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

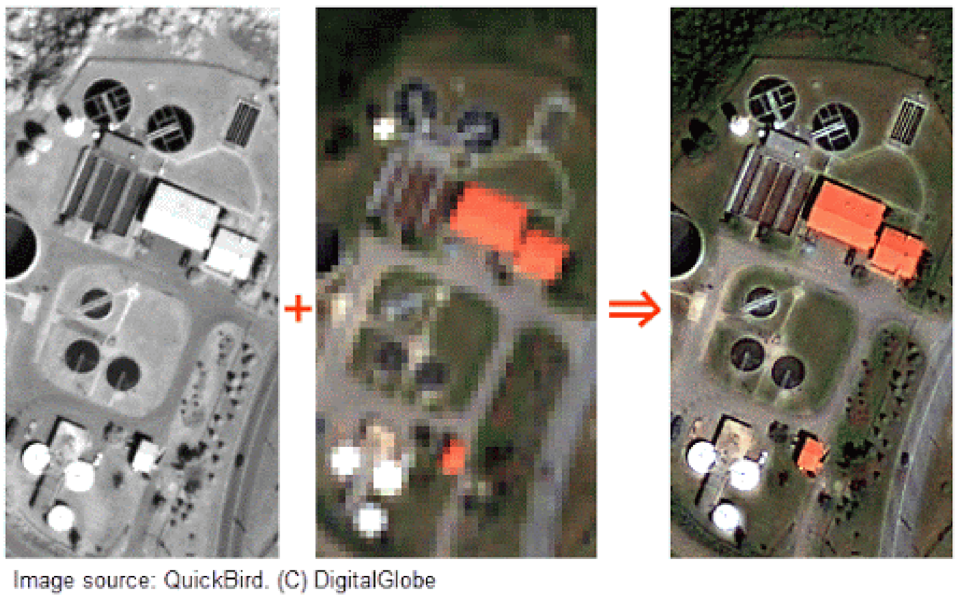

Figure 1: Pansharpening example

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

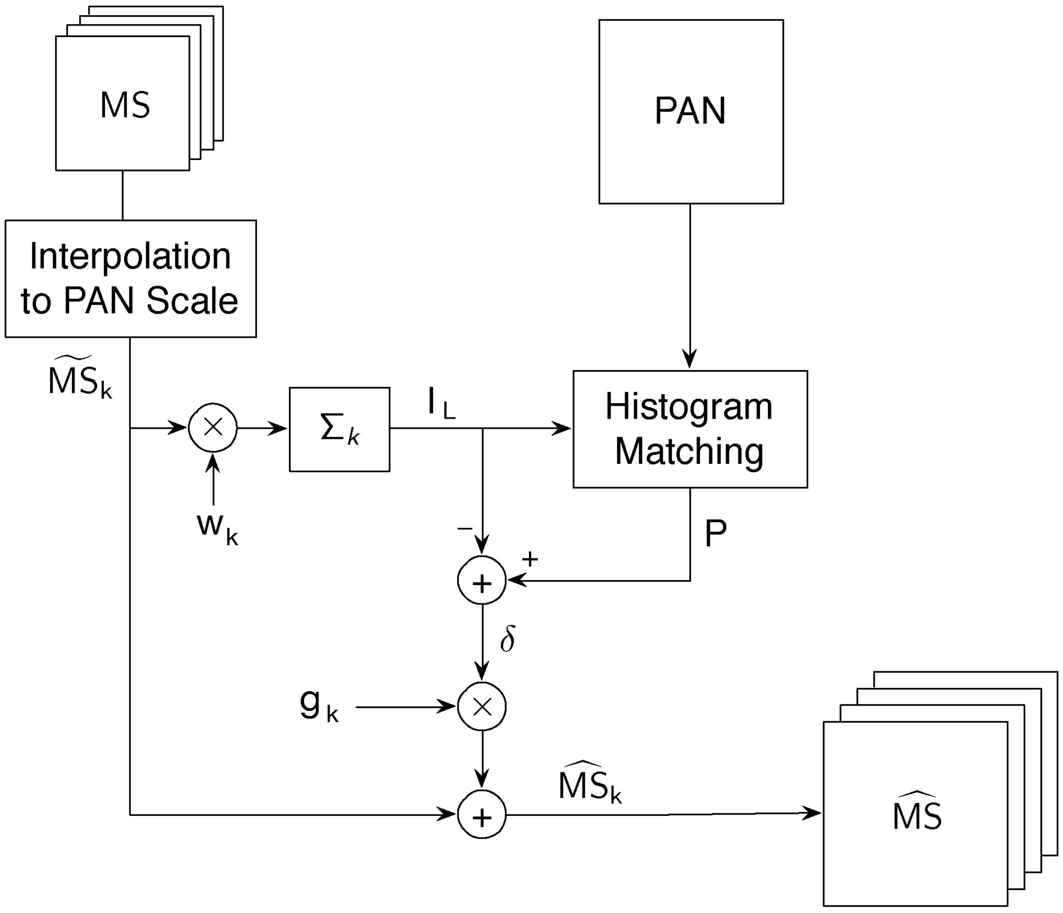

Figure 1.1: Flowchart of CS approach [20]

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:

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

in which $\text{cov}(\cdot,\cdot)$ is the covariance between two images and $\text{var}(\cdot)$ 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

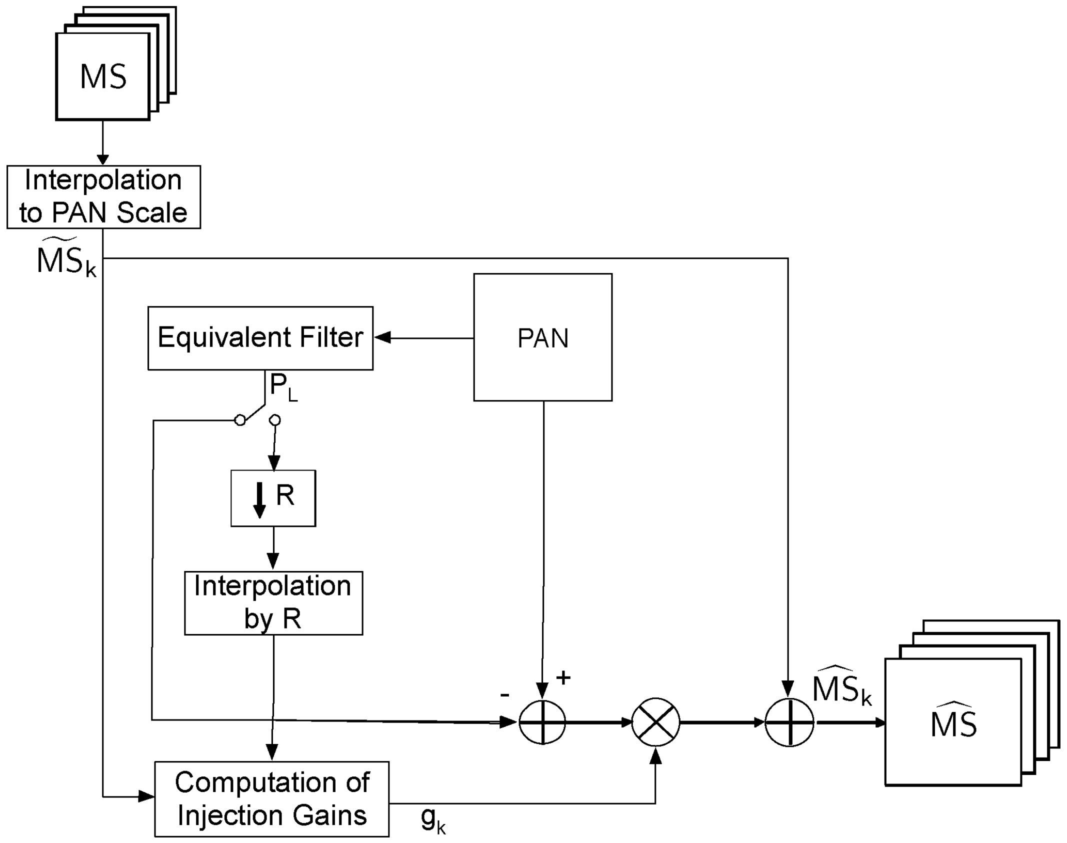

Figure 1.2: Flowchart of MRA approach [20]

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 :

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

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 with the Perceptron [29] and in the 60s with the invention of backpropagation that is the most commonly used algorithm in the present day to train a Neural Network. The Neural Network is a model constituted by several Perceptron, also called neurons, divided into different layers that give the ability to learn, extract and distinguish different features from the data given in input. To learn these capabilities, the network should be subject to a training phase that requires an incredible amount of computational power. Indeed, in the beginning, this type of algorithm was discarded because the computers in the

60s was not powerful enough and there wasn't a great data availability. 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 reference. It is calculated the error between the output of the prediction and the reference and this error is propagated in all the neurons of the model. This error modify the weights of all the neurons so that the next prediction for the same input it will much similar to the reference output. How this error is calculated and what means ”similar” is established by the loss function that calculates the error. The backpropagation uses the Gradient Descendent algorithm, that it allows the network to understand how to change the weights to minimize the loss. 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 function.

LeCun in 1989 for the first time used a backpropagation algorithm to train a convolutional neural network, a particular type of networks constituted from different convolutional layers to classify handwritten digits.

These days, the data collected with the 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, has 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 only one output but can be more inputs as described in the Fig. 2.1. A perceptron uses only one linear or non-linear activation function. In the following paragraph it will discuss what is an activation function. 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 for example the separation of a plane described by the XOR function. But 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 the different type of layers.

2.2.2 Activation Function

An activation function is used to transform the weighted data (input multiplied weights) in outputs, in deep learning often a non-linear transformation. Indeed, using a non-linear transformation, we create new relationships between the points and 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 rectified linear

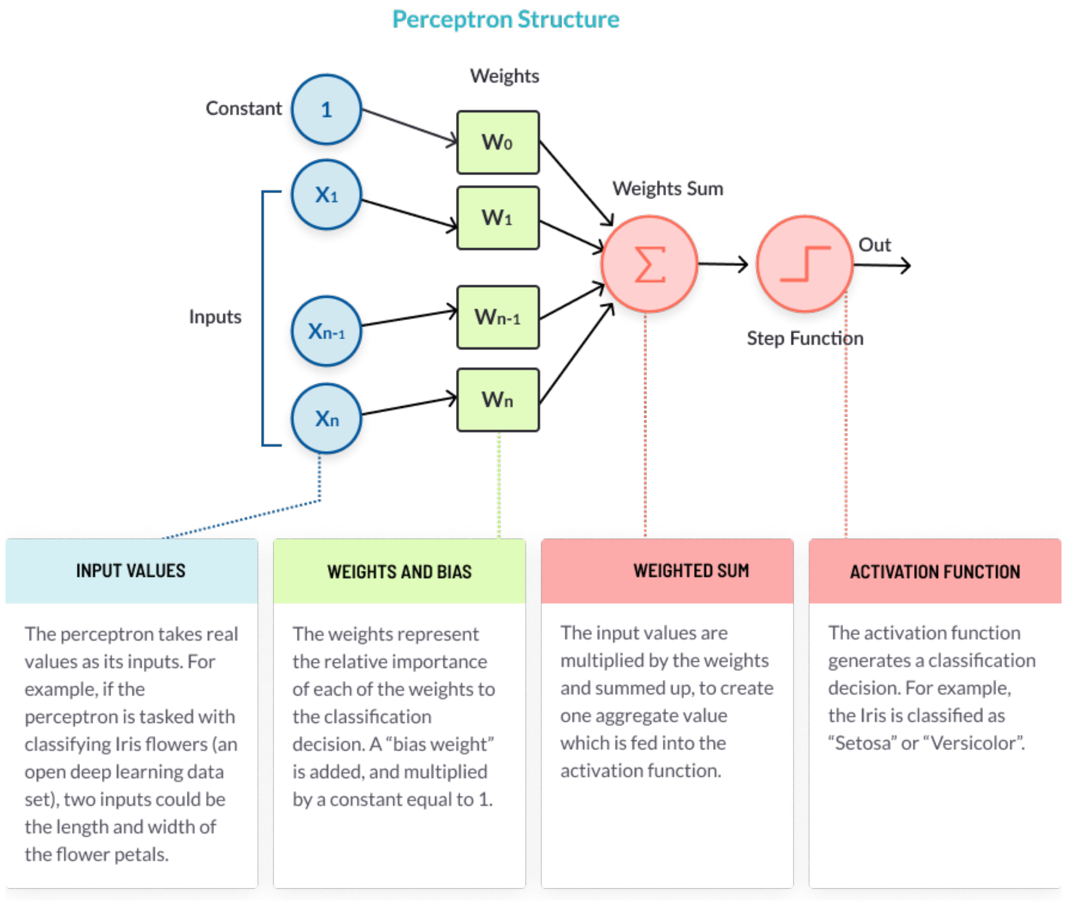
Perceptron Structure
Weights
Constant
1
X1
Xn-1
Xn
Inputs
W0
W1
Wn-1
Wn
Weights Sum
Σ
Out
Step Function
INPUT VALUES
The perceptron takes real values as its inputs. For example, if the perceptron is tasked with classifying Iris flowers (an open deep learning data set), two inputs could be the length and width of the flower petals.
WEIGHTS AND BIAS
The weights represent the relative importance of each of the weights to the classification decision. A "bias weight" is added, and multiplied by a constant equal to 1.
WEIGHTED SUM
The input values are multiplied by the weights and summed up, to create one aggregate value which is fed into the activation function.
ACTIVATION FUNCTION
The activation function generates a classification decision. For example, the Iris is classified as "Setosa" or "Versicolor".

Figure 2.1: Perceptron in detail [30]

function or also colled 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. Gradient equal 1 means that the training will be much faster compared with other activation functions like logistic sigmoid.

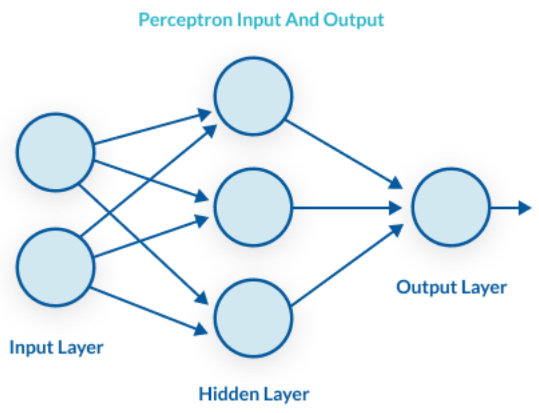

Figure 2.2: Multilayer architecture example [30]

2.2.3 Layers

The DL network can have different layers type that differ in how the perceptrons inside them are organized and how they modify their weights during the training. A layer of neurons can be expressed as a function, mostly non-linear, that apply a transformation of the input into the output as described in Fig. 2.2. The first experimented layer was the dense layer in which all the neurons are connected with all the neurons of the next layer. Every neuron has his weights and with all the connections with other neurons generates very fast a large number of weights to train, so the necessity of a larger dataset and a longer time for the training.

A layer that gave an important improvement in computer vision applications is the convolutional layer. 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

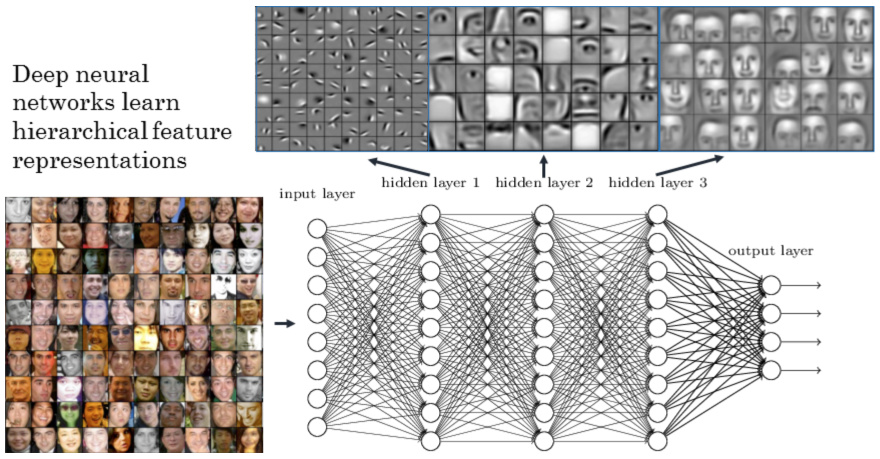

Figure 2.3: Feature extraction of a deep model [31]

used to extract features from an image. The DL model is capable to extract and recognize features in the data as complex as the model dimension growth. The features can be in a second phase, analyzed by a different type of layer, for example a dense layer, that can elaborate them for a classification or other tasks (Fig. 2.3). The filters weights of the convolutional layer are learned in the training and applied to the whole input and the neurons that constitute the layer, share the same weights. This layer, respect to a dense layer, has a lower weights count, is much faster to train and also allocate a minor amount of memory. Different layers in sequence can extract features more and more complex. LSTM and GRU layers were created to allow the model and 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 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 used and require a high knowledge of the particular context.

In the DL, the net is trained also for the features extraction, but ths require a more complext architecture.

Deep Learning gave a strong impulse to the development of very efficient algorithms in the computer vision field. Indeed, these days there are a lot of tools on the internet that simplifies the use of complex models trained to recognize hundreds of objects in an image. Also, there is the possibility to tune this kind of nets to recognize a different set of objects with a technique called fine-tuning.

The fine-tuning is based on the concept (described also in Fig. 2.4) that the initial layers of a model, in computer vision always convolutional layers, are trained to extract features from the input. This features will be used by the final layers of the model to accomplish different tasks, for example classification.

The idea of fine-tuning is to reset the weights of the final layers so that

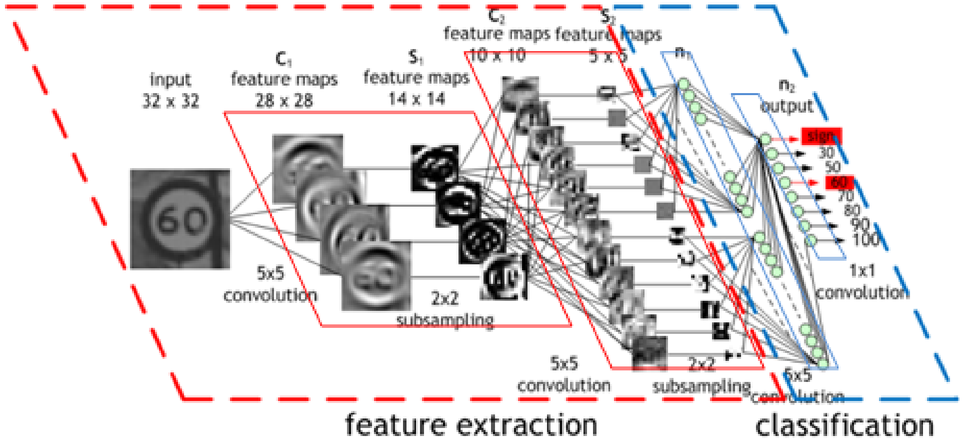

Figure 2.4: Image by Maurice Peemen

the model, from the same features extracted, can accomplish another task, for example classify different objects. With this technique is not necessary to train the model for days using very large datasets because is not necessary to train the entire network but only the final layers.

2.4 Pansharpening Applications

Recently it was proposed in [4] a new pansharpening method based on a convolutional neural network. It was specialized a network built for super-resolution [32] to accomplishes the pansharpening task. Three different models for GeoEye1, IKONOS and WorldView2 sensors have been built. This because the net weights can be specialized to predict the sensor characteristics and give better performance. Not only the weights but also the layers are slightly different from each other because the authors run different test for each sensor to increase performance. An important issue in this field is that it's difficult to found good images because of the high cost for a high-resolution images. For this reason, it can't be built a deep network because it became hard to train. The authors choice was to use three convolutional layers, illustated in Fig. 2.5. Using only convolutional layers have also the advantage that no matter the number of columns and rows of the images, the model will fits the images dimensions and it can execute the pansharpening on anyway.

The downside of convolutional layers is that it reduce the image dimensions layer by layer because essentially, this type of layer apply a convolution between the input and some filters of a pre-determined kernel size without adding padding. For this reason, the output dimension will be reduced by half the kernel size in each direction.

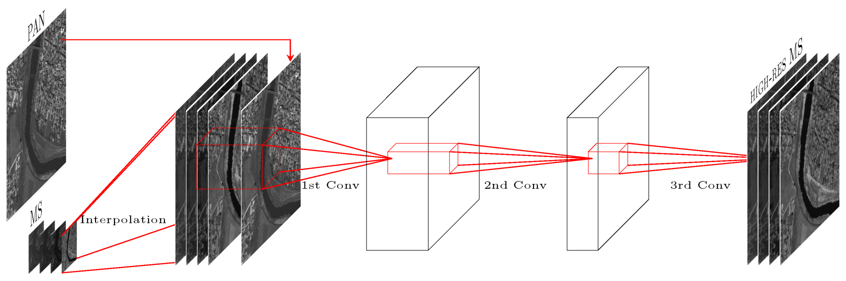

Figure 2.5: CNN architecture for pansharpening[4]

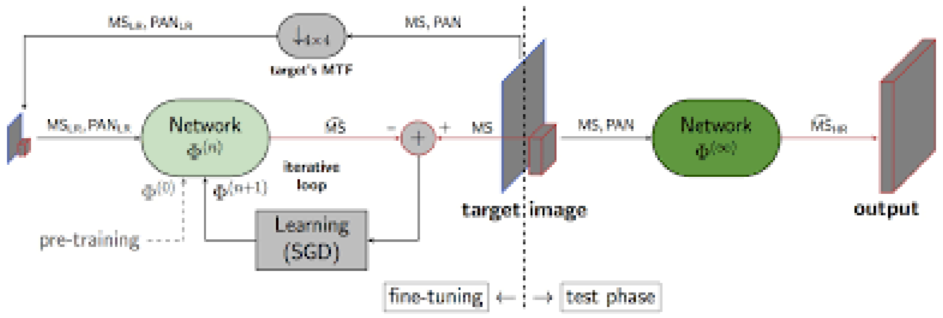

Figure 2.6: Training and Test[33]

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 appended to the MS downgraded and given in input to the model. The MSE error between MS and the fused image is calculated and this error is used for the training of the weights.

An improvement illustrated into the paper was to append also some radiometric relevant indexes for the applications to adding information abound the image. It was showed that the network avoids learning this indexes already provided and it could focus to learn other information. Another important step introduced in [33] was to run a fast session of fine-tuning before the application of the model to an image with the same procedure of the training.

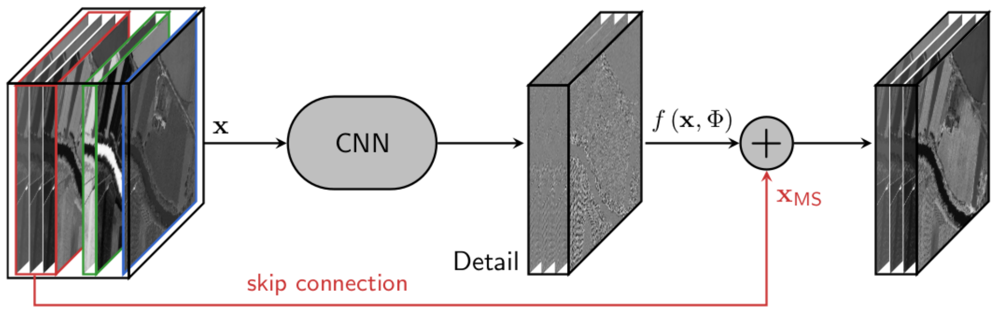

Figure 2.7: Residual-based version [34]

With the fine-tuning, the net can learn to fuse the new image in a downgraded version. This can train the network also for the new image and produce better performance. But, as described in the conclusion of [33] performance in down-sampling domain is relatively relevance and no-reference measures in training can have a major impact. Another noteworthy downside of this method is that with misaligned images, the model is trained with a fraction of the misalignment depends on the scale ratio between PAN and MS. This is an example that the degradation changes the sensor model in which we want to apply the pansharpening.

In another paper [34], the authors has tried different architectures and strategies to improve the network released in [4]. An important result it was reached with a residual version of the original network. The architecture is showed in Fig. 2.7.

Essentially, the residual version it was not trained to reproduce the whole image, but only the high-pass component. Indeed, the low-pass component is

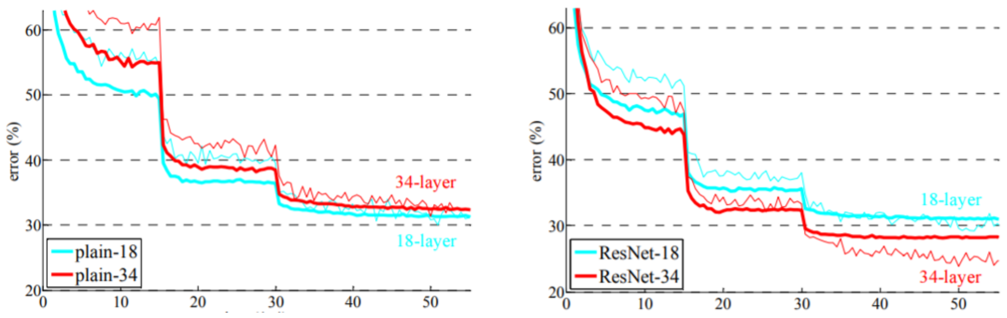

Figure 2.8: Performance comparison between residual (right) and non residual (left) networks [38]

represented by the multispectral image used in input. This means that the network reconstruct only the missing parts.

Residual-based architecture was used in different papers ([35] and [36]) and in particular in the deep learning in [37] and [38].

In general, 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 [38] are showed in Fig. 2.8.

Chapter 3

Proposed Solution

3.1 Introduction

In this chapter will be described the concept of our solution and all the steps and issues encountered. The main goal it was to select a correct loss function and implement it using modern tools.

3.2 Loss Function Issue

In general, deep learning models should not be specialized too much for the training set, because the model could have worse performance in real applications in which the net is not trained. To avoid this situation, is necessary to create a validation set during the training. The validation set is a dataset in which there are data that the net is not trained for. Performance on the validation set are more similar to real performance. To generate a reference image, necessary for the quality assessment, it was downsampled an MS and PAN. The original MS it was used as Ground Truth (GT) and the downgraded version will be called MS and PAN for simplicity. To build the training set for the reduced resolution method, selected an image to performe the fine-tuning, it was used the downgraded version of PAN and MS with the goal for the model to reproduce the MS. At every epoch, the PAN and MS were used to produce a fused image with the updated weights and compared with the GT and so, these images constitute a validation set. This procedure can give a real performance assessment of the whole model. During the training, the training set have always better performance because the model is optimizing the error of these data. Only the performance of the validation set are really important.

In the fine-tuning, it is noticed that some images have worse performance with the updated weights respect to the original ones without fine-tuning. These because the model is fine-tuned with the downgrade version of the original image in which it should be performed the fusion. The models 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.

The thesis' aim is to change the training process using the MS and PAN combined with a no-reference index and compare the results with the reduced resolution method. The degraded images were given in input to the two methods and the result was compared with the reference, the original image. With the RR method (reduced-resolution), the inputs were downgraded another time for the fine-tuning, but with the NOREF method (no reference), the inputs were just used for the training with a no-reference loss function. At every epoch, reference indexes like SAM, ERGAS and Q and also no reference index QNR and HQNR were calculated between the images and the original image. These indexes were calculated with the MatLab toolbox functions provided by [20] using the MatLab engine for python [39].

3.3 Automatic Differentiation

In mathematics, the automatic differentiation is a set of techniques to efficiently calculate, with a computer, the gradient of a function. 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. The Numerical differentiation is the standard definition of a derivative.

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

This type of differentiation is easy to code but the evaluation have a really high cost.

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

Symbolic differentiation is inefficient in terms of performance. The complexity can in a lot of cases be 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 simplier ones and build a computation graph. Each node of the graph have some attributes like value, primitive operation and parents. For each primitive operation, it must be defined a Vector-Jacobian Product (VJP), a product based on the Jacobian that is the matrix of partial derivatives. Combining all the VJP of all the nodes return 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, but the first one operates on computer programs, with numerical values and it was created specifically for the computer manipulation and the second one operates on mathematical expressions and with 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 define a custom loss function, Tensorflow provide an API. This API, define all the operations betweens Tensors that are the basic datatype of all operations in Tensorflow. In the Table 3.1, the math module API is showed (more used module during the thesis).

Tensors in this algorithm stage does not hold real values, but only sizes and other tensors' attributes. Only during the training, when the gradient is calculated layer by layer, the tensors acquire a value. This allow to validate all operations before the actually computation and compile these safely.

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 only necessary to create a $Model$ class. And it's 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 that the layer should have and so on depends on the class. Use such a widespread framework has the advantage of having a large community that can help in difficult situations but also a much larger compatibility on Operating Systems, GPU and hardware in general.

abs(...);	Computes the absolute value of a tensor.
accumulate_n(...);	Returns the element-wise sum of a list of tensors.
acos(...);	Computes acos of x element-wise.
acosh(...);	Computes inverse hyperbolic cosine of x element-wise.
add(...);	Returns x + y element-wise.
add_n(...);	Adds all input tensors element-wise.
angle(...);	Returns the element-wise argument of a complex (or real) tensor.
argmax(...);	Returns the index with the largest value across axes of a tensor.
argmin(...);	Returns the index with the smallest value across axes of a tensor.
asin(...);	Computes the trignometric inverse sine of x element-wise.
asinh(...);	Computes inverse hyperbolic sine of x element-wise.
atan(...);	Computes the trignometric inverse tangent of x element-wise.
*atan*2(...);	Computes arctangent of y/x element-wise, respecting signs of the arguments.
atanh(...);	Computes inverse hyperbolic tangent of x element-wise.
*bessel_i*0(...);	Computes the Bessel i0 function of x element-wise.
*bessel_i*0*e*(...);	Computes the Bessel i0e function of x element-wise.
*bessel_i*1(...);	Computes the Bessel i1 function of x element-wise.
*bessel_i*1*e*(...);	Computes the Bessel i1e function of x element-wise.
betainc(...);	Compute the regularized incomplete beta integral .
bincount(...);	Counts the number of occurrences of each value in an integer array.
ceil(...);	Return the ceiling of the input, element-wise.
confusion_matrix(...);	Computes the confusion matrix from predictions and labels.
conj(...);	Returns the complex conjugate of a complex number.

cos(...);	Computes cos of x element-wise.
cosh(...);	Computes hyperbolic cosine of x element-wise.
count_nonzero(...);	Computes number of nonzero elements across dimensions of a tensor.
cumprod(...);	Compute the cumulative product of the tensor x along axis.
cumsum(...);	Compute the cumulative sum of the tensor x along axis.
cumulative_logsumexp(...);	Compute the cumulative log-sum-exp of the tensor x along axis.
digamma(...);	Computes Psi, the derivative of Lgamma (the log of the absolute value o.
divide(...);	Computes Python style division of x by y.
divide_no_nan(...);	Computes a safe divide which returns 0 if the y is zero.
equal(...);	Returns the truth value of (x == y) element-wise.
erf(...);	Computes the Gauss error function of x element-wise.
erfc(...);	Computes the complementary error function of x element-wise.
erfinv(...);	Compute inverse error function.
exp(...);	Computes exponential of x element-wise. .
*expm*1(...);	Computes exp(x) - 1 element-wise.
floor(...);	Returns element-wise largest integer not greater than x.
floordiv(...);	Divides x / y elementwise, rounding toward the most negative integer.
floormod(...);	Returns element-wise remainder of division. When x ¡ 0 xor y ¡ 0 i.
greater(...);	Returns the truth value of (x ¿ y) element-wise.
greater_equal(...);	Returns the truth value of (x ¿= y) element-wise.

igamma(...);	Compute the lower regularized incomplete Gamma function P(a, x).
igammac(...);	Compute the upper regularized incomplete Gamma function Q(a, x).
imag(...);	Returns the imaginary part of a complex (or real) tensor.
in_top_k(...);	Says whether the targets are in the top K predictions.
invert_permutation(...);	Computes the inverse permutation of a tensor.
is_finite(...);	Returns which elements of x are finite.
is_inf(...);	Returns which elements of x are Inf.
is_nan(...);	Returns which elements of x are NaN.
is_non_decreasing(...);	Returns True if x is non-decreasing.
is_strictly_increasing(...);	Returns True if x is strictly increasing.
l2_normalize(...);	Normalizes along dimension axis using an L2 norm.
lbeta(...);	Computes , reducing along the last dimension.
less(...);	Returns the truth value of (x ¡ y) element-wise.
less_equal(...);	Returns the truth value of (x ¡= y) element-wise.
lgamma(...);	Computes the log of the absolute value of Gamma(x) element-wise.
log(...);	Computes natural logarithm of x element-wise.
log1p(...);	Computes natural logarithm of (1 + x) element-wise.
log_sigmoid(...);	Computes log sigmoid of x element-wise.
log_softmax(...);	Computes log softmax activations.
logical_and(...);	Logical AND function.
logical_not(...);	Returns the truth value of NOT x element-wise.
logical_or(...);	Returns the truth value of x OR y element-wise.

logical_xor(...);	Logical XOR function.
maximum(...);	Returns the max of x and y (i.e. x ¿ y ? x ; y) element-wise.
minimum(...);	Returns the min of x and y (i.e. x ¡ y ? x ; y) element-wise.
mod(...);	Returns element-wise remainder of division. When x ¡ 0 xor y ¡ 0 i.
multiply(...);	Returns an element-wise x * y.
multiply_no_nan(...);	Computes the product of x and y and returns 0 if the y is zero, even if x is NaN or infinite.
ndtri(...);	Compute quantile of Standard Normal.
negative(...);	Computes numerical negative value element-wise.
nextafter(...);	Returns the next representable value of x1 in the direction of x2, element-wise.
not_equal(...);	Returns the truth value of (x != y) element-wise.
polygamma(...);	Compute the polygamma function .
polyval(...);	Computes the elementwise value of a polynomial.
pow(...);	Computes the power of one value to another.
real(...);	Returns the real part of a complex (or real) tensor.
reciprocal(...);	Computes the reciprocal of x element-wise.
reciprocal_no_nan(...);	Performs a safe reciprocal operation, element wise.
reduce_all(...);	Computes the "logical and" of elements across dimensions of a tensor.
reduce_any(...);	Computes the "logical or" of elements across dimensions of a tensor.
reduce_euclidean_norm(...);	Computes the Euclidean norm of elements across dimensions of a tensor.

reduce_logsumexp(...);	Computes log(sum(exp(elements across dimensions of a tensor))).
reduce_max(...);	Computes the maximum of elements across dimensions of a tensor.
reduce_mean(...);	Computes the mean of elements across dimensions of a tensor.
reduce_min(...);	Computes the minimum of elements across dimensions of a tensor.
reduce_prod(...);	Computes the product 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.
reduce_variance(...);	Computes the variance of elements across dimensions of a tensor.
rint(...);	Returns element-wise integer closest to x.
round(...);	Rounds the values of a tensor to the nearest integer, element-wise.
rsqrt(...);	Computes reciprocal of square root of x element-wise.
scalar_mul(...);	Multiplies a scalar times a Tensor or IndexedSlices object.
segment_max(...);	Computes the maximum along segments of a tensor.
segment_mean(...);	Computes the mean along segments of a tensor.
segment_min(...);	Computes the minimum along segments of a tensor.
segment_prod(...);	Computes the product along segments of a tensor.
segment_sum(...);	Computes the sum along segments of a tensor.
sigmoid(...);	Computes sigmoid of x element-wise.
sign(...);	Returns an element-wise indication of the sign of a number.
sin(...);	Computes sine of x element-wise.

$sinh(...);$	Computes hyperbolic sine of x element-wise.
$sobol\_sample(...);$	Generates points from the Sobol sequence.
$softmax(...);$	Computes softmax activations.
$softplus(...);$	Computes softplus; log(exp(features) + 1).
$softsign(...);$	Computes softsign: features / (abs(features) + 1).
$sqrt(...);$	Computes element-wise square root of the input tensor.
$square(...);$	Computes square of x element-wise.
$squared\_difference(...);$	Returns (x - y)(x - y) element-wise.
$subtract(...);$	Returns x - y element-wise.
$tan(...);$	Computes tan of x element-wise.
$tanh(...);$	Computes hyperbolic tangent of x element-wise.
$top\_k(...);$	Finds values and indices of the k largest entries for the last dimension.
$truediv(...);$	Divides x / y elementwise (using Python 3 division operator semantics).
$unsorted\_segment\_max(...);$	Computes the maximum along segments of a tensor.
$unsorted\_segment\_mean(...);$	Computes the mean along segments of a tensor.
$unsorted\_segment\_min(...);$	Computes the minimum along segments of a tensor.
$unsorted\_segment\_prod(...);$	Computes the product along segments of a tensor.
$unsorted\_segment\_sqrt\_n(...);$	Computes the sum along segments of a tensor divided by the sqrt(N).
$unsorted\_segment\_sum(...);$	Computes the sum along segments of a tensor.
$xdivy(...);$	Returns 0 if x == 0, and x / y otherwise, elementwise.
$xlog1py(...);$	Compute x * log1p(y).
$xlogy(...);$	Returns 0 if x == 0, and x * log(y) otherwise, elementwise.

$zero\_fraction(...)$;	Returns the fraction of zeros in value.
$zeta(...)$: Compute the Hurwitz zeta function .	

3.4.1 Loss Function

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

$$f(x) = 1 - \widetilde{QNR} \tag{3.2}$$

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. With this loss function, the result after the training presents a lot of negative values and also after a clip by values of the image with 0 and 1 as minimum and maximum, the performance was unsatisfactory (results illustated in the Chapter 4). This because the model want to minimize the loss functions and so, maximize the QNR considering (25) no matter the sense of output values. After some tests, it was considered to solve this problem using a different D_s, the $D_s reg$ [41] and migrate to an approximation of HQNR instead of the QNR. It was discovered that it's really difficult to manipulate quaternions and hypercomplex numbers in TensorFlow in such a way that the function remains differentiable so, the exactly HQNR it was impossible to implement. Also a different kind of D_λ was implemented because the first one used by the original QNR did not reflect great performance according to the scientific community and the 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 and 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 program was implemented in such a way that all the possible algorithms, methods, parameters, learning rate and so on, can be defined on the terminal with arguments using the standard *argparse* library. This allow to run, with a bash commands, 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 increase in performance and after, a stable phase in which the index has not a notable mutation.

4.2 Experimental Settings

To compare the different methods, it was also trained the model with the reference image itself using the MSE loss function. The purpose is to compare the previous described methods with the best possible solution and to find the best index to use for the loss function, but is not applicable in a real approach. Using the reference image, it is clear that the model will reproduce the performance and the behaviour of the indexes during the traning can suggest which index could be used for a real non-reference training.

All those type 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 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.

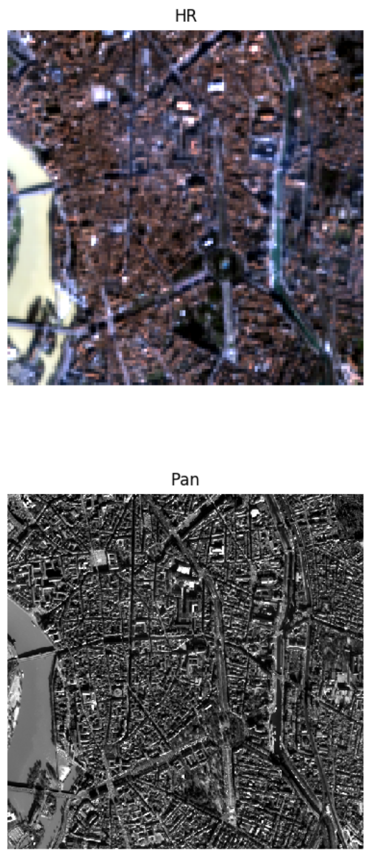

Figure 4.1: Multispectral and Pancromatic Toulouse images used for all the tests

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. 4.2 4.3 4.4, even the QNR itself. This because the algorithm, to minimize the error builded with the QNR developed in python, generate an image with negative values. An

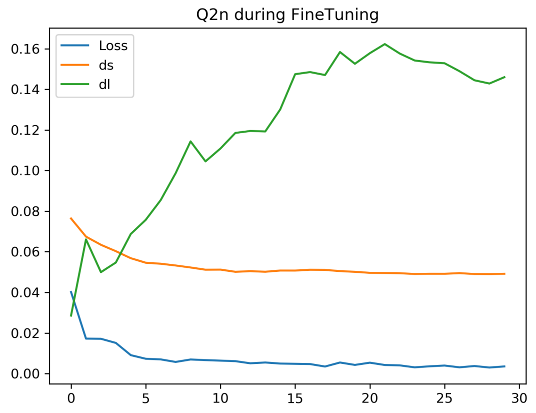

Figure 4.2: Loss, DS and DL of QNR during the training

input with negative values is not expected by any of the indexes used and this create a discrepancy between the QNR developed in python and the QNR of the Matlab Toolbox. Indeed, the QNR developed in python and used in the loss function it was tested with regular images with values between 0 and 1 and it returned same results of the Matlab version.

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

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. 4.5) the results are illustated for the Ikonos Toulouse image.

With this type of loss function it was recorded an increase in all the indexes

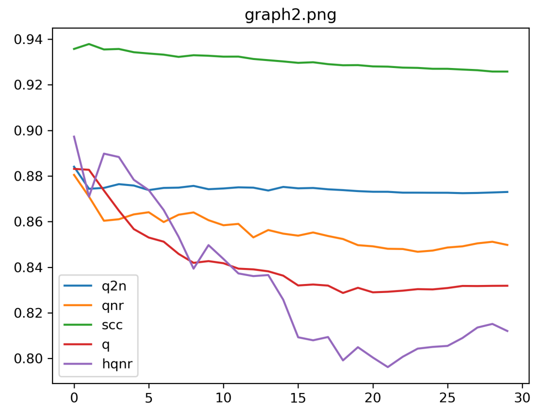

Figure 4.3: Q2N, QNR, SCC, Q and HQNR during the training

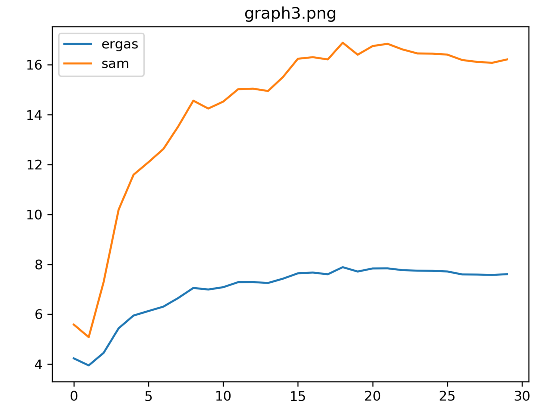

Figure 4.4: ERGAS and SAM during the training

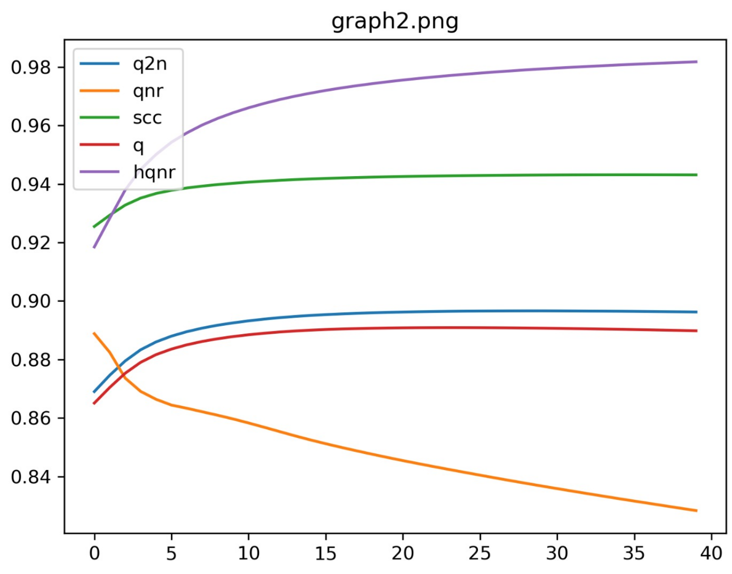

Figure 4.5: Q2N, QNR, SCC, Q and HQNR during the training using the loss function with HQNR

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, test with different values has been lauched. At the beginning with steps of 0.5 and after for particupar ranges with steps of 0.25.

In Fig. 4.6 and 4.7, in the legend the first argument is alpha and the second is beta. They are the inverse respectively of p and q.

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. 4.8. GT is the method that use the Ground Truth during

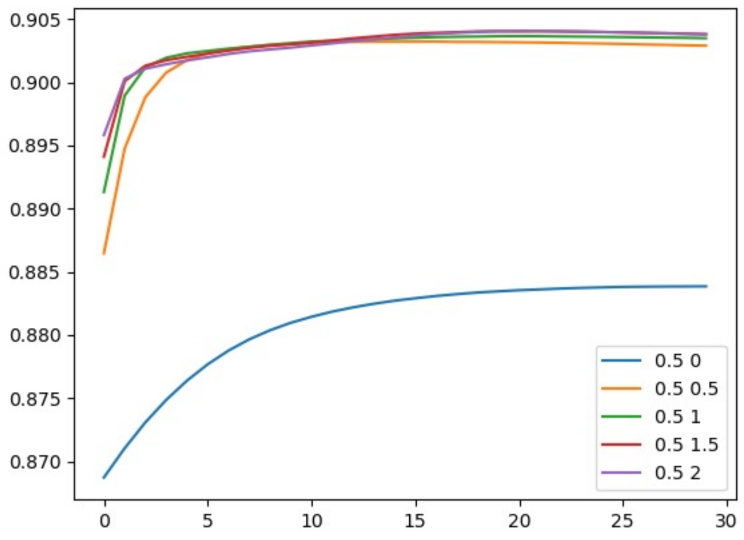

Figure 4.6: Alpha and Beta with a 0.5 step

the training to have the best performance, RR is the method used in [4] with a reduced 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.

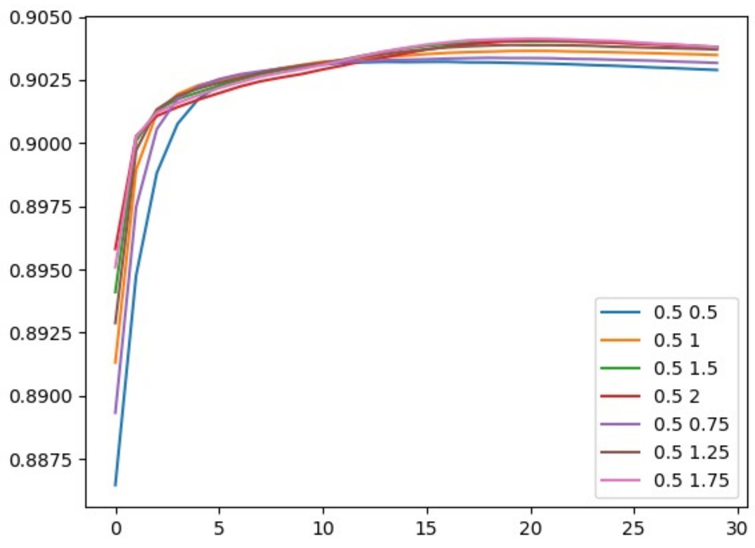

Figure 4.7: Beta with a 0.25 step between beta 0.5 and beta 2 and alpha setted to 0.5

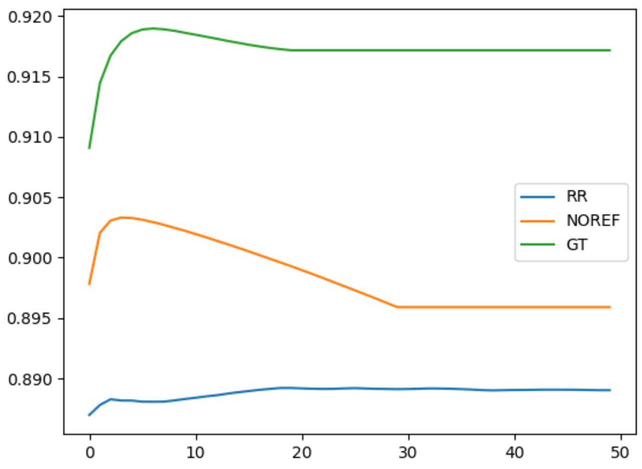

Figure 4.8: Experiment results with GT, RR and NOREF

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

[36] 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.

[37] 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.

[38] 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.

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

[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.

.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
```