SPATIAL ENHANCEMENT OF REMOTELY SENSED IMAGES USING CONVOLUTIONAL NEURAL NETWORKS

Ugo Palatucci

July 2020

Supervisors:

Prof. Restaino Rocco

Contents

Introduction

Pansharpening refers to a particular data fusion issue where two images, one panchromatic and one multispectral, representing the same area can be combined to enhance the peculiarity of both. The panchromatic image is acquired with a wide spectrum sensor that can have a higher spatial resolution compared to a multispectral one. However, the sensor cannot acquire different bands. A multispectral image, instead, has several bands in a lower spatial resolution. As physical constraints occur, the creation of a sensor specialized in both resolution's type would not be possible. For this reason, the fusion of both images can be the only possibility to have a new image with higher spatial and spectral resolution. Pansharpening is an urgent topic for remote sensing, indeed, the result can be used upstream of another process such as change detection [1], object recognition [2], visual image analysis and scene interpretation [3]. An example of pansharpening can be observed in the Fig. 1.

Based on a convolutional neural network, a new pansharpening method has been proposed recently [4]. Using a degraded version of the PAN and MS images, the network's weights were trained to fuse the images, optimizing

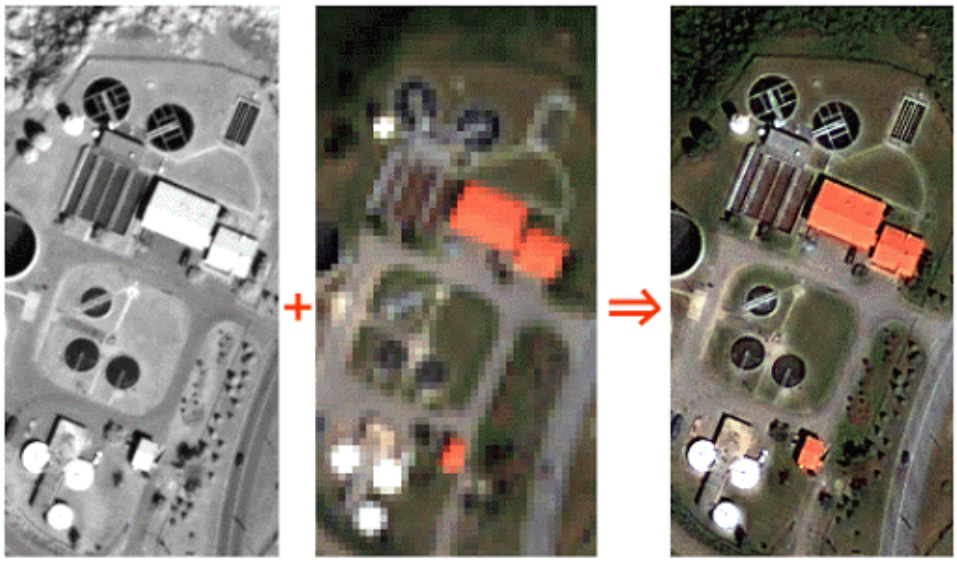

Figure 1: Pansharpening example

a reference index with the gradient descendent algorithm: the mean squared error. After the training, the same weights were applied to fuse the original PAN and MS images. The purpose of this work is to improve the current methodology using the original PAN and MS images for a training with a no-reference index like QNR or HQNR indexes. Furthermore, the intention is to remove the error added using the degraded images that do not reflect the models were the pansharpening algorithm should be utilised. In [4] the code was written in python 2.7 using the Theano library. The Theano project has been not updated since 2018 [5]. For this reason, the project is incompatible with new Cuda libraries and NVIDIA drivers and many issues to run the training process on the GPU have been founded. To solve that, a new software has been created using TensorFlow. Tensorflow is a more modern and popular

DeepLearning library maintained by Google that allow more compatibility with the newest libraries and a larger community helping the development in case of uncommon errors. As Theano does, Automatic Differentiation has been implemented by the new library [6]. Indeed, Automatic Differentiation can be defined as critical feature that allows writing differentiable functions and subsequently using them for the core algorithm of the neural network's backpropagation training: the gradient descendent algorithm.

Chapter 1

Pansharpening state of the art

According to the current methodology, the pansharpening techniques are divided into two main areas: component substitution (CS) and the multiresolution analysis (MRA). The techniques belonging to the first class consist in representing the MS and PAN in a different domain that can entirely split the spatial information from the spectral information. In this domain, the spatial information part of the MS image can be replaced with the PAN image. After this substitution, the MS image can be back-transformed in the original domain. Clearly, the less the PAN is correlated with the replaced component, the more distortion is introduced. The most famous techniques of this class are intensity-hue-saturation (IHS) [7] [8], in which the images are represented in the IHS domain, principal component analysis (PCA) [9] [10] and Gram-Schmidt (GS) spectral sharpening [11]. On the one side, those techniques preserve the PAN spatial information. On the other side they can produce a

high spectral distortion. This is because PAN and MS are obtained in spectral ranges that only partially overlap.

The second class of techniques, MRA, are based on the introducing of spatial details extracted from the PAN image into the up-sampled version of the MS. This approach promises a better spectral fidelity but often present spatial distortions.

The lack of the reference image is the principal issue in the evaluation of the pansharpening methods. When a couple of images are fused, the result cannot be compared with anything else. The sensors used for the acquisition cannot reach alone both spatial and spectral resolution of the result. As the two models are different, the result cannot be compared with another image acquired with a different sensor. For this reason, there is no universal measure of quality for the pansharpening. The scientific community common practice is to use the verification criteria that were proposed in the most credited work [12]. This study defines two properties to use for the evaluation of the fused product: consistency and synthesis. The first means that the original MS image should be obtained with a degradation of the fused result. The second property describe that the fused image should preserve both the features of each band and the mutual relations among them. The definition of an algorithm that accomplishes these properties and of an index that can guarantee the correct evaluation are an open problem. But, no matter what index is decided to use, the unavailability of a reference image is a huge problem and a visual inspection is always mandatory. There are two techniques that can be

used for the quality assessment. The first is to reduce both the images given in input to the pansharpening algorithm and use the original MS image as a reference for the result evaluation. The downside of this method is the assumption of invariance between scales, which justifies that the same algorithm operates similarly at reduced scale. The cited hypothesis is not always verified as documented here [9] [12]. A second technique is the use of an index that does not require a reference image.

1.1 CS

The CS family is based on converting the MS image into a domain in which the spatial and spectral pieces of information can be better separated. In this domain, the component containing the spatial information can be replaced by the PAN image. The greater the correlation between the PAN image and the replaced component, the lower the distortion introduced by the fusion. For this reason, the histogram matching of the PAN with the component that contains the spatial part of the MS information is preliminarily performed. After the substitution, the data can be represented in the original space with an inverse transformation. This approach is applied to the whole image in the same way. Techniques of this category have high fidelity regarding the fusion of spatial details and are fast and easy to implement. But as the acquisition spectrum of the sensors used to produce the PAN and MS image differ each other, the process may produce significant spectral distortions [13] [14]. In the studies

[15] [16] [17] [18] [19], it was shown that, when a linear transformation is used, the substitution and fusion can be obtained without the explicit forward and backward transformation of the images but with a precise injection scheme. This scheme can be formalized according to the following equation:

$$\widehat{MS}_k = \widetilde{MS}_k + g_k(P - I_L), \qquad k = 1, \ldots, N \tag{1.1}$$

in which k indexes the spectral bands, g_k are the injection gains, $\widehat{MS}_k$ is the k-th band of the pansharpened image, $\widetilde{MS}_k$ is the k-th band of the MS image interpolated to the PAN scale and I_L is the intensity component derived from the MS image according to the relation:

$$I_L = \sum_{i=1}^{N} w_i \widetilde{MS}_i \tag{1.2}$$

The weight vector $w = [w_1, w_2, \ldots, w_k]$ is the first row of the forward transformation matrix and depends on the spectral overlap among MS channels and PAN.

The CS approach procedure is illustrated in Fig. 1.1. Four important steps can be noticed: 1) interpolation of MS image for matching the PAN scale; 2) calculation of I_L using Eq. (1.2); 3) histogram matching between PAN and intensity component; 4) details injection according to Eq. (1).

The various CS techniques such as IHS [7, 8], PCA [10, 1] and GS [11, 15] define different $w_{k,i}$ and g_k.

In the IHS pansharpening method is used the IHS transformation. This is

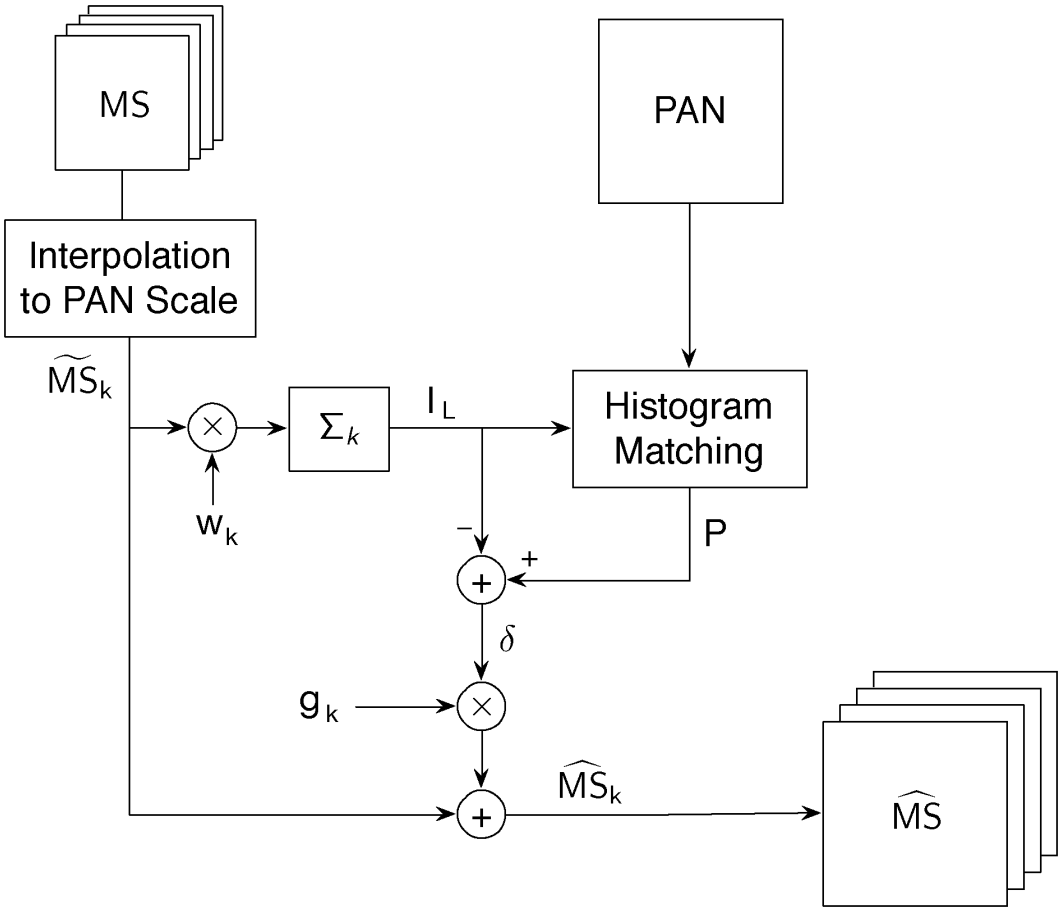

Figure 1.1: Flowchart of CS approach [20]

the major limitation of this technique because it can transform only images in RGB and often the MS image has 4 or also 8 and more bands. As a workaround, the authors of paper [18] has proved that GIHS, a generalization of the IHS transformation for more bands, can be formulated for any arbitrary set of nonnegative spectral weights as described in the following equation:

$$\widehat{MS_k} = \widetilde{MS_k} + \left(\sum_{i=1}^{N} w_i\right)^{-1} (P - I_L), \qquad k = 1, \ldots, N \tag{1.3}$$

in which w_i are all equal to $1/N$ [7]. With the injection gains defined such that:

$$g_k = \frac{\widetilde{MS_k}}{I_L}, \qquad k = 1, \ldots, N \tag{1.4}$$

$\widehat{MS_k}$ can be calculated as

$$\widehat{MS_k} = \widetilde{MS_k} \cdot \frac{P}{I_L} \tag{1.5}$$

which is the known Brovey Transform.

In the PCA pansharpening method, it is used the PCA transformation, also called Karhunen-Loeve transform. It is a linear transformation that can be implemented for a multidimensional image, so it is not limited as the IHS method, and consists into the projection of all the components along the eigenvectors of the covariance matrix. This means that each component is orthogonal and statistically uncorrelated from the others. The hypothesis introduced in this step is that the spatial information is concentrated in the first component, the component with the higher eigenvalue. The PCA can be implemented by using Eq. 1.1, in which w is the first row of the forward transformation matrix; g is the first column of the backward transformation matrix.

The GS transformation is a common technique used to orthogonalize a set of vectors in linear algebra. First of all, the $\widetilde{MS}$ bands are organized in vectors to obtain a two dimensional matrix in which the columns are constituted by the bands organized as vectors. The mean of each band is subtracted from all the columns. The orthogonalization procedure is used to create a low-resolution

version of the PAN image, i.e. I_L. The last step is the replacement of I_L with the histogram matched PAN before the inverse transformation. GS is a generalization of PCA in which PC1 may be any component and the remaining ones are calculated to be orthogonal with PC1. Also the GS procedure can be described by Eq. 1.1 if g_k is defined as:

$$g_k = \frac{\mathrm{cov}(\widetilde{MS}_k, I_L)}{\mathrm{var}(I_L)}, \qquad k = 1, \ldots, N \tag{1.6}$$

in which $\mathrm{cov}(\cdot, \cdot)$ is the covariance between two images and $\mathrm{var}(\cdot)$ is the variance. There are several version of this technique that differ on how the I_L is created. The simplest way is to set $w_i = 1/N$. This version is called GS mode 1 [11]. It was proposed also an *adaptive* version of this mode called GSA in [15] in which I_L is generated by a weighted average of the MS bands. Another technique defined in [11] and called GS mode 2 suggests to generate the I_L by applying a low pass filter to the PAN image. This last step leads the GS mode 2 that belongs to the MRA class of techniques.

Another noteworthy technique is described in [19] that introduces the concept of *partial replacement* of the intensity component. An intensity image is created for every band of the MS from the PAN image; it is calculated with the following equation:

$$P^{(k)} = CC(I_L, \widetilde{MS}_k) \cdot P + (1 - CC(I_L, \widetilde{MS}_k)) \cdot \widetilde{MS}'_k \tag{1.7}$$

in which $\widetilde{MS}'_k$ is the k-th MS band histogram-matched to PAN and CC is

the correlation coefficient. I_L is defined using in Eq. 1.2 a vector w obtained with a linear regression of $\widetilde{MS}'_k$ on P_L, the degraded version of the PAN. The injection gains are the result of:

$$g_k = \beta \cdot CC(P_L^{(k)}, \widetilde{MS}_k) \cdot \frac{std(\widetilde{MS}_k)}{\frac{1}{N}\sum_{i=1}^{N} std(\widetilde{MS}_i))} L_k \tag{1.8}$$

β is empirically tuned and is a factor that normalizes the high frequencies. $P_L^{(k)}$ is a low-pass-filtered version of $P^{(k)}$, and L_k is an adaptive factor that removes the local spectral instability error between the synthetic component image and the MS band defined as:

$$L_k = 1 - |1 - CC(I_L, \widetilde{MS}_k)\frac{\widetilde{MS}_k}{P_L^{(k)}}|. \tag{1.9}$$

1.2 MRA

In the MRA class of techniques, the pansharpened image is defined as:

$$\widehat{MS}_k = \widetilde{MS}_k + g_k(P - P_L), \qquad k = 1, \ldots, N. \tag{1.10}$$

$P - P_L$ is the operation performed to obtain the high-frequency details of the PAN image. The algorithm to create the P_L and the chosen g_k weights differentiate the MRA pansharpening techniques of this class.

However, in general, all the techniques follow the algorithm described in Fig. 1.2. First of all, the MS image is interpolated to the PAN scale. The

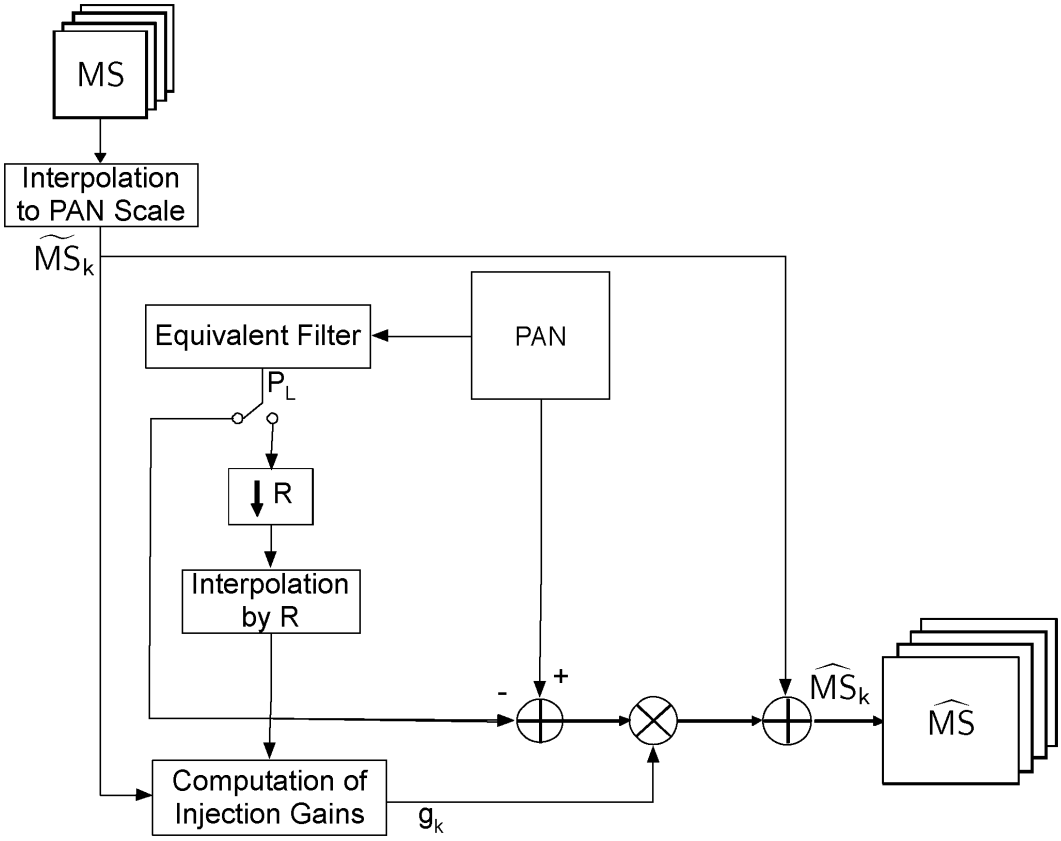

Figure 1.2: Flowchart of MRA approach [20]

second step is to calculate P_L, the low pass version of PAN obtained by means of an equivalent filter. The vector of injection weights g_k can be computed using the $\widetilde{MS_k}$ in combination with P_L. Interpolation is less crucial in MRA respect to CS methods. A method to produce the P_L image consists in applying a low pass filter h_{LP} to the PAN image P. So Eq. (1.2) can be rewritten as:

$$\widehat{MS_k} = \widetilde{MS_k} + g_k(P - P * h_{LP}), \qquad k = 1, \ldots, N \tag{1.11}$$

where $*$ is the convolution operation. A more general method to obtain the

P_L is called *Pyramidal Decompositions* and the number of filterings can be one or more. A filter type that proves to be a good choice is a Gaussian filter that closely matches the sensor MTF. A noteworthy option is the MTF-GLP with a context-based decision (MTF-GLP-CBD) [21] where the injection gains are defined as follows:

$$g_k = \frac{cov(\widetilde{MS}_k, P_L^{(k)})}{var(P_L^{(k)})} \tag{1.12}$$

It is context-based because it can be applied on nonoverlapping image patches to improve the quality of the final product.

1.3 Quality Assessment

As explained above, the lack of a reference image is the main limitation. The community has proposed two assessment procedures as a workaround. The first procedure consists in using the images at a lower spatial resolution and use the original MS image as a reference. However, the output of an algorithm can have different performance at different scales, as it is showed in [22]. This because the performance assessment depends intrinsically to the image scale, mostly in case of pansharpening methods that apply spatial filters. The second procedure consists in using non-reference quality indexes. Both types of procedures require also a visual inspection for spectral distortions and spatial details.

The Wald's protocol is composed by three requirements:

1. $\widehat{MS}_k$ degraded to the original MS scale should be as identical as possible

to the MS_k.

2. The fused image $\widehat{MS_k}$ should be as identical as possible to the MS_k that the sensor would acquire at the highest resolution

3. The MS set of synthetic images $\widehat{MS} = \{\widehat{MS}\}_{k=1,\ldots,N}$ should be as identical as possible to the MS set of images $HRMS = \{HRMS\}_{k=1,\ldots,N}$ that the corresponding sensor would observe at the highest resolution.

In the previous definitions HRMS is the reference image. For a reduced-resolution assessment, the filter choice for the downsampling is crucial in the validation. A bad filter choice results in an image degradation that does not reflect the sensor characteristics at a lower scale. So the algorithm, after the degradation, is applied to images that reflect the wrong sensor model. This means that the same algorithm can have a more different result at the original and lower scales. On the contrary, with a good filter that preserves the sensor characteristics at the lower resolution, the algorithm has much more possibility to reflect the quality of the original resolution. Indeed, the filter used for the MS degradation should simulating the transfer function of the remote sensor and so, it should match the sensor's MTF [23]. Similarly, the PAN image has to be degraded in order to contain the details that would have been seen if the image were acquired at the reduced resolution. The fused image obtained from the degraded PAN and MS, can be evaluated different indexes using the MS as a reference image.

The Spectral Angle Mapper (SAM) is a vector measure that is useful

to evaluate the spectral distortion. In simple terms, denoting by $I_{(n)} = [I_{1,n}, \ldots, I_{N,n}]$ a pixel vector of the MS image, with N bands, the SAM between the corresponding pixel vectors of two images is defined as:

$$SAM(I_i, J_i) = arcos(\frac{< I_i, J_i >}{||I_i||||J_i||}) \tag{1.13}$$

$< I_i, J_i >$ is the scalar product and $||I_i||$ is the vector l_2-norm. Applying this equation to every pixel results in a so-called SAM map. Averaging all the pixel of the SAM map returns the SAM index for the whole image. The optimal value of the SAM index is 0.

RMSE is used to calculate the spatial/radiometric distortions. It is defined as:

$$RMSE(I, J) = \sqrt{E[(I - J)^2]} \tag{1.14}$$

The ideal value of RMSE is zero and is achieved if and only if $I = J$. But it is not an efficient index because it is not considered the error for each band, but is global. So, to better measure the error for each band, the ERGAS index is used. The ERGAS index evaluates the RMSE error with a different weight for each band.

$$ERGAS = \frac{100}{R}\sqrt{\frac{1}{N}\sum_{k=1}^{N}\left(\frac{RMSE(I_k, J_k)}{\mu(I_k)}\right)^2} \tag{1.15}$$

Obviously, the ERGAS is composed of a sum of RMSE, so the optimal value is also 0. Another important index is the Universal Image Quality Index (UIQI)

or also called Q-index, proposed in [24]. Its expression is:

$$Q(I,J) = \frac{\sigma_I J}{\sigma_I \sigma_J} \frac{2\bar{I}\bar{J}}{\bar{I}^2 + \bar{J}^2} \frac{2\sigma_I \sigma_J}{(\sigma_I^2 + \sigma_J^2)} \tag{1.16}$$

where σ_{IJ} is the covariance of I and J, and $\bar{I}$ is the mean of I. The first fraction represents an estimation of the covariance, the second is a difference in the mean luminance and the third is the difference in the mean contrast. The Q-index varies in the range $[-1, 1]$ with 1 as the optimal value.

Q4 is an extension of the UIQI for images with 4 bands [25]. Let a, b, c and d denote the radiance values of the given image pixel in four bands, and let the quaternions:

$$z_A = a_A + ib_A + jc_A + kd_A \tag{1.17}$$

$$z_B = a_B + ib_B + jc_B + kd_B \tag{1.18}$$

The Q4 is defined as :

$$Q4 = \frac{4|\sigma_{z_A z_B}||\dot{z}_A||\dot{z}_B|}{(\sigma_{z_A}^2 + \sigma_{z_B}^2(|z_A|^2 + |z_B|^2))} \tag{1.19}$$

If eventually, the bands are more than 4, the Q4 can be replaced with Q average.

An index used for the validation at full-resolution is the Quality with no reference (QNR) index [26]. It is defined by the following equation:

$$QNR = (1 - D_\lambda)^\alpha (1 - D_S)^\beta \tag{1.20}$$

α and β are two coefficients which can be tuned to weight more the spectral or the spatial distortion, respectively.

The maximum theoretical value of the index is 1 and is reached when D_λ and D_S are 0. The spectral distortion is calculated with D_λ using this equation:

$$D_\lambda = \sqrt[P]{\frac{1}{N(N-1)} \sum_{i=1}^{N} \sum_{j=1, j\neq i}^{N} |d_{i,j}(MS, \widehat{MS}|^P)} \tag{1.21}$$

where $d_{i,j}(MS, \widehat{MS}) = Q(MS_i, MS_j) - Q(\widehat{MS}_i, \widehat{MS}_j)$, $\widehat{MS}$ is the fused image and p is a parameter typically set to one [26]. The objective is to create an image with the same spectral features of the original MS image.

The spatial distortion is calculated by:

$$D_S = \sqrt[q]{\frac{1}{N} \sum_{i=1}^{N} |Q(\widehat{MS}_i, P) - Q(MS_i, P_{LP})|^q} \tag{1.22}$$

where P_{LP} is the low-resolution PAN image at the same scale of the MS image and q is usually set to one [26].

Khan protocol [27] extends the consistency property of Wald's protocol. The pansharpened image is considered as a sum of a lowpass term plus a high pass term. The lowpass term is the original interpolated low resolution MS image and the highpass term corresponds to the spatial details extracted to the PAN and injected into the MS image. A Gaussian model of the sensor's MTF is used to build the filters. The similarity between the lowpass component and the original MS image can be calculated using the Q4 index or any other

similarity measure for images with more bands. The similarity between PAN and the spatial component is measured as the average of UIQI calculated using the PAN and each band of MS. The same similarity is calculated also between the original MS and the degraded version of the PAN.

The QNR and the spectral distortion of Khan's protocol can be combined to yield another quality index, the HQNR [28]:

$$HQNR = (1 - D_{\lambda}^{(K)})(1 - D_s) \tag{1.23}$$

in which $D_{\lambda}^{(K)}$ is :

$$D_{\lambda}^{(K)} = 1 - Q4(\widehat{M_L}, M) \tag{1.24}$$

The $\widehat{M_L}$ is the fused image degraded to the resolution of the original MS image.

Chapter 2

Pansharpening applications of Deep Learning

2.1 Introduction

Early works in the field of Deep Learning have been made in the 1940s starting from the concept of Perceptron [29] and afterwards in the 60s with the invention of backpropagation, the most commonly used algorithm in the present day to train a Neural Network. The Neural Network is a model constituted by several Perceptrons, also called neurons, that are divided into different layers with the ability to learn, extract and distinguish different features from the data given into input. In order to obtain these capabilities, the network should be subjected to a training phase that requires an incredible amount of computational power. As in the early 60s the computers were no powerful enough

and there was not a great data availability, Deep Learning had no reason to be implemented as today.

This field of study is characterized by different phases. One of them, the training, is a critical phase in which the model learns from new data and can differ for the type of issues that the model should solve. Most of the cases, the model gives a prediction and the result is compared with a target. Initially, the error is calculated between the output of the prediction and the target. After that, when the error is propagated in all the neurons of the model, the weights of all the neurons would be modified so that the next prediction for the same input would be much similar to the target output. How this error is calculated and what means "similar" is established by the loss function that calculates the error. The propagation use an algorithm called Gradient Descendent algorithm, that allows the network to understand how to change the weights and minimize the loss. In order to use the Gradient Descendent, the loss function must be differentiable so that, the gradient operator can be applied more times. Many layers the net have, many times the algorithm apply the gradient to the error.

LeCun (1989) for the first time used a backpropagation algorithm to train a neural network to classify handwritten digits.

Nowadays, the data collected through internet, the incredible amount of computational power exhibited by data centers and GPUs, the performance of this type of algorithm and a large number of applicative fields, have encouraged the growth of Machine Learning and in particular, Deep Learning.

2.2 Neural Networks

2.2.1 Perceptron

Essentially a perceptron is an element in which the output is a result of a weighted sum of the inputs. There is always only one output but it can be more inputs as described in the Fig. 2.1. A perceptron uses only one linear or non-linear activation function. The following paragraph will discuss what is an activation function and how it works. With a non-linear transfer function, perceptron can build a nonlinear plane separating data points of different classes. In 1969, it was proved that the perceptron itself may fail in certain simple tasks, as instance the separation of a plane described by the XOR function. However, 3 perceptron organized in 2 layers can separate an XOR plane. This opened up the development of multi-layer models and subsequently, for training optimization for specific applications, the creation of different type of layers.

2.2.2 Activation Function

Activation function is used to transform the weighted data (input multiplied weights) in outputs. By the use of a non-linear transformation, we create new relation between the points. This consent to the machine learning model to create increasingly complex features with every layer. Features of many layers that uses pure linear transformations can be reproduced by a single layer that use a non-linear function. Most common activation functions is the so called

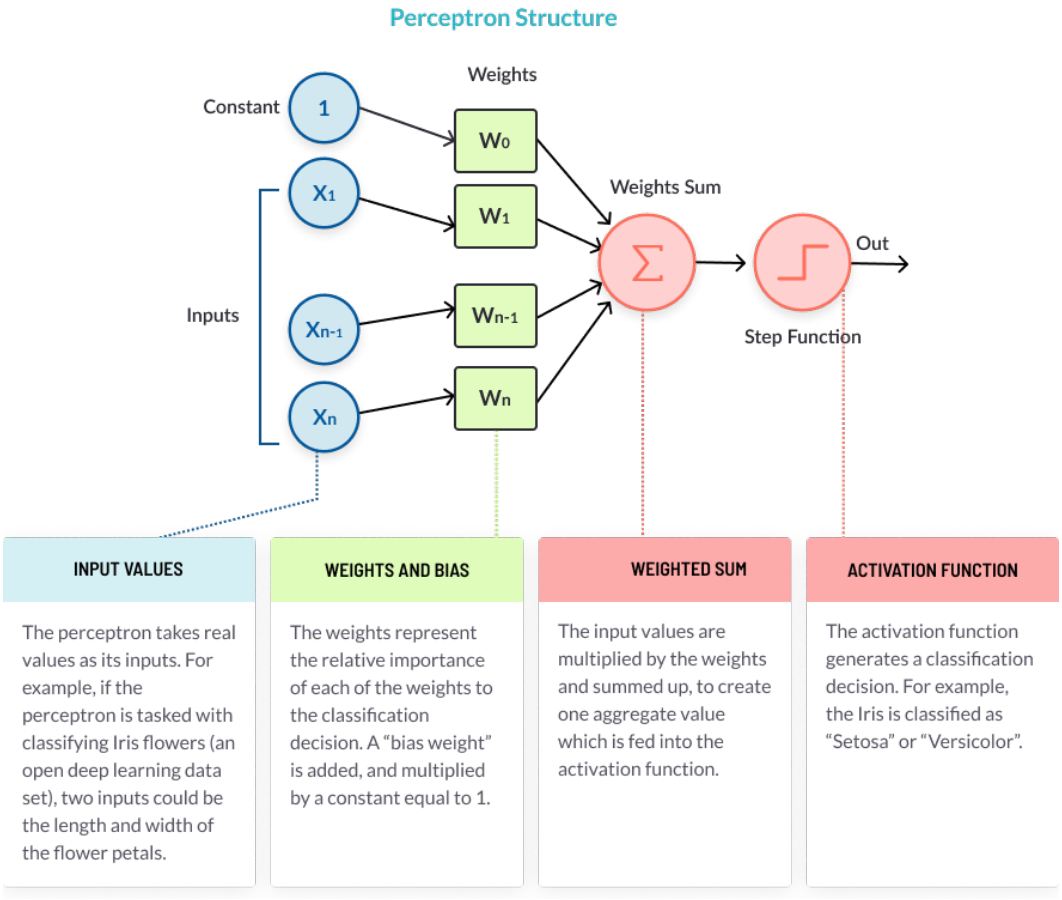

Figure 2.1: Perceptron in detail [30]

ReLu that can be described as $y = max(0, x)$. The gradient is always x when the value of x is positive, and 0 when negative. This means that during the training, negative gradients will not update the weights. A Gradient with equal 1 means that the training will be much faster compared with other activation functions like logistic sigmoid.

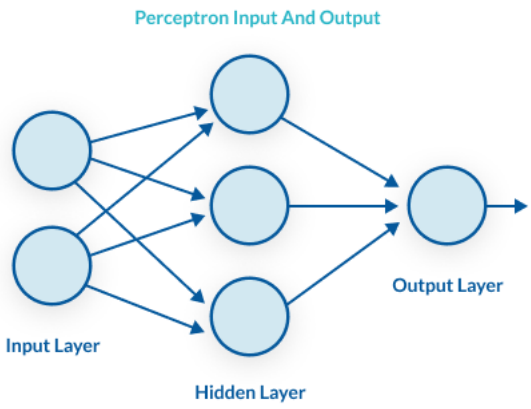

Figure 2.2: Multilayer architecture example [30]

2.2.3 Layers

The DL network can have different type of layers that differ in how the perceptrons inside them are organized and how they modify their weights during the training. The Fig. 2.2 shows an example of multilayer model. The first experimented layer was the dense layer where all neurons are connected with all neurons of the next layer. Every neuron has his own weights and considerin all the neurons connections, this type of layer generates promptly a large number of weights. It will be necessity a large dataset and a longe time for the training.

The convolutional layer gave an important improvement in computer vision applications. This layer can apply different filters to the data at the same time.

Commonly, several convolutional layers at the start of a neural network are used to extract features from an image. On a first phase The DL model is able to extract features in the data as complex as the model dimension growth as

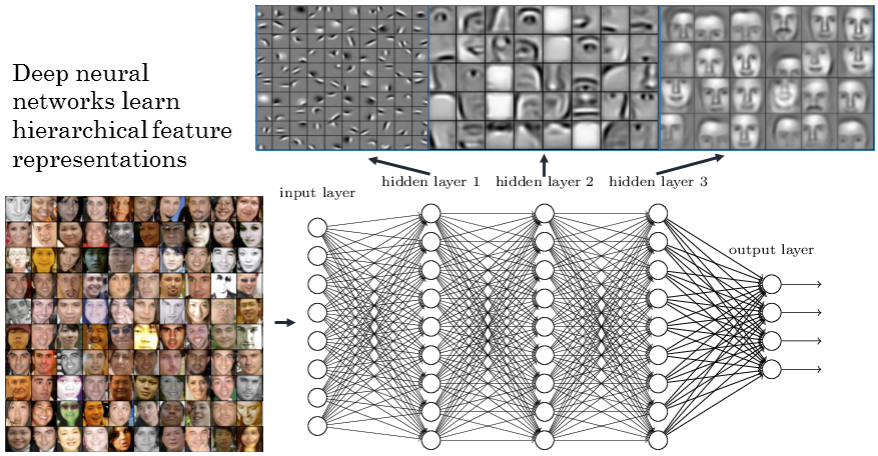

Figure 2.3: Feature extraction of a deep model [31]

the Fig. 2.3 shows. In a second phase, the features can be analyzed by different layers, for instance dense layers can elaborate features for classification or for other tasks. The filters weights of the convolutional layer are learned in the training and applied to the whole input where the neurons share the same weights. For this reason, respect to a dense layer, the convolutional layer has a lower weights count, is much faster to train and allocate a minor amount of memory. LSTM and GRU layers were created to allow the model to recognize patterns among a sequence of data such as video or audio. Other layers like the pooling layer, max-pooling layer and dropout were created to optimize the training.

2.3 The Deep Learning Paradigm

Deep Learning is a subfield of Machine Learning focalized in the study of deep neural networks. The model architecture is made in such a way to extract

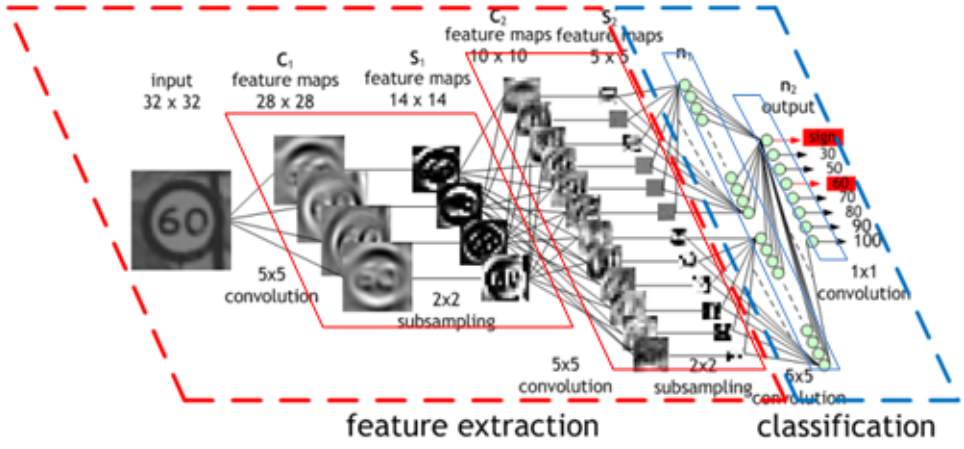

Figure 2.4: Image by Maurice Peemen

features from the data and after lean from them. This is the main difference between the Deep Learning and other Machine Learning techniques. Other Machine Learning techniques are focused on learning from handcrafter features. The extraction process of this features should be tuned for the specific data structure and require a high knowledge of particular context.

In DL, the net is trained also for features extraction, but it require a more complex architecture. The Fig. 2.4 propose a structure of a DL model.

Moreover, it gave a strong impulse to the development of very efficient algorithms in the computer vision field. Indeed, nowadays there are a lot of tools that simplifies the use of complex models trained to recognize hundreds of objects in an image. The so called fine-tuning technique allows net to recognize a different set of objects.

The purpose of fine-tuning is to reset the weights of final layers in a way that the model from the same features extracted, can accomplish another task.

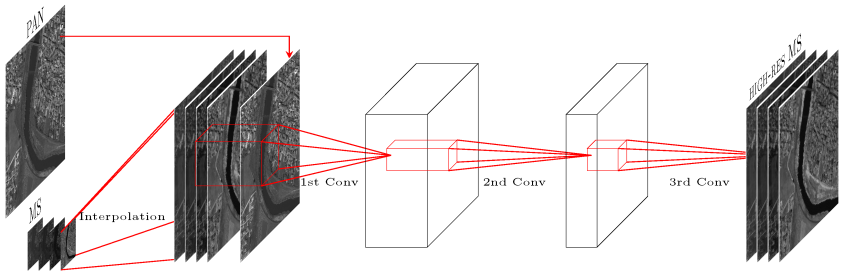

Figure 2.5: CNN architecture for pansharpening[4]

With this technique it will be just necessary train final layers instead of the entire model for days and using large datasets.

2.4 Pansharpening Applications

Recently a new pansharpening method based on a convolutional neural network has been proposed [4]. To accomplish that, it has been specialized a network built for super-resolution [32]. From that model, other three different models has been assested for GeoEye1, IKONOS and WorldView2 sensors with the aim of predict the sensor characteristics and give better performance.

An important issue in this field is that it is difficult to found good images because of the high cost for high-resolution images. For this reason, it cannot be possible train deep network. The choice of the authors was to use three convolutional layers, illustated in Fig. 2.5. The advantage of using only convolutional layers is that the input can have differente sizes.

The training phase showed in Fig. 2.6 uses a reference approach in which the images are downsampled and fused. After the downgrande, the P_L image

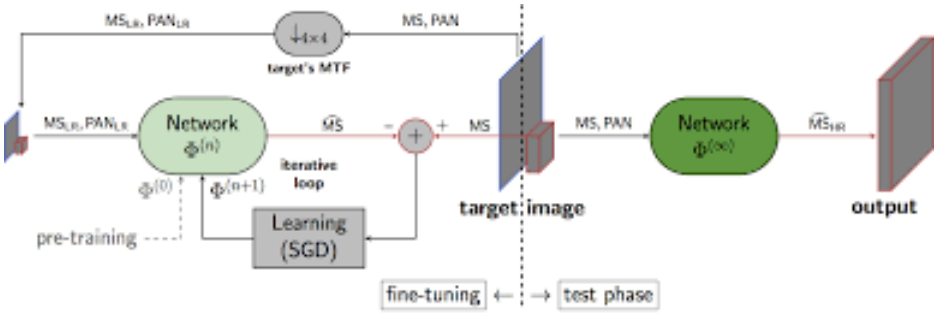

Figure 2.6: Training and Test[33]

is attached to the MS downgraded in order to provide input to the model. The MSE error between MS and the fused image would be calculated and this error will be used for the training of the weights.

Radiometric relevant indexes has been used to improve the results [4]. It was showed that the network avoids learning the indexes provided with more focus on other information. Another important step introduced in [33] was to run a fast session of fine-tuning before the application of the model. With the fine-tuning, the net can learn how to fuse the input in a downgraded version and after apply the prediction in the original images with better results. However, as described in the conclusion of [33], performance in downsampling domain is not as much relevante as no-reference measures that have a major impact to performance. Another important disadvantage of the method is that when the model works with misaligned images it is trained with images that have a fraction of the misalignment depending on the scale ratio between PAN and MS.

In another paper [34], the authors has tried different structures and strate-

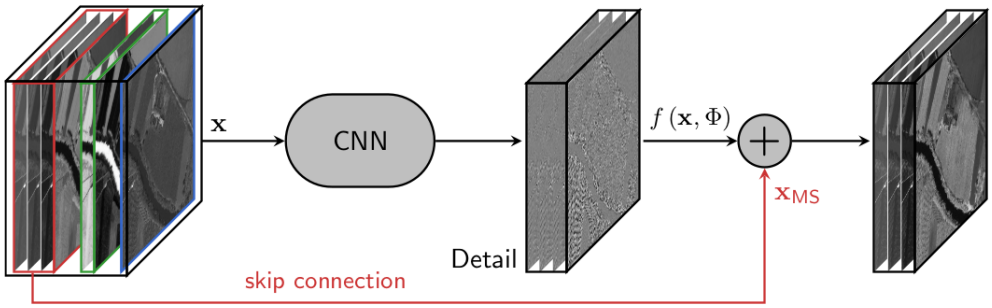

Figure 2.7: Residual-based version [34]

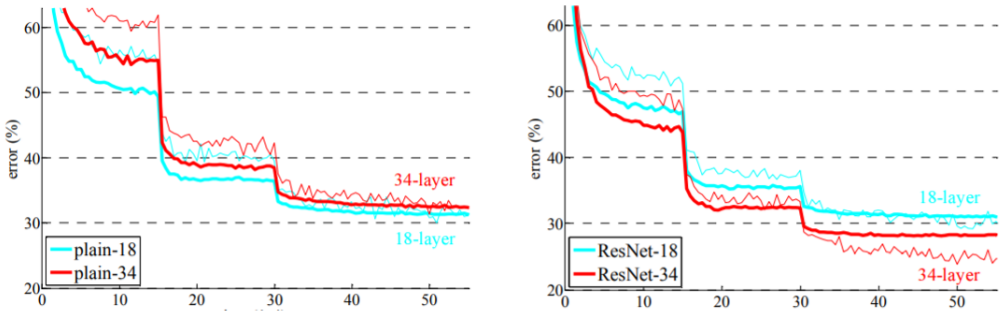

Figure 2.8: Performance comparison between residual (right) and non residual (left) networks [35]

gies to improve the network released in [4]. An important result was achived with a residual version of the original network. The structure is showed in Fig. 2.7.

The residual version was not trained to reproduce the whole image, but just the high-pass component. Indeed, the low-pass component is represented by the multispectral image provided by input. This means that the network reconstructs just the missing parts.

Residual-based structures were used in different papers ([36] and [37]), in

particular in the deep learning in [38] and [35].

As overall, it was observed that it is easier to train a neural network for differences instead of reproducing an ouput really similar to the input.

Results of [35] are indicated in Fig. 2.8.

Chapter 3

Proposed Solution

3.1 Introduction

The aim of this chapter is to discuss and evaluate the difference between the training of the reduced resolution approach in [4] and the training of the no-reference approach developed in the research. The analysis will cover the Automatic Differentiation and the implementation in Tensorflow.

3.2 Loss Function Issue

To assest the training set for the reduced resolution method (also called RR) described in [4] , the algorithm synthetized in the Fig. 3.1 has been implemented.

Both PAN and MS are downsampled into two images called MS_L and PAN_L. The MS_L is upsampled to match the PAN_L size. The images represent

the input of the model that compute the eror by the use of MS.

Instead of this procedure, the reseachers decided to use the no-reference approach (NOREF) illustrated in Fig. 3.2. As the figure shows, the images are not downsampled. The great advantage is that the model is trained with origianl images and not with the downgraded version. Also in this case, the MS is upsampled to match the PAN size. Without the upsampling, it will not be possible to stack the images and apply the 3D filters of the convolutional layers.

In this case, the loss function does not use a target. This is because the model is trained with the images at the higher possible resolution. Indeed, the loss computation is done using a no-reference index. For this reason, this approach belong into the Unsupervised Learning field. Opposite to Supervised Learning, Unsupervised learning is a subfield of Machine Learning where the training process is not leaded by a target.

3.3 Automatic Differentiation

Automatic differentiation is a set of techniques to efficiently calculate the gradient of a function with a computer. Every computer program executes a sequence of elementary operations. There are different kind of techniques to differentiate a function: Automatic Differentiation, Numerical Differentiation and Symbolic Differentiation. Numerical differentiation is the standard defi-

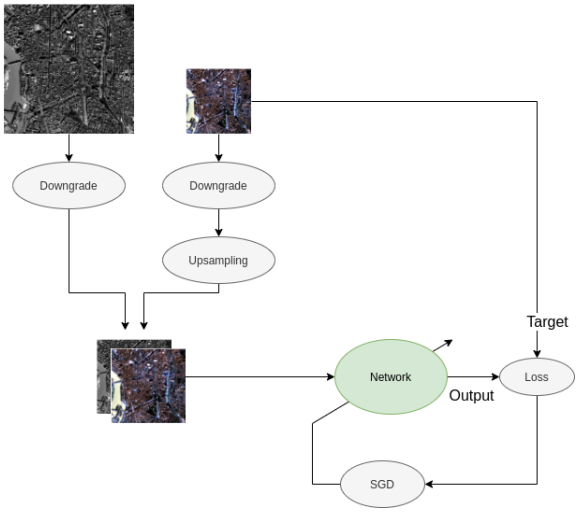

Figure 3.1: Graph of the training algorithm released by [4]

nition of a derivative.

$$\frac{df(x)}{dx} = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \tag{3.1}$$

As described in [39], this type of differentiation is easy to code. However, the $O(n)$ cost of the evaluation is so high that Machine Learning, where n can be millions or also billions, would not be accessible.

Symbolic differentiation is a technique that use the chain rule, product rule and other rules to split the expression in known derivative primitive to obtain result.

Symbolic differentiation is inefficient in terms of performance. The complexity can be, in a lot of cases, exponential. Furthermore, techniques that belong into this class are really difficult to convert into a computer programs.

Automatic differentiation systems explicitly split the operations in a sim-

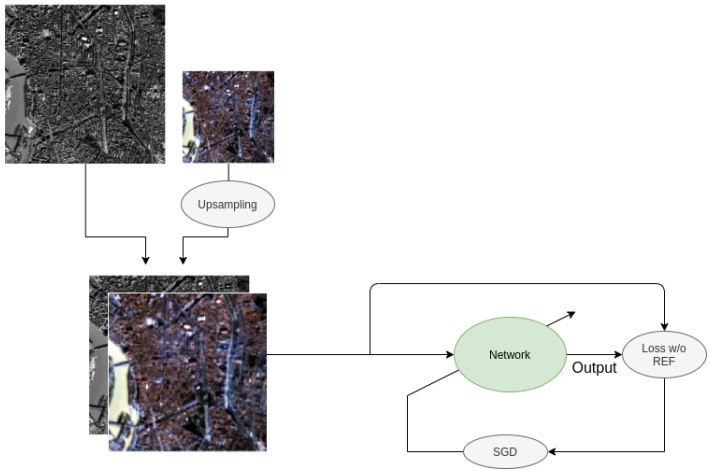

Figure 3.2: Graph of the training algorithm developed

plier ones and build a computation graph. Each node of the graph have some attributes like value, primitive operation and parents.

Jacobian matrixes are used to represente the derivative of each output y with respect to each input x.

$$J_f = \begin{bmatrix} \frac{dy_1}{dx_1} & \cdots & \frac{dy_1}{dx_n} \\ \vdots & \ddots & \\ \frac{dy_m}{dx_1} & & \frac{dy_m}{dx_n} \end{bmatrix} \tag{3.2}$$

For each primitive operation, it must be defined a Vector-Jacobian Product (VJP). Combining all the VJP nodes, the result would be the value of the gradient.

Automatic differentiation provide different modes like Forward mode and Reverse mode. In Forward mode, automatic differentiation and symbolic differentiation are equivalent as described in this paper [40]. They both apply the

chain rule and other differentiation rules and actually create expression graphs. However, the first one was created specifically for the computer manipulation and includes numerical values. The second one, symbolic differentiation, operates on mathematical expressions and symbols. Indeed, the automatic differentiation can handle also control flow statements like "if", "while" and "loops".

3.4 Tensorflow and custom loss function implementation

Tensorflow is the most famouse machine learning and deep learning tool that implement the automatic differentiation. To build a custom loss function, Tensorflow provide APIs. This APIs, define all the operations betweens Tensors that are the basic datatype in Tensorflow. The Tensorflow APIs used in the research code are showed in the Table 3.1. The external package tensorflow probability APIs used are showed in the Table 3.2.

After the definition of the custom loss function, at runtime Tensorflow translate the python code in C/C++ for efficience purpouse, compile it, and build the computation graphs. Also, tensorflow from a couple of years integrate on it Keras, another framework that simplify the creation of the model with all common layers well-defined into Classes. To create a model it is necessary to create a $Model$ class. It is possible to add a layer using $model.add(Layer(args))$. $Layer$ is the corresponding class of the layer and

args are the arguments like the input shape, the number of filters the layer should have. Firstly, using such a widespread framework has the advantage of a large community that support difficult situations; secondly it will be a larger compatibility on Operating Systems, GPUs and hardware in general.

Table 3.1: Table of tensorflow API used in the thesis

abs(...);	Computes the absolute value of a tensor.
add(...);	Returns x + y element-wise.
cast(...) :	Casts a tensor to a new type.
constant(...) :	Creates a constant tensor from a tensor-like object.
divide(...);	Computes Python style division of x by y.
$expand_dims$(...) :	Returns a tensor with an additional dimension inserted at index axis.
multiply(...);	Returns an element-wise x * y.
ones(...) :	Creates a tensor with all elements set to one.
pad(...) :	Pads a tensor.
reduce_mean(...);	Computes the mean of elements across dimensions of a tensor.
reduce_std(...);	Computes the standard deviation of elements across dimensions of a tensor.
reduce_sum(...);	Computes the sum of elements across dimensions of a tensor.
sqrt(...);	Computes element-wise square root of the input tensor.
square(...);	Computes square of x element-wise.
squeeze(...) :	Removes dimensions of size 1 from the shape of a tensor.

Table 3.2: Table of tensorflow probability API used in the thesis

stats.covariance(...);	Sample covariance between observations indexed by event_axis.

3.4.1 Loss Function

To build the first loss function, it was used a QNR approximation.

$$f(x) = 1 - \widetilde{QNR} \tag{3.3}$$

where $\widetilde{QNR}$ is the approximated QNR.

The D_s and D_λ were calculated using the Qavg that is the average of Q evaluation for each band. The Q was calculated considering the whole image and not dividing it into small patches. The result shows series of negative values instead of values between 0 and 1 as the original images format. This is why the performance was unsatisfactory (results illustated in the Chapter 4). Indeed, the model wants to minimize the loss functions (maximize the QNR considering 3.3) no matter the sense of output values. After some tests, it was considered to solve this problem using the D_sreg described in [41] instead of D_s and migrate to an approximation of HQNR instead of the QNR. Implement the exact HQNR it was impossible due to the difficulty on manupulating quaternions and hypercomplex numbers in TensorFlow in such a way that the function would be kept differentiable. In order to gain better performance, different kind of D_λ has been implemented according to the scientific community and the QNR author itself.

Chapter 4

Implementation details and experimental results

4.1 Description

The code was written in python 3.7 using the file main.py as a entrypoint. It is compatible with Linux, Windows and MacOS and can run on dedicated GPU as well as on CPU. Obviously, on CPU with a large decrease in performance in the training. The software was implemented using the standard *argparse* library in such a way that all the possible algorithms, methods, parameters, learning rate and so on, can be defined with arguments on the terminal. This allow to run, with a bash file, all the experiments in series without changing the code.

In the beginning, the QNR function has developed with an approximation.

In the Appendix .1 the reader can find all the principal functions written for the QNR. As described in the code, the Q has obtained as an average of the Q calculated band-by-band. This means is not calculated in patches as the original paper of the QNR describe. The rest of the function is as similar as possible to the original paper. As formalized in the Eq. 1.22, to obtain the D_lambda, for each band of MS and the fused image, the Q is calculated between the band and the others. This can gives a consistency measure between bands. To obtain the D_s, according to the Eq. 1.21, for every band of the MS and the fused image it is calculated the Q between the band and pan degraded and pan, respectively. This gives a quality measurements of the details inserted into the image.

Because the QNR poor performance, it was decided to use the HQNR. Also the implementation of this index have some approximations due to the lack of tensorflow hypercomplex API. To calculate the D_λ, instead of using a Q4, it was used the same implementation of the Q average used for the QNR. The team has opted for a D_sreg, an implementation of the D_s that take advantages from the $rsquared$ calculation.

The implementation is illustated in the Appendix .2. The D_sreg is obtained as $1 - rsquared$ between the fused image and the pan. The D_lambda is the result of $1 - Q$ between the fused image filtered with a gaussian filter and the MS. The filter used can be different depends on the sensor used for the image acquisition. This because the filter should match the sensor MTF. The filters were created with the Matlab Toolbox, exported in the .mat format and

imported in the project at runtime. It was decided to not use padding after the filtering process and cut the extra-pixel from the MS for the Q calculation.

An important choice was the learning rate. The learning rate is multiplied with the gradinet of the error for each layer. This means that determines how quickly the descent will be. With a too fast descent, the model could not reach the minimum but with a too low learning rate, it can take long time to reach the convergence or get stuck in a local minimum. The best approach used by the community is to use a learning rate scheduler like Adam or SGD. At the start of the training, the model can have a high learning rate so that can explore all the loss function and after reaching a great descent, gradually degrade the learning rate and avoid oscillations. The learning rate is one of the parameter that require some experiments to be tuned. The best case is to reach the maximum possible quality with a constant increase in performance during the training. The main index of performance was the q2n. This because the q2n is, in this moment, the best index and being a reference index means to be really robust respect to a non reference one. It is important in the training to avoid a bell shape in the index graph. The bell shape indicates that the q2n increase really fast at the beginning and after reaching the pick value of optimum, decrease really fast. With this behaviour, it can be really difficult in real application stops the training in the right epoch because it is not possible to calculate the q2n and at some point the model can have a performance degradation.

Without the bell shape during the training, the model can only have an

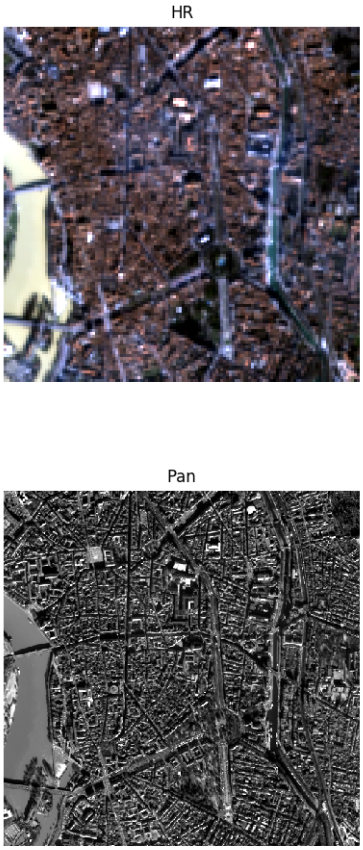

Figure 4.1: Multispectral and Pancromatic Toulouse images used for all the tests

increase in performance and after, a stable phase in which the index has not a notable mutation.

4.2 Experimental Settings

All types of training have been executed in a test image illustated with a true color representation in Fig. 4.1. Toulouse dataset: An IKONOS image has

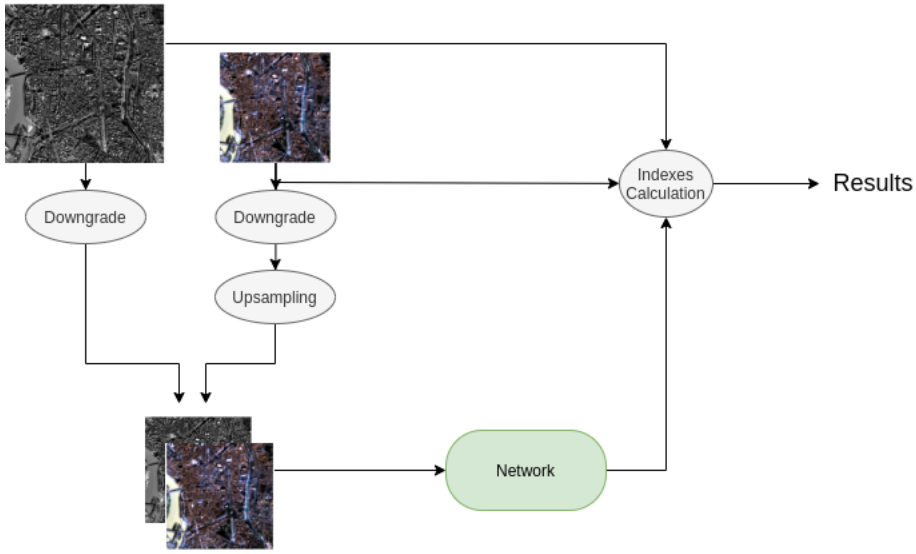

Figure 4.2: Validation process for RR and NOREF method

been acquired over the city of Toulouse, France, in the 2000. The MS sensor acquires an image of 512×512 pixels in the visible and near-infrared spectrum range in 4 different bands and with a 4m spatial sampling interval (SSI). The pancromatic image is constitute by an image of 2048×2048 pixels with a 1m SSI.

In general, deep learning models should not be specialized too much for the training set. The reason why is that the model could have worse performance for data in which the it is not trained for, like in the real world. To avoid this situation, is necessary to create a validation set for the training. The validation set is a dataset in which there are data that net is not trained for. Performance on the validation set are more similar to real performance. The validation process is described in the Fig. 4.2.

To generate a reference image for the quality assessment, MS and PAN has

been downsampled. The original MS has been used as Ground Truth (GT). For the reason of brevity, the two downgraded images will be called MS_L and PAN_L for simplicity.

At every epoch, the PAN_L and MS_L were used to produce a fused image with the updated weights and compared with the GT. This means that these images constitute the validation set. Reference indexes like SAM, ERGAS and Q and also no reference indexes QNR and HQNR were calculated between the output of the model and the GT. Those indexes has been calculated with the MatLab toolbox functions provided by [20] and MatLab engine for python [42]. This procedure can give a real performance assessment of the whole model. The training set have always better performance as the model optimize the error of these data. It needs to be highlight that only the performance of the validation set are relevant.

For the training, as described in the Paragraph 3.2, the PAN_L and MS_L in the RR approach are downsampled again because the model should have a target different from the GT. Training the model with the GT as target have no valid applications. In real world, the model would not have a GT as target. Additionally, the validation set would have no relevant performance because the model is just trained with the same data.

It has been noticed that, for some images, the model that use the fine-tuning with the reference approach has worse performance to the original ones. The reason is that the model is fine-tuned with the downgrade images.

The characteristics of the sensors, in general, are not scale invariance and

downsampling the input does not guarantee similar performance like using the original images as described also in the previous chapters.

In the NOREF approach, the images are not downsampled again because the model use a no-reference loss function.

To compare the different methods, it was also trained the model with the GT image itself using the MSE loss function. The purpose is to compare the previous described methods with the best possible solution, but it is not applicable in a real approach.

4.3 Results

Using the original QNR function has not produce any positive results. During the training, while the error continued to be minimized, the indices calculated with the Matlab Toolbox worsened as showed in Fig. 4.3 4.4 4.5, even the QNR itself. This because the model, to minimize the error, generate an image with negative values. An input with negative values is not expected by any of the indexes used and also by the loss function. This create a discrepancy between the QNR developed in tensorflow and used in the loss function and the QNR of the Matlab Toolbox. Indeed, with inputs having values between 0 and 1, the QNR developed in tensorflow and the Matlab one returns the same results.

The loss tends to reach his minimum value but this is not reflected in an increase of the quality indexes.

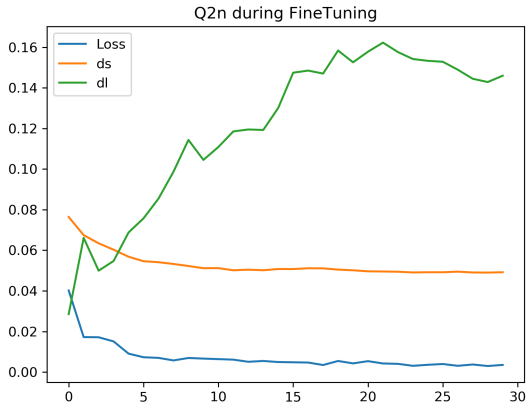

Figure 4.3: Loss, DS and DL of QNR during the training

After these tests, it was decided to change the loss function using another no-reference index: the HQNR. But this index is more complex than the QNR because it involve quaternions and operations with hypercomplex numbers.

As described in the previous chapter, it was applied an approximated version of HQNR and in the following graphs (Fig. 4.6) the results are illustated for the Ikonos Toulouse image.

With this type of loss function it was recorded an increase in all the indexes except for the QNR. The QNR, as evidenced also by the previous results, does not seems to be a good index for the quality assessment.

To explore all the possible values of q and p in the Eq. 1.21 1.22, tests in different cases have been lauched. At the beginning with steps of 0.5 and after for particupar ranges with steps of 0.25.

In Fig. 4.7 and 4.8, in the legend the first argument is alpha and the second is beta. They are the inverse respectively of p and q.

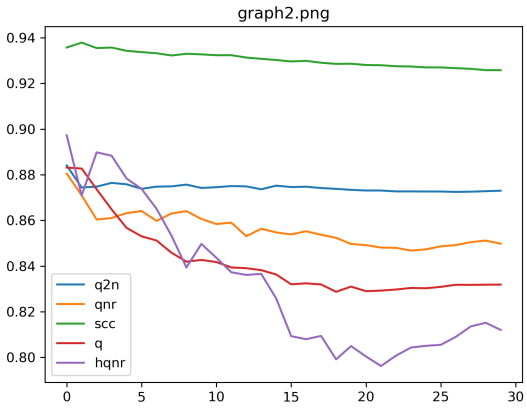

Figure 4.4: Q2N, QNR, SCC, Q and HQNR during the training

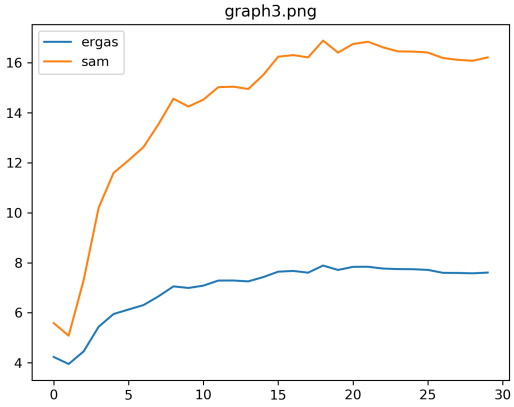

Figure 4.5: ERGAS and SAM during the training

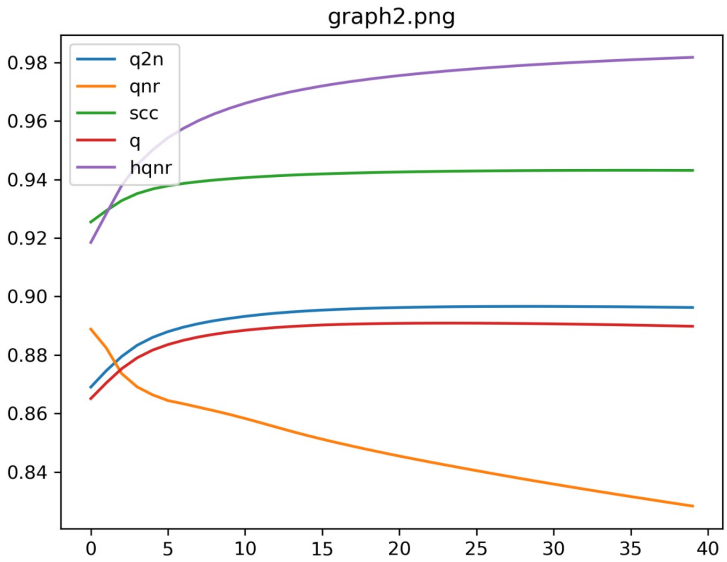

Figure 4.6: Q2N, QNR, SCC, Q and HQNR during the training using the loss function with HQNR

The experiment conclusion was that best result are reached with alpha 0.5 and beta 2.

After this experiments, all the methods have been compared and the results are available in Fig. 4.9. GT is the method that use the Ground Truth during the training to have the best performance, RR is the method used in [4] with a reduc ed resolution training and the NOREF is the method implemented in this thesis with a full-resolution training and HQNR index described in the previous paragraphs.

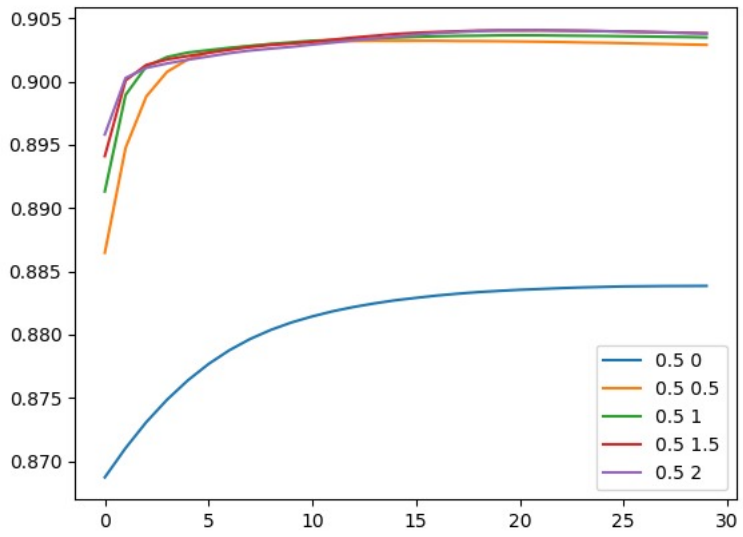

Figure 4.7: Alpha and Beta with a 0.5 step

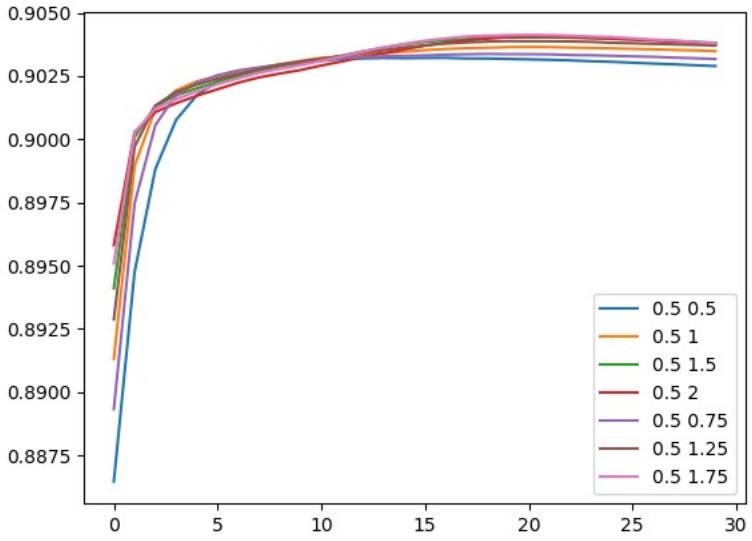

Figure 4.8: Beta with a 0.25 step between beta 0.5 and beta 2 and alpha setted to 0.5

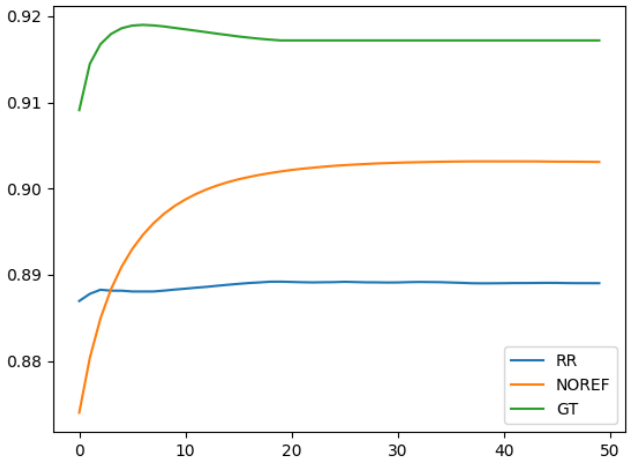

Figure 4.9: Experiment results with GT, RR and NOREF

Conclusions

"I always thought something was fundamentally wrong with the universe" [20]

Bibliography

[1] C. Souza Jr. L. Firestone L. M. Silva and D. Roberts. *Mapping forest degradation in the Eastern Amazon from SPOT 4 through spectral mixture models.* Remote Sens. Environ., 2003.

[2] A. Mohammadzadeh A. Tavakoli and M. J. Valadan Zoej. *Road extraction based on fuzzy logic and mathematical morphology from pansharpened IKONOS images.* Photogramm. Rec., 2006.

[3] F. Laporterie-Déjean H. de Boissezon G. Flouzat and M.-J. Lefèvre-Fonollosa. *Thematic and statistical evaluations of five panchromatic/-multispectral fusion methods on simulated PLEIADES-HR images.* Inf. Fusion, 2005.

[4] Giuseppe Masi Davide Cozzolino Luisa Verdoliva and Giuseppe Scarpa. *Pansharpening by Convolutional Neural Networks.* Remote Sensing, 2016.

[5] Pascal Lamblin. *MILA and the future of Theano.* `https://groups.google.com/forum/#!topic/theano-users/7Poq8BZutbY`, 2018.

[6] tensorflow.org. *Introduction to Gradients and Automatic Differentiation.* `https://www.tensorflow.org/guide/autodiff?hl=da`, 2020.

[7] W. Carper T. Lillesand and R. Kiefer. *The use of intensity-hue-saturation transformations for merging SPOT panchromatic and multi-spectral image data,.* Photogramm. Eng. Remote Sens., 1990.

[8] P. S. Chavez Jr. S. C. Sides and J. A. Anderson. *Comparison of three different methods to merge multiresolution and multispectral data: Landsat TM and SPOT panchromatic,.* Photogramm. Eng. Remote Sens., 1991.

[9] C. Thomas and L. Wald. *Analysis of changes in quality assessment with scale.* Proc. 9th Int. Conf. Inf. Fusion, 2006.

[10] V. P. Shah N. H. Younan and R. L. King. *An efficient pan-sharpening method via a combined adaptive-PCA approach and contourlets.* IEEE Trans. Geosci. Remote Sens.,, 2008.

[11] C. A. Laben and B. V. Brower. *Process for enhancing the spatial resolution of multispectral imagery using pan-sharpening,.* U.S. Patent 6 011 875, 2000.

[12] L. Wald T. Ranchin and M. Mangolini. *Fusion of satellite images of different spatial resolutions: Assessing the quality of resulting images.* Photogramm. Eng. Remote Sens, 1997.

[13] L. Wald C. Thomas, T. Ranchin and J. Chanussot. *Synthesis of multispectral images to high spatial resolution: A critical review of fusion*

methods based on remote sensing physics. IEEE Trans. Geosci. Remote Sens., 2008.

[14] M. Vega R. Molina I. Amro, J. Mateos and A. K. Katsaggelos. *A survey of classical methods and new trends in pansharpening of multispectral images*. EURASIP J. Adv. Signal Process., 2011.

[15] B. Aiazzi S. Baronti and M. Selva. *Improving component substitution pansharpening through multivariate regression of MS+Pan data*. IEEE Trans. Geosci. Remote Sens., 2007.

[16] T.-M. Tu S.-C. Su H.-C. Shyu and P. S. Huang. *A new look at IHS-like image fusion methods*. Inf. Fusion, 2001.

[17] T.-M. Tu P. S. Huang C.-L. Hung and C.-P. Chang. *A fast intensity-hue-saturation fusion technique with spectral adjustment for IKONOS imagery*. IEEE Trans. Geosci. Remote Sens., 2004.

[18] W. Dou Y. Chen X. Li and D. Sui. *A general framework for component substitution image fusion: An implementation using fast image fusion method*. Comput. Geosci., 2007.

[19] J. Choi K. Yu and Y. Kim. *A new adaptive component-substitution based satellite image fusion by using partial replacement*. IEEE Trans. Geosci. Remote Sens., 2011.

[20] Gemine Vivone Luciano Alparone Jocelyn Chanussot Mauro Dalla Mura andrea Garzelli Giorgio A. Licciardi Rocco Restaino and Lucien Wald. *A*

Critical Comparison Among Pansharpening Algorithms. IEEE Transactions on Geoscience and Remote Sensing, 2015.

[21] L. Alparone et al. Comparison of three different methods to merge multiresolution and multispectral data: Landsat tm and spot panchromatic. 2007.

[22] F. Laporterie-Déjean H. de Boissezon G. Flouzat and M.-J. Lefèvre-Fonollosa. *Thematic and statistical evaluations of five panchromatic/-multispectral fusion methods on simulated PLEIADES-HR images*. Inf. Fusion, 2005.

[23] B. Aiazzi L. Alparone S. Baronti A. Garzelli and M. Selva. *MTF-tailored multiscale fusion of high-resolution MS and Pan imagery,*. Photogramm. Eng. Remote Sens., 2006.

[24] Z. Wang and A. C. Bovik. *A universal image quality index*. IEEE Signal Process. Lett., 2002.

[25] S. Garzelli A. Alparone, L. Baronti and Nencini F. *A global quality measurement of pan-sharpened multispectral imagery*. IEEE Geosci. Remote Sens., 2004.

[26] L Alparone et al. *Multispectral and panchromatic data fusion assessment without reference*. Photogramm. Eng. Remote Sens., 2008.

[27] Khan M. M. Alparone L. and Chanussot J. *Pansharpening quality assessment using the modulation transfer functions of instruments.* IEEE Trans. Geosci. Remote Sens., 2009.

[28] B. Aiazzia L. Alparonea S. Barontia R. Carl A. Garzellia L. Santurri. *Full scale assessment of pansharpening methods and data products.* Photogramm. Eng. Remote Sens., 2008.

[29] Warren S. McCulloch and Walter Pitts. *A Logical Calculus of Ideas Immanent in Nervous Activity.* University of Chicago, 1943.

[30] MissingLink. *Perceptrons and Multi-Layer Perceptrons: The Artificial Neuron at the Core of Deep Learning.* https://missinglink.ai/guides/neural-network-concepts/perceptrons-and-multi-layer-perceptrons-the-artificial-neuron- at-the-core-of-deep-learning.

[31] RSIPvision. https://www.rsipvision.com/exploring-deep-learning.

[32] K. Tang X Dong C. Loy C. He. *Image super-resolution using deep convolutional networks.* IEEE Trans. Pattern Anal. Mach. Intell., 2006.

[33] Giuseppe Scarpa Sergio Vitale and Davide Cozzolino. *CNN-based pansharpening of multi-resolution remote-sensing images.* Conference: 2017 Joint Urban Remote Sensing Event (JURSE).

[34] Giuseppe Scarpa Sergio Vitale and Davide Cozzolino. *Target-adaptive CNN-based pansharpening.* 2017.

[35] K He X Zhang S Ren and J Sun. *Deep residual learning for image recognition.* IEEE Conf on Computer Vision and Pattern Recognition (CVPR), 2016.

[36] R Zeyde M Elad and M Protter. *On single image scale-up using sparse-representations.* Curves and Surfaces Springer, 2012.

[37] R Timofte V De and L V Gool. *Anchored neighborhood regression for fast example-based super-resolution.* IEEE Int Conf on Computer Vision (ICCV), 2013.

[38] J K L J Kim and K M Lee. *Accurate image super-resolution using very deep convolutional networks.* IEEE Conf on Compute Vision and Pattern Recognition (CVPR), 2016.

[39] Atilim Gunes Baydin Barak A Pearlmutter Alexey Andreyevich Radul and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. 2018.

[40] Sören Laue. *On the Equivalence of Forward Mode Automatic Differentiation and Symbolic Differentiation.* 2019.

[41] Luciano Alparone Andrea Garzelli Gemine Vivone. *Spatial Consistency for Full-Scale Assessment of Pansharpening.* IEEE International Geoscience and Remote Sensing Symposium, 2018.

[42] Matlab. *https://it.mathworks.com/help/matlab/matlab-engine-for-python.html.*

.1 QNR

```
import tensorflow as tf
import tensorflow_probability as tfp

def q_index(y_true, y_pred):
    two = tf.constant(2.0, tf.float32)

    cov_b = tfp.stats.covariance(y_true,y_pred, [0, 1], None)
    true_b_std = tf.math.reduce_std(y_true, [0, 1])
    pred_b_std = tf.math.reduce_std(y_pred, [0, 1])

    true_b_mean = tf.cast(tf.reduce_mean(y_true, [0, 1]), tf.float32)
    pred_b_mean = tf.cast(tf.reduce_mean(y_pred, [0, 1]), tf.float32)

    q1_b = cov_b / (true_b_std * pred_b_std)
    q2_b = (two * true_b_mean * pred_b_mean) / (tf.square(true_b_mean) +
    q3_b = (two * true_b_std * pred_b_std ) / (tf.square(true_b_std) + t

    q_b = q1_b * q2_b * q3_b
    return tf.reduce_mean(q_b)


def d_lambda(ms, fused, p, b):
    result = tf.constant(0.0, tf.float32)
    for l in range(b-1):
        for r in range(l+1, b):
            result += tf.abs(tf.cast(q_index(fused[:, :, l:l+1], fused[:
            tf.cast(q_index(ms[:, :, l:l+1], ms[:, :, r:r+1]), tf.float3

    b = tf.constant(b, tf.float32)

    s = ( b * ( b - tf.constant(1.0, tf.float32) ) ) / tf.constant(2.0,

    result = result / s
    result = result ** (1.0/p)
    return result
```

```
def d_s(ms, fused, pan, pan_degraded, q, b):
    result = tf.constant(0.0, tf.float32)

    for l in range(b):
        result += tf.abs(tf.cast(q_index(fused[:, :, l:l+1], pan), tf.fl
         tf.cast(q_index(ms[:, :, l:l+1], pan_degraded), tf.float32))**q

    b = tf.constant(b, tf.float32)

    result = result / b

    r = result**(1./q)
    return r


def qnr(fused, ms, pan, pan_degraded, alpha, beta, p, q, bands):
    a = (1-d_lambda(ms, fused, p=p, b=bands))**alpha
    b = (1-d_s(ms, fused, pan, pan_degraded, q, b=bands))**beta
    return a*b
```

.2 HQNR

```
def filter_image(origin_image, kernel):
    image = tf.expand_dims(origin_image, 0)
    kernel = tf.expand_dims(kernel, -1)
    output = tf.nn.depthwise_conv2d(image, kernel, strides=(1, 1, 1, 1),
    output = tf.squeeze(output)
    return output

def gaussian_filtered_image(image, sensor):
    if sensor == "WV2":
        gauss_kernel = hWV2
    elif sensor == "WV3":
        gauss_kernel = hWV3
    elif sensor == "GeoEye1":
        gauss_kernel = hGE1
    else:
        gauss_kernel = hIK

    return  filter_image(image, gauss_kernel)
```

```
def d_s_reg(ms, pan):
  ms = tf.reshape(ms, (ms.shape[0] * ms.shape[1], ms.shape[2]))
  pan = tf.reshape(pan, (pan.shape[0] * pan.shape[1],1))

  alpha = tf.matmul(pinv(ms), pan)
  fi = tf.matmul(ms, alpha)
  return 1 - r_squared(pan, fi)


def d_lambda(ms, fused, p, b, sensor):
  fused_filtered = gaussian_filtered_image(fused, sensor)
  return 1 - q_index(fused_filtered, ms[R:-R, R:-R, :])

def hqnr(fused, ms, pan, pan_degraded, alpha, beta, p, q, bands, sensor)
  a = ( 1 - d_lambda_consistence(ms, fused, p=p, b=bands, sensor=sensor)
  b = ( 1 - d_s_reg(fused, pan)) ** beta
  return a*b
```