Silesian
University
of Technology

Politechnika Śląska
Wydział Automatyki, Elektroniki i Informatyki
Kierunek Informatyka

Praca dyplomowa magisterska

Data augmentation for super–resolution reconstruction using deep convolutional neural networks

Autor: Maciej Ziaja

Promotor: dr hab. prof. Pol. Śl. Michał Kawulok

Gliwice, lipiec 2020

Abstract

The aim of this work is to enhance super–resolution satellite imaging by using data augmentation techniques based on deep learning algorithms. Super–resolution is a technology that enables upscaling images to a higher resolution with more refined details and improved quality. Such image–enhancing techniques are nowadays undergoing rapid development thanks to advancements in deep learning and convolutional neural networks. Deep learning is an approach in which training data plays a key role in the outcome and quality of the solution. Size and quality of the dataset used to train super–resolution networks are crucial to achieve best results. This is especially significant when working with satellite images, which are effortful to acquire in large numbers. Thus, when training a super–resolution network, it may be worth incorporating data augmentation techniques. Data augmentation is a process that intends to enlarge and improve training datasets for machine learning by transforming, multiplying or generating data. This process has been traditionally done using trivial techniques, however this work aims to use deep learning to generate datasets for training super–resolution algorithms. Following chapters provide an overview of modern super–resolution solutions and a proposal of deep learning algorithms to enhance the training datasets. Results of the work are evaluated by testing super–resolution networks which were trained on the datasets created during the project.

Contents

1 Introduction **1**
- 1.1 Super–resolution technology . . . 1
- 1.2 Purpose of data augmentation and available solutions . . . 2
- 1.3 Aim of the work and motivation . . . 4

2 Overview of super–resolution imaging techniques **5**
- 2.1 Characteristics of satellite imagery . . . 5
- 2.2 Machine learning for image processing . . . 6
  - 2.2.1 Neural networks and deep learning . . . 6
  - 2.2.2 Encoder–decoder mechanism . . . 7
  - 2.2.3 Measuring quality of image–generating neural networks . . . 8
- 2.3 Super–resolution with HighRes–net . . . 9
  - 2.3.1 Architecture overview . . . 9
  - 2.3.2 Super–resolution inference process . . . 10
  - 2.3.3 Registered loss calculation . . . 11
- 2.4 Other super–resolution architectures . . . 12

3 Scope of the work **13**
- 3.1 Selection of data types and augmentation techniques . . . 13
  - 3.1.1 Data types in super–resolution . . . 13
  - 3.1.2 Data augmentation techniques and appproaches . . . 15
- 3.2 Plan of experiments . . . 16
  - 3.2.1 Required data . . . 16
  - 3.2.2 Experiment flow . . . 19
- 3.3 Selection of tools and technologies . . . 19
  - 3.3.1 Language and libraries . . . 19
  - 3.3.2 Data and experiment management . . . 21
- 3.4 Project strucutre . . . 23
- 3.5 Additional remarks . . . 23

4 Data augmentation with usage of deep learning **25**
4.1 Proba–V preprocessing 25
4.2 Augmentation network architectures 27
4.2.1 Simple fully–convolutional network 27
4.2.2 Fully–convolutional encoder–decoder network 28
4.2.3 Generative adversarial network 28
4.3 Training details 31
4.4 Intermediate results 34
4.5 Implementation details 36

5 Super–resolution training and evaluation **39**
5.1 Training HighRes–net 39
5.1.1 Cross validation as stop condition 39
5.2 Evaluation and results 41

6 Summary **47**

Appendices **49**

A Model and training parameters of the augmentation networks **51**

Chapter 1

Introduction

1.1 Super–resolution technology

Super–resolution is a group of techniques that upscale images and improve their quality. Such an algorithm can be treated as a function that takes an image and returns it with a resolution n times larger. Algorithms taken into account in the process usually upscale images two or three times.

It is important to distinguish between super–resolution and traditional upscaling algorithms. The later use interpolation to enlarge images; however, they hardly improve the quality of the image. The intent of super–resolution is not only to upscale images, but to improve the quality and detailing. Nowadays, such an effect is achieved using machine learning, precisely—deep learning—a technology that utilizes multi–layered neural networks trained with large datasets. Deep learning networks that process image data usually utilize convolutional layers. Such layers contain a number of image filters that are tuned during training. Like all the rest of machine learning algorithms, the deep learning based super–resolution works in a statistical manner. This means that the extra details created during the image enhancement process state an imaginary approximation of image features. It is important to keep the statistical nature of machine learning algorithms in mind.

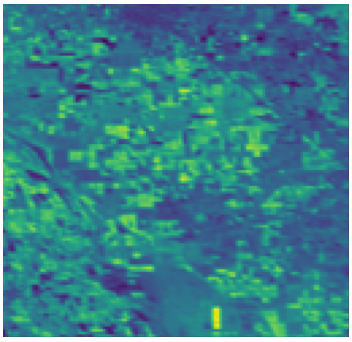
Two kinds of super–resolution algorithms can be outlined: *one–to–one* and *many–to–one*. The first one is the obvious approach, where one low–resolution image is translated into high–resolution one. The latter is more advanced technique, which utilizes multiple low–resolution images of the same scene to produce one high resolution picture. The usual approach is to have multiple low–resolution images that are slightly shifted. Data from these multiple images is merged together to produce an image of greater quality. This approach can lead to best results in super–resolution. In some scenarios the data fusion can lead to recreation of high resolution details that are hardly visible in any single low–resolution

image. Moreover, it should be noticed that the super–resolution networks trained on domain–specific data often cannot be used to enhance images with different contents. For example, if a network was trained on a dataset with human faces, it is likely to perform poorly on satellite images. Network architecture can also be domain–specific, for example, utilizing different bands of a multispectral image.

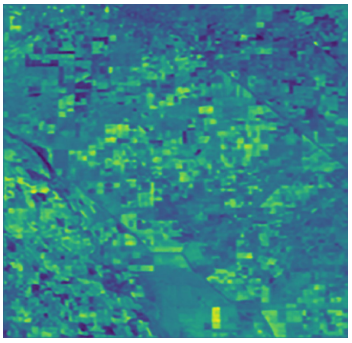
Super–resolution imaging is a technique relevant in the fields of satellite imaging remote sensing and geoscience. The most common reason for image enhancement is for aesthetic reasons. This application is viable in satellite imaging; however, super–resolution can lead to other practical advantages. Image enhancing techniques can be used as a preprocessing step in remote sensing pipelines. For this reason super–resolution can be especially useful when considered in the context of satellite imagery. A visual demonstration of modern super–resolution applied on satellite images can be seen in figures 1.1 and 1.2. In these examples a set of nine low–resolution images per each scene was super–resolved into a single high–resolution one. A sample of those nine images is shown in the comparison.

1.2 Purpose of data augmentation and available solutions

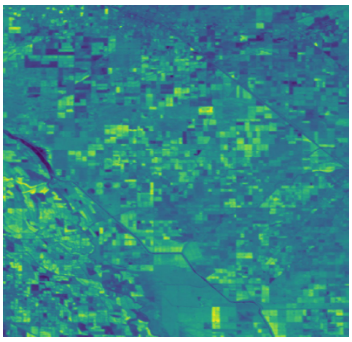
Deep learning, utilized in the modern super–resolution techniques, requires a lot of data to train successfully. Increase in quality and size of dataset can lead to far better results when training a neural network. This is why data augmentation techniques are often used to improve performance of deep networks. Data augmentation incorporates various transformation to improve, multiply or generate training data. Data created in the augmentation process is often called *synthetic*. Common techniques to improve image data include zooming, resizing, shifting, flipping, rotating, distorting, adding noise, modifying colors and exposure. These operations may be application–specific. To give an example, one should beware of distorting or flipping data containing with constrained geometry, like road signs. The mentioned augmentation techniques can be considered classic and rather trivial. However, more advanced approach can be taken to generate data. It is possible to create deep neural networks to create data augmentation transformations. This approach can be especially useful when a dataset is available that it is too small to train the desired network. A smaller network can be made and trained on an existing small dataset to multiply the data. Then the augmentation network can be used to generate more data for the original model to learn. With deep learning capabilities networks can learn to multiply, transform or even generate data without direct input [3].



(a) One of nine low–resolution images

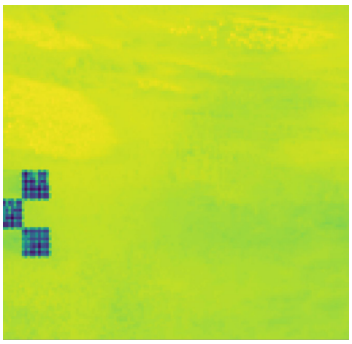


(b) Super–resolution reconstruction of the scene

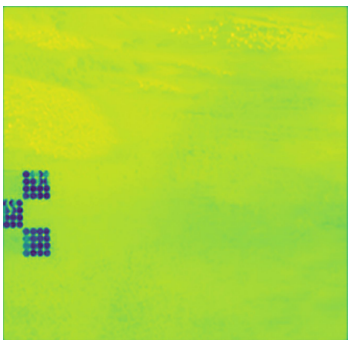


(c) Real–life high–resolution picture of the scene

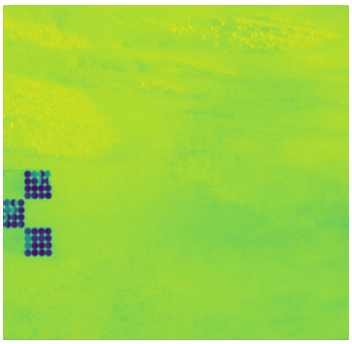
Figure 1.1: A demonstration of super–resolution technique, where nine low–resolution satellite images of a farming area are turned into one high–resolution image



(a) One of nine low–resolution images



(b) Super–resolution reconstruction of the scene



(c) Real–life high–resolution picture of the scene

Figure 1.2: A demonstration of super–resolution technique, where nine low–resolution satellite images of a barren landscape are turned into one high–resolution image

1.3 Aim of the work and motivation

The objective of the work is to create a set of augmentation networks for enhancing super–resolution training data. Subsequent chapters will present considered super–resolution architectures with greater details and propose neural network models for data augmentation. The task of the augmentation algorithms is to create low–resolution images for high–resolution ones, to make training pairs for super–resolution networks.

The nature of super–resolution technology and satellite imagery imposes certain ways in which data augmentation should be applied to the training data. Super–resolution is trained using pairs of data—low resolution image with high resolution image (or a set of low resolution images with high resolution image in case of many–to–one network). This requirement renders compiling training sets a challenge, especially in the field of satellite imagery. In such a scenario the most common technique is to use resizing algorithms on single image datasets. A set of training pairs can be created by downscaling high–resolution images. In the case of many–to–one networks, a single high–resolution image can be multiplied and shifted before shrinking to create more low–resolution images. Such a technique may work well; however, it infuses the data with information about resizing algorithms. The way high–resolution and low–resolution images relate in such a set depends on the interpolation algorithm (e.g., bicubic, bilinear, nearest, lanczos). The network trained on such datasets will likely learn to invert given interpolation methods. This does not match exactly real–life scenarios, most images are not created using resizing algorithms. Another approach utilizes pairs of real low–resolution and high–resolution images of the same scene, taken by cameras of different quality. The con of this method is challenging data acquisition process—satellite images are rarely taken in pairs. Such a dataset has to be deliberately made with super–resolution in mind, which makes such data less common. The main idea of the project is to use such a dataset of real–life data to train an augmentation network. Such a network would learn to create low–resolution images for high–resolution, without imprinting resampling algorithms mechanisms into the data. The relation between low and high–resolution images in such an augmented dataset would resemble the relation between same image taken by cameras of different quality. The augmentation neural network can be then used on other satellite image datasets to generate training data pairs for super–resolution. Different data, models and generation techniques can be used to achieve desired results. Possible variations are discussed in the course of this work to improve super–resolution datasets.

Chapter 2

Overview of super–resolution imaging techniques

2.1 Characteristics of satellite imagery

This work centers around super–resolution technique in the sphere of satellite imagery. As mentioned in the introduction, image enhancing can be domain specific. This is especially crucial when satellite photos are taken into account. Pictures taken from aerospace devices differ substantially from normal photography. Mulit–image observation is usually favoured over single–image. Satellites often take a series of photos of a single scene. This puts emphasis on the mulit image super–resolution techniques in the many–to–one fashion. Another unique feature of satellite observations is the usual spectral width of the imagery. Scientific *hyper–spectral* apparatus present on satellites often take photos in a very wide spectrum that may not include frequencies of visible light. This specific kind of image with large spectral dimension is often called a *hyper–spectral cube*, because it can be represented as a three–dimensional tensor (cube) with height, width and spectral dimensions. Spectral bands in the cube can contain wavelengths such as infrared, near–infrared, panchromatic[1], radio frequencies and more. Such images are often stored in special file formats or in a series of high bit–depth standard lossless image formats, such as PNG or TIFF. These can take up to 12 bands in different files per a single satellite photography. Another crucial property of satellite imagery is the GSD (*ground sample distance*) parameter, which denotes spacial distance between pixels of a digital image. For example, one–meter GSD states that that location of adjacent pixels is one meter apart on the ground.

[1] A spectral range similar to the range of traditional monochromatic grayscale photography. This range is usually highlighted because of connections with pre–digital imaging of the past century.

2.2 Machine learning for image processing

2.2.1 Neural networks and deep learning

Machine learning is a computer science technique that solves problems by fitting algorithms to data, by optimization algorithms and statistics. This approach contrasts with the traditional imperative problem–solving, where algorithms are designed with step–by–step attitude. *Artificial neural networks* are machine learning structures modeled after living organisms. The traditional neural networks consist of layers of densely connected neurons. Each of the neurons contains a set of inputs with connected weights. The output of a neuron is passed through a nonlinear activation function. The fitting process of such a network consists in adjusting the input weights. This kind of machine learning architecture has been initially used with a set of predefined image filters. A set of such filters would include basic geometric shapes. These small filters would be convoluted with the input image. The results of such an operation would be then fed into the neural network to get the final result of image processing. With the advancements in the machine learning area, a new kind of neural network layer was created—a *convolutional layer*. These layers consist of (one, two or even three dimensional) filters that can be convoluted with the input image. However, in contrast to the older technique, these filters are adjusted in the fitting process of the network. Elements in the filter tensor are treated like neuron weights, and they are accommodated during gradient descent. This enables creation of much better and flexible image processing neural networks.

However, the creation of convolutional neural networks leads to increasing complexity and number of parameters in models. This issue can be addressed by using very large datasets for the fitting process. Nowadays, the smallest datasets for training modern neural networks contain thousands of images. Such trainings require a lot of time and processing power; they usually must be performed using (even multiple) GPUs and may last a few days. This combination of three factors: complex mulit–layered neural networks (often with media–oriented specialized layers), very large datasets (often with many classes and objects) and utilization of expensive time and resource–consuming trainings constitute what is called *deep learning*. This kind of machine learning has proven, in the last ten years, to hold a revolutionary potential, pushing forward techniques such as image and audio processing beyond what is possible with older methods. Super–resolution, which this work revolves around, is possible thanks to advancements in the deep learning.

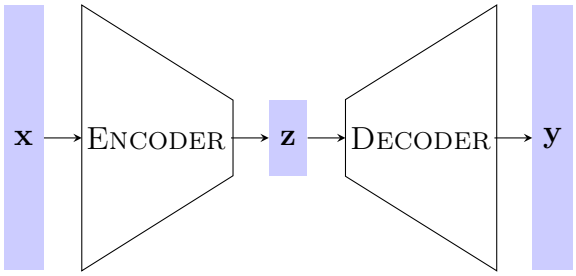


Figure 2.1: Schematic of encoder–decoder mechanism

2.2.2 Encoder–decoder mechanism

Encoder–decoder network architecture is a common pattern in generative image processing. It is used both in super–resolution models and in the data augmentation network presented in the latter chapters. Encoder–decoder translates input data into abstract state during encoding, then reconstructs it when decoding. The mid–point of the architecture usually bottlenecks the information containing compressed–like data. Convolutional interpretation of the encoder–decoder is usually used when working with images. During the encoding process the depth of input is usually increased and spatial dimensions are shrunken. This is achieved by subsequent usage of convolutional and pooling layers. After encoding, the compressed data can undergo some form of processing. For example, it can be flattened and passed through a densely connected layer, although this is rarely applied in the super–resolution, because fully–connected layers break the fully convolutional nature of a network (meaning that it cannot process images of varying spatial size). The decoding process commonly reconstructs depth dimensions into spatial size by upsampling or transposed convolution. The output may match the input dimension; however, it is not necessary. In super–resolution it is common to output data of different size than the input. Encoder–decoder architecture is appropriate for image–to–image transformations in machine learning. The inner workings of such an architecture are shown in the figure 2.1, where x and y denote input and output and z is the encoded hidden state.

Encoder–decoder mechanism is often enhanced with *residual connections*. These are often called *skip connections* because they form parallel branches in networks that skip certain operations. These skip routes are then summed with the result of an operation, resulting in additional direct flow of information during forward and direct gradient flow on the backward pass. Residual connections applied between arms of an encoder–decoder create what is called *U–Net* architecture. In the case of super–resolution processing, the forward skips can be viewed as routes for transporting unprocessed low–frequency information. This information can be used during the decoding step in the encoder–decoder scheme. Another way to

understand residual connection is to look at them as local ensembles of shallow networks.

2.2.3 Measuring quality of image–generating neural networks

Both super–resolution networks and data augmentation networks input and output images. Quantitive evaluation of such networks require comparison of two images—the network output and the ground truth reference image. Images are usually compared using metrics like *mean absolute error*, *mean square error* and *peak signal to noise ratio (PSNR)*. These calculate error between pairs of corresponding pixels in different ways. However, these metrics may be insufficient for super–resolution related problems. Calculating pixel–wise differences doesn't resemble the way humans estimate image quality. Images of varying perceived quality can have the same *PSNRs* compared to the reference image.

To measure image similarity in a more reliable way *structural similarity index (SSIM)* [11] was introduced. *SSIM* calculates image quality in three main components:

- Average *luminance*.
- *Contrast* as standard deviation of pixels.
- *Structure* as luminance difference divided by standard deviation.

However, these values are not calculated globally. Instead, *SSIM* values are measured using windows with pixel weights determined by Gaussian distribution. Values of *SSIM* components are combined using a compound formula. The precise mathematical description of the SSIM metric can be found in the bibliography. Advantages of *structural similarity index* render it suitable for super–resolution related image quality evaluation. However, modified versions of the previously mentioned traditional metrics can also prove to be useful in the image comparison, one of them being the *cPSNR*. The traditional PSNR has a potential drawback of being sensitive to bias in image brightness. This metric equalizes average brightness of compared images before calculating standard PSNR to alleviate this problem. Various of the mentioned metrics were in this work, in accord to specific requirements of each step in the augmentation and super–resolution training process.

Another challenge often encountered during super–resolution evaluation consists in aligning image pairs correctly. Often two images that are to be compared are slightly shifted; it is common for these dislocations to lie in sub–pixel domain. The process of aligning two similar images is called *registration*. Registration can be performed either with traditional or deep learning based algorithms.

2.3 Super–resolution with HighRes–net

In recent years many super–resolution architectures have emerged due to advancements in deep learning techniques. At the moment the state–of–the art model is RAMS (*Residual Attention Multi-image Super-resolution*). However, in this work *HighRes–net* architecture is utilized. *HighRes–net*, which is few months older, achieves slightly worse results; however, it is simpler and faster to train [12]. Because the aim of the work is to compare different data generation techniques, not the super–resolution algorithms themselves, the more manageable architecture was chosen. A brief description of the more sophisticated architecture is also given to provide a wider context.

2.3.1 Architecture overview

HighRes–net [4] is a super–resolution network based on generative deep learning. It falls into the category of *multi–frame super–resolution (MFSR)* algorithms, which takes *many–to–one* (or *multi–image*) approach to output generation. In MSFR systems input is a series of images, taken with a slight shift, perhaps with a small time interval. The input series contains more information, than a single image, as a result of random displacements, noise disturbances and atmospheric conditions. MSFR tackles the problem of aliasing in sampled data. Low frequency parts of the image, with large geometry and little detail don't differ much between many images. However, MSFR is crucial when enhancing small detailing. Upscaling small details from a single image can be non–reliable due to aliasing. Applying MSFR techniques and multiple low–resolution images fusion leads to de–aliasing information contained in the images. *HighRes–net* processing is divided into four subtasks:

1. **Co–registration**, which estimates relative geometric differences between input images. These include divergences, due to shifts, rotations, deformations, etc.)

2. **Fusion**, which combines multiple input images into a single one, that is more refined.

3. **Up–sampling**, which upscales low into high–resolution image.

4. **Registration–at–the–loss**, which estimates relative geometric differences of high–resolution prediction and ground truth, for more representative loss calculation. After calculating shift between super–resolution output and reference image, they are aligned using Lanczos resampling and then loss is measured. The registration and alignment are learned by a model inspired by a *ShiftNet* network architecture.

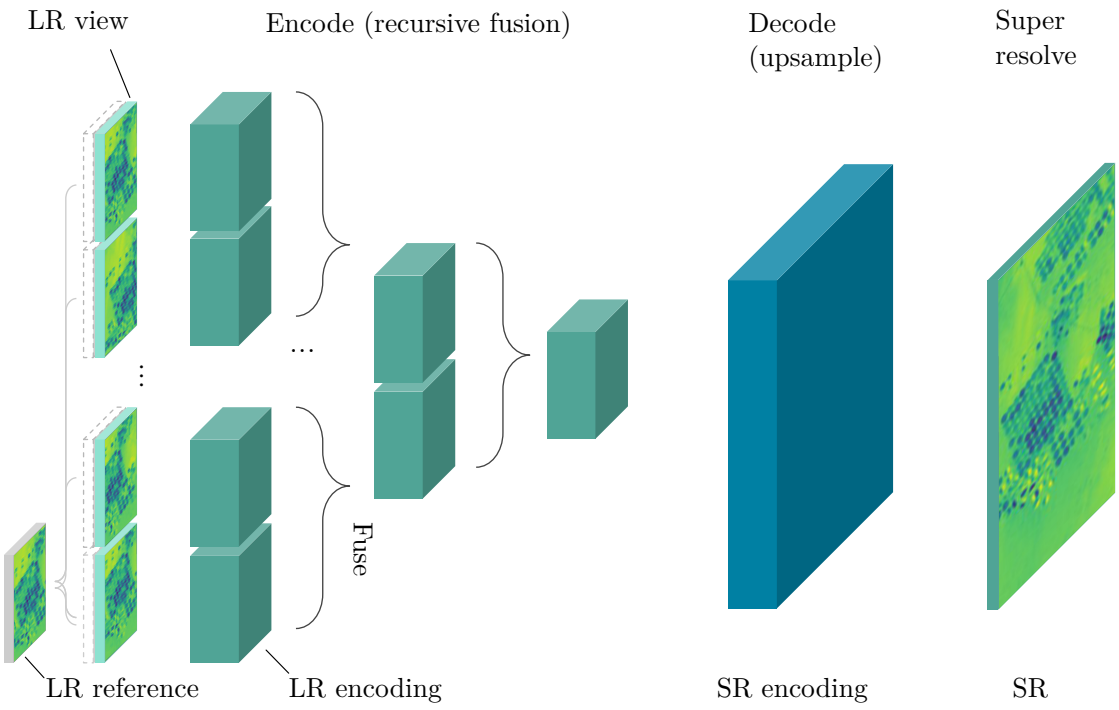


Figure 2.2: Schematic of inference in *HighRes–net* [4]

The unique feature of *HighRes–net* is that all of its parts are learned in a single fit in an end–to–end fashion.

2.3.2 Super–resolution inference process

The inference pipeline of HighRes–net is shown in the figure 2.2. The consecutive paragraphs will walk through each step in the process and explain how super–resolution is performed.

The key element of *HighRes–net* is achieving *multi–frame super–resolution* by *recursive fusion*. Image generation is done by a neural network organized in an encoder–decoder scheme. The input of the encoder is constructed from a series of low–resolution images. If necessary, the input set is padded with zero–valued images to ensure that the number of low–resolution images is a power of 2, which is required by the network architecture. For each input series, a *reference image* is computed using median values of images. Then the reference picture is paired with the input images. Each low–resolution and reference pair is processed through an embedding function. Embedding layers consists of a convolutional layer and two residual blocks with PReLu activations. For input of length n, output of the

encoding consists of n images, each convolved with the reference image. In this scheme embedding learns to perform a process called *implicit co–registration*, which is responsible for adjusting geometric differences between images in the input. It is important to notice that the embedding block is a single instance shared between input pairs.

The next step in the *HighRes–net* architecture is *recursive–fusion*. In this process output images are recursively fused together, pair by pair. Fusion operation consists of two steps—co–registration of the input pair and the actual fusion. The co–registration of fused images is similar to the co–registration of the input–reference pairs. It is done by convolutional layer with PReLu activation and two residual layers. Then the fusion itself is done, again by a combination of a convolutional layer and PReLu (this part doesn't include local residual layer). The whole co–registration–fusion includes a residual connection. Similarly to the embedding block, the fusion operator has a single instance that is shared for all steps of the recursion.

The last step of super–resolution process is to upscale the image by decoding the hidden state. This is done with transposed convolutional layer with PReLu activation. The transposition of the output of convolution makes the data grow in spatial dimensions, instead of the usual increase of depth when convolving. The final image is constructed by applying convolution of size one, which doesn't change the size of the image.

2.3.3 Registered loss calculation

As stated before, registration is an important part of *HighRes–net* architecture. It is especially crucial at the loss calculation step. Without registration, the network would learn to output blurry images as a result of shift between predictions and targets. Previous steps of *HighRes–net* include an *implicit co–registration*, where registration mechanisms learned by the network don't have to be necessarily based on shifts, but also other geometric distortions. During evaluation it is desired to register image shifts explicitly, thus the *registration–at–loss* differs from the registration performed during encoding and fusion. At the final step, the sub–pixel registration is done by the *ShiftNet–Lanczos* network. *ShiftNet* [13] was introduced before *HighRes–net*, in a separate research. It was created with image inpainting via *Deep Feature Rearrangement*. Because this kind of filling in missing picture areas works by reusing and transferring existing data, it is suitable to be used as a registration mechanism. It implements a modified *U–Net* [9] architecture. As mentioned in the introduction, U–Nets follow the encoder–decoder pattern with multiple residual connections. Pairs of convolution and deconvolution layers in the contracting and expanding arms of a U–Net feature a residual connection. The *ShitNet* variant of *U–Net* architecture contains an additional *shift* operation

for one of these residual connections. More about *ShiftNet* can be found in the publication.

2.4 Other super–resolution architectures

As mentioned, other super–resolution architectures are available, with RAMS [10] being the best performing one. RAMS utilizes a novel technique called *feature attention mechanism*, which enables the network to focus on high–frequency information that can be used to produce more detailed outputs. This leads to overcoming main locality limitations of convolutional operations. Mechanisms used in RAMS are specifically aimed at multi–image super–resolution of remote sensing data. RAMS approach takes into account the nature of satellite imagery—relatively low spatial resolution and high depth and temporal resolution. The attention mechanism works with three–dimensional convolutions to explore all possible directions. This architecture puts emphasis on simultaneous data exploration and from spatial and temporal dimension resulting in the best quality of multi–image super–resolution.

Chapter 3

Scope of the work

3.1 Selection of data types and augmentation techniques

3.1.1 Data types in super–resolution

The introductory chapters provide a general overview of super–resolution, data augmentation mechanism and types of satellite imagery. To lay out a plan of experiments, a subset of selected techniques should be defined. The previously described types of super–resolution training data are presented in the form of a graph in the figure 3.1. The graph features a distinction that has not been mentioned before—a difference between fully and semi–simulated mulit–image datasets. When multi–image training data generation is considered, one of two approaches can be taken. If multiple high–resolution images of the same scene are available, then low–resolution training images can be created by shrinking each of the distinct photos. This way of data generation is denoted as *semi–simulated*. Otherwise, one can create many low–resolution images from a single high–res one. This can be done by shifting the original image randomly multiple times and then, applying the shrinking transformation on each displaced copy. This procedure is denoted as a *(fully) simulated* data generation and is presented in the form of pseudocode as algorithm 1.

A selection of approaches to be taken into account in this work has been made and marked with color in the graph in the figure 3.1. The choice is rather arbitrary and aims to cover the most common, yet uncomplicated variants of data generation. Thus, it was decided to perform data augmentation for mulit–image super–resolution with simulated data with deep learning and resizing algorithms.

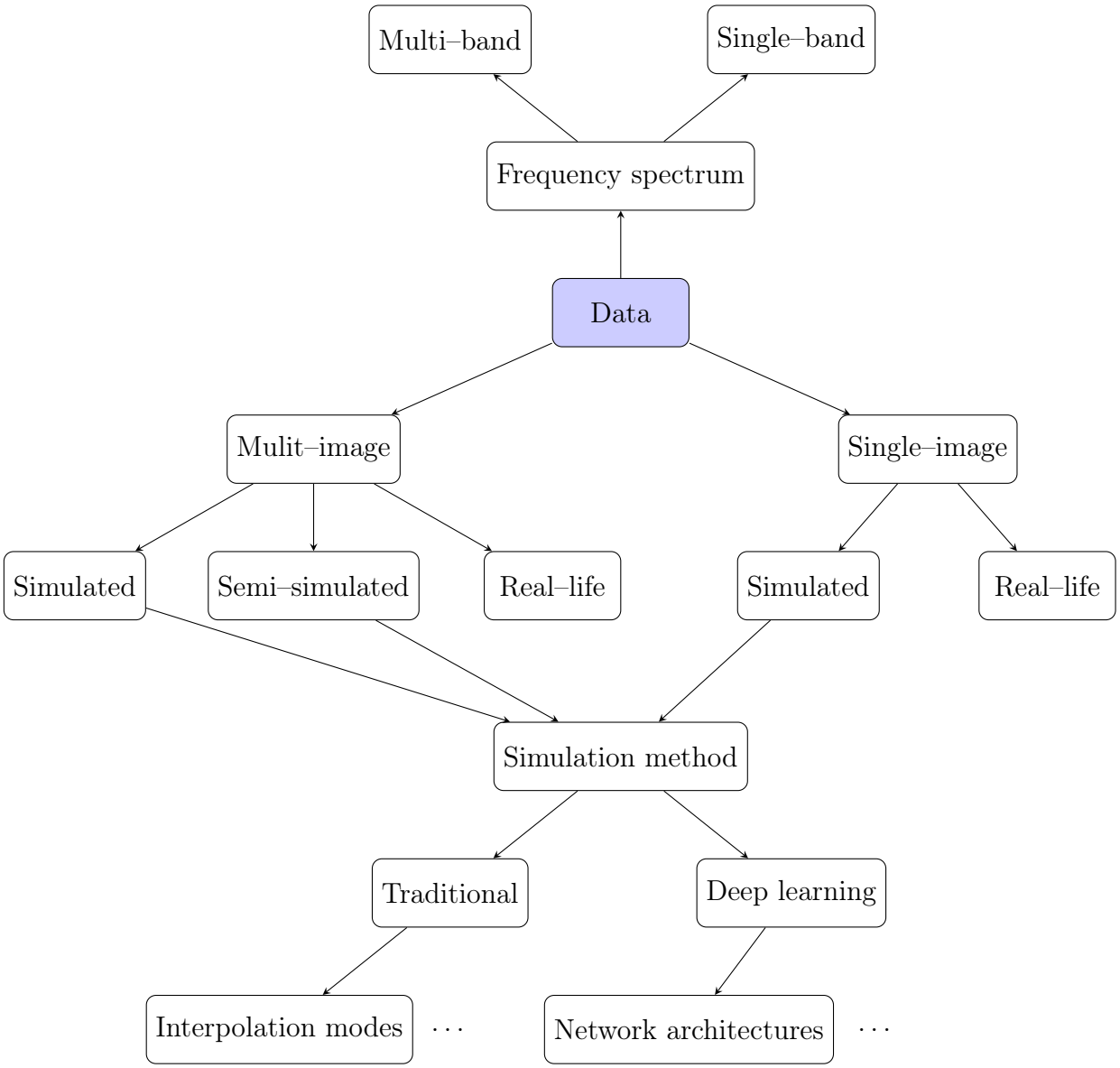


Figure 3.1: Graph of various training data generation techniques

Algorithm 1 Approach to generating fully simulated mulit–image datasets

Require: n_lrs, $shrink\_ratio$
  $max\_shift = shrink\_ratio$
  for all hr_img **do**
    $cropped\_hr\_img = crop\_border(hr\_img, max\_shift)$
    for $i = 0$ **to** n **do**
      $hr\_shift = generate\_shift(max\_shift)$
      $translated\_hr = translate(hr\_img, hr\_shift)$
      $lr\_image = shrink(translated\_hr, shrink\_ratio)$
      $cropped\_lr\_images_i = crop\_border(lr\_image)$
    end for
    $save\_scene(cropped\_hr\_img, cropped\_lr\_images)$
  end for

3.1.2 Data augmentation techniques and appproaches

Augmentation via deep learning

Augmentation with deep learning techniques is the key point of this work. In the subsequent chapters three augmentation architectures with varying level of complexity are introduced. It should be kept in mind that this kind of augmentation requires a separate dataset to fit the augmentation networks prior to the export of the proper super–resolution training data.

Augmentation via interpolation algorithms

In the process of the work, the deep learning–based augmentation methods are to be compared with traditional resizing algorithms which are based on interpolation techniques. The *bicubic interpolation* was chosen as a reference point. Furthermore, some of the resized images were subject to several transformations to enhance their quality. The brightness, contrast and noise of the interpolated images were adjusted using normal distributions with parameters:

Noise with 0.0 mean and standard deviation of 30.

Contrast with 1.0 mean and standard deviation of 0.2.

Brightness with 0.0 mean and standard deviation of 200.

Gaussian blur with sigma of 0.5 was applied on each of these low–resolution images. These transformations to the dataset created by bicubic interpolation were chosen outside the scope of this work.

Translations between low–resolution images in multi–image super–resolution data

As stated before, in this work multi–image super–resolution is taken into account. When generating images with both interpolation and deep learning techniques, the same approach to creating translations between low–resolution images was taken. These were created using uniform distribution between -0.95 and 0.95 in the high resolution domain (given that the max low–resolution images shift in this domain should be subpixel). The translations were applied in the vertical and horizontal directions.

3.2 Plan of experiments

3.2.1 Required data

The broad aim of the work is to train augmentation neural networks on small real–world datasets and then use the trained models to export a larger, synthetic dataset which should be used to fit super–resolution algorithm. A number of datasets is required to perform this task. To clarify the demands for specific datasets can be listed as:

1. Dataset containing high and low–resolution images for training an augmentation network.

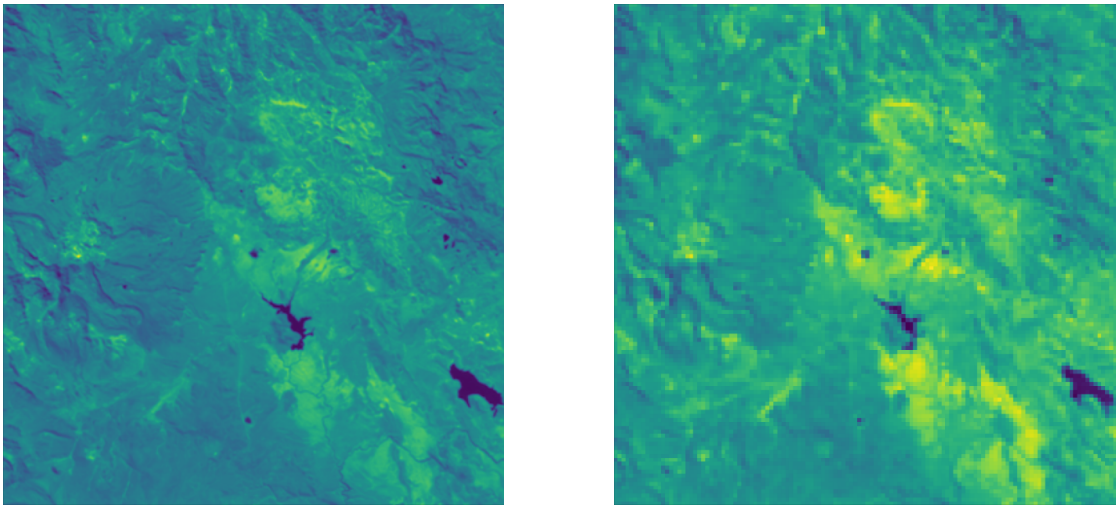
2. Dataset containing high and low–resolution images for testing the augmentation network.

3. Dataset that should be augmented with the network. This set doesn't need to contain high and low–resolution pairs. The low–resolution images for existing high–resolution ones should be generated by the augmentation network. Result of the augmentation will be used to train the super–resolution network.

4. Dataset of high and low–resolution images for testing the super–resolution network.

Proba–V as an augmentation training dataset

The *Proba–V* dataset [1] can be used to fill the first two requirements. It contains images taken during the *Proba–Vegetation* satellite mission launched by the *European Space Agency* in 2013. The dataset contains imagery taken in multiple spectral bands. Two subsets are regarded in this work—the *RED* band (610—690



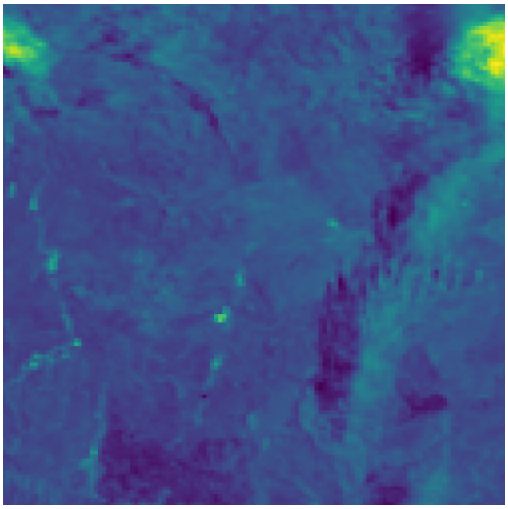
(a) High–resolution image (b) Low–resolution image

Figure 3.2: Sample image pair from *Proba–V NIR train dataset*

nm wave length) and the *NIR (near–infra–red)* band (777–893 nm wave length). Proba contains multiple real–life low–resolution images per one high–resolution image. Images of the same scene were taken in moments, thus they vary slightly in framing and atmospheric conditions. Unprocessed Proba high–resolution images are 384 by 384 pixels, low–resolution photos are 128 by 128 pixels. A selected pair of high and low–resolution samples from Proba–V dataset is shown in the figure 3.2. Furthermore, Proba–V features a set of binary masks for low–resolution images. These masks indicate areas of photos that may not be suitable for processing, such as clouds or blank spaces. An example of Proba–V image with a binary mask designating clouds is shown in the figure 3.3.

Sentinel–2 as a super–resolution training dataset

The super–resolution training part of the data demands can be met by utilizing *Sentinel–2* dataset [2]. Contrary to Proba, it doesn't feature high and low–resolution scenes, for this reason, it suits the role of the dataset to undergo data generation process. Sentinel–2 is part of a European earth observation programme that has lasted since 2015. It gathers data from two twin satellites that acquire optical imagery at high spatial resolution from 10 to 60 meters. Sentinel images are hyperspectral and feature a total of 13 bands. Since it was decided not to include multispectral super–resolution only band eight is to be used (which is close to infrared). A sample image from Sentinel–2 dataset is shown in the figure 3.4.



(a) Sample Proba–V image with invalid area that contains clouds

(b) Binary mask for a sample Proba–V image with invalid area

Figure 3.3: Proba–V sample with invalid area and its binary mask

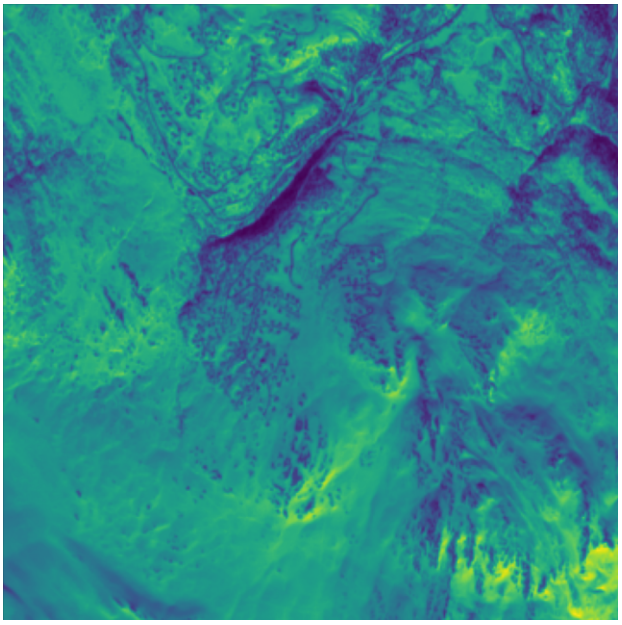


Figure 3.4: Sample image from *Sentinel–2 dataset* (eight band)

Dataset for super–resolution evaluation

Additional datasets are needed to evaluate final results. The goal of the tests is to measure super–resolution generalization capabilities and data selection impact on the metrics. This can be done in four ways with different data:

- Each augmented Sentinel dataset should contain a test subset.
- Test subset generated in different ways can be tested in cross–validation scheme.
- Separate real–world Sentinel images can be used for tests. Sentinel doesn't include high and low–resolution pairs, so numeric tests are not possible. However, it is still viable to perform a visual test in search of artifacts.
- The Proba-V test subset used to evaluate the augmentation process can be used to measure super–resolution performance. Since the initial tests are not a part of any decision process, there is no information leak and the Proba subset can be used twice.

3.2.2 Experiment flow

Given the selection of data augmentation methods and available, data an experiment flow can be laid out in the form of a graph in the figure 3.5. The graph follows the assumption of training various augmentation models with Proba–V dataset, exporting Sentinel–2 images for super–resolution training and final evaluation using cross validation and tests on real–world data.

3.3 Selection of tools and technologies

3.3.1 Language and libraries

Python, which is an industry standard for neural network research and scientific computing has been chosen as a main language for the implementation. It combines ease of prototyping new code with efficient use of existing libraries written in more performant languages. Python version 3.8 was chosen for the project as a reasonably recent, yet mature and field–tested version.

The main part of the work, consisting in writing neural network architectures and training them was written using *Tensorflow* library. Tensorflow has been a leading deep learning library in recent years and is widely used in the industry and academia. The version 2.5 of the library was chosen for the project. From the second version Tensorflow includes *Keras* neural network prototyping interface

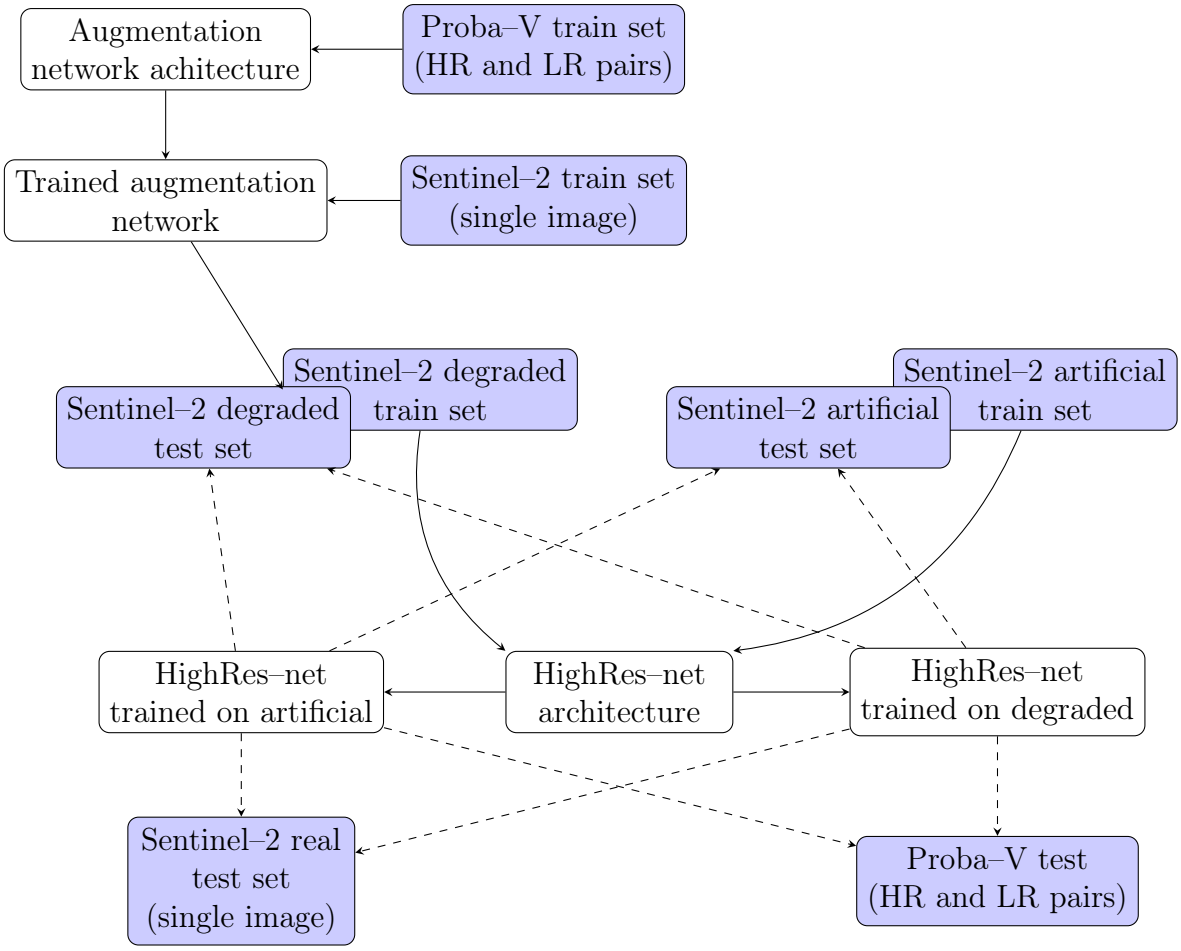


Figure 3.5: Graph of the experiment flow, solid lines indicate trainings, dashed arrows designate evaluations

natively as a default and recommended way for model building. This work utilizes the Keras API as well as some underlying Tensorflow function for fine tuning and expanding models. Tensorflow supports GPU acceleration via *Nvidia CUDA* libraries, which is crucial for conducting efficient trainings.

The image manipulation tasks are performed using *SciKit–image* library which offers a variety of IO functions, transformations and analysis tools. *Matplotlib*, another wide–spread Python library was used to create plots and heatmaps in the work.

NumPy, the most popular numerical calculations library was chosen as a tool for matrix manipulation and mathematical operations. Arrays created with NumPy are stored in contiguous memory layout, which enables utilization of performance optimizations such as vectorized operations. NumPy ubiquity in the world of scientific computing with Python proves to be very convenient. It is compatible with other popular libraries; it is interoperable with parts of Keras API, Scikit–image stores images as NumPy arrays, Matplotlib can plot data from NumPy arrays out of the box. NumPy arrays' performance comes from their static nature, their size is predetermined. This constrain is prerequisite to the contiguous memory layout of the arrays, which enables utilization of fast mathematical operations. However, this comes at a cost of flexibility; resizing the NumPy arrays is costly and often requires substantial memory reallocations.

The set of required libraries with proper versions can be installed using *requirements.txt* files. It is best to use it inside a Python *virtualenv*, running the command: `pip install -r requirements.txt`.

To ensure code quality and avoid mistakes, a *continuous integration* system was used. Continuous integration is based on automatic code building on the server with the help of version control systems. Every time a new commit is pushed to the upstream, a set of checks is run remotely. The integration system reports any errors which prevents integrating invalid code into the main system. The built–in GitHub continuous integration and delivery system, called *GitHub Actions* was used in the project. Integration checks are performed inside a Ubuntu Docker image and consist of various *flake8* linter analytics.

3.3.2 Data and experiment management

The codebase in the work was managed with *Git* version control system. Git is the most popular tool for tracking changes in code and text files. However, it was not created with media and datasets in mind. Git may work very slowly with large datasets and popular Git server providers such as *GitHub* offer limited storage for repositories. For this reason, *dvc*—a supplementary tool was used. *Dvc* consists of two main components: data versioning and experiment tracking. The data versioning part works similarly to systems like *Git LFS*; it stores metadata

inside the Git repository and the actual data in other storage (e.g. Google Drive). Information about the proper data is stored in the form of *MD5* hashes in the metadata files. The metadata versioned with Git can be used to download proper data from the external cloud storage. *Dvc* is analogous to Git in operation, it uses a command line interface with a similar structure. To download the data in the project one should run command `dvc pull`. Sine the metadata is versioned with Git, when going back in the commit history one should run `dvc checkout` after `git checkout` to conceal data versions.

The experiment tracking part of Dvc ensures reproducibility and result caching. The layout of experiments is stored in the *dvc.yaml* file where each stage of experiment is described with its input dependencies and outputs. The results of each stage can be cached and rebuilt automatically depending on changes in the dependencies. The experiments in this work usually depend on data and files with model architecture. Training models can be parametrized using a configuration file, here named *params.yaml*. Parameters of models and training can be changed by editing this file. Another convenience of dvc is that the outputs are automatically rebuilt on changes in the parameters. Trained models are cached and bound with code and parameters used for training. Similarly to the data versioning system, models and artifacts are also tracked with metadata and git. When going back with history, `dvc checkout` will also conceal artifacts and data. The stage of the experiment pipeline is stored in the *dvc.lock* file that also contains metadata. The command `dvc repro` is used to reproduce experiments.

Scripts in the project can also be run independent of dvc. Training can be run with command: `python -m cnn_res_degrader.train [-s] [-a] [-g] training_name` where the optional arguments decide which architecture to train and the positional argument is the experiment name. The trainings done automatically by the Dvc system contain word *dvc* as the experiment name. Validation on Proba–V dataset can be run with command: `python -m cnn_res_degrader.test [-s|-a|-g] weights_path output_dir` where the positional stores model architecture and positional arguments hold paths to trained model weights and output directory.

The augmented sentinel datasets can be created by running command: `python -m export_sentinel.py [-s|-a|-g] [-r] [-d] weights_path`, where architecture and path to weights are denoted as described earlier. The last two of the switches enable generating random translations, instead of using ones saved in sentinel files and running a demo export on one image instead of using the whole dataset.

3.4 Project strucutre

Files in the project have the following structure and function:

.github/workflows/build.yml continuous integration system configuration.

cnn_res_degrader main part of the source code, contains the augmentation models, training and testing scripts.

dataset_augmentation source code for exporting augmented Sentinel–2 datasets.

data super–resolution datasets and artifacts controlled by Dvc.

dvc.lock metadata to track experiment progress in current commit.

dvc.yaml dvc experiment description.

params.yaml models and experiment configuration.

requirements.txt libraries requirements.

Some configuration files and minor parts of project were omitted in this description.

3.5 Additional remarks

The following chapters will include many visualizations and image previews. They often include a side–by–side comparison between high and low resolution images of the same scene that were generated using different methods. When computers display single–chanel images (like the pictures presented in this work), a colormap is often used. The most popular colormap is grayscale which displays white for max values, blacks for min values and shades of gray in between. To improve visibility of details, a yellow and blue colormap called *Viridis* is often used in this work. Viridis is an example of a *Perceptually Uniform Color Map*, which aims to guarantee an even perceived contrast of all color shades in the map. However, when comparing a number of side–by–side images it is important to apply the colormap consistently across all pictures. Modern image processing libraries may try to stretch colormap to the range of input image. If images that are to be compared differ in maximal and minimal values, the way colormap is applied will be different. It is important to ensure that images in the same scene display pixels with the same value in the same color. For this reason, when several images are to be compared, they should be displayed with common colormap. At the same time, an outlier pixel in one of the images can result in the colormap not being well suited to display the rest of the pictures. The solution to choosing a

good colormap range is to use maximum and minimum values using pixel mean values and standard deviation, as presented in the formula 3.1. This way, only pixels within the range of three standard deviations from mean are taken into consideration when creating a common colormap for image comparison.

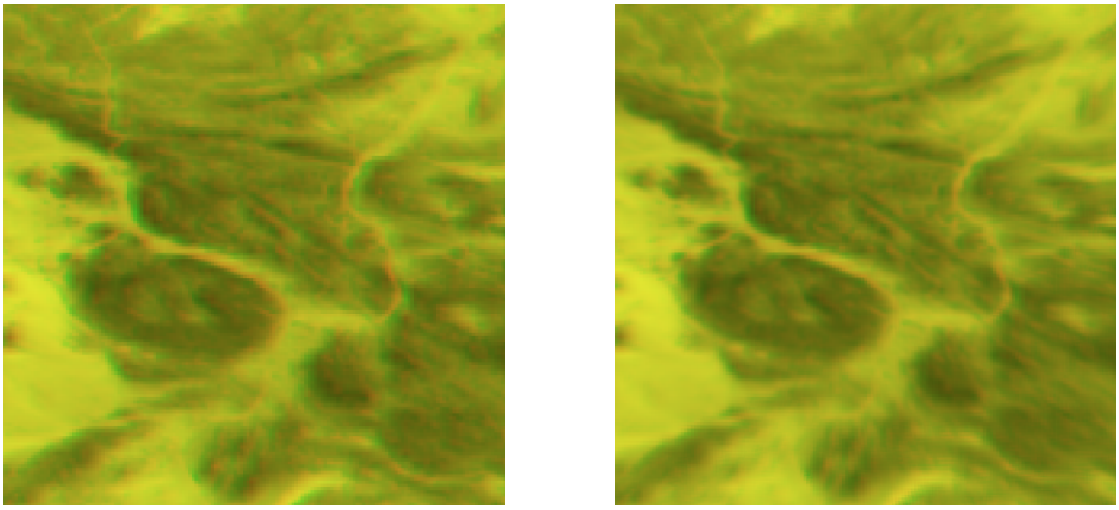
$$c_{max,min} = \bar{p} \pm 3 \cdot \sigma_p \tag{3.1}$$

Chapter 4

Data augmentation with usage of deep learning

4.1 Proba–V preprocessing

As stated before, it was decided to use Proba-V as the augmentation training dataset. As a preprocessing step, before training the high and low–resolution images should be aligned. Some architectures, like the *HighRes–net* feature deep learning based built–in mechanisms for image registration. However, for training a simple image–downscaling augmentation network, a more traditional approach to image alignement can be implemented. The preparation of dataset requires to turn the single high–resolution and multiple low–resolution imageset into high and low–res pairs. This can be done by multiplying high–resolution images and aligning each copy with the corresponding low–resolution one. Registering shift between images can be done using *phase–correlation* [6]. This method utilizes two–dimensional *Fourier transform*. To register sub–pixel translations, images can be upscaled before applying the algorithm. Since images to be aligned in Proba are of different resolution, they should be resized to common shape before registration. It was decided to perform registration in the high–resolution image domain. Shifting images will create blank columns and rows on their borders, which were removed by cropping the images. It is known that low–resolution images in Proba have sub–pixel dislocation, for this reason they should be cropped with one pixel border. High–resolution images are three times larger, consequently, a margin of three pixels should be removed from them. After this step of preprocessing high–res images are 378 by 378 pixels and low–resolution photos are 126 by 126 pixels. It is valuable to visualize the effects of the registration process to ensure its correctness. Because Proba images have single depth dimension, they can be visualized as different color channels. This technique is utilized in the figure 4.1, where red



(a) Proba–V images overlay before registration

(b) Proba–V images overlay after registration

Figure 4.1: Sample image pair from *Proba–V NIR train dataset*

and blue channels contain two pictures that are to be registered. If the pictures are aligned perfectly, the image should be yellow because red and blue channels overlay perfectly. The more unaligned the pictures are the more red and blue is visible in the picture, which is most visible at distinct edges. The visualization proves that the phase–correlation based registration is suitable for the Proba–V dataset—after registration the picture becomes more yellow and the unaligned edges are less visible.

Photos in the Proba–V dataset are saved as sixteen-bit images; however, only fourteen–bit values are utilized. It should be noted, to scale the images properly. Using Proba–V dataset with real–life high and low–resolution images poses an additional challange. As stated, the images contained in the dataset are taken at different moments, thus they differ in exposure and contrast. Images in high and low–res pairs differ in brightness, both at the level of local details and global average values. Distributions of image exposure in the dataset are presented in the figures 4.3 and 4.3 for RED and NIR subsets.

Having a dataset with high and low–resolution images of different brightness and contrast can be an obstacle for learning a neural network. For this reason, *histogram equalization* was applied to the learning data. This technique enhances image contrast and evens the histogram of pixel values. After equalization, the cumulative distribution of brightness is close to linear.

As noted previously, Proba–V includes binary masks denoting areas of images that are suitable for training. Some images include noticeable large areas of unus-

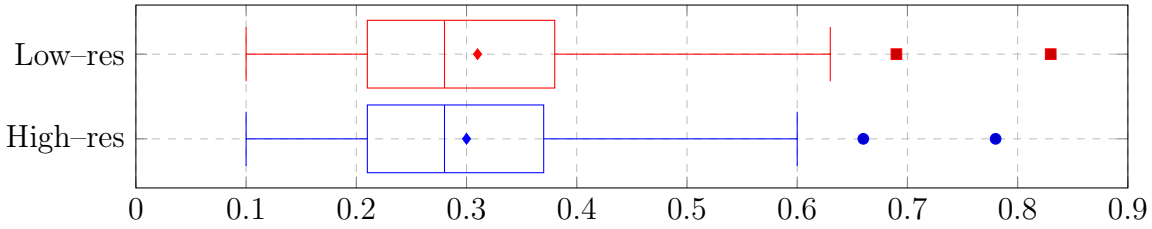


Figure 4.2: Exposure distributions in the RED Proba–V dataset

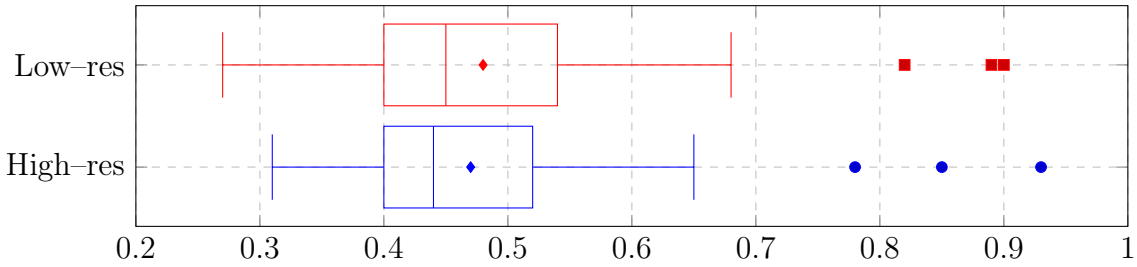


Figure 4.3: Exposure distributions in the RED Proba–V dataset

able pixels. For this reason, nine images with the least the percentage of invalid pixels were chosen per scene as training data. This excludes incorrect data from the training set.

4.2 Augmentation network architectures

The augmentation network is to perform a rather simple task of reducing the size of input pictures. Three architectures that perform this task have been created, they are presented with an increasing complexity.

4.2.1 Simple fully–convolutional network

The most simplistic approach to shrinking images with deep learning is to build a basic, fully convolutional network. The most simplistic implementation of this idea uses three convolutional layers with kernels of size three by three. The midmost convolution slides the filter window with a stride of three to decrease the size of input by three times. The last convolutional layer features one filter to reduce the output depth to a single channel. $ReLU$ activation function was used for each applicable layer. A precise description of the architecture is presented in the table 4.1.

Table 4.1: Simple fully–convolutional network architecture for data augmentation

Layer type	Output Shape	Number of parameters
$Input$	378, 378, 1	0
$Conv2D(filters = 64, stride = 1)$	378, 378, 64	640
$ReLu$	378, 378, 64	0
$Conv2D(filters = 64, stride = 3)$	126, 126, 64	36928
$ReLU$	126, 126, 64	0
$Conv2D(filters = 1, stride = 1)$	126, 126, 1	577
$Sigmoid$	166, 126, 1	0

4.2.2 Fully–convolutional encoder–decoder network

The more complex architecture that can be applied is based on the previously mentioned encoder–decoder network scheme. It is one of the most popular image–to–image neural network architectures. The encoder shrinks the image by a factor of six instead of three. Then the image is upscaled two times. In result, the output image is overall three times smaller; however, it has been processed by more convolutional layers than in the simple convolutional network.

There are various ways to upscale the image during the decoding process. The main ones are traditional upscaling and a *transposed convolution* (sometimes called *deconvolution*). The later performs a convolution and then transposes the output, which makes the spatial dimension grow in size. However, the transposed convolution layers can produce visible checkerboard artifacts on the image [8]. For this reason, the simpler layer architecture based on upscaling was chosen. Like the simple architecture, the encoder–decoder features a convolution with a single filter at the end. The detailed description of the network is shown in the table 4.2.

4.2.3 Generative adversarial network

In recent years, a new approach to training generative neural networks has emerged. The traditional supervised learning described in the previous sections consists in providing the network with an input and comparing the generated output with a ground truth to compute loss. The GAN—*generative adversarial network* approach requires creating two networks, a *generator* and *discriminator* [5]. The former is tasked with generating data, while the latter learns to differentiate images created by the generator from real ones. In the GAN scheme the discriminator learns like a traditional binary classifier—it is provided with real and generated images, which are labeled accordingly. Then discriminator loss is computed using standard classification metrics like *binary cross–entropy*. However, the generator learns in a

Table 4.2: Encoder–decoder network architecture for data augmentation

Layer type	Output Shape	Number of parameters
$Input$	378, 378, 1	0
$Conv2D(filters = 64, stride = 1)$	378, 378, 64	640
$ReLU$	378, 378, 64	0
$Conv2D(filters = 64, stride = 3)$	126, 126, 64	36928
$ReLU$	378, 378, 64	0
$Conv2D(filters = 64, stride = 2)$	126, 126, 64	36928
$ReLU$	378, 378, 64	0
$UpSampling2D(stride = 2)$	126, 126, 64	0
$Conv2D(filters = 1, stride = 1)$	126, 126, 1	577
$Sigmoid$	166, 126, 1	0

more unique way; it creates an output image that is fed into a discriminator. The generator loss is calculated depending on how well it produces data that may be classified as *real* by a discriminator. If the generated image is recognized as a *fake* one, it receives a big penalty in the form of a large loss.

GANs can have many variations, the most common variant utilizes unsupervised or semi–supervised data generation. A great advantage of this kind of network is the ability to create images without direct input. Usually a GAN generator creates output from random values. These values are randomly initiated inside proper *latent space*. However, adversarial networks can also work in fully–supervised way. In this work a GAN is proposed that transforms high–resolution images to low–resolution, with loss calculated in regard to the discriminator's judgements. Such networks are often called picture–to–picture GANs. An algorithm for training such a network is provided in the pseudocode listing 2. One further optimization can be applied to GAN—usually better results are achieved if small noise is applied to the labels. The fitting process of a GAN network can also be visualized in the form of a graph, as presented in the figure 4.4.

The discriminator architecture reassembles a common binary classifier with convolutional and polling layers. A *leaky ReLu (rectified linear unit)* was used as an activation function. After applying a series of convolutions, the image is flattened and then densely connected. A common problem encountered during GAN networks is too rapid fitting of the discriminator. Because it is tasked with a much simpler task than the generator, the discriminator tends to overwhelm its opponent. For this reason, the discriminator is often handicapped in a way. In this case it is missing a large densely connected layer after flatten operation, which would be otherwise typically used in a binary classifier. Moreover, a stride of size

Algorithm 2 GAN training flow

Require: x, y_{gt}

$y_{fake} = generator(x)$
$y_{pred} = discriminator(cat(y_{fake}, y_{gt}))$
$loss_d = lossfn_d(y_{pred}, cat(\mathbb{0}, \mathbb{1}))$
$optimze_d(loss_d)$
$freeze\_learning(discriminator)$
$y_{pred} = discriminator(generator(x))$
$loss_g = lossfn_g(y_{pred}, \mathbb{1})$ {Notice the misleading labels matrix}
$optimize_g(loss_g)$

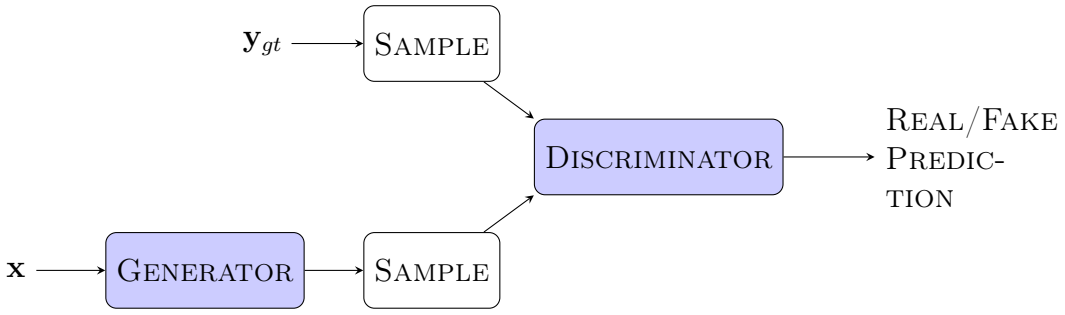


Figure 4.4: Schematic of GAN network inner–workings

Table 4.3: GAN architecture discriminator for data augmentation

Layer type	Output Shape	Number of parameters
$Input$	126, 126, 1	0
$Conv2D(filters = 64, stride = 2)$	63, 63, 64	640
$LeakyReLU$	63, 63, 64	0
$MaxPool2D(stride = 2)$	31, 31, 64	0
$Conv2D(filters = 64, stride = 2)$	16, 16, 64	36928
$LeakyReLU$	16, 16, 64	0
$MaxPool2D(stride = 2)$	8, 8, 64	0
$Flatten$	4096	0
$Dropout(rate = 0.5)$	4096	0
$Dense$	1	4097
$Sigmoid$	1	0

two is combined with a pooling operation to perform more rapid spatial shrinking of the image and put the discriminator at a disadvantage. Furthermore, a dropout layer was used right before the end of the network. This kind of layer drops a part of data to prevent overfitting (and also slow down the learning process). A detailed description of the discriminator part of GAN can be found in the table 4.3. The generator is structured similarly to the previously presented fully–convolutional layers, as it performs a similar task. The full architecture is the same as presented in the table 4.1.

4.3 Training details

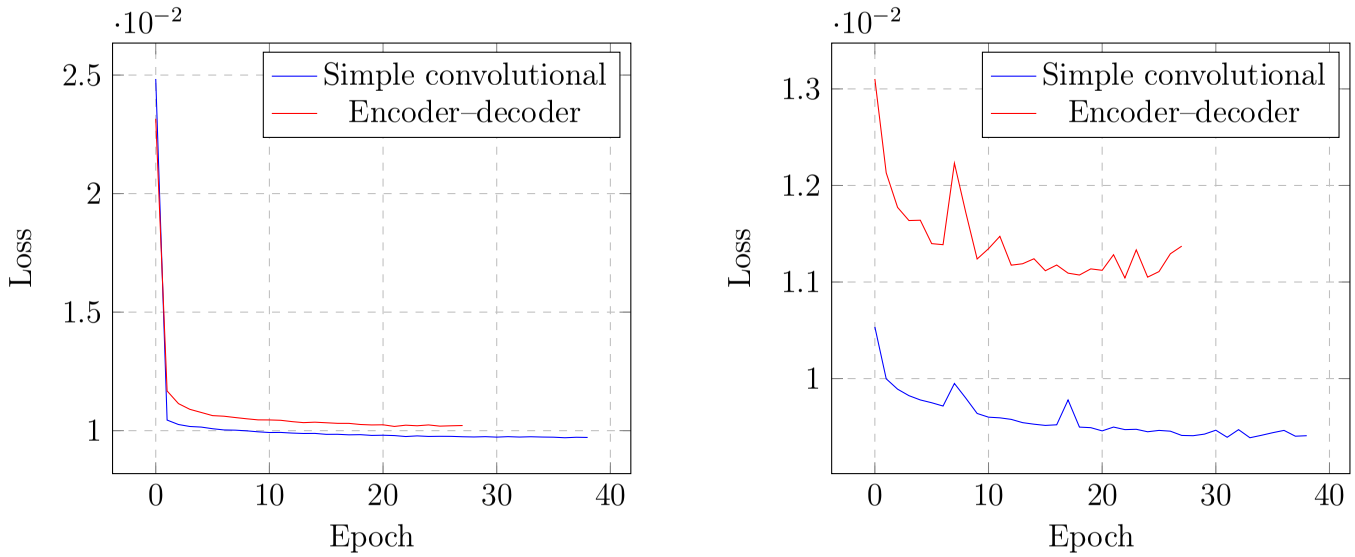
The NIR subset of Proba–V dataset was used to train augmentation networks. As mentioned before, Proba contains a set of binary masks designating areas of low–resolution images that may be invalid. This has been taken into account in the two of the described networks—the simple–fully convolutional network and encoder–decoder. If masks are used during training, the masked regions of low–res images do not participate in loss calculation. This feature works only with per–pixel losses such as MAE and MSE. Because SSIM works in a more structural way, it is not possible to remove some pixels from the evaluation when using this metric.

Both simple and encoder–decoder networks use *ADAM (Adaptive Moment Estimation)* optimizer for gradient descent. This kind of optimization algorithm combines advantages of *momentum* technique and *RMSprop*. The momentum component uses a moving average of gradients from consecutive steps to update

weights, instead of single gradient value, as traditional algorithms would use. This modification helps the gradient descent to avoid slow downs on plateaus in the search space. The RMSprop–alike part of ADAM is responsible for adaptive scaling of learning rate based on value of squared gradients. Alongside ADAM optimizer, MSE was used as the loss function during the final training of the two simpler architectures.

To supervise the training process in an unbiased, way a validation data subset should be used. Validation data is a part of the training set that doesn't take part in the gradient calculation, but is used to calculate loss and metrics every epoch. This way the ongoing training can be evaluated on the data that is not directly used for the fitting process. It is important to create the validation subset out of the training data, not the test dataset. If the test data was used for validation, a data leak from the evaluation step to the training would occur. In the training process of all models 20% of the training dataset was used for the validation. Both of the simpler architectures utilize *early stopping* mechanism. This technique stops training after a designated number of epochs without an improvement in the validation loss value. Early stopping helps to avoid overfitting and provides a more rational stop condition than training for an arbitrary number of epochs. Training history of the simple convolutional and the encoder–decoder network is presented in the figure 4.5. The subfigures show the simultaneous decrease of training and validation loss. The decreasing loss curves reassemble an exponential decay function, which is a sign of proper training.

Training a GAN can pose a significant challenge when stopping condition is taken into consideration. More ordinary networks utilize early stopping mechanism on the validation dataset to limit the number of training epochs, as it was described previously. When training GAN, it is less obvious what should be used as a metric during validation. The losses calculated during training do not explicitly express the quality of the outcome. The generator loss indicates how well it performs relatively to the discriminator's ability to differentiate real and generated images. This doesn't necessarily mean that the generator outputs high quality images. It is better to apply a different metric on evaluation that omits the adversarial part of the network. In this work, validation is based on calculating SSIM value of the generator output and ground truth. This approach is not applicable to all GAN networks because their training scenarios usually lack the possibility of comparing with any reference data. Having an easily readable validation metric is beneficial for network evaluation; however, in this work it was not used to utilize early stopping with the GAN network. Adversarial networks tend to have more unstable and varied trainings than conventional networks. Thus, it was decided to train the GAN with a large fixed number of a hundred epochs. History of training loss of discriminator and generator parts of a the GAN network are presented in



(a) Training loss history

(b) Validation loss history

Figure 4.5: Training history of the simple convolutional and encoder–decoder netowrks

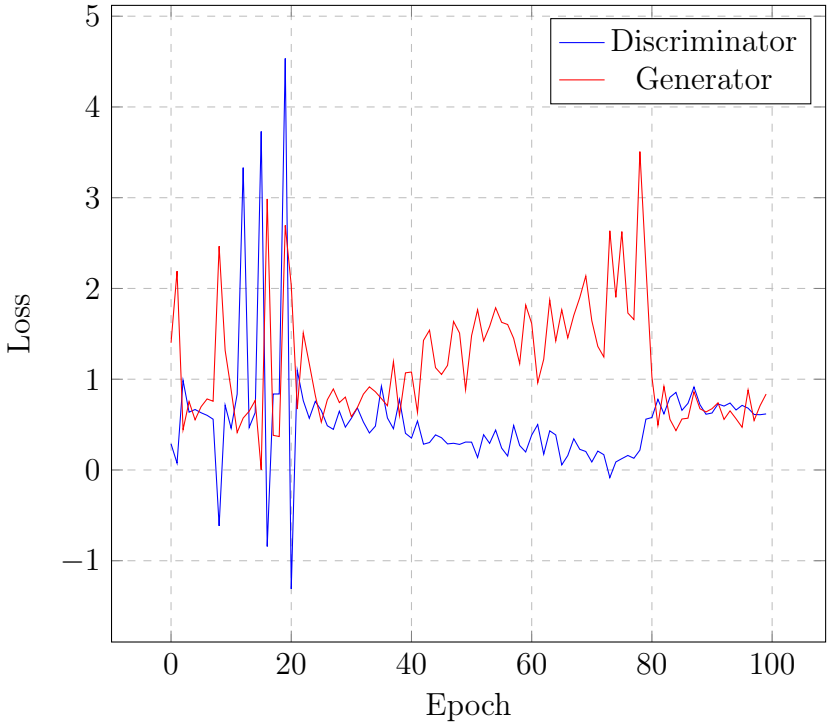


Figure 4.6: Training history of GAN generator and discriminator subnetworks

the figure 4.6. The model from the epoch 94, with the highest SSIM validation score, was chosen as the best one and used to export augmented dataset in the further course of the work.

The contents of the *params.yaml* file used for training the augmentation networks are included in the appendix, in the chapter A.

4.4 Intermediate results

The experiment workflow assumes that an intermediate evaluation step can be performed between training augmentation and super–resolution networks. The Proba–V test dataset can be used to test how well the low–resolution images are recreated by the neural networks. It should be noted that the following results are an intermediate step to sanity–check if augmented pictures are reconstructed correctly. At this step, the results do not guarantee the quality of super–resolution training. To outline a fuller picture the results can be compared with common resizing algorithms. The results are presented in the table 4.4. The results can also be examined visually by displaying images side–by–side. Figure 4.7 presents an example of image shrinking using various methods, figure 4.8 presents an en-

Table 4.4: Intermediate results of evaluation on Proba–V test dataset (SSIM metric)

	Real.	Simple conv	Encoder–decoder	GAN	Nearest	Bilinear	Bicubic	Lanczos
Real	1.0	0.946	0.935	0.923	0.887	0.947	0.945	0.94
Simple conv	0.946	1.0	0.987	0.963	0.938	0.995	0.996	0.992
Encoder–decoder	0.935	0.987	1.0	0.958	0.91	0.982	0.982	0.978
GAN	0.923	0.963	0.958	1.0	0.892	0.967	0.966	0.962
Nearest	0.887	0.938	0.91	0.892	1.0	0.937	0.949	0.949
Bilinear	0.947	0.955	0.982	0.967	0.937	1.0	0.996	0.989
Bicubic	0.945	0.996	0.982	0.966	0.949	0.996	1.0	0.998
Lanczos	0.94	0.992	0.978	0.962	0.949	0.989	0.998	1.0

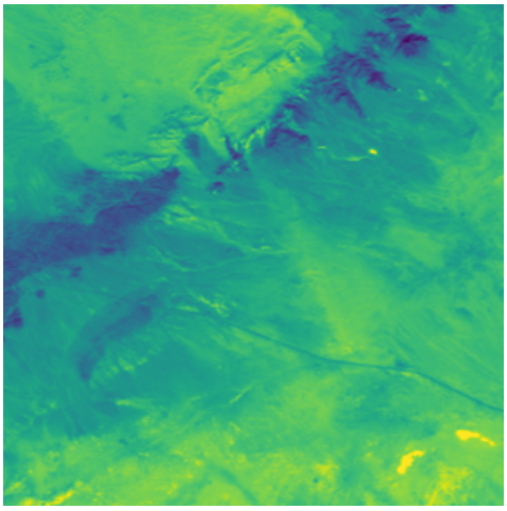
larged part of the same image. The visual comparison has been created using the simple convolutional augmentation network. The perceptible results for other architectures look very similar, for this reason they are not included in the figure.

The intermediate results indicate that the augmentation process works reasonably well and can be used for creating new datasets. The numeric metrics look fairly similar for all given methods, their usefulness is to be determined during super–resolution training and evaluation. Visual results also look well, low–resolution images resemble high–resolution ones.

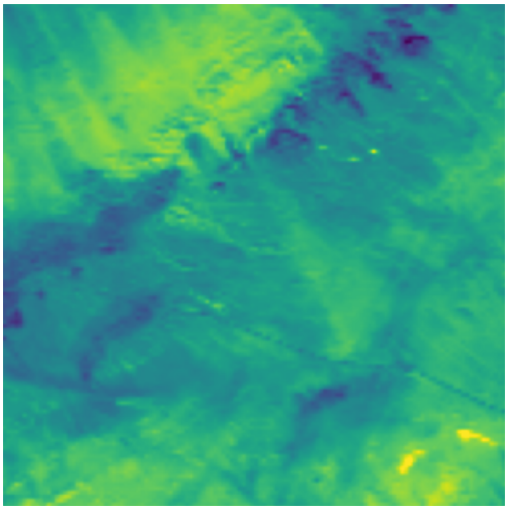
4.5 Implementation details

As stated before, all networks were implemented in *Python* using *Tensorflow 2* library. The built–in Keras interface offers several ways of building models; the most flexible one is called *model subclassing*. This approach enables using binary masks during training. Model subclassing involves inheritance to implement custom training loops for Tensorflow models. The simple convolutional and endcoder–decoder networks utilize this kind of model instantiation. The GAN model uses a mixture of subclassing and *Sequential API* which defines the model in a declarative way. The two components of GAN—the generator and the discriminator are instantiated using the sequential approach, then they are combined into one using model subclassing.

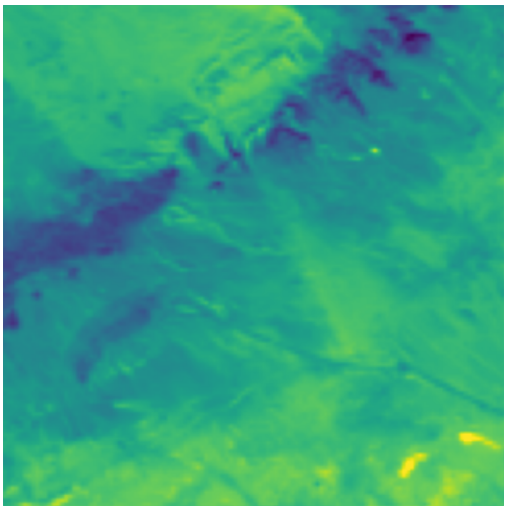
Tensorboard is a package accompanying Tensorflow that provides a convenient interface for training supervision. It runs as a web server that displays a web page with the live progress of training in the form of interactive plots. Tensorboard data is created using Keras' callbacks. Callbacks are functions hooks called during specific steps of training. They can provide features like previously discussed early stopping. *Model checkpoint* is another useful callback that saves the model at the end of the training epoch. Tensorflow provides an interface for creating custom callbacks. An inference preview mechanism on validation image at the end of each epoch was implemented as a custom callback. This mechanism is very convenient when training the GAN network. Keras also enables writing custom losses and metrics. This possibility has been used to write training losses based on PSNR and SSIM. Both of them are based on internal Tensorflow implementations that were adjusted to serve as loss functions.



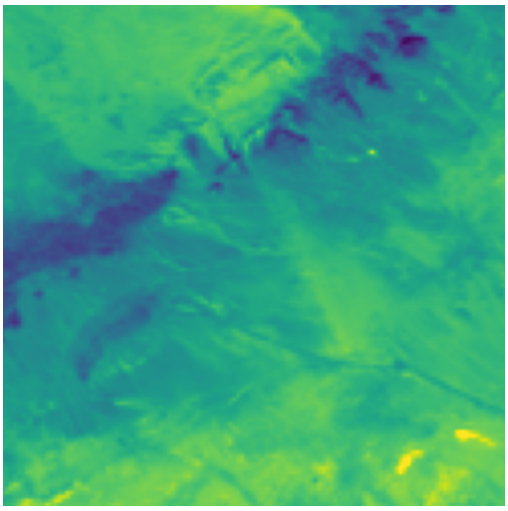
(a) High–resolution



(b) Low–resolution

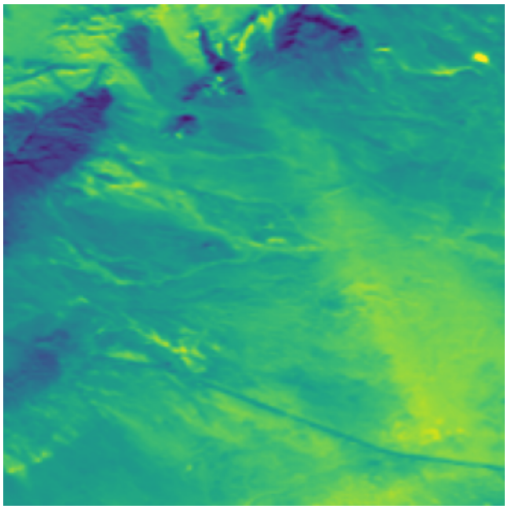


(c) Low–resolution augmented

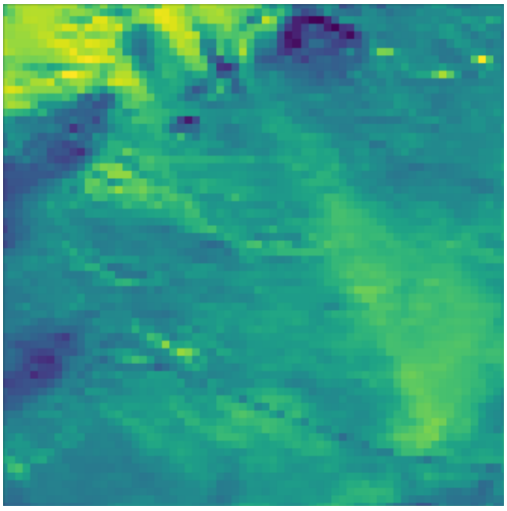


(d) Low resolution resized with bicubic

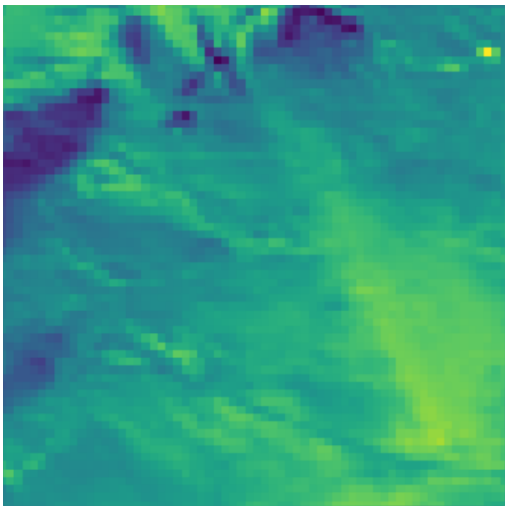
Figure 4.7: Intermediate results of evaluation on Proba–V test dataset for simple convolutional augmentation network



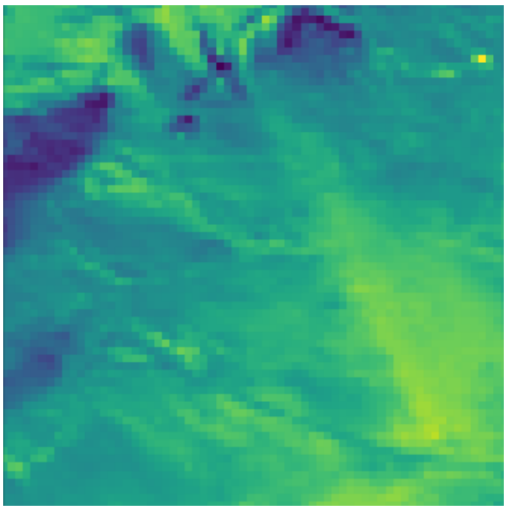
(a) High–resolution



(b) Low–resolution



(c) Low–resolution augmented



(d) Low resolution resized with bicubic

Figure 4.8: Zoomed intermediate results of evaluation on Proba–V test dataset for simple convolutional augmentation network

Chapter 5

Super–resolution training and evaluation

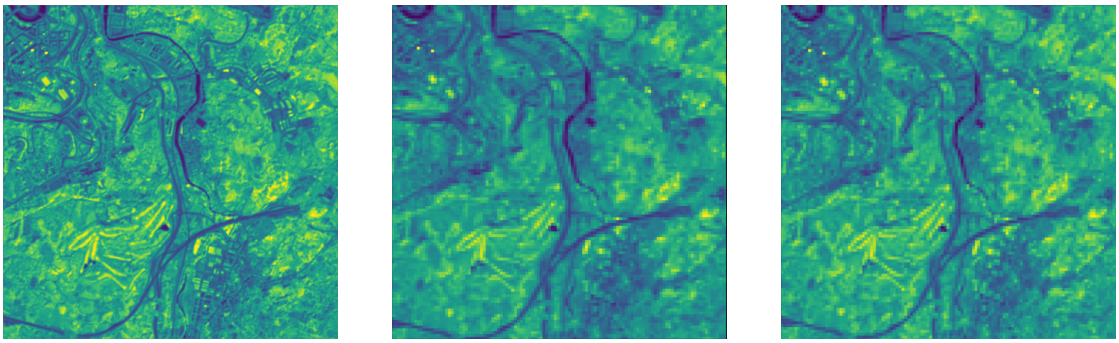
5.1 Training HighRes–net

As stated in the experiment layout, the final step consists in training HighRes–net super–resolution model with different data. One final step before conducting the training is to examine visually the results of augmentation on the Sentinel–2 image. These are presented for the simple convolutional augmentation network architecture in the figure 5.1 and 5.2.

As planned, the super–resolution is to be trained with multi–image data in a single–band (eight band) mode. Training is configured in such a way that only weights for the best validation score are saved. However, automatic stopping is not enabled, for this reason fitting was interrupted manually around epoch one hundred when all training curves plateaued. Training history has been plotted to visualize the loss fitting progress.

5.1.1 Cross validation as stop condition

Having multiple datasets generated in different ways enables an alternative way of validating the training process. Instead of taking a subpart of the training set as validation data, one may use images generated in a different way. Such a technique may help to investigate the generalization capabilities of the super–resolution network. The datasets created using the simple convolutional network and resizing algorithm were used for the cross validation. In each of them, one set was used for fitting and the other was used for validation. This time the increasing validation loss indicates overfitting in early epochs, compared to previous trainings. This may indicate that the vast part of the fitting process does not contribute

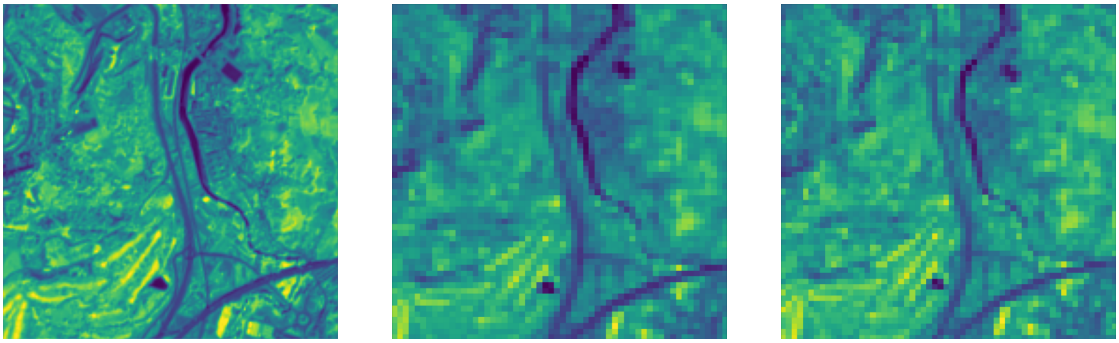


(a) High–resolution

(b) Low–resolution augmented

(c) Low resolution resized with bicubic

Figure 5.1: Example of Sentinel–2 training data export with simple convolutional augmentation network



(a) High–resolution

(b) Low–resolution augmented

(c) Low resolution resized with bicubic

Figure 5.2: Zoomed example of Sentinel–2 training data export with simple convolutional augmentation network

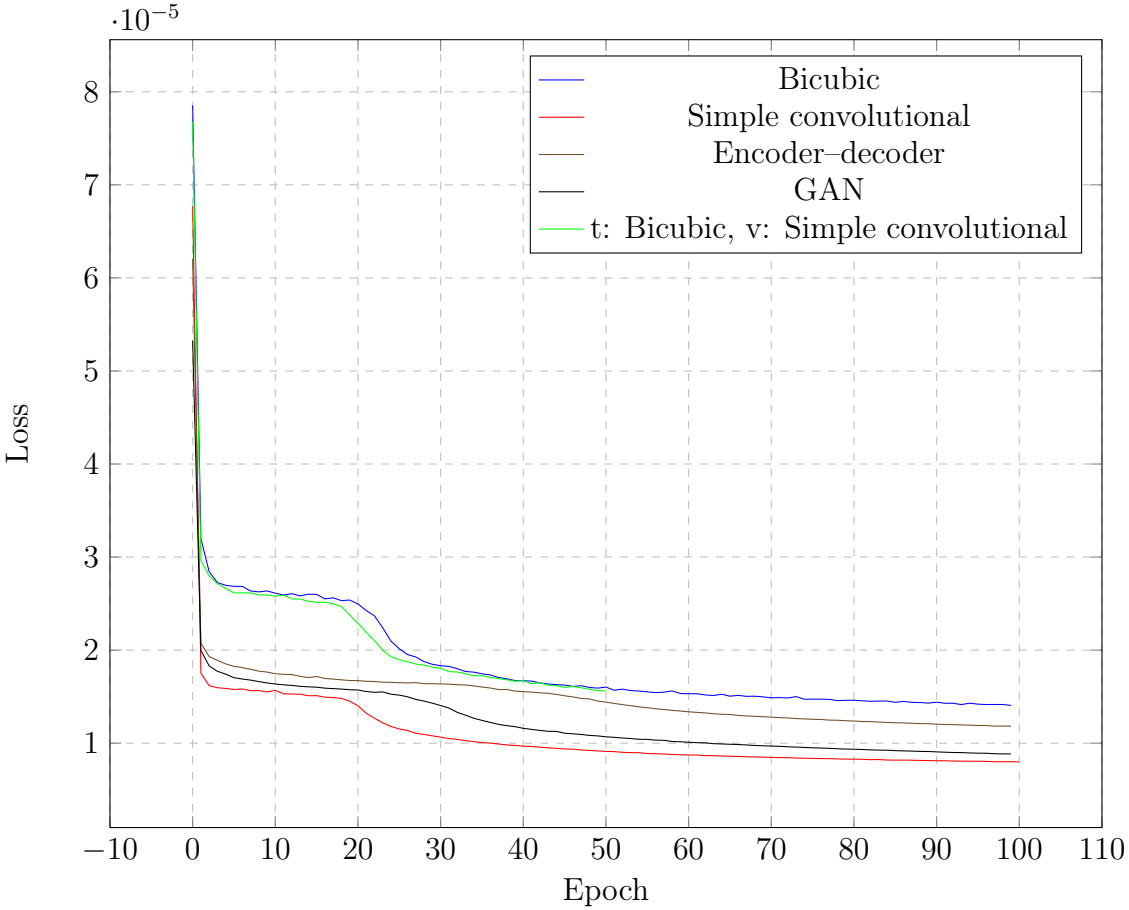


Figure 5.3: HighRes–net training loss history

to overall generalization capability of the super–resolution network. The cross–validated trainings were stopped earlier; since only weights for the epoch with the best validation score are saved, it is pointless to run longer fittings. Figure 5.3 shows the loss history for all discussed HighRes–net trainings, figure 5.4 presents validation loss improvement.

5.2 Evaluation and results

According to the experiment layout, the trained super–resolution models are to be evaluated on a variety of test datasets to examine generalization capabilities and robustness. As stated before, the evaluation datasets include:

- Test subsets from all augmented Sentinel–2 datasets that can be used for calculating numerical metrics.

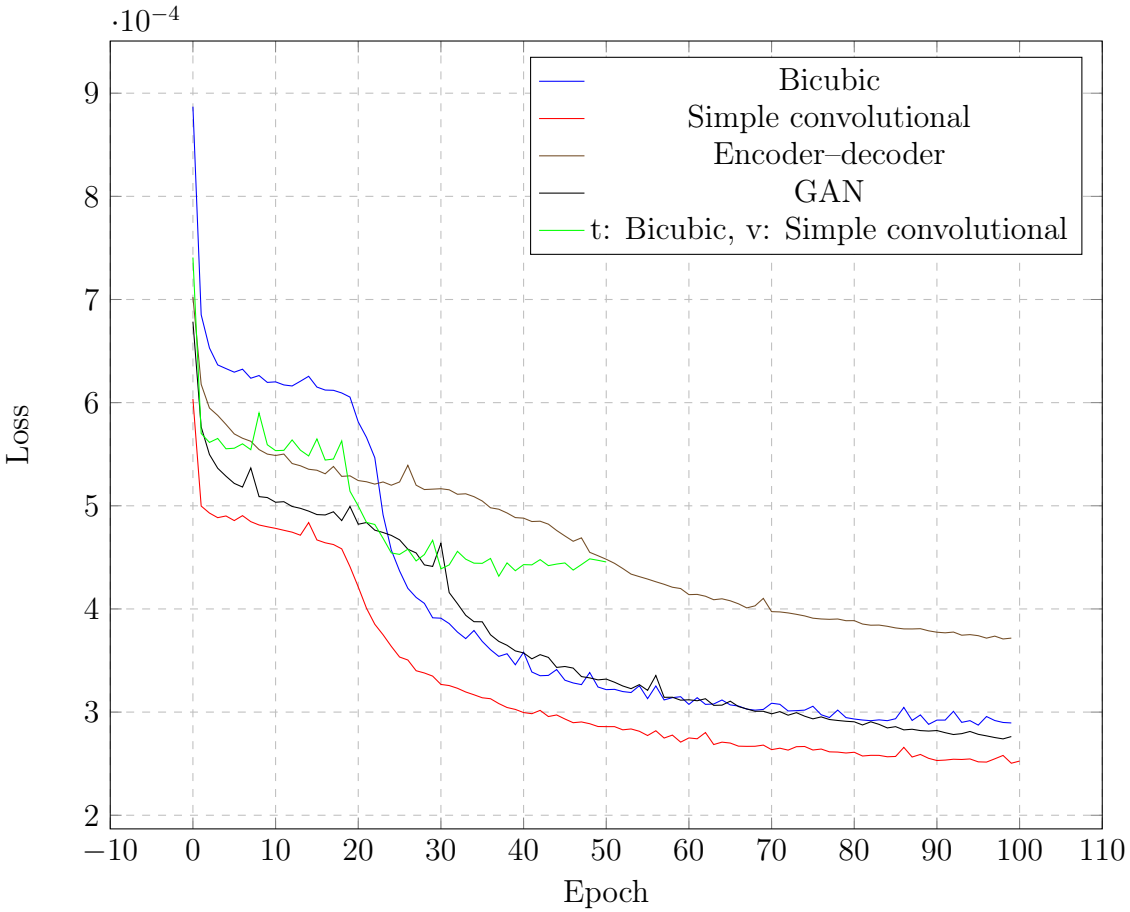


Figure 5.4: HighRes–net validation loss history

- Real–life Sentinel–2 images that were not used in the training process. These do not include high and low–resolution pairs, so only visual examination can be performed.

- Proba–V test dataset that can be used for calculating numerical metrics.

It should be noticed that the later two test sets differ in the GSD parameter from the synthetic low–resolution images in the Sentinel–2 training datasets. This data is still valid for evaluation and investigation because it is desirable that super–resolution, and machine learning algorithms in general, work on objects of any scale and size. Limiting tests to single GSD would draw an incomplete picture of the results. The results of evaluation are presented in the table 5.1, where rows designate test datasets and columns indicate models trained on data created in different ways. For the cross–validation training scenarios, letters t and v indicate training and validation datasets respectively. The $cPSNR$ score was used as the super–resolution evaluation metric.

The best results are observed on the diagonal of the table, which means that the algorithm works best on test data created in the same way as training data for a given model. This result is expected and indicates that no gross mistakes were made in the course of the work. As stated before, the results on the Sentinel–2 real–life image were evaluated visually by observing artifacts in the images. The figure 5.5 shows artifacts presence on test image for networks trained on augmented data with and without cross–validation. The artifacts take form of a mosaic–like overlay or checkerboard effect distorting the image, especially in high frequency areas. The cross–validated model that was stopped in the earlier epoch shows less visible artifacts. The same observation can be made looking at the figure 5.6, which shows the same phenomenon for the training set created using bicubic interpolation. These two observations are marked in the results table 5.1.

The results in the table 5.1 can be summarized in the following statements:

- Data created with augmentation techniques, both traditional and deep learning–based is suitable for training super–resolution networks.

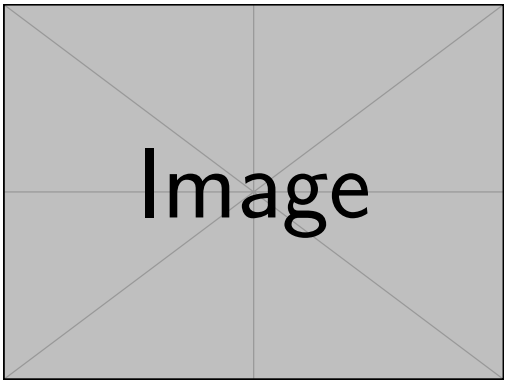
- The different deep learning architectures give fairly similar results, although the simplest one works best.

- At the moment, the bicubic interpolation gives slightly better results, possible reasons for that and suggestions of improvement are included in the summary.

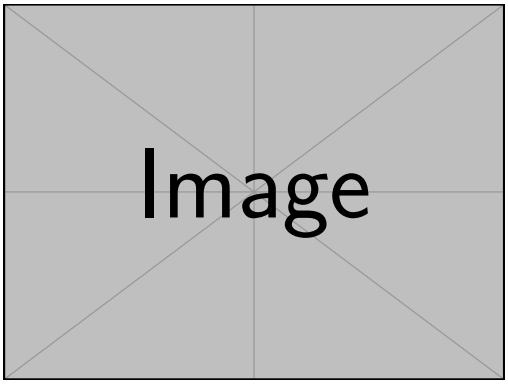
- The cross–validation proves that all training are prone to overfitting to the data generation techniques. Early stopping based on different datasets can leads to this conclusion.

Table 5.1: Evaluation of super–resolution training on different test sets

cPSNR	Bicubic	Simple conv	Encoder–decoder	GAN	t: Bicubic v: Simple convolutional
Bicubic	35.78	28.96	24.55	25.80	35.26
Simple convolutional	33.69	36.15	27.16	29.83	33.67
Encoder–decoder	31.72	31.78	34.43	30.22	31.76
GAN	28.63	28.63	27.27	35.72	30.28
Artifacts on real Sentinel–2	yes	yes	yes	yes	no
Proba–V	43.36	42.02	40.43	40.91	43.51

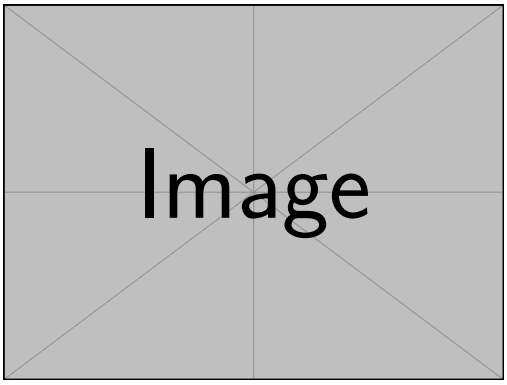


(a) Trained without cross–validation

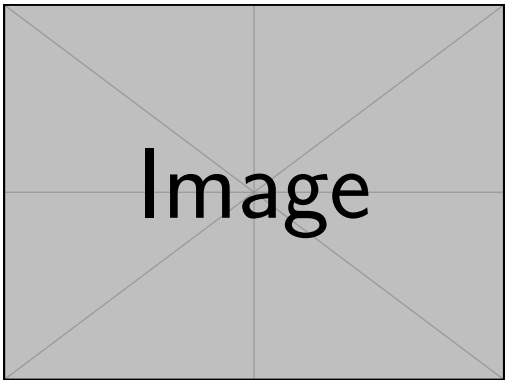


(b) Trained with cross–validation on data created by bicubic interpolation

Figure 5.5: Artifacts presence on Sentinel real–life data for network trained on dataset augmented with simple convolutional network



(a) Trained without cross–validation



(b) Trained with cross–validation on data augmented with simple convolutional network

Figure 5.6: Artifacts presence on Sentinel real–life data for network trained on dataset created by bicubic interpolation

Chapter 6

Summary

In the course of the work various approaches to data augmentation were explored. Three different deep learning architectures were introduced and compared with the traditional approach. The augmentation networks were trained on real–life data and used to export new datasets to fit the super–resolution algorithm. Then the super–resolution models were evaluated on various synthetic and real–file images; both in a numerical and visual way. The cross–validation based approach to training supervision confirmed to be valuable. The deep learning based–methods proved to be of utility, yet slightly below what would be expected since they didn't surpass the bicubic interpolation approach. There may be a variety of reasons for that to explore. Many possibilities were not explored in a full way; translations between low–resolution images may be investigated in a greater detail. Modeling them after distribution extracted from a real–world dataset may lead to enhanced results. The semi–simulated approach for data generation was not included in the scope of the work; however, it would be interesting to investigate this option in the future.

The most novel observation in the work is connected with the cross–validation technique during training. Early stopping training based on this approach leads to good results. This indicates a greater overfitting problem in regard to the data generation method. The robust data augmentation problem remains open, however valuable observations about the issue of super–resolution generalization has been made.

There is a value in the opportunity for further development of the work and more detailed investigation. Achieving robustness and greater generalization of super–resolution techniques through better data augmentation is an important step towards productization of this technique. A super–resolution algorithm independent of satellite data type would be useful in the image processing pipeline during aerospace missions, remote sensing tasks and Earth observations.

Appendices

Appendix A

Model and training parameters of the augmentation networks

As stated in the work the models and training process are parametrized by set o variables in a *params.yaml* file. This file is supervised by Git and Dvc to ensure reproducibility and artifacts caching. Contents of the configuration file during the training of augmentation networks are presented in the listing 1.

```yaml
# WARNING: Some options are mutually exclusive.
# List of incompatible options:
# 'metrics.use_lr_masks: true' and metrics: 'metrics.loss: SSIM'.

SIMPLE_CONV:
  load:
    # Supported datasets: 'NIR', 'RED'
    dataset: NIR
    # Must be divisible by three.
    input_shape:
      - 378
      - 378
      - 1
    # Supported options: int or 'null'
    limit_per_scene: null
    batch_size: 8
    validation_split: 0.8
    preprocess:
      equalize_hist: true
      artificial_lr: false
      # Supported modes: 'NEAREST', 'BILINEAR', 'BICUBIC', 'LANCZOS'
```

```
      interpolation_mode: BICUBIC

  train:
    lr: 0.0001
    epochs: 100
    # Supported losses: 'MAE', 'MSE', 'SSIM'
    loss: MSE
    use_lr_masks: true

    callbacks:
      tensorboard: true
      modelcheckpoint: true
      save_best_only: true
      earlystopping: true
      stopping_delta: 0.0
      stopping_patience: 5
      preview: true

ENCODERDECODER:
  load:
    # Supported datasets: 'NIR', 'RED'
    dataset: NIR
    # Must be divisible by three.
    input_shape:
      - 378
      - 378
      - 1
    # Supported options: int or 'null'
    limit_per_scene: null
    batch_size: 8
    validation_split: 0.8
    preprocess:
      equalize_hist: true
      artificial_lr: false
      # Supported modes: 'NEAREST', 'BILINEAR', 'BICUBIC', 'LANCZOS'
      interpolation_mode: BICUBIC

  train:
    lr: 0.0001
    epochs: 100
```

```
    # Supported losses: 'MAE', 'MSE', 'SSIM'
    loss: MSE
    use_lr_masks: true

    callbacks:
      tensorboard: true
      modelcheckpoint: true
      save_best_only: true
      earlystopping: true
      stopping_delta: 0.0
      stopping_patience: 5
      preview: true

GAN:
  load:
    # Supported datasets: 'NIR', 'RED'
    dataset: NIR
    # Must be divisible by three.
    input_shape:
      - 378
      - 378
      - 1
    # Supported options: int or 'null'
    limit_per_scene: null
    batch_size: 8
    validation_split: 0.8
    preprocess:
      equalize_hist: true
      artificial_lr: false
      # Supported modes: 'NEAREST', 'BILINEAR', 'BICUBIC', 'LANCZOS'
      interpolation_mode: BICUBIC

  train:
    lr:
      discriminator: 0.0001
      generator: 0.0001
    epochs: 100
    loss: BINARY_CROSSENTROPY
    use_lr_masks: false
```

```yaml
callbacks:
  tensorboard: true
  modelcheckpoint: true
  save_best_only: false
  # Earlystopping is currently not supported for GAN
  earlystopping: false
  stopping_delta: 0.0
  stopping_patience: 0
  preview: true
```

Listing 1: Contents of the *params.yaml* file used for training

List of Figures

1.1 A demonstration of super–resolution technique, where nine low–resolution satellite images of a farming area are turned into one high–resolution image 3
1.2 A demonstration of super–resolution technique, where nine low–resolution satellite images of a barren landscape are turned into one high–resolution image 3

2.1 Schematic of encoder–decoder mechanism 7
2.2 Schematic of inference in *HighRes–net* [4] 10

3.1 Graph of various training data generation techniques 14
3.2 Sample image pair from *Proba–V NIR train dataset* 17
3.3 Proba–V sample with invalid area and its binary mask 18
3.4 Sample image from *Sentinel–2 dataset* (eight band) 18
3.5 Graph of the experiment flow, solid lines indicate trainings, dashed arrows designate evaluations 20

4.1 Sample image pair from *Proba–V NIR train dataset* 26
4.2 Exposure distributions in the RED Proba–V dataset 27
4.3 Exposure distributions in the RED Proba–V dataset 27
4.4 Schematic of GAN network inner–workings 30
4.5 Training history of the simple convolutional and encoder–decoder netowrks 33
4.6 Training history of GAN generator and discriminator subnetworks . 34
4.7 Intermediate results of evaluation on Proba–V test dataset for simple convolutional augmentation network 37
4.8 Zoomed intermediate results of evaluation on Proba–V test dataset for simple convolutional augmentation network 38

5.1 Example of Sentinel–2 training data export with simple convolutional augmentation network 40

5.2 Zoomed example of Sentinel–2 training data export with simple convolutional augmentation network 40
5.3 HighRes–net training loss history 41
5.4 HighRes–net validation loss history 42
5.5 Artifacts presence on Sentinel real–life data for network trained on dataset augmented with simple convolutional network 45
5.6 Artifacts presence on Sentinel real–life data for network trained on dataset created by bicubic interpolation 45

List of Tables

4.1 Simple fully–convolutional network architecture for data augmentation 28
4.2 Encoder–decoder network architecture for data augmentation 29
4.3 GAN architecture discriminator for data augmentation 31
4.4 Intermediate results of evaluation on Proba–V test dataset (SSIM metric) 35

5.1 Evaluation of super–resolution training on different test sets 44

List of Algorithms

1 Approach to generating fully simulated mulit–image datasets 15
2 GAN training flow 30

List of Listings

1 Contents of the *params.yaml* file used for training 54

Bibliography

[1] European Space Agency. Earth Online (official web resources about Proba–V mission and data). `https://earth.esa.int/eogateway/missions/proba-v`. accessed: 25.07.2021.

[2] European Space Agency. Sentinel Online (official web resources about Sentinel-2 mission and data. `https://sentinel.esa.int/web/sentinel/missions/sentinel-2`. accessed: 25.07.2021.

[3] Adrian Bulat, Jing Yang, Georgios Tzimiropoulos. To learn image super-resolution, use a GAN to learn how to do image degradation first. *CoRR*, abs/1807.11458, 2018.

[4] Michel Deudon, Alfredo Kalaitzis, Israel Goytom, Md Rifat Arefin, Zhichao Lin, Kris Sankaran, Vincent Michalski, Samira Ebrahimi Kahou, Julien Cornebise, Yoshua Bengio. Highres-net: Recursive fusion for multi-frame super-resolution of satellite imagery. *CoRR*, abs/2002.06460, 2020.

[5] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, Y. Bengio. Generative adversarial networks. *Advances in Neural Information Processing Systems*, 3, 06 2014.

[6] Manuel Guizar-Sicairos, Samuel T. Thurman, James R. Fienup. Efficient subpixel image registration algorithms. *Opt. Lett.*, 33(2):156–158, Jan 2008.

[7] Marcus Märtens, Dario Izzo, Andrej Krzic, Daniël Cox. Super-resolution of PROBA-V images using convolutional neural networks. *CoRR*, abs/1907.01821, 2019.

[8] Augustus Odena, Vincent Dumoulin, Chris Olah. Deconvolution and checkerboard artifacts. *Distill*, 2016.

[9] Olaf Ronneberger, Philipp Fischer, Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

[10] Francesco Salvetti, Vittorio Mazzia, Aleem Khaliq, Marcello Chiaberge. Multi-image super resolution of remotely sensed images using residual attention deep neural networks. *Remote Sensing*, 12(14), 2020.

[11] Zhou Wang, A.C. Bovik, H.R. Sheikh, E.P. Simoncelli. Image quality assessment: from error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4):600–612, 2004.

[12] Papers with Code. Multi–Frame super–resolution ranking leaderboard on Proba–V dataset. `https://paperswithcode.com/sota/multi-frame-super-resolution-on-proba-v`. accessed: 25.07.2021.

[13] Zhaoyi Yan, Xiaoming Li, Mu Li, Wangmeng Zuo, Shiguang Shan. Shift-net: Image inpainting via deep feature rearrangement. *CoRR*, abs/1801.09392, 2018.