SPATIAL ENHANCEMENT OF REMOTELY SENSED IMAGES USING CONVOLUTIONAL NEURAL NETWORKS

Ugo Palatucci

July 2020

Supervisor:

Prof. Restaino Rocco

Contents

Introduction

Pansharpening refers to a particular data fusion issue where two images, one panchromatic (PAN) and one multispectral (MS), 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 sensor, instead, collects several channels at a lower spatial resolution. As physical constraints occur, the creation of a sensor specialized in both types of resolution 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].

Based on a convolutional neural network, a new pansharpening method has been proposed recently [4]. Using degraded versions of the PAN and MS images, the network weights were trained to fuse the images, optimizing 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 Quality with no reference (QNR) or hybrid QNR (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 Deep Learning library maintained by Google that allows more compatibility with the newest libraries and a larger community, helping the development in case of uncommon errors. Moreover, Tensorflow implements the *automatic differentiation* [6], a critical feature for this research. *Automatic differentiation* 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 existing literature, the pansharpening techniques are divided into two main areas: component substitution (CS) and 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 introduction of spatial details, extracted from the PAN image through a multgiscale decomposition, 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 definitions 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.

Accordingly, there are two techniques that can be used for the quality assessment. The first is to degrade 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 in [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.

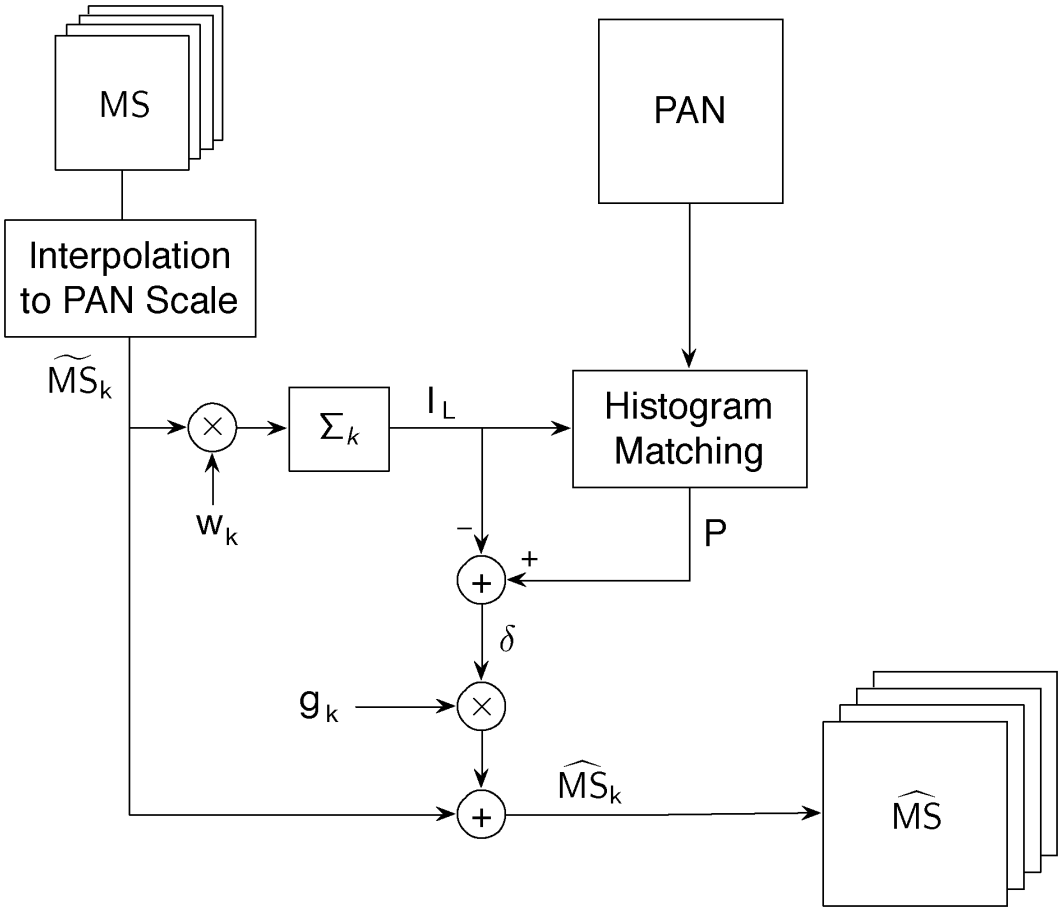

Figure 1.1: Flowchart of CS approach [20]

In the IHS pansharpening method the IHS transformation is used. This is the major limitation of this technique because it can transform only RGB images 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 well-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 larger 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]. In [15], also an *adaptive* version of this mode was proposed. It is called GSA and the 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] and 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:

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

1.2 MRA

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

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

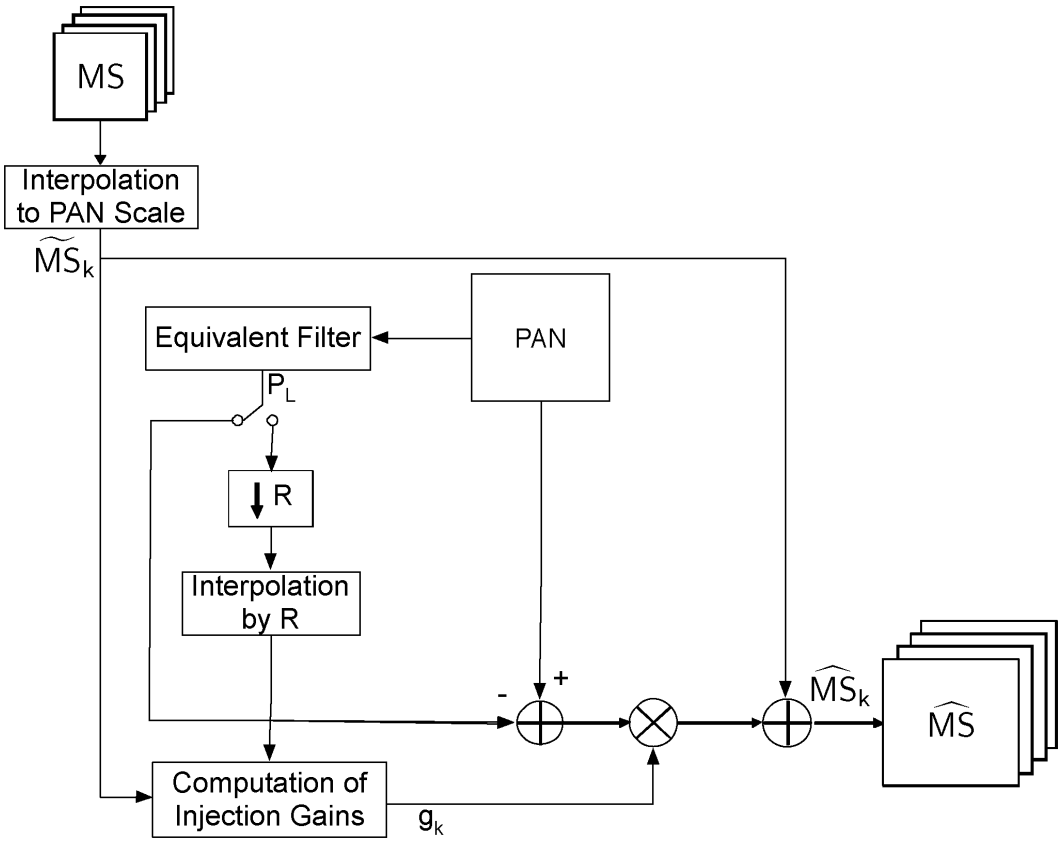

Figure 1.2: Flowchart of MRA approach [20]

$P - P_L$ 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 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 happens because the performance assessment depends intrinsically on 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 simulate 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 by means of 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.

Root Mean Square Error (RMSE) is used to calculate the spatial/radiometric distortions. It is defined as:

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

where $E[]$ is the arithmetic mean. 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 Erreur Relative Globale Adimensionnelle de Synthèse

(ERGAS) 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 RMSEs, 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 construct 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 :

$$Q4 = \frac{4|\sigma_{z_A z_B}| \cdot |\bar{z_A}| \cdot |\bar{z_B}|}{(\sigma^2_{z_A} + \sigma^2_{z_B}(|z_A|^2 + |z_B|^2))} \tag{1.19}$$

where $\bar{z}$ is the complex conjugate. 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 from 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 UIQIs 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]:

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

in which α and β are coefficient tuned to weight more the spectral or the

spatial distortion as in the Eq. 1.20 and $D_\lambda^{(K)}$ 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 (DL) 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, DL 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 uses 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 (ML) and in particular, DL.

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 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

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

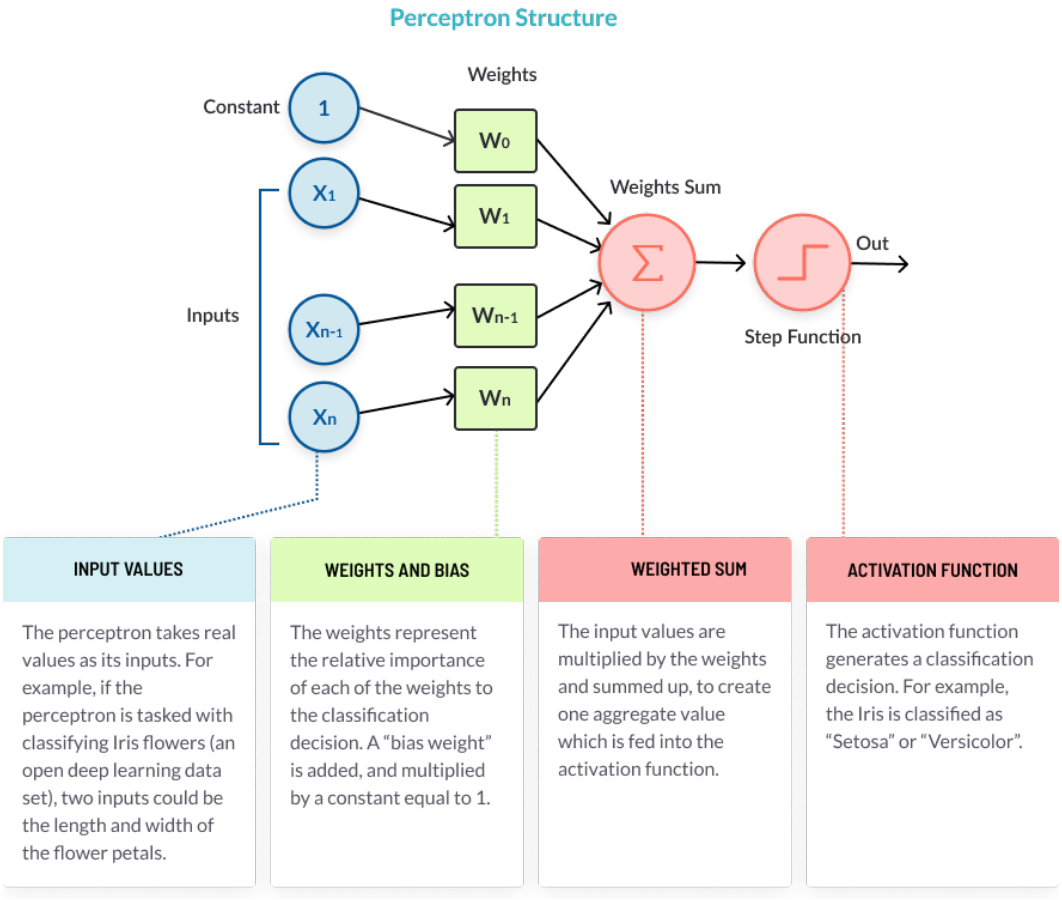

Figure 2.1: Perceptron in detail [30]

called 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 equal to 1 means that the training will be much faster compared with other activation functions like logistic sigmoid.

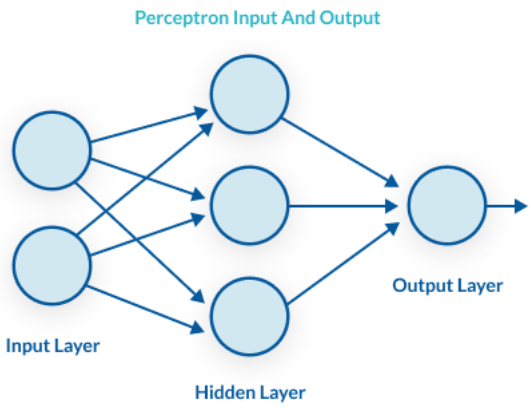

Figure 2.2: Multilayer architecture example [30]

2.2.3 Layers

The DL network can have different type of layers that differ in how the perceptrons are organized and how they modify their weights during the training. Fig. 2.2 shows an example of a 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 considering all the neurons connections, this type of layer generates promptly a large number of weights. This requires 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 with complexity that grows with the model dimension, as the Fig. 2.3 shows. In a second phase, the features can be

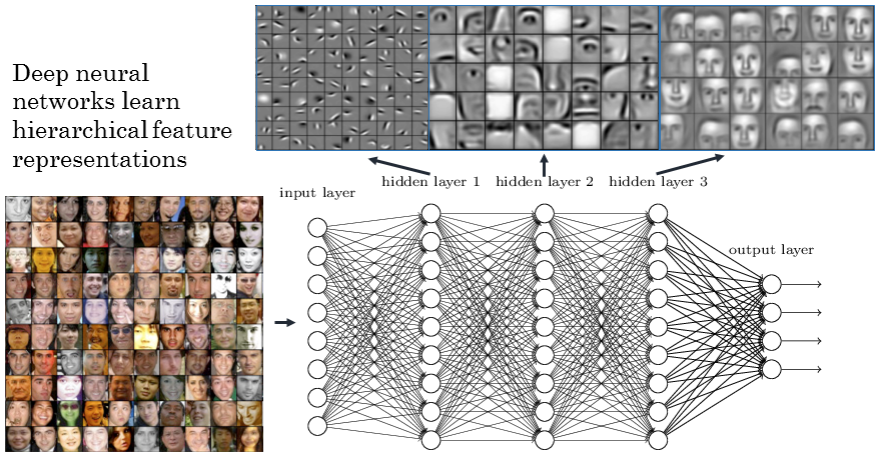

Figure 2.3: Feature extraction of a deep model [31]

analyzed by different layers, for instance dense layers can elaborate features for classification or for other tasks. The filter 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 allocates a minor amount of memory.

To recognize patterns among a sequence of data such as video or audio, Long short-term memory (LSTM) and Gated Recurrent Unit (GRU) are used ad, unlike standard layers, have feedback connections that allow the model to memorize data for subsequent inputs. Other layers like the pooling layer, max-pooling layer and dropout were created to optimize the training.

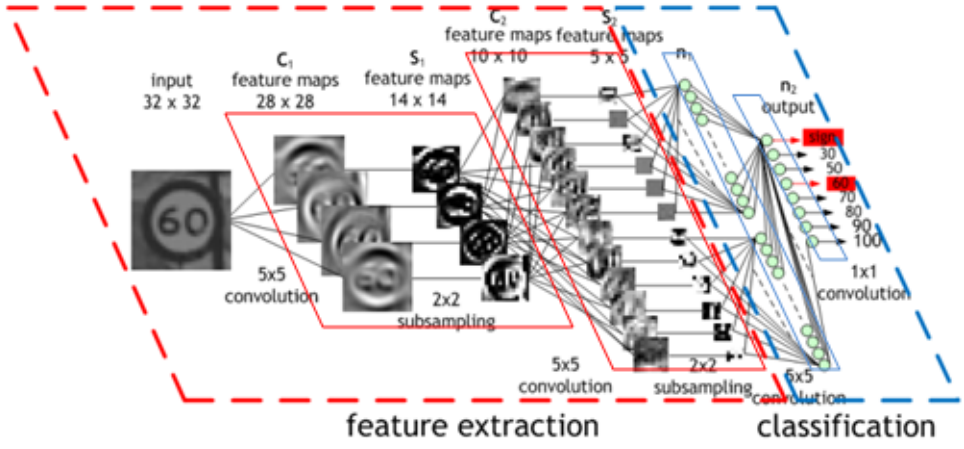

Figure 2.4: Image by Maurice Peemen

2.3 The Deep Learning Paradigm

DL is a subfield of ML focalized in the study of deep neural networks. The model architecture is made in such a way to extract features from the data and recognize patterns from them. This is the main difference between the DL and other ML techniques. Other ML techniques are focused on learning from handcrafter features. The extraction process of this features should be tuned for the specific data structure and requires a high knowledge of the particular context.

In DL, the net is trained also for feature extraction, but it require a more complex architecture. Fig. 2.4 illustrates 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 simplify 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 learned on some given features, can accomplish another task. With this technique it will be just necessary to 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 have been tuned for GeoEye1, IKONOS and WorldView-2 sensors with the aim of predicting the sensor characteristics and giving 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 to train deep network. The choice of the authors was to use three convolutional layers, illustrated in Fig. 2.5. The advantage of using only convolutional layers is that the input can have different 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 is attached to the downgraded MS in order to provide input to the model. The MSE error between the original MS and the network output would be

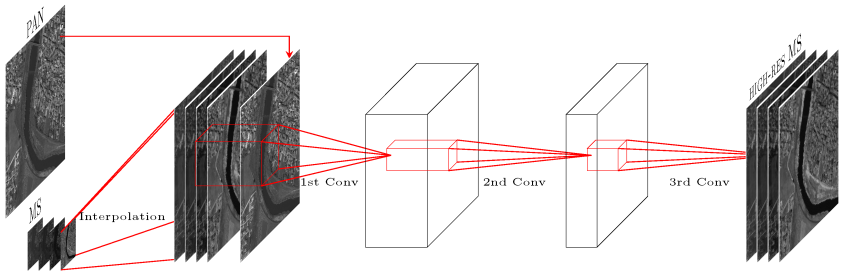

Figure 2.5: CNN architecture for pansharpening[4]

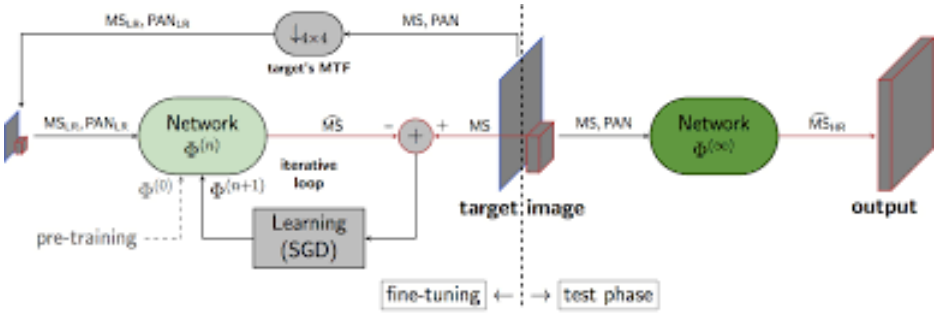

Figure 2.6: Training and Test[33]

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. 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 specialize the weights for the specific environment described by the downgraded image and after realize on the original images, the fusion with better results. However, as described in the conclusions of [33], performance with this approach is not as much relevant as *no-reference* approach 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

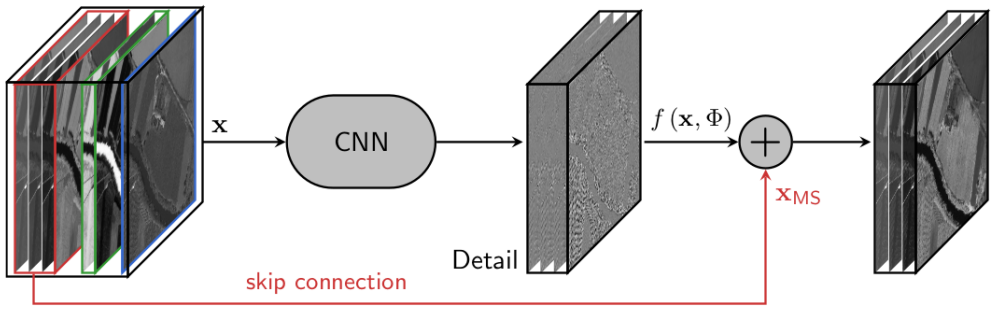

Figure 2.7: Residual-based version [34]

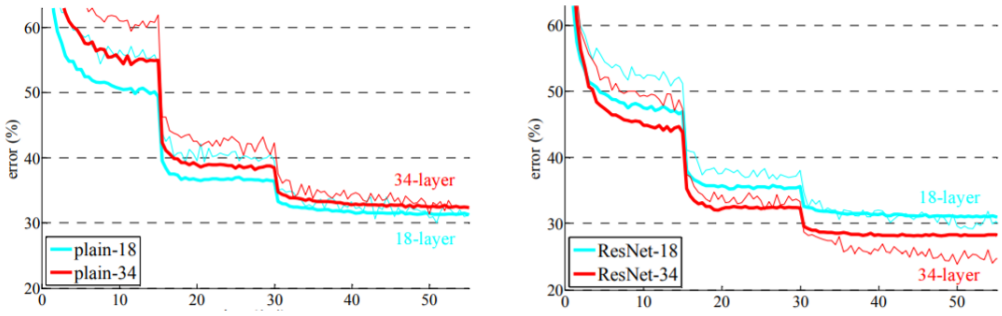

Figure 2.8: Performance comparison between residual (right) and non residual (left) networks [35]

fraction of the misalignment depending on the scale ratio between PAN and MS.

In another paper [34], the authors has tried different structures and strategies to improve the network released in [4]. An important result was achieved 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, 37]), in particular in the DL in [38, 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.

The main results of [35] are shown 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 proposed 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 assess the training set for the *reduced resolution* (RR) method described in [4], the algorithm synthesized 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 error by the use of MS. To update

the model weights, a learning rate scheduler is used: the SGD [39]. A learning rate scheduler consent to modify the learning rate during the training in order to optimize performance and reach easier the minimum error. Other details about learning rate and learning rate scheduler can be find in the paragraph 4.1

Instead of this procedure, we 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 original 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 highest possible resolution. Indeed, the loss computation is done using a *no-reference* index. For this reason, this approach belong into the *unsupervised learning* field. Oppositely to *supervised learning*, unsupervised learning is a subfield of ML 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 by a computer. Every computer program executes a sequence of elementary operations. There are different kind of techniques to differentiate

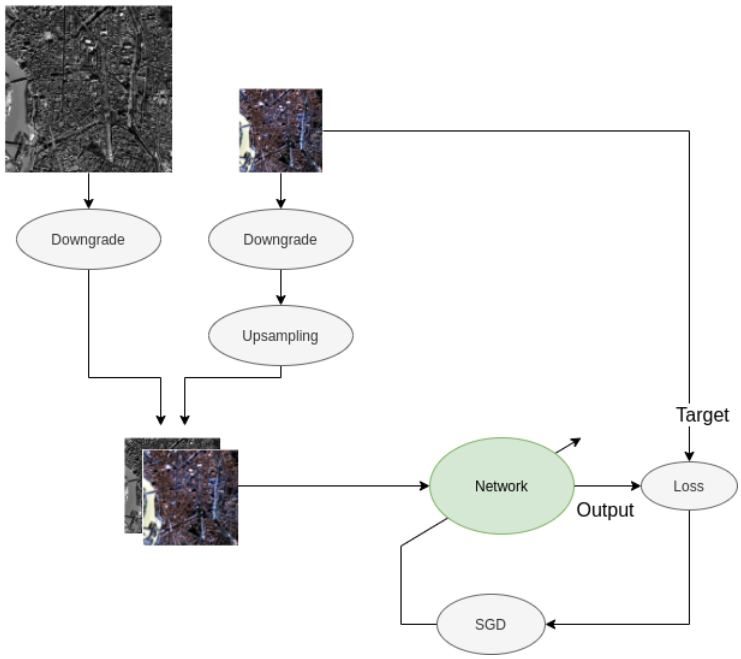

Figure 3.1: Graph of the training algorithm released by [4].

a function: *automatic differentiation*, *numerical differentiation* and *symbolic differentiation*. The Fig. 3.3 shows an example of all techniques that evaluate the derivative of the same function. The various approach would be described.

Numerical differentiation is the finite approximation of the derivatives.

$$\frac{df(x)}{dx} \approx \frac{f(x+h) - f(x)}{h} \tag{3.1}$$

In this technique, the computer evaluate the function described in Eq. 3.1 at some sample points. As described in [40], this type of differentiation is easy to code. However, the $O(n)$ cost of the evaluation in n dimensions is so high that ML, where n can be millions or also billions, would not be accessible.

Symbolic differentiation is a technique that uses the chain rule, the product

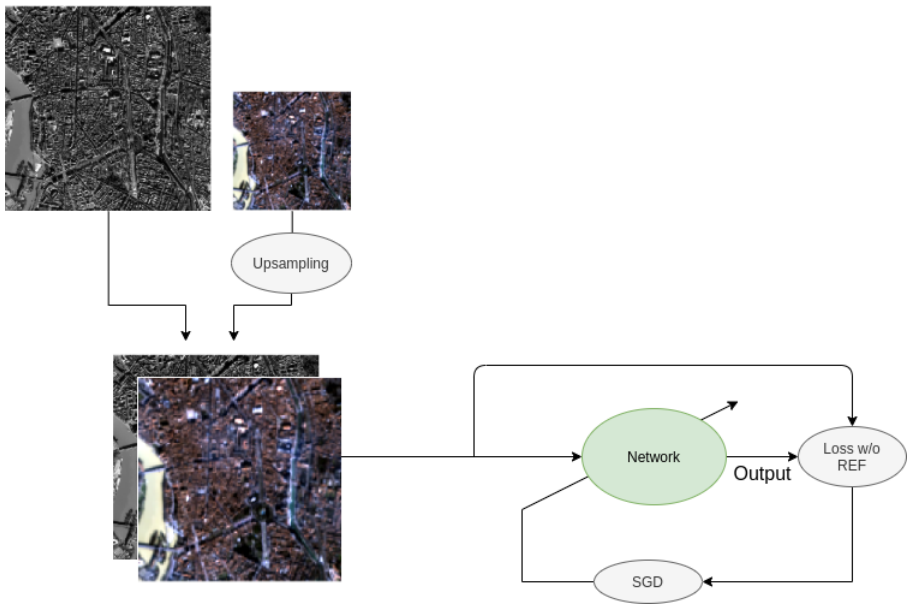

Figure 3.2: Graph of the training algorithm developed.

rule and other rules to split the expression in known primitive derivative to obtain the result. The Eq. 3.2 shows how product rule and sum rule are applied.

$$\begin{aligned} \frac{d}{dx}(f(x)+g(x)) &= \frac{d}{dx}f(x)+\frac{d}{dx}g(x) \\ \frac{d}{dx}(f(x)g(x)) &= \left(\frac{d}{dx}f(x)\right)g(x)+f(x)\left(\frac{d}{dx}g(x)\right) \end{aligned} \tag{3.2}$$

Techniques that belong to this class are really difficult to convert into a computer programs.

Automatic differentiation systems explicitly split the operations in simplier ones and build a computation graph. Each node of the graph have some attributes like values, primitive operations and parents.

Jacobian matrixes are used to represent the derivative of each output y respect to each input x of a graph node.

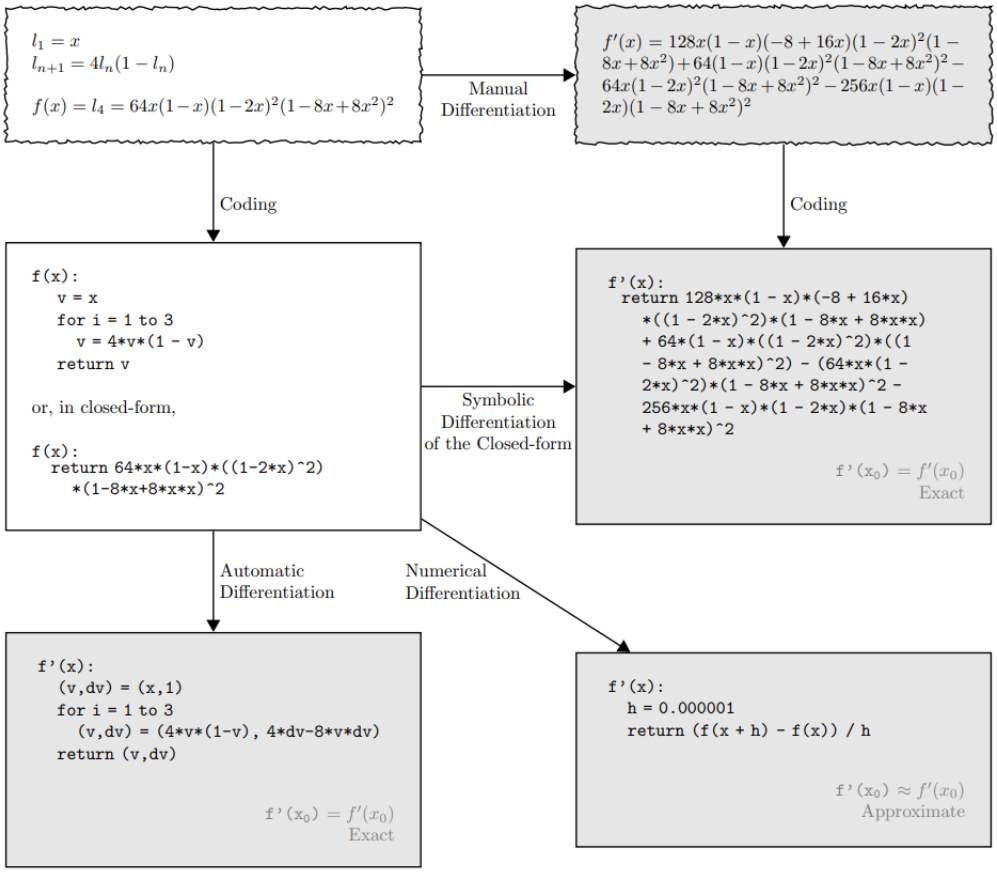

Figure 3.3: Example with all type of differentiation [40]

$$J_f = \begin{bmatrix} \frac{dy_1}{dx_1} & \cdots & \frac{dy_1}{dx_n} \\ \vdots & \ddots & \\ \frac{dy_m}{dx_1} & & \frac{dy_m}{dx_n} \end{bmatrix} \tag{3.3}$$

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 the paper [41]. 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. Indeed, the *automatic differentiation* can handle also control flow statements like "if", "while" and "loops". The second one, *symbolic differentiation*, operates on mathematical expressions and symbols.

3.4 Tensorflow and custom loss function implementation

Tensorflow is the most famous ML and DL tool that implements the *automatic differentiation*. To build a custom loss function, Tensorflow provide APIs. These 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 efficiency purpose, compile it, and builds the computation graphs. Also, Tensorflow for 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 supports 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

At the beginning, to build the loss function, it was used a QNR approximation

$$f(x) = 1 - \widetilde{QNR} \tag{3.4}$$

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

The D_s and D_λ were calculated using the Qavg, that is the average of Q indexes computed for each band. The Q index was calculated considering the whole image and not dividing it into small patches. The result shows a series of negative values instead of values between 0 and 1 as the original

images format. This is why the performance was unsatisfactory (results are illustrated in the Chapter 4). Indeed, the model minimizes the loss functions (namely, it maximizes the QNR, considering 3.4) no matter the sense of output values. After some tests, it was considered to solve this problem using the $D_s reg$ described in [42] instead of D_s and migrate to an approximation of HQNR instead of the QNR. Implementing the exact HQNR was unfeasible 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 implemented code has been written in Python 3.7 with the use of file `main.py` as a entrypoint. It is compatible with Linux, Windows and MacOS and it can run on dedicated GPU as well as on CPU.

By the use of the standard `argparse` library, all the possible parameters can be codified on terminal. This allows to run, with a bash file, all the experiments in series without changing the code.

At the beginning, the QNR function has been developed with approximations. In the Appendix A.1 are presented the most important functions for the QNR evaluation. As showed in the Appendix A.1, the Q has been obtained

as an average of the Q calculated band-by-band instead of being calculated in patches as the original paper of the QNR described. In order to obtain the spectral consistency measure D_λ, for each band of MS and fused image, it has been necessary the a calculation of Q between bands as formalized in the Eq. 1.22. D_s, according to the Eq. 1.21, has been calculated as the Q between MS bands and PAN_L and also between the fused image and the PAN. This gives a quality measurements of the details inserted into the image.

Due to QNR poor performance, it has been decided to use an approximation of HQNR. Approximations are given due to the lack of Tensorflow hypercomplex API. Instead of using Q4, for calculating D_λ, the Qavg used for the QNR has been adopted. The D_λ is the result of $1 - Q$ between the fused image filtered through a Gaussian filter and the MS.

We opted for a D_sreg, an implementation of the D_s that takes advantages from the *coefficient of determination* calculation. The coefficient of determination, also called R-squared (R2), is a statistical measure that represents the proportion of the variance for a dependent variable that is predictale from the indipendent variables.

The D_sreg is obtained as $1 - rsquared$ between the fused image and the PAN. The implementation is illustrated in the Appendix A.2. Depending on the sensor used for the image acquisition, the filter would be different to 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 gradient of the error for each layer and it will determines how quickly the error decrease.

If the learning rate is too high, the error will drop down quickly, but difficulties will be encountered in finding minimum. With a too low learning rate, it can take long time to reach the convergence or get stuck in a local minimum. According to the community, the best approach is to use a learning rate scheduler like Adam [43] or SGD [39]. At the start of the training, the model can have a high learning rate so that it can explore all the loss function. After that, a great descent would be reached and the learning rate would gradually degrade in order to avoid oscillations. The learning rate is one of the parameters 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 Q2n. At the moment, Q2n is the best index used with reference image. It is important in the training to avoid a bell shape in the index graph. Bell shape indicates that the Q2n increase really fast at the beginning. After reaching the pick value of optimum, it decrease really fast. This characteristic would made the training really difficult in real applications. The reason why is that the calculation would be impossible due to the inexistence of reference image.

Without the bell shape during the training, the model can only have an increase in performance. After that, stable phase would be achieved by per-

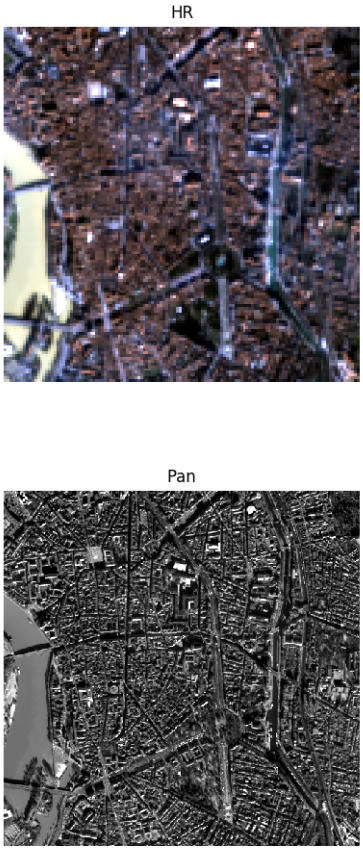

Figure 4.1: Multispectral and Pancromatic Toulouse images used for all the tests

formance.

4.2 Experimental Settings

All types of training have been executed in a test image illustrated with a true color representation in Fig. 4.1 identified as Toulouse dataset. Toulouse dataset is an IKONOS image acquired over the city of Toulouse, France, in

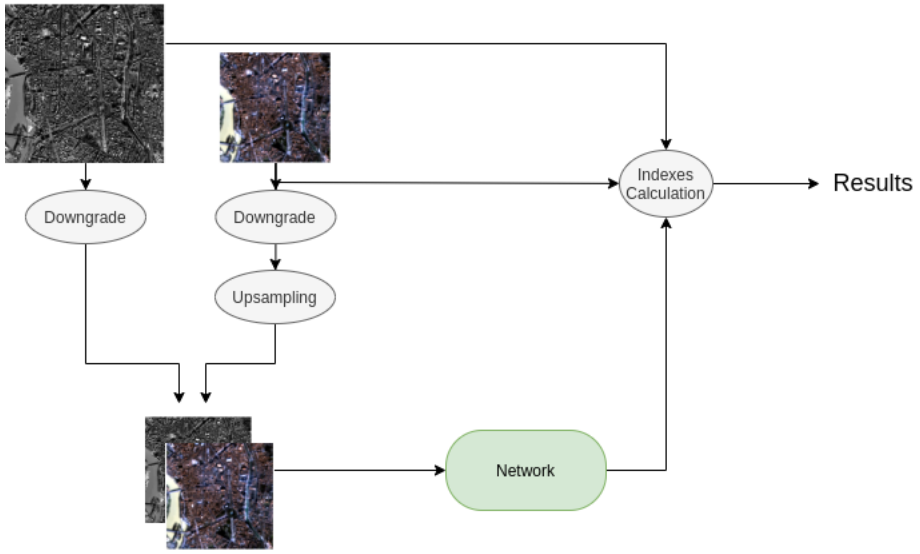

Figure 4.2: Validation process

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 constituted by an image of 2048×2048 pixels with a 1m SSI.

DL models should not be specialized too much for the training set. As the model could have worse performance for data not trained for like in the real applications. To avoid this situation, it is necessary to create a validation set in which there are different data. Performance on the validation set are more similar to real world 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).

At every epoch, the downsampled images PAN_L and MS_L were used to

produce a fused image with the updated weights and compared with the GT. This means that the images will constitute the validation set. Reference indexes like SAM, ERGAS and Q together with *no-reference* indexes QNR and HQNR were calculated between the output of the model and the GT. Those indexes has been obtained with the MatLab toolbox functions provided by [20] and MatLab engine for python [44]. The procedure can give a real performance assessment of the whole model. The training set have always better performance as the model optimizes the error of these data. It needs to be highlighted that only the performance of the validation set are relevant.

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

It has been noticed that, for some images, the model that uses the fine-tuning with the reference approach has worse performance than the original one. The reason is that the model is fine-tuned with the downgraded images.

The characteristics of the sensors, in general, do not satisfy the scale invariance hypothesis 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 as the

model use a *no-reference* loss function.

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

4.3 Results

The use of the original QNR function has not produced any positive results. During the training, while the error continued to be minimized, the Matlab Toolbox indexes get worse as showed in Fig. 4.3, 4.4 and 4.5, even the QNR itself. The reason is related to the fact that, in order to minimize the error, it has been generated an image with negative values. An input with negative values is not expected by any of the indexes used. This creates a discrepancy between the QNR 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 even though this is not reflected in an increase of the quality indexes.

After those tests, it has been decided to change the loss function using another *no-reference* index: the HQNR. However, this index is more complex than the QNR, as it involves quaternions and operations with hypercomplex numbers.

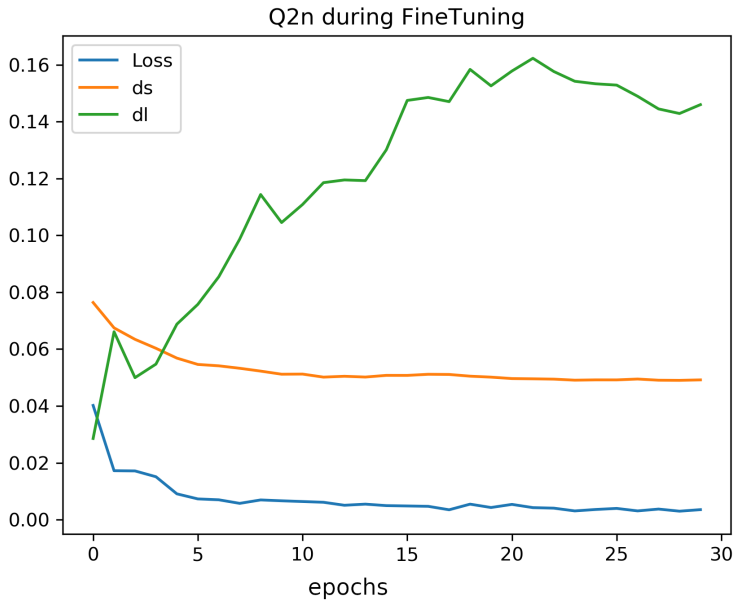

Figure 4.3: Loss, DS and DL of QNR during the training.

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 carried out, firstly, with steps of 0.5 and secondly, for particular ranges, with steps of 0.25.

In Fig. 4.7 and 4.8, α and β are defined in the Eq. 1.23

The conclusion of the experiment shows that best results are reached with

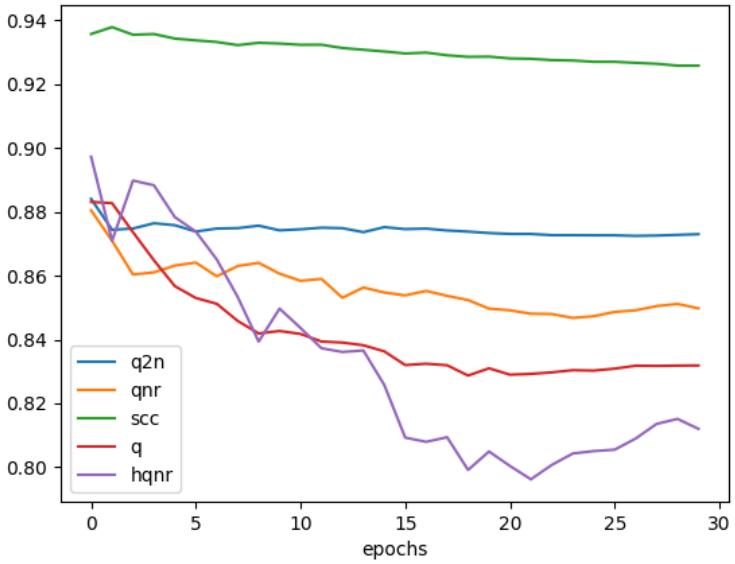

Figure 4.4: Q2N, QNR, SCC, Q and HQNR during the training.

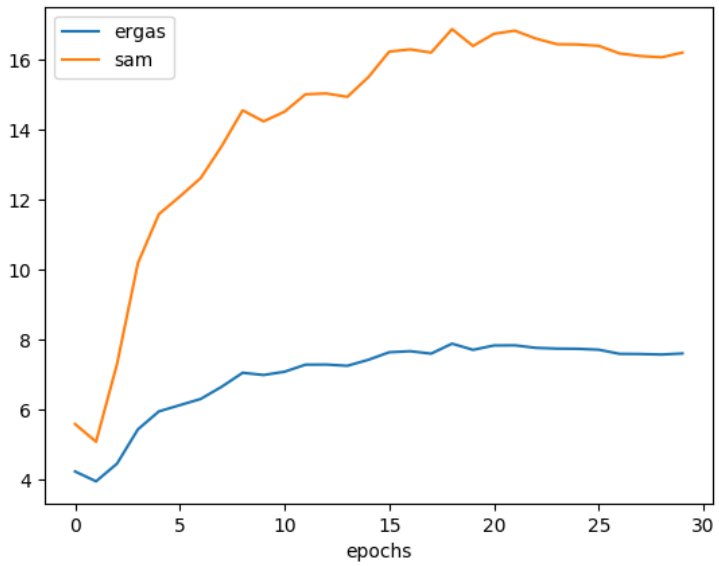

Figure 4.5: ERGAS and SAM during the training.

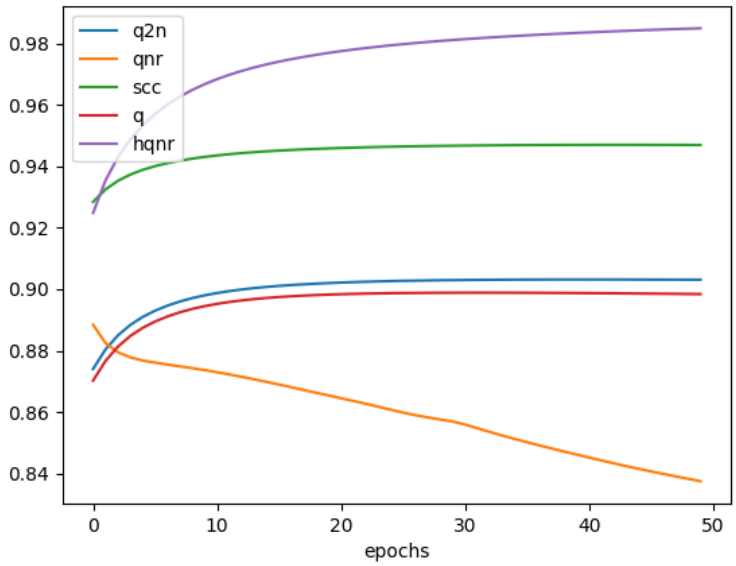

Figure 4.6: Q2N, QNR, SCC, Q and HQNR during the training using the loss function with HQNR.

α 0.5 and β 2.

After this experiments, all methods have been compared. While GT is the method that uses the Ground Truth during the training to have the best performance, RR is the method used in [4] with reduced resolution training. In addition to that approaches, a *no-reference* method to train the model with full-resolution images described in the previous paragraphs has been implemented.

Fig. 4.9 shows the comparison between the three different methodologies. The graph indicates that GT Q2n trend (in green) has a high margin of performance respect others. The expectation was that the model would reach a level close to 1. However, with the actual architecture, the model cannot

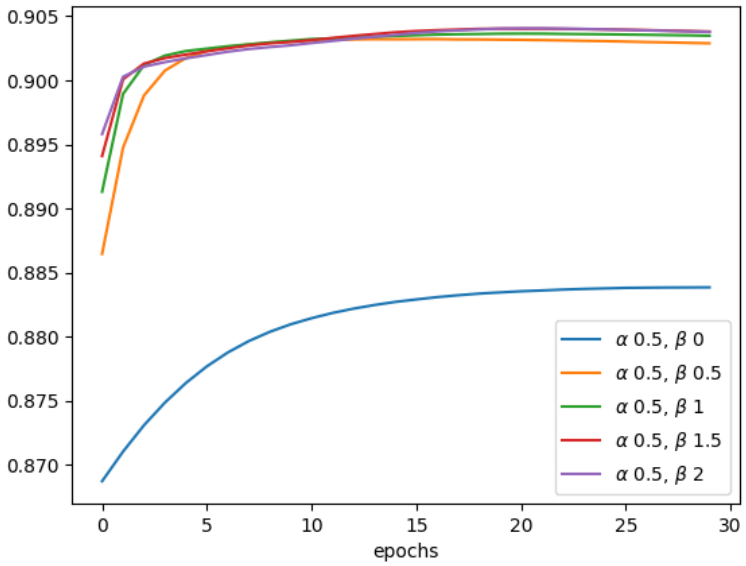

Figure 4.7: Alpha and Beta with a 0.5 step.

achieve higher performance. Worst performance has been obtained by RR (in blue) where the trend had increased until the second epoch and after it remains stuck below 0.89. The NOREF (in yellow) has a continuous growth and it remains constant from the 30th epoch. The model has better performance than RR despite his recent application.

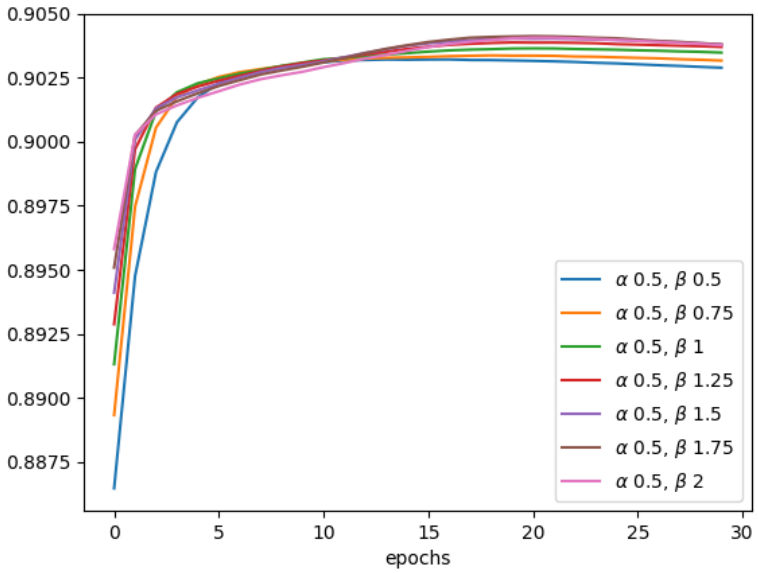

Figure 4.8: Beta with a 0.25 step between β 0.5 and β 2 and α setted to 0.5.

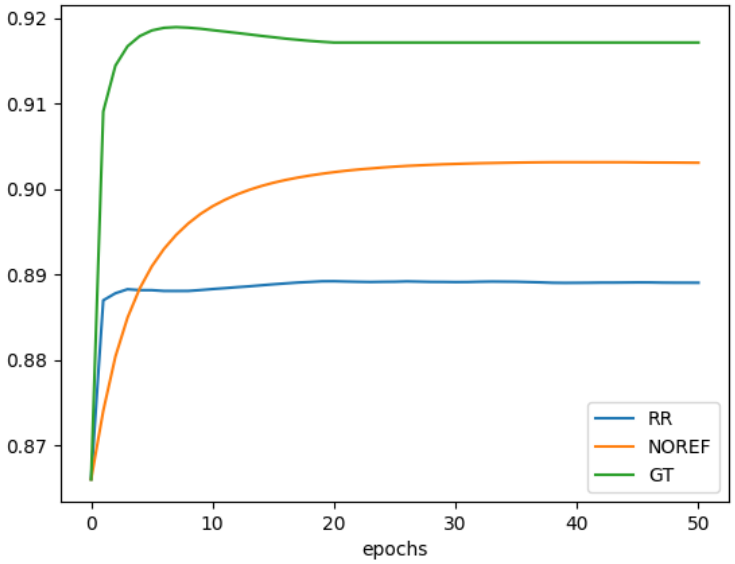

Figure 4.9: Q2n experiment results with GT, RR and NOREF.

Chapter 5

Conclusions

In conclusion, the use of a hybrid approach of RR and NOREF can have even better performance as it inherits the best aspect of both methods. Indeed, a hybrid model would use reduced resolution images (RR) with a robust reference index and, at the same time, images at the original resolution (NOREF) that would compensate any problems due to scale variance.

Furthermore, different architecture can be considered. An architecture with only convolutional layers includes a limited number of filters and every filter would be applied to the whole image. This implicates that a filter is trained to optimize the performance of the whole image instead of a particular area. Images can have different environments (like rural or urban) and the model could benefit by neurons divided to specialize different patches.

This results can be achieved with different techniques. At first glance, it would be possible to generate multiple small models fine-tuned for patches.

Furthermore, changing the model layers would deliver images processed in a specialized way. To this extent, in a dense layer, all neurons have their own specific weights that would be trained for the exact pixels in input. Indeed, the actual limit in *pansharpening neural networks* (PNN) is treating images without considering the different morphology of them.

This research lay the foundation for future development in the field of pansharpening. A *no-reference* model with a more implemented and specialized image processing can overcome the limits of the model in use.

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] George Dahl Ilya Sutskever1, James Martens. On the importance of initialization and momentum in deep learning. 2013.

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

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

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

[43] Jimmy Ba Diederik P. Kingma. Adam: A method for stochastic optimization. 2014.

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

Appendix A

A.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) + tf.square(pred_b_mean))
    q3_b = (two * true_b_std * pred_b_std ) / \
        (tf.square(true_b_std) + tf.square(pred_b_std))

    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[:, :, r:r+1]\
                    ), tf.float32) - \
            tf.cast(\
```

```
                q_index(ms[:, :, l:l+1], ms[:, :, r:r+1]), tf.float32))**p

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

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

    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.float32) - \
         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
```

A.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), padding="VALID")
    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(\
      ms, fused, p=p, b=bands, sensor=sensor))**alpha
  b = ( 1 - d_s_reg(fused, pan)) ** beta
  return a*b
```