see that the network with data augmentation is way less efficient than the one without.

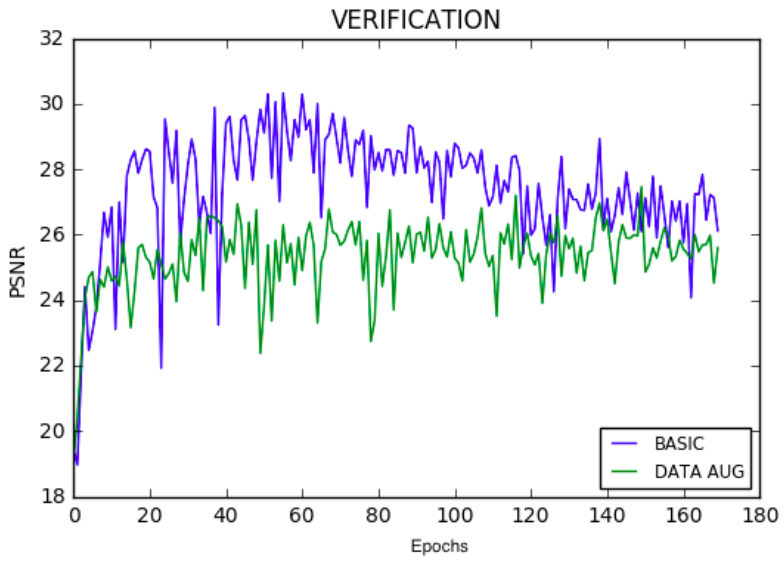
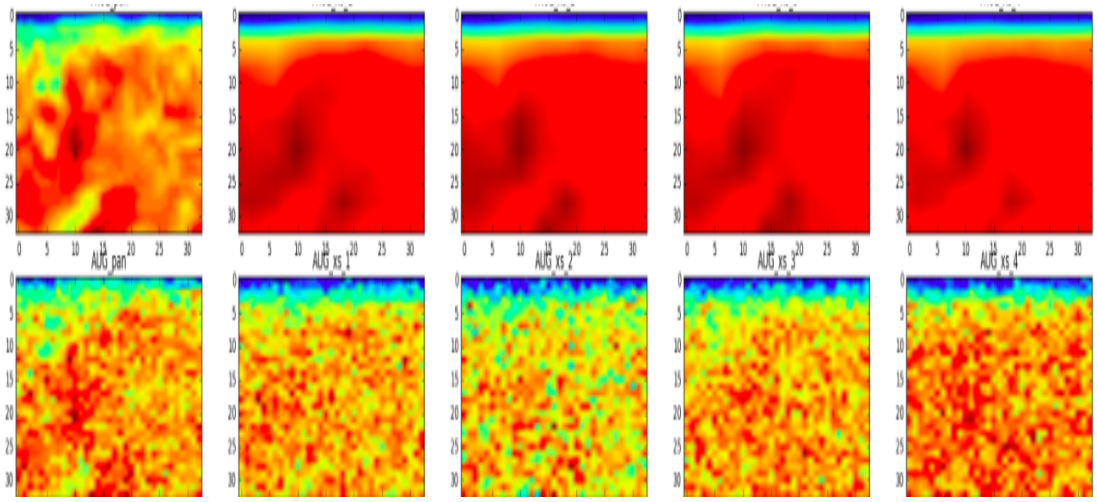
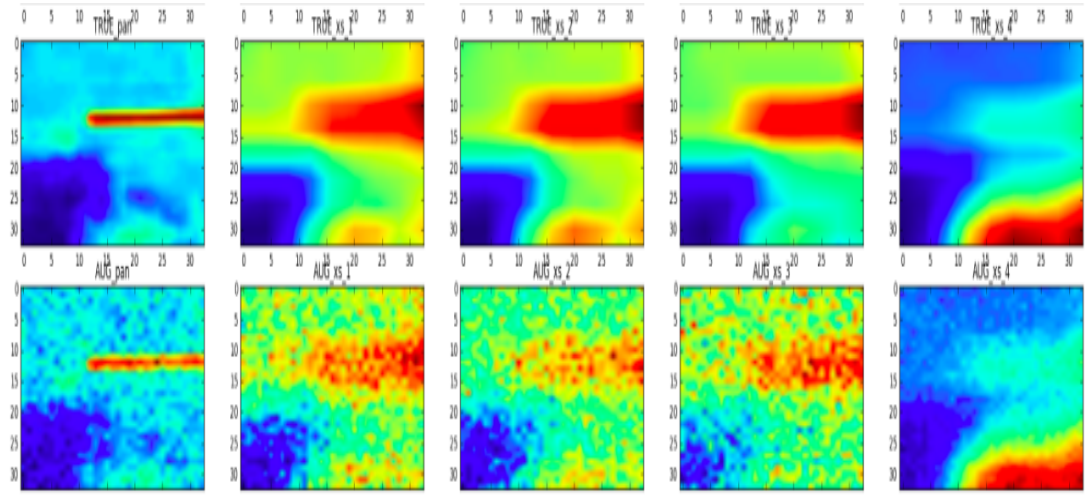


Figure 21: Comparison Validation PSNR between Basic Model with and without data augmentation

While having a look to input data after data augmentation, figure 22, we see that the distortion on each channel is a bit brutal which is maybe what disturbs the network instead of helping it. One solution might be to only distort the high resolution panchromatic input patch and especially because deforming patches up sampled (the 4 XS channels) is deforming over a little amount of pixels already averaged which is really violent transformation. Another interesting point which can be seen in the validation PSNR curve is that there is a peak around 50-60 epochs and it drops after it which might lead to think that early stopping could really be consider. In the last part, these two methods, data augmentation on one channel, the panchromatic one is considered coupled to early stopping but from now on we will keep on investigation with the original dataset.



(a) Sample 1



(b) Sample 2

Figure 22: Samples of input patches. First row is the true input and second is the augmented one

7.2 Network size

The network size tuned only deals with the filter width of the unique non linear mapping layer because SRCNN paper highlights that using more layers and more filters doesn't bring further improvements. I choose size of 1x1, 3x3 and 5x5 to test and figure 23 shows the PSNR values for the validation set along the epochs. We see that at the beginning, 1x1 is doing the best but the variance in the PSNR is really large and it finally drops anyway, whereas 5x5 keeps being more stable and is the highest at the end. Comparing test image, it is impossible to distinguish the difference of performance with human eye. We will therefore stick to what the PSNR curve demonstrates and keep the 5x5 filter.

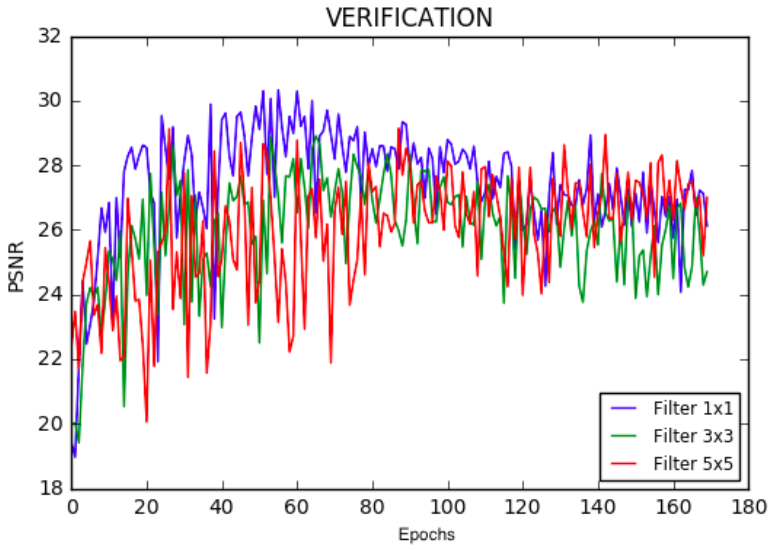


Figure 23: Comparison Validation PSNR with different filters' width

7.3 Batch Size Reduction

Reducing the batch size leads to better generalization so batch sizes of 50, 36, 24 and 12 have been tested. The PSNR curve in Figure 24 for the validation set shows that the lower the size the longer it takes for the PSNR to get higher values but it gives values with smaller variances so more stable. A good trade off is a batch size of 36. The quality of the test image follows the final values of PSNR at the end of the training with batch size of 12 giving the poorest result.

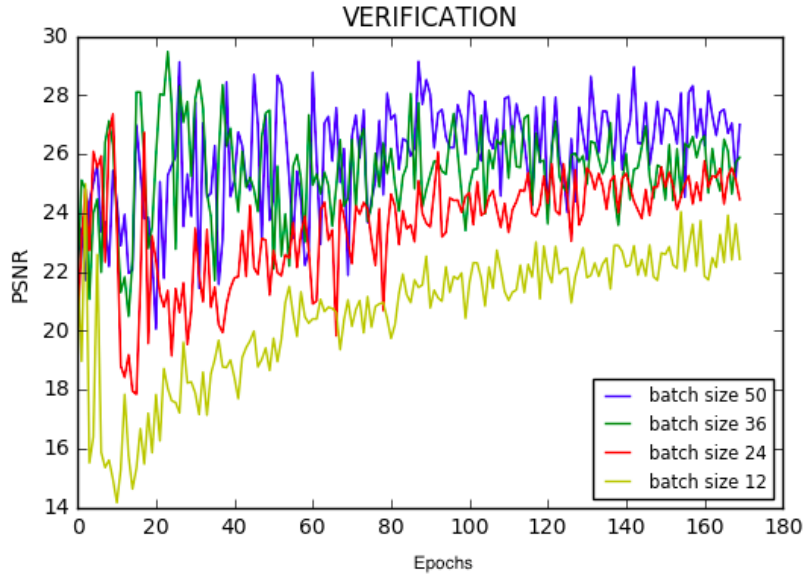


Figure 24: Comparison Validation PSNR with different batch sizes

7.4 Dropout

Dropout brings regularization as some nodes are « forgotten » at each iteration. The probability of dropping a node is tested for 0.6, 0.7 and 0.8. As seen in the curve of PSNR, a probability of dropout of 0.7 gives the highest values from far (see Figure 25).

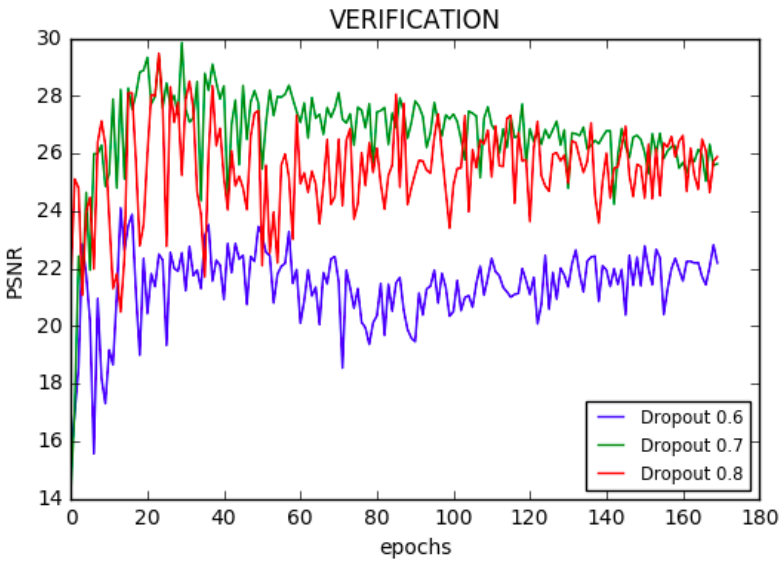


Figure 25: Comparison Validation PSNR with different dropout probabilities

7.5 Learning rate

To finish with, several initial learning rates for the Adam optimization have been tested from 0.001, 0.0001 and 0.00001 and the number of epochs have been increased for 0.00001 as the algorithm takes smaller steps. We see, in Figure 26 that the smallest learning rate 0.00001 gives the best result and small variance. Following the uprising trend, it gives the hint that training over a way longer number of epochs could give better results which is done in the last part.

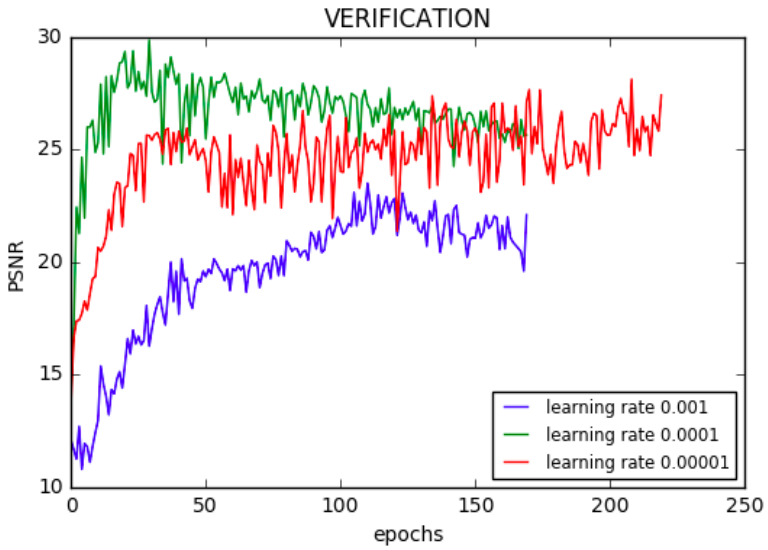


Figure 26: Comparison Validation PSNR with different learning rate values

7.6 Training over a long time with and without Data Augmentation

As mentioned in the previous parts, considering we lowed down the learning rate and that we only trained over a certain number of epochs, it was interesting to consider how the situation would evolve after a great number of iterations. Figure 28 shows the results between data augmentation setting (only on the panchromatic channel) and data non augmented. We find that both techniques lead to almost same results but as the validation PSNR for data augmented was still increasing, I retrained the network from the current model with data augmented to see if the validation PSNR would drop like it does for the data non augmented or keep on augmenting. Figure 27 shows that the Validation PSNR keeps being stable around 24 and therefore data augmentation doesn't bring any further improvement.

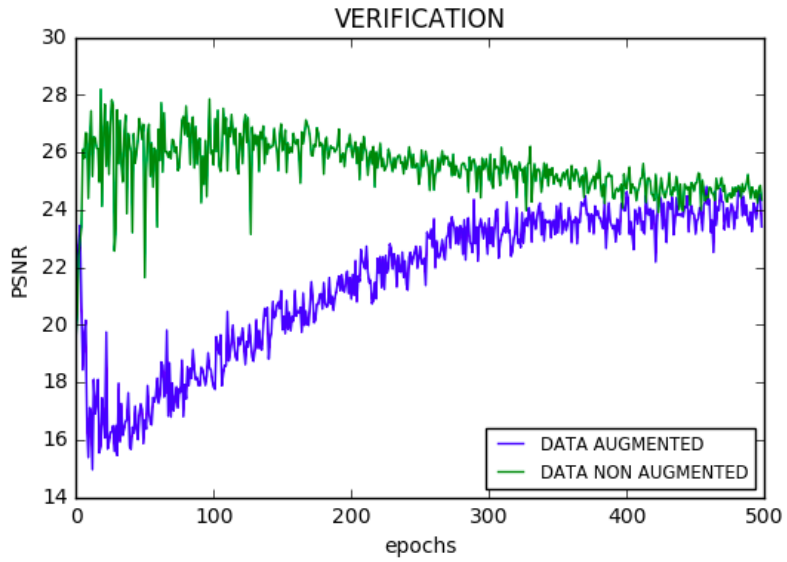


Figure 27: Comparison Validation PSNR with and without data augmentation over a great number of epochs

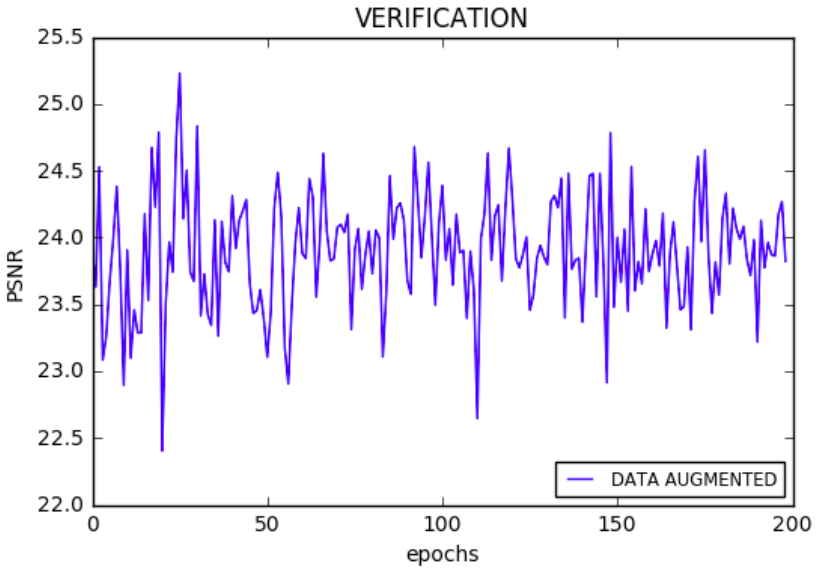


Figure 28: Validation PSNR for data augmented model retrained

Figure 29 shows the test image out of the algorithm with and without data augmented compared to the true pansharpened image and at this scale it's a bit difficult to see but by zooming on big images we see that the algorithm without data augmentation reconstructs a bit better the colors. I also compared visually the image out of the first BASIC version described in 7 and the results seem slightly better for dark hues with long trained advanced network without data augmentation and the first basic version but the contrary operates

for light hues. Therefore, talking about visual results, the improvements don't seem to be striking even after a very long period of training.

(a)

(b)

(c)

Figure 29: Up: output with Data augmented (panchromatic channel), Middle: output without Data augmented, Down: True pansharpened image

7.7 Early Stopping

Early Stopping has been tried so that models that lead to highest validation PSNR are considered. Figure 30 shows Validation and Training PSNR for the most advanced model we kept.

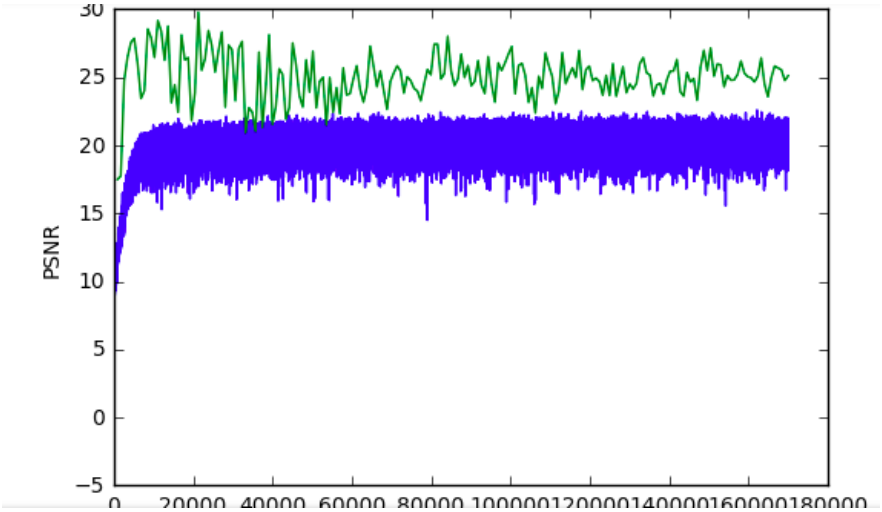
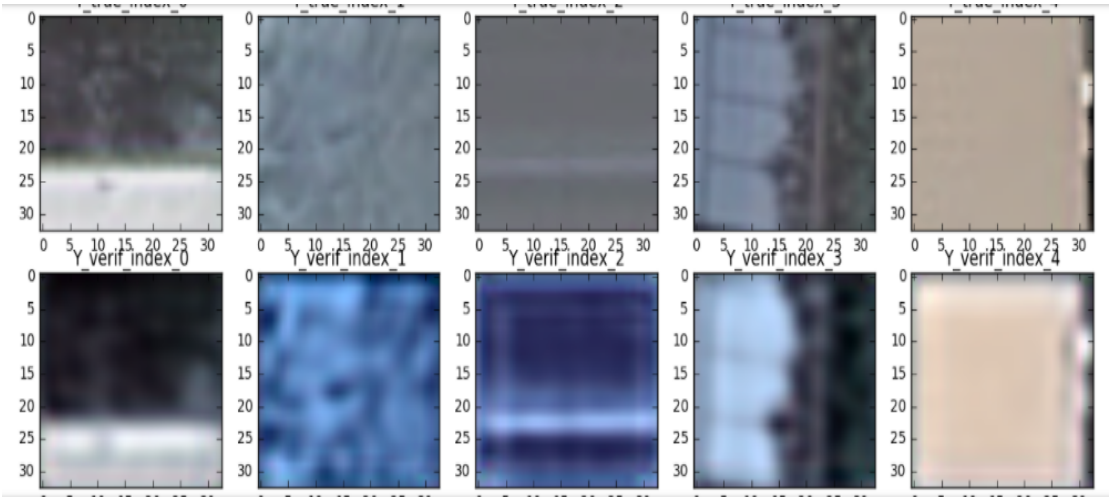
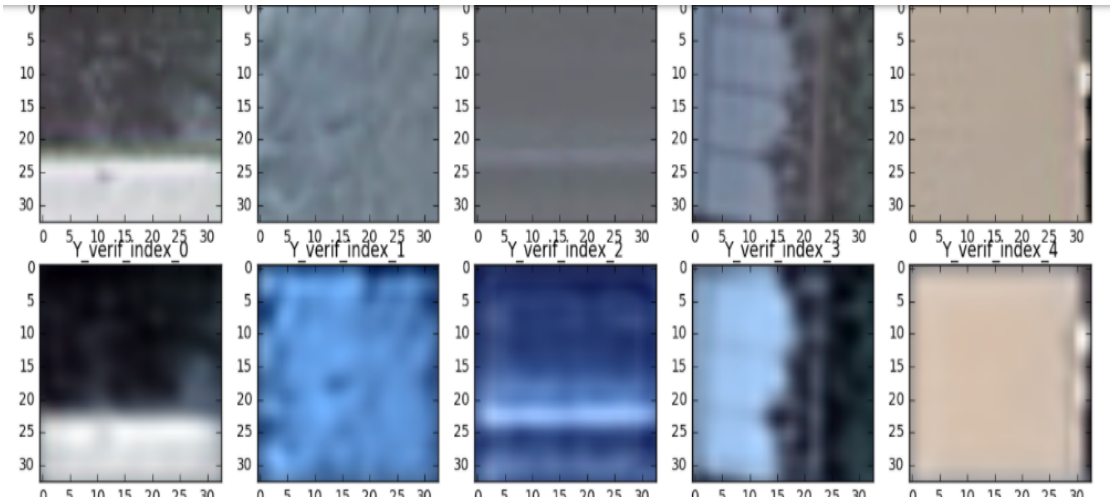


Figure 30: Training PSNR (blue) and Validation PSNR (green) Early Stopping

Epoch 12 (Validation PSNR: 29.2) and epoch 22 (Validation PSNR: 21.8) have been investigated to compare the visual results on validation patches in Figure 31. We see that PSNR measure correlates poorly with visual results as almost 8 PSNR of difference don't seem to give greater improvement. This problem is one mentioned in EnhanceNet paper. Therefore, even if early stopping seems to give greater PSNR, that doesn't mean the visual result is actually better.



(a)



(b)

Figure 31: (a) Validation PSNR of 29.2 (epoch 12) [Up:True and Down:Estimation] (b) Validation PSNR of 21.8 (epoch 22) [Up:True and Down:Estimation]

8 Discussion and Conclusions

8.1 Discussion

We observed that tuning parameters and training over a very long time were not seeming to bring greater improvements and that the PSNR seems to not being a trustful measure. The high resolution seems to be correctly retrieved but the global hue is shifted towards blue for a strange reason. However, results are encouraging and training over an even greater number of input images could lead to better results even if the paper [3] does not say so, but, again, our problem is a bit different than the one dealt with in this paper.
One surprising point is still that the training PSNR is lower than the validation one and I can't find any explanation for this point except maybe dropout influence.

No computational cost has been considered, neither for training nor the prediction, as the goal was to develop an algorithm that could work first but when the method will be improved, it will definitely be a point to consider. How quick is the prediction, and how does it generalize to many different images with different exposure or noise of measurement? Considering that the main goal of this project was to challenge traditional methods of fusion algorithms with deep learning methods, it would be interesting to compare the two algorithms, with PSNR or any other suitable measure.

8.2 EnhanceNet

One of the major problems of our implementation of SRCNN is that we keep having a blue hue over the images but the high resolution seems fairly reconstructed thanks to the high resolution panchromatic channel in input. As explained in part 5.3, residual blocks of the EnhanceNet help to stabilize the training and reduce color shifts in the output during training. However, the EnhanceNet in his initial form takes only low resolution channels in input to output high resolution channels whereas the input of our problem is a mixture between low and high resolution channels. Considering that the high resolution seems to be restored pretty well, the colors actually are the really features which still gives trouble to retrieve with SRCNN.

I tried to find an implementation of EnhanceNet but the only one available is a pretrained model which can't be modified internally which is an issue considering the type of input with have. Moreover, the network introduced in the EnhanceNet paper is not an easy one to recode. Therefore, if it has to be used, EnhanceNet would have to be recoded from scratch.

Just to play with the implementation of EnhanceNet, I tried to retrieve a high resolution pansharpened image (4 high resolution RGBA channels Figure 34) from our dataset which was downsampled in input (Figure 32) to simulate low resolution and the result is very good (Figure 33).

Figure 32: Input of EnhanceNet (dowsampled pansharpened)

Figure 33: Output of EnhanceNet

Figure 34: True Pansharpened image

Glossary

CNN Convolutional Neural Network

HR High Resolution

LR Low Resolution

XS Multi Spectral Bands

GPU Graphics Processing Units

CPU Central Processing unit

References

[1] 37 reasons why your neural network is not working. https://blog.slavv.com/37-reasons-why-your-neural-network-is-not-working-4020854bd607.

[2] Artificial neuron network. https://en.wikipedia.org/wiki/Artificial_neural_network.

[3] Kaiming He Xiaoou Tang Chao Dong, Chen Change Loy. Image super-resolution using deep convolutional networks. *Neural Netw.*, Computer Vision and Pattern Recognition, Jul 2015.

[4] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[5] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. Rectifier nonlinearities improve neural network acoustic models. 2013.

[6] O. Ronneberger, P.Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, volume 9351 of *LNCS*, pages 234–241. Springer, 2015. (available on arXiv:1505.04597 [cs.CV]).

[7] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[8] Mehdi S. M. Sajjadi, Bernhard Schölkopf, and Michael Hirsch. Enhancenet: Single image super-resolution through automated texture synthesis. *CoRR*, abs/1612.07919, 2016.

[9] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[10] Martinez Wilson. The general inefficiency of batch training for gradient descent learning. *Neural Netw.*, 16(10):1429-51., 2003 Dec.