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: inż. Maciej Ziaja

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

Gliwice, wrzesień 2021

Abstract

This work aims 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. The size and quality of the dataset used to train super-resolution networks are crucial to achieving the 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. In the field of super-resolution, the data creation process has been traditionally done using resampling techniques; however, this work aims to use deep learning to generate the datasets for training super-resolution algorithms. The following chapters provide an overview of modern super-resolution solutions and a proposal of a set of deep learning algorithms to generate the training datasets. The results of the work are evaluated by testing super-resolution networks that were trained on the datasets created during the project.

Keywords: super-resolution, deep learning, data augmentation

Contents

Chapter 1

Introduction

1.1 Super-resolution technology

Super-resolution is a group of techniques that upscale images and improve their quality. A super-resolution algorithm can be treated as a function that takes an image and returns it with a resolution n times larger [**?**]. Algorithms that were taken into account in the process usually upscale images two or four times. In this study, due to the size of images in the available datasets (Proba-V [**?**] and Sentinel-2 [**?**], more on the utilized data can be found in Chapter 3), a ratio of three was used.

It is important to distinguish between super-resolution and traditional upscaling algorithms. The latter use interpolation to enlarge pictures; however, they hardly improve the image quality. The intent of super-resolution is not only to upscale images but to improve the quality and details. 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 include convolutional layers. Such layers contain a number of image filters whose weights are tuned during the training process. Like all the rest of the 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 imagined approximation of image features. It is important to keep the statistical nature of machine learning algorithms in mind.

Two main kinds of super-resolution algorithms can be outlined: *one-to-one* and *many-to-one*. The first one is the obvious approach where a single low-resolution image is translated into a high-resolution picture. The second method is to combine multiple low-resolution images into a single high-resolution one. The first technique is often called *single-image* and the latter *multi-image* super-resolution.

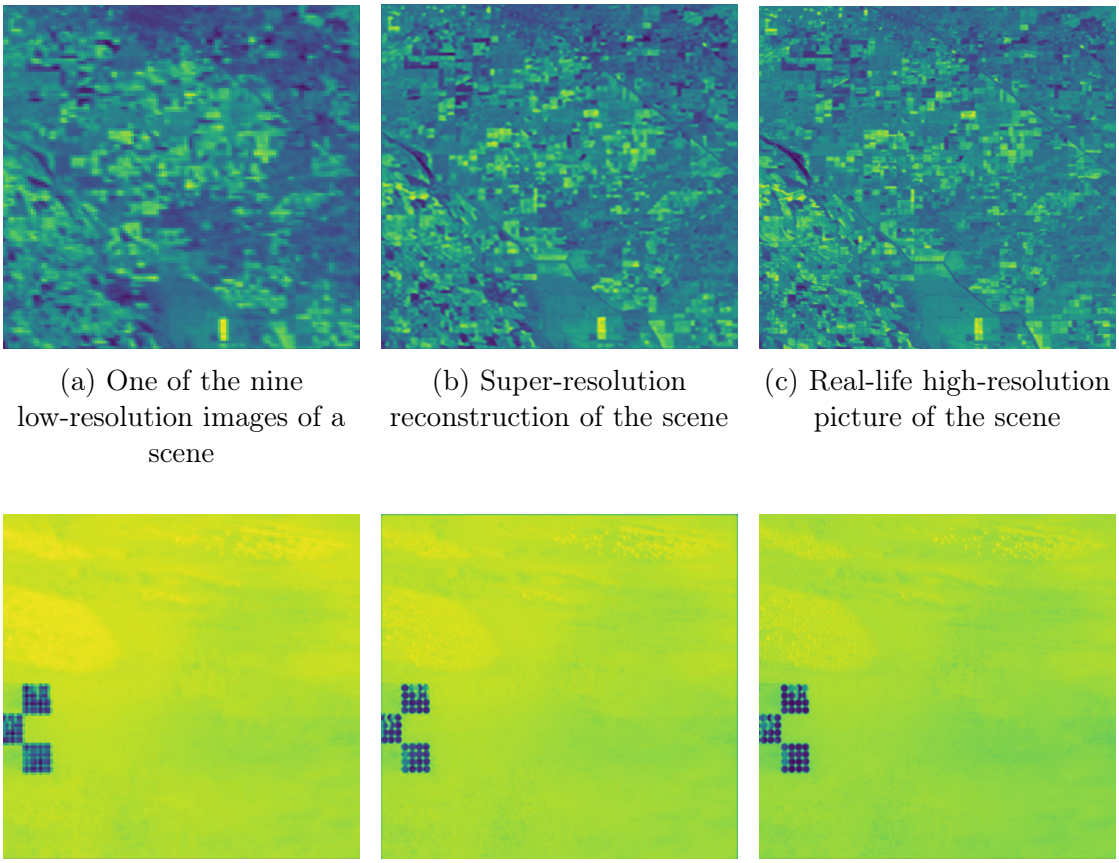
Super-resolution is a technique relevant in satellite imaging, remote sensing, and geoscience. This work focuses on the application of super-resolution technology in these fields. The most common cause for image enhancement is aesthetics. This application is viable in satellite imaging; however, super-resolution has other practical advantages. Image enhancing techniques can be used as a preprocessing step in remote sensing pipelines. For this reason, super-resolution may be especially useful when considered in the context of satellite imagery. Multi-image super-resolution in satellite applications can be treated as a data fusion operation. A visual demonstration of modern super-resolution applied on satellite images is presented in Figure 1.1. In these examples, a set of nine low-resolution images per scene was super-resolved into a single high-resolution one. A sample of those nine images is shown in the comparison.

Deep-learning based image-enhancing algorithms may be domain specific. Different model architectures and data are used to create super-resolution algorithms for satellite imagery and human faces.

1.2 Purpose of data augmentation and available solutions

Data augmentation is a suite of techniques that enhance the size and quality of datasets used for training machine learning models [**?**] [**?**]. Deep learning, utilized in the modern super-resolution solutions, requires a lot of data to train successfully. An increase in the quality and size of a dataset can lead to far better results when training a neural network. This is why data augmentation techniques are often used to improve the performance of deep networks. Data augmentation incorporates various transformations to improve, multiply, or generate training data. This process often multiplies data by modifying existing examples; however, more advanced generation techniques can produce fully artificial datasets. Completely new data created in the augmentation process is often called *synthetic* [**?**]. Classical augmentation techniques utilize simple operations like image resizing, flipping, mirroring, recoloring, etc. However, a different approach can be taken to generate data. It is possible to train deep neural networks to perform data augmentation transformations. This approach can be especially useful when the available dataset is too small to train the desired network. The small dataset can be used to fit a simpler network that is taught to create new data, based on the existing training set. Then, this small 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.

Generating more diverse training datasets is often used as a method of *reg-*



(a) One of the nine low-resolution images of a scene

(b) Super-resolution reconstruction of the scene

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

(d) One of the nine low-resolution images of a scene

(e) Super-resolution reconstruction of the scene

(f) Real-life high-resolution picture of the scene

Figure 1.1: A demonstration of super-resolution technique, where nine low-resolution satellite images of different landscapes are turned into one high-resolution image; 1.1a, 1.1b, and 1.1c show pictures of a rural area; 1.1d, 1.1e, and 1.1f present pictures of a barren landscape with silos (these examples comes from the HighRes-net model applied on the Proba-V dataset) [**?**]

ularization. Regularization methods prevent overfitting machine learning models to the training datasets which leads to better generalization capabilities [**?**]. The problem of overfitting and the role of data augmentation in preventing it are covered in more detail in Sections 2.2.1 and 2.2.6.

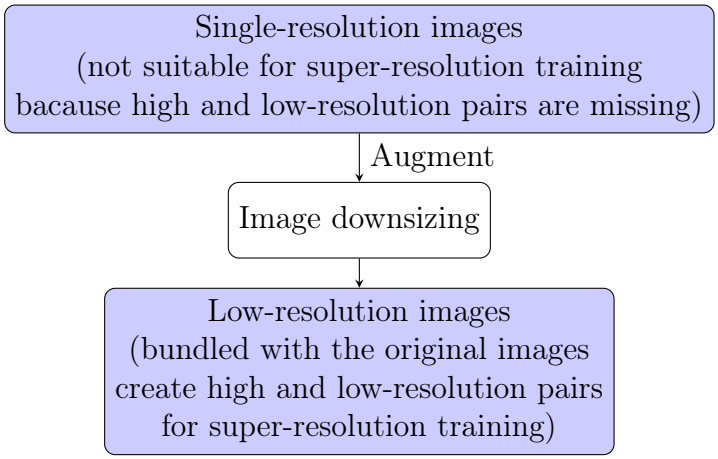
1.3 Aim of the work and motivation

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

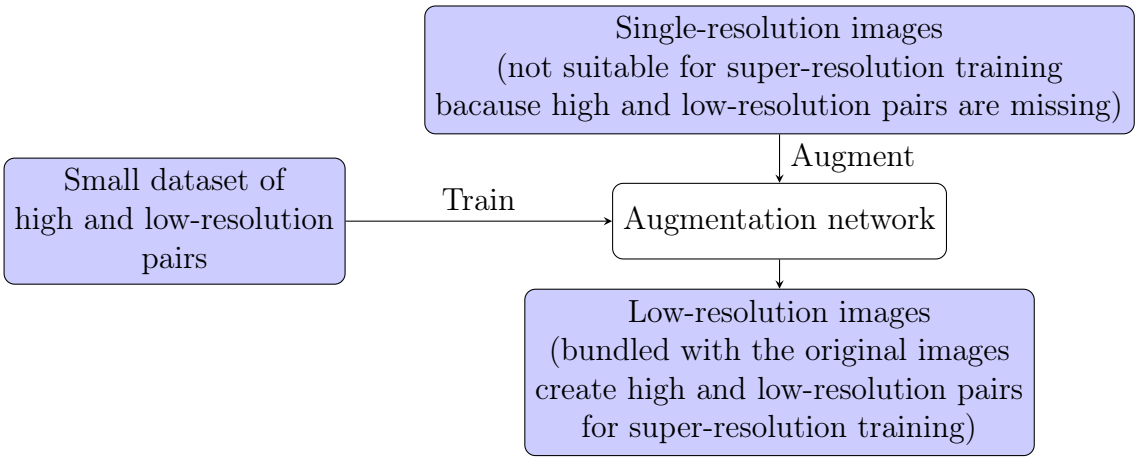
The nature of super-resolution technology and satellite imagery imposes certain ways in which augmentation should be applied to the training data. Single image super-resolution is trained using pairs of high and low-resolution images. On the other hand, multi-image super-resolution learns from examples of one high-resolution image bounded with multiple low-resolution ones. These requirements render compiling training sets a challenge, especially in the field of satellite imagery. Even though single satellite images are widely available, it is difficult to find collections containing pairs of the same scenes with different resolutions.

Data augmentation and generation techniques may be the solution to the scarcity of satellite data suitable for super-resolution trainings. Pairs of high and low-resolution images can be created from single-image satellite datasets by resizing existing photographs. 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-neighbor, Lanczos). The network trained on alike datasets will likely learn to invert given interpolation methods. This may not match exactly real-life scenarios where images are not created using resizing algorithms.

However, another method of data augmentation based on deep learning is possible. Similar to the super-resolution trainings, it requires pairs of real low and high-resolution images of the same scene, taken by cameras of different quality. The main idea of this project is to use a dataset of real-life data to train an augmentation network. Such a network would learn to create low-resolution images for high-resolution ones, without imprinting resampling algorithms mechanisms into



(a) Super-resolution data augmentation with traditional resizing algorithms



(b) Super-resolution data augmentation with deep-learning

the data. The relation between low and high-resolution images in this augmented dataset would resemble the relation between the same image taken by cameras of different quality. The deep learning approach to data augmentation has proven beneficial in other fields of super-resolution imaging [**?**].

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. For clarity, the different methods of data augmentation are illustrated in Figures 1.2a and 1.2b. Each of these starts with a dataset of single-resolution images. To train a super-resolution algorithm a set of high and low-resolution pairs is needed. Both of these methods create low-resolution counterparts for ex-

isting images to generate the training pairs. The approach based on traditional resizing algorithms creates missing low-resolution pictures by downsizing the existing ones. However, with deep learning a set of high and low-resolution images is used to train the augmentation model. This model learns to downsize existing images. In a sense, the augmentation network learns to perform a task opposite to the super-resolution algorithm. Then the augmentation model can be used to shrink existing images and create low-resolution images to be eventually used in the super-resolution training.

1.4 Content outline

The consecutive chapters provide a description of the theoretical background of regarded topics, implementation of data augmentation networks, and final evaluation. The detailed content of the chapters is as follows:

Chapter 1 (this chapter) broadly introduces topics of super-resolution and data augmentation. The main goal of this work is set.

Chapter 2 provides an overview of satellite imagery, explains topics of deep learning, super-resolution, augmentation, and image interpolation methods.

Chapter 3 explains the specific goals of the work. This chapter lays out the experiment plan and provides a rationale for chosen methods and tools.

Chapter 4 presents the main part of the work, contains data augmentation neural network architectures description.

Chapter 5 includes the results of super-resolution training and evaluation of different datasets.

Chapter 6 summarizes the work and provides a commentary on the results.

Chapter 2

Analysis of the related work

This chapter provides an overview and analysis of the related works. Subsequent sections introduce characteristics of satellite imagery, discuss topics of deep learning and super-resolution. The following part of the work provides the theoretical background for techniques used in the latter Chapters 3, 4, and 5.

2.1 Characteristics of satellite imagery

This work centers around super-resolution techniques in the sphere of satellite imagery. Pictures taken from aerospace devices differ substantially from normal photography. Multi-image observation is usually favored over single-image (in terms of possible quality of outcome). Satellites often take a series of photos of a single scene. This puts emphasis on the multi-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 installed on satellites often acquire photos in a very wide spectrum that may not include frequencies of visible light. Spectral bands in satellite imagery can contain wavelengths such as infrared, near-infrared, panchromatic[1], radio frequencies, and more. This specific kind of image with a 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 have the same width and sample adjacent parts of the spectral range. The *multi-spectral* devices take pictures in multiple different bands of different wavelengths, which often do not border each other. Multi-spectral images can be treated as discrete sampling points of the

[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.

spectrum whereas hyper-spectral ones strive to resemble a continuous range of wavelengths [?]. Hyper-spectral images usually feature higher spectral and lesser spatial resolution than multi-spectral pictures [?]. Satellite images with multiple bands are often stored in special file formats or in a series of high bit-depth standard lossless image formats, such as Portable Network Graphics (PNG) or Tagged Image File Format (TIFF). These can take up to 13 bands or more in different files per single satellite photography.

One more crucial property of satellite imagery is the *Ground Sampling Distance (GSD)* parameter, which denotes a spatial distance between pixels of a digital image. For example, one-meter GSD states that the location of adjacent pixels is one meter apart on the ground. The GSD parameter determines the size of objects visible in the satellite pictures.

Super-resolution for satellite imagery has been developing rapidly in recent years. The growth of this field was accelerated by the Proba-V super-resolution competition organized by the European Space Agency [?]. The challenge lasted from the end of 2018 to June 2019 (over half a year). The competition consisted in creating a multi-image super-resolution network trained on the Proba-V dataset. Proba-V is unique in the satellite dataset group—it contains both low-resolution and high-resolution images of the same scenes, making it directly fit for super-resolution network training. The high-resolution images have GSD of $1/3 \times 1/3$ kilometer per pixel and the low-resolution ones feature GSD of 1×1 kilometer per pixel [?]. Architectures submitted in the competition have pushed super-resolution beyond previous baseline performance scores. A more detailed description of the Proba-V dataset can be found in Section 3.2.1.

2.2 Machine learning for image processing

Both the augmentation process and the super-resolution implementations in this work are based on machine learning—especially deep learning. The following chapters provide an overview of these methods for image processing.

2.2.1 Neural networks and deep learning

Machine learning is a computer science technique that solves problems by fitting models to data using 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 and the structure of the brain. 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 the sum of the weighted inputs passed through a nonlinear activation function [?]. In contrast to more advanced and specialized layers, the classical neurons are often called *dense*, *densely-connected* or *fully-connected*.

Training a neural network in a supervised manner requires a set of examples bounded with ground truth labels. The fitting process of such a network consists in adjusting the input weights. Learning is done in steps called *epochs*, during each epoch the training set is passed through the network. The output of the network is then compared with the ground truth labels. This is done according to a given *loss function* which serves as a metric between the actual and expected output of the network. Then the loss is used to optimize the weights via *gradient descent* methods. The gradients are computed using the *backpropagation* algorithm which is based on the chain rule of derivatives. There are various loss functions and optimizers to choose from and apply according to the given problem. In modern deep learning, datasets may be too big to apply backpropagation in one pass. For this reason, weights are usually updated using small subsets called *minibatches* [?].

This kind of machine learning architecture has been initially used with manual feature extraction. The utilization of a set of predefined filters or feature maps may be an example of this approach in the domain of image processing. A set of such feature maps would include basic geometric shapes to detect various edge types. These filters would be matched with regions of the input image. The results of such an operation are then fed into a neural network or other machine learning algorithm to get the final result of image processing (an example of this approach can be found in [?]).

With the advancements in the machine learning area, a new kind of neural network layer was created—a *convolutional layer* (often abbreviated to *conv layers*). These layers consist of (one, two, or even three-dimensional) filters that can be convolved with the input image. However, in contrast to the manual feature extraction 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 the gradient descent. This enables the creation of much better performing and flexible image processing neural networks. Convolutional layers can also be viewed as dense layers with shared weights between groups of pixels. This way, all pixels can be used in the fitting process without connecting every value in the image with every neuron, which would result in very big and hard-to-train networks [?]. A single convolutional layer usually consists of a number of filters. After passing a standard image through such a layer its depth increases to the number of filters. It is a common practice to apply a *pooling* (*maximum pooling, average pooling, global pooling*) operation after the convolutional layer to decrease spatial dimension of the image. Pooling layers reduce the image size by combining multiple pixels into a single one between neural layers (for example,

by averaging pixel values). Alternatively, a convolution with a substantially large stride can be used to shrink the image during a passthrough [?].

However, the creation of convolutional neural networks leads to increasing complexity and high numbers of parameters in models. These complex models demand 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 a Graphics Processing Unit (GPU) (or even multiple GPUs) and may last a few days. This combination of three factors: complex multi-layered neural networks (often with media-oriented specialized layers), very large datasets (often with many classes and objects), and the utilization of expensive time-consuming trainings constitutes modern *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. Modern super-resolution, which this work revolves around, is possible thanks to advancements in deep learning.

Generalization capabilities are a valid concern in the field of deep learning. Machine learning algorithms may be subject to the problem of *overfitting* where the model wights are overadjusted with regard to the training data. This problem leads to poor performance on data outside of the training set [?]. To evaluate the generalization capabilities of a neural network a *test set* is usually utilized. This dataset contains samples outside of the training data. One should beware of using test data in any step of the architecture modeling or training process. The test data should be separate from the model creation, the situation where information from the test set participates in the training process is called *data leakage* [?]. However, some form of evaluation is desired during the training process. Often, to prevent overfitting a subset of the training set is separated from the fitting process. This part of data is called *validation subset* [?]. When training loss continues to decrease, but the validation loss increases, the overfitting starts to occur and the training should be stopped. This work aims to explore the generalization capabilities of super-resolution networks trained on datasets created in different ways.

2.2.2 Encoder-decoder mechanism

Encoder-decoder network architecture is a common pattern in generative image processing. It is used both in data augmentation networks and super-resolution model utilized in Chapters 4 and 5. Encoder-decoder translates the input data into an abstract state during the 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

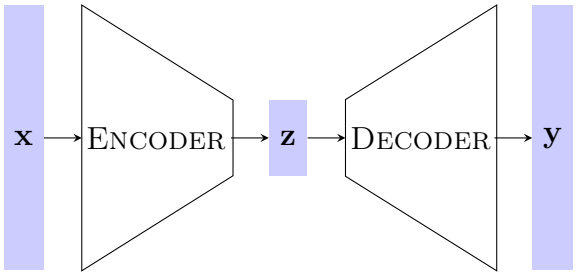


Figure 2.1: Schematic of encoder-decoder mechanism

commonly 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 fully-connected layer, although this is rarely applied in the super-resolution domain because densely-connected layers break the fully-convolutional [**?**] nature of a network (meaning that with a dense layer in the middle it cannot process images of varying spatial size). The decoding process commonly reconstructs depth dimensions into the 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 a different size than the input. Encoder-decoder architecture is an appropriate architecture for image-to-image transformations in machine learning. The inner workings of such an architecture are shown in Figure 2.1, where x and y denote the input and the output and z is the encoded hidden state.

The 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 the additional direct flow of information during the forward pass and a direct gradient flow on the backward pass. Residual connections applied between arms of an encoder-decoder create what is called a *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.

2.2.3 Generative Adversarial Networks

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 sample and comparing the generated output

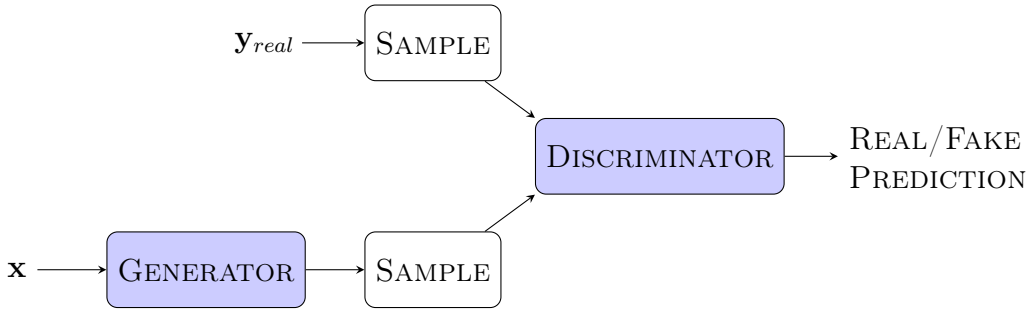


Figure 2.2: Schematic of a GAN network inner-workings

with a ground truth label to compute the loss value. The *Generative Adversarial Network (GAN)* approach requires creating two networks, a *generator*, and a *discriminator* [?]. 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 the discriminator loss is computed using standard classification metrics like *binary cross-entropy*. However, the generator learns in a more unique way; it creates an output image that is fed into the discriminator. The generator loss is calculated depending on how well it produces data that may be classified as *real* by the discriminator. If the generated image is recognized as a *fake* one, the generator receives a big penalty in the form of a large loss. The inner-workings of a GAN network can be visualized in the form of a graph, as in Figure 2.2.

GANs can have many variations, the most common type utilizes unsupervised or semi-supervised data generation. This means that in many GANs the generator can be fed with values from proper random distribution (often called *latent space*) to generate new data. The ability to create images without direct input is a great advantage of adversarial networks.

In general, adversarial network architectures provide vast generative capabilities. For this reason, GANs are widely used for data augmentation and creation [?, ?, ?, ?]. A variant of GAN architecture is later used in this work for creating super-resolution training data in Chapter 4.

2.2.4 Measuring quality of image-generating neural networks

Both super-resolution networks and data augmentation networks input and output images. Quantitative evaluation of such networks requires comparing two images—the network output and the ground truth reference image. Images are usually compared using metrics like *Mean Absolute Error (MAE)*, *Mean Squared Error (MSE)*, and *Peak Signal-to-Noise Ratio (PSNR)*. These calculate the error

between pairs of corresponding pixels in different ways. However, these metrics may be insufficient for super-resolution-related problems. Calculating pixel-wise differences does not 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 Measure (SSIM)* [**?**] was introduced. SSIM calculates image quality in three main components:

- Average *luminance*.
- *Contrast* as the standard deviation of pixels.
- *Structure* as the luminance difference divided by the 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 [**?**]. Advantages of SSIM render it suitable for super-resolution-related image quality evaluation.

However, modified versions of the previously mentioned traditional metrics can also be useful for image comparison, one of them being the *cPSNR* score. The traditional PSNR has the potential drawback of being sensitive to the bias in image brightness. This metric equalizes the average brightness of compared images before calculating standard PSNR to alleviate this problem. The improved version of PSNR was introduced and used for scoring in the Proba-V super-resolution competition [**?**]. Various of the mentioned metrics are used in this work, in accord with specific requirements of each step in the augmentation and super-resolution training process.

2.2.5 Image registration

Another challenge often encountered during super-resolution training and 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 the subpixel domain. The process of aligning two similar images is called *image registration*. Registration can be performed either with traditional or deep learning-based algorithms. To register the subpixel translations, images can be upscaled before using a matching algorithm.

Registration algorithms can be divided by the applied methodology of image processing. The first group of methods works on image features and properties. These algorithms can detect image area characteristics like boundary regions, corners and frequency properties. Image features can also be detected by

similarity measures and image descriptors like *Scale-Invariant Feature Transform (SIFT)*, *Speeded-Up Robust Features (SURF)*, and statistical properties, e.g., *cross-correlation* (more detailed description of various methods can be found in [?]). Pictures can also be characterized by frequency-domain-based traits like *phase-correlation* [?]. This method utilizes a two-dimensional *Fourier transform* on images and is later used in this work (Chapter 4).

Another branch of image registration techniques takes advantage of modern machine-learning technologies. These can utilize approaches like deep learning with similarity metrics or unsupervised trainings [?]. A modified version of an image inpainting deep neural network can be used as a well-working registration algorithm [?, ?]. A more detailed explanation of such an approach is included in Section 2.3.1.

2.2.6 Data augmentation in machine-learning

Data augmentation techniques are briefly characterized in the introductory part (Section 1.2). However, this field is a very wide area of research, ranging from applying simple modifications to existing samples to generating synthetic datasets [?]. Traditional augmentation techniques include operations like zooming, resizing, shifting, flipping, rotating, distorting, adding noise, random erasing, image mixing, applying predefined filters, modifying colors, and exposure. Data augmentation is useful for preventing overfitting and training complex networks on relatively small datasets [?]. These operations may be application-specific. To give an example, one should beware of distorting or flipping data containing constrained geometry, like road signs. The artificial data should resemble real-life data as much as possible.

The traditional augmentation techniques lead to great results; however, more modern approach based on deep learning can be used to widen data generation capabilities. The popularization of GAN networks has led to great advancements in data generation. Thanks to the adversarial networks entirely artificial samples can be generated [?] even online during the training process [?].

2.3 Overview of super-resolution techniques

As mentioned in Chapter 1 super-resolution techniques can be divided into single and multi-image categories where one or many low-resolution images are turned into a high-resolution one. The latter is a more advanced technique, which utilizes multiple low-resolution images of the same scene to produce one high-resolution picture. The usual approach is to utilize multiple low-resolution images that are slightly shifted (in the subpixel domain). Data from these multiple images is merged together to produce an image of greater quality [?]. This approach can

lead to the best results in super-resolution. In some scenarios, the data fusion can lead to the recreation of high-resolution details that are hardly visible in any single low-resolution image. In the case of multi-image super-resolution, better image quality comes at a cost of obtaining a series of input pictures instead of one, as it is in the single-image approach.

The simplest super-resolution methods predate deep-learning techniques and utilize interpolation for multi-image data. Multiple low-resolution images with subdomain shifts can be used to achieve more detailed interpolation for missing values in the reconstructed high-resolution picture [**?**]. Nowadays super-resolution is mainly done with deep learning techniques which undergo rapid development. Deep learning-based super-resolution can be divided in regard to: supervision (supervised or unsupervised learning), application domain, network architecture, learning strategy, and evaluation technique [**?**, **?**]. Machine learning techniques that enable modern super-resolution are discussed in Section 2.2. There are many deep learning-based modern super-resolution models that may vary in architecture. Examples of such networks are: *Residual Attention Multi-Image Super-Resolution (RAMS)* [**?**], *HighRes-net* [**?**], and *Deep Neural Network for Super-Resolution of Unregistered Multitemporal Images (DeepSUM)* [**?**]. An in-depth analysis of a super-resolution neural network is provided in Section 2.3.1.

2.3.1 Super-resolution with HighRes-net

In recent years many super-resolution architectures have emerged due to advancements in deep learning techniques. At the moment, RAMS architecture achieves one of the best scores. However, in this work HighRes-net architecture is utilized. HighRes-net, which is a few months older, achieves slightly worse results; however, it is simpler and faster to train [**?**]. 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.

Architecture overview

HighRes-net [**?**] is a super-resolution network based on generative deep learning. It falls into the category of *Multi-Frame Super-Resolution (MFSR)* algorithms, which takes a *many-to-one* (or *multi-image*) approach to output generation. In MFSR systems input is a series of images, taken with a slight shift, and in different moments in time. The input series contains more information, than a single image, as a result of random displacements, noise disturbances, and atmospheric conditions. MFSR tackles the problem of aliasing in sampled data. Low-frequency parts of the image, with large geometry and little detail, do not differ much between

many images. However, MFSR is crucial when enhancing small details. Upscaling small geometry from a single image can be non-reliable due to aliasing. Applying MFSR 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 an input low-resolution image into a high-resolution one.

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

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

Super-resolution inference process

The inference pipeline of HighRes-net is shown in Figure 2.3. 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 consist of a convolutional layer and two residual blocks with *Parametric Rectified Linear Unit (PReLu)* activations. For input of length n, the 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*

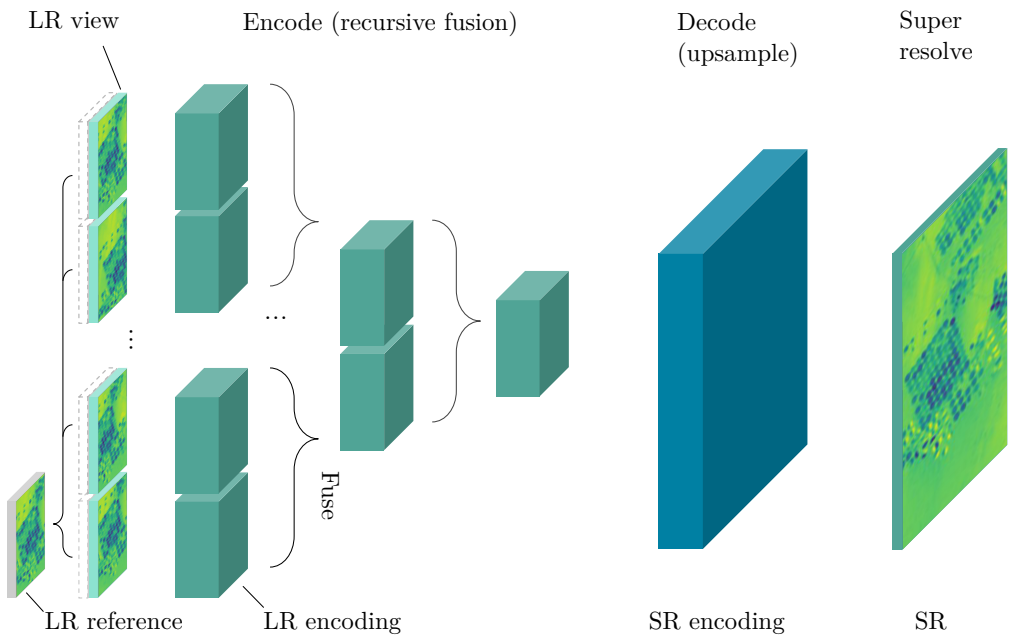


Figure 2.3: Schematic of inference in HighRes-net [?]

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. The 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 a 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 does not include local residual layer). The whole co-registration-fusion includes a residual connection. As in 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 a 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.

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 a shift between predictions and targets. Previous steps of HighRes-net include an *implicit co-registration*, where registration mechanisms learned by the network do not have to be necessarily based on shifts, but also other geometric distortions. During the 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 subpixel registration is done by the *ShiftNet-Lanczos* network. ShiftNet [**?**] was introduced before HighRes-net, in a separate research. It was created for image inpainting[2] via *Deep Feature Rearrangement* technique. 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* [**?**] architecture. As mentioned in Section 2.2.2, 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 ShiftNet 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 that introduces it.

2.3.2 Other super-resolution architectures

As mentioned, other super-resolution architectures are available, with RAMS [**?**] being one the best performing. 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 from spatial and temporal dimensions resulting in the best quality of multi-image super-resolution.

[2] *Inpainting* is a process of reconstructing or readjusting missing or damaged parts of an image.

2.4 Resizing images with interpolation techniques

Image resizing is a relevant topic in super-resolution, as a reference point and a tool for visualization. It is useful as a visual baseline for super-resolution. Traditional image resizing algorithms use interpolation to enable changing image dimensions; however, they do not create new details in the image. A well-working super-resolution should recreate missing features when enhancing images. Thus, it is expected that any super-resolution algorithm gives better results than image interpolation techniques. In this work, image interpolation is used for comparisons during both the augmentation and super-resolution processes.

Bicubic interpolation

Bicubic interpolation is one of the most prominent image interpolation techniques. Compared to other interpolation methods it is regarded as the most advanced and time-consuming of the commonly used solutions [**?**, **?**]. This interpolation mode fills missing values by fitting third-degree polynomials between existing pixels. The gradients of existing values are taken into account during the fitting process, so the interpolation splines' steepness matches the existing derivatives. Bicubic interpolation is usually calculated in neighborhoods of four by four pixels. Fitting polynomials to existing pixels may lead to *overshooting* values. This phenomenon often causes a slight increase in local contrast, which overall increases the *acutance* (perceived sharpness) of an image. Other image resizing modes operate on a simpler basis or consist in fitting simpler interpolation functions. For this reason, bicubic is the most advanced and often best out of the common solutions.

2.4.1 Other interpolation techniques

The bicubic interpolation has been chosen as the main reference point; however, there are other popular image resizing techniques that may be taken into account in comparisons.

The *nearest neighbor interpolation* is the simplest out of widely used techniques. In this approach, the missing points are filled in with values of the closest existing pixels. This technique often resolves in jagged edges and coarse details.

Another widely used approach is *bilinear interpolation*. The bilinear mode works similarly to bicubic one; however, it operates on simpler terms. Instead of fitting polynomials, it uses straightforward linear functions to find missing values. For his reason, it cannot take into account image gradients and often results in a less plausible outcome. Furthermore, in contrast to the bicubic interpolation, bilinear usually operates in two by two neighborhoods.

Lanczos is another interpolation method, with greater complexity and quality similar to bicubic. In contrast to other techniques, it uses sinus function for interpolation. Fitting is done using *Lanczos filters* which may vary in function order and neighborhood size.

2.5 Image visualisation techniques

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-channel images (like the pictures presented in this work), a colormap is often used. The most popular colormap is grayscale which displays white for maximal values, black for minimal values, and shades of gray in between. To improve the visibility of details, a yellow and blue colormap called *Viridis* [**?**] is mainly 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. An example of Proba-V satellite image with Viridis colormap applied is shown in Figure 2.4.

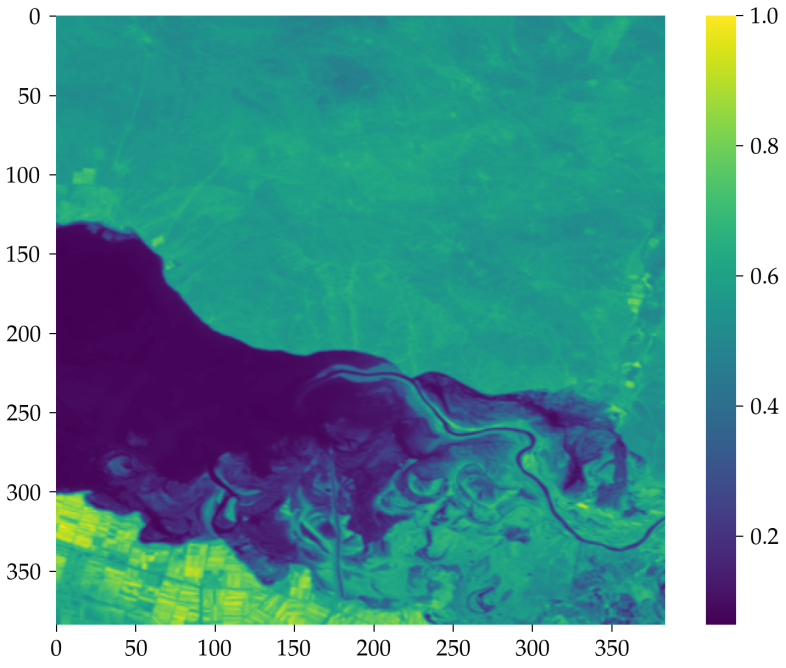


Figure 2.4: An example of Proba-V satellite image with Viridis colormap applied

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 the 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 a 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:

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

where $c_{max,min}$ denotes maximum and minimum values of the color mappings, $\bar{p}$ is the mean value of the pixels, and σ_p is the standard deviation of the pixel values. This way, only pixels within the range of three standard deviations from the mean are taken into consideration when creating a common colormap for image comparison.

Chapter 3

Scope of work

This chapter provides a detailed plan of the proposed work and a rationale for chosen methods and technologies. A subset of super-resolution and augmentation techniques is chosen to include in the course of the work. Based on the selection, an experiment plan is laid out. A brief description of utilized tools, technologies, and methodology to perform deep learning trainings is included.

3.1 Selection of data types and augmentation techniques

Various data types in the field of super-resolution and approaches to data augmentation were introduced in Chapters 1 and 2. A subset of possible techniques should be chosen to determine the scope of the experiments.

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 Figure 3.1. The graph features a distinction that has not been mentioned before—a difference between fully and semi-simulated multi-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 can be denoted as *semi-simulated*. Otherwise, one can create many low-resolution images from a single high-resolution one. This can be done by shifting the original image randomly multiple times and then, applying

the shrinking transformation to each displaced copy. This procedure is denoted as a *(fully) simulated* data generation and is presented in the form of pseudocode as Algorithm 1.

Algorithm 1 Approach to generating fully simulated multi-image datasets

Require: n_{lr}, $ratio_{shrink}$
  $shift_{max} = ratio_{shrink}$
  for all hr_{img} **do**
    $hr_{cropped} = crop\_border(hr_{img}, shift_{max})$
    for $i = 0$ **to** n_{lr} **do**
      $hr_{shift} = generate\_shift(shift_{max})$
      $hr_{translated} = translate(hr_{img}, hr_{shift})$
      $lr_{img} = downscale(hr_{translated}, ratio_{shrink})$
      $lr^{i}_{img} = crop\_border(lr_{img})$
    end for
    $save\_scene(hr_{cropped}, lr^{i...n_{lr}}_{img})$
  end for

A selection of approaches to be taken into account in this work has been made and marked with blue color in the graph in 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 multi-image super-resolution on single-band pictures with simulated data using deep learning and resizing algorithms.

3.1.2 Data augmentation techniques and approaches

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 levels 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, the images created with bicubic interpolation were subject to several trans-

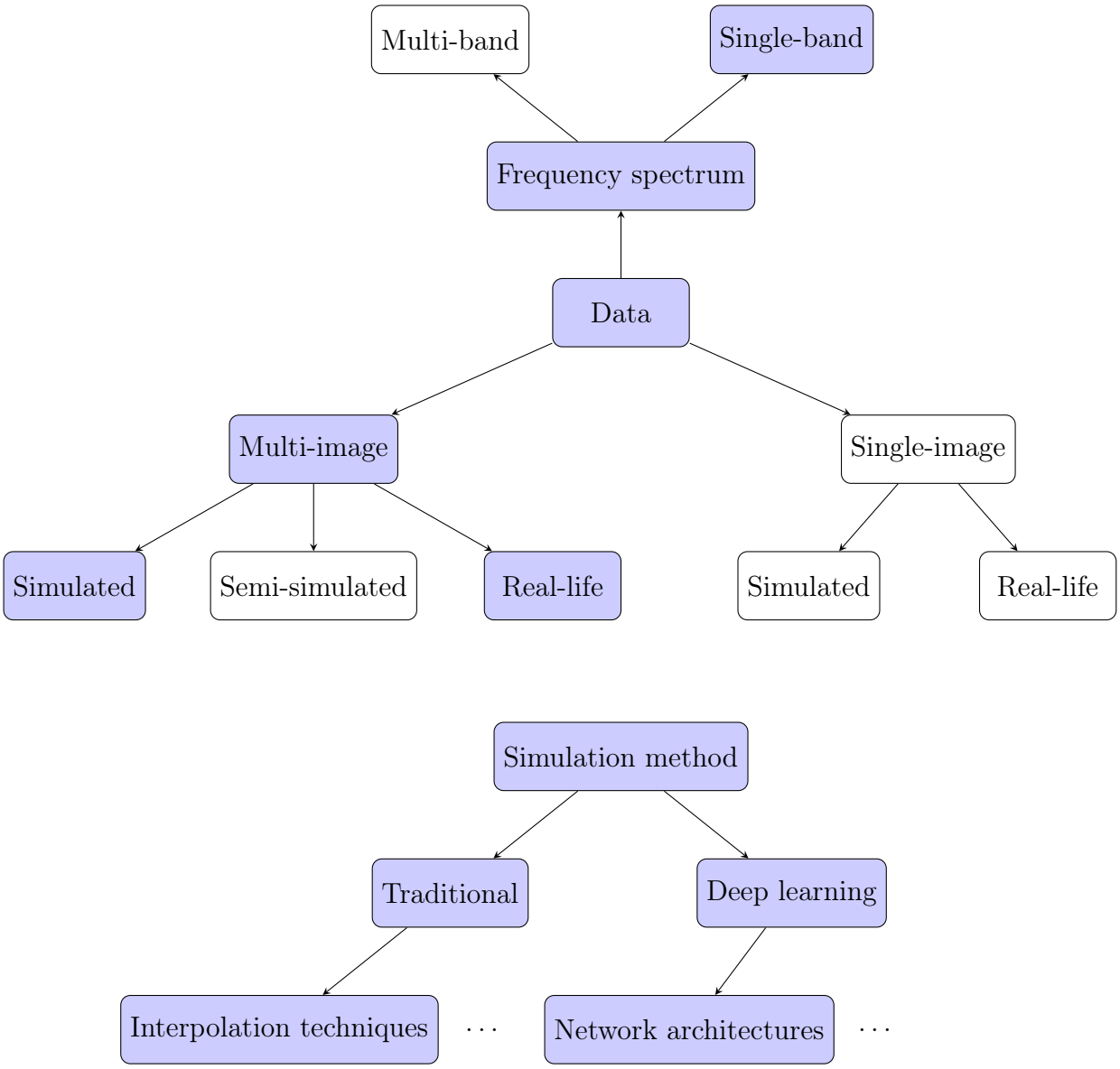


Figure 3.1: Graph of various training data generation techniques

formations to enhance their quality. The brightness, contrast, and noise of the interpolated images were adjusted using normal distributions with parameters:

Gaussian noise with 0.0 mean and standard deviation of 30 was applied to each image.

Contrast was adjusted by multiplying each image by random scalar value from gaussian distribution with 1.0 mean and standard deviation of 0.2.

Brightness was adjusted by adding a random scalar value from gaussian distribution with 0.0 mean and standard deviation of 200 to each image.

Gaussian blur with a standard deviation of 0.5 was applied to each of these low-resolution images.

These were applied to Sentinel-2 images with values in 14-bit (maximum value is 16384) range. These transformations applied to the dataset created by bicubic interpolation were chosen outside of 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. The translations were applied in the vertical and horizontal directions.

3.2 Plan of experiments

After choosing a subset of augmentation techniques to examine, an experiment plan should be laid out.

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 a super-resolution algorithm. A number of datasets are required to perform this task. To clarify the demands for specific datasets can be listed as:

1. A dataset containing high and low-resolution images for training an augmentation network.

2. A dataset containing high and low-resolution images for testing the augmentation network.

3. Dataset that should be augmented with the network. This set does not need to contain high and low-resolution pairs. The low-resolution images for existing high-resolution ones should be generated by the augmentation network. Results of the augmentation will be used to train the super-resolution network.

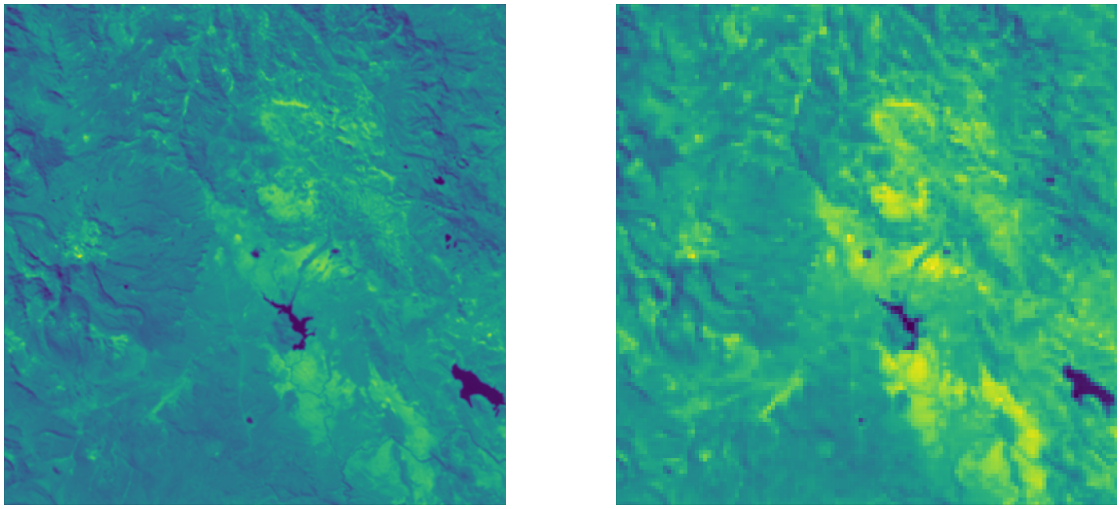
4. Dataset of high and low-resolution images for testing the super-resolution network.

Proba-V as an augmentation training dataset

The *Proba-V* dataset [**?**] 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 Light Spectrum (RED)* band (610–690 nm wavelength) and the *Near-Infrared Light Spectrum (NIR)* band (777–893 nm wavelength). Proba-V contains multiple real-life low-resolution images per one high-resolution image. Images of the same scene were taken in different moments, thus they vary slightly in framing and atmospheric conditions. Unprocessed Proba-V high-resolution images are 384 by 384 pixels, low-resolution photos are 128 by 128 pixels. As mentioned in Chapter 2 high-resolution images have GSD of $1/3 \times 1/3$ kilometer per pixel and the low-resolution ones feature GSD of 1×1 kilometer per pixel [**?**]. A selected pair of high and low-resolution samples from the Proba-V dataset is shown in 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 a Proba-V image with a binary mask designating clouds is shown in Figure 3.3. Proba-V pictures are saved as 16-bit images; however, only 14 bits are used to store pixel values.

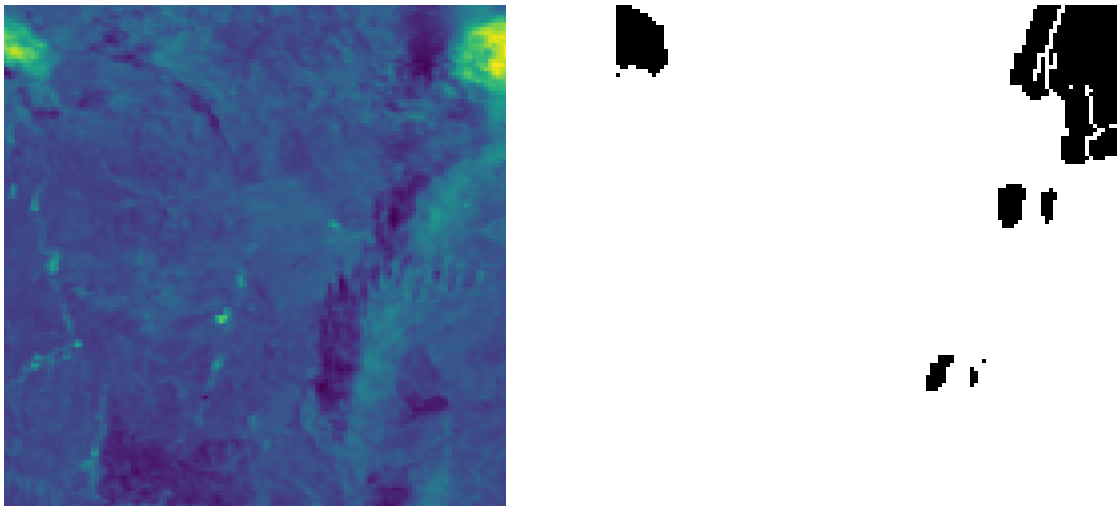
Sentinel-2 as a super-resolution training dataset

The super-resolution training part of the data demands can be met by utilizing *Sentinel-2* dataset [**?**]. Contrary to Proba-V, it does not feature high and low-resolution scenes, for this reason, it suits the role of the dataset to undergo the data generation process. Sentinel-2 is a part of the European Earth Observation



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

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



(a) Sample Proba-V image with invalid areas that contain 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

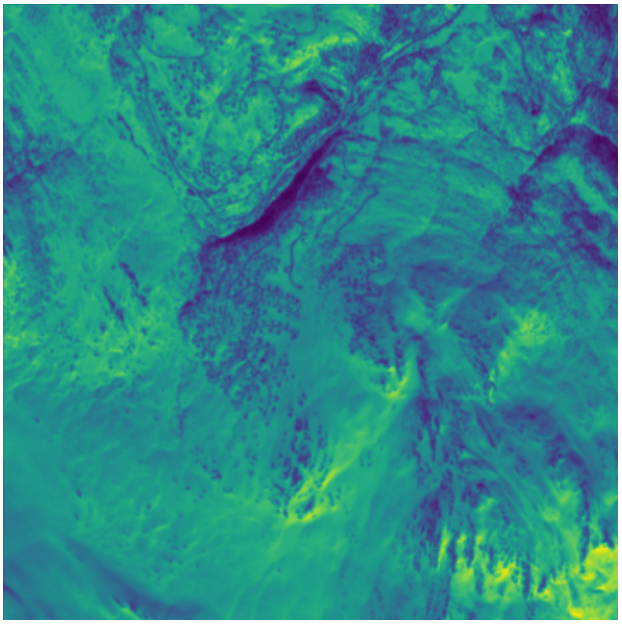


Figure 3.4: Sample image from *Sentinel-2 dataset* (eight band)

Program 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-2 images are multi-spectral and feature a total of 13 bands with GSD values of 10×10, 20×20 and 60×60 meters per pixel [**?**]. Since it was decided not to include multi-spectral super-resolution only band eight is to be used. This band features spectrum close to infrared (835.1 nm central wavelength, 145 nm bandwidth for satellite Sentinel-2 A and 833 nm central wavelength, 133 nm bandwidth for satellite Sentinel-2 B) with GSD of 10×10 meters per pixel). A sample image from the Sentinel-2 dataset is shown in Figure 3.4. As with Proba-V, Sentinel-2 is saved using 16-bit image format; however, only 14 bits store useful pixel values.

Dataset for super-resolution evaluation

Additional datasets are needed to evaluate the 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-2 dataset should contain a test subset. Each super-resolution network can be evaluated on a test set generated in the same way as that used for the training.
- Test subsets generated in different forms can be used in a cross-type evaluation scheme. In this work, the augmented datasets are created in a variety of

ways. The network trained on one version of an augmented dataset can be tested on a dataset created using a different method. For example, a super-resolution network trained using the data created with bicubic interpolation can be tested on the data generated by a neural network (and the other way). These cross-type tests can be used to examine generalization capabilities.

- Separate real-world Sentinel-2 images can be used for tests. Sentinel-2 does not include high and low-resolution pairs, so numeric tests are not possible. However, it is still viable to perform a visual test, for example, 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-V 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 Figure 3.5. The experiment flow presented in the graph is as folows:

1. Use existing real-life Proba-V dataset to train augmentation networks.

2. Take the existing Sentinel-2 dataset (which features single-resolution images). Use the augmentation networks to create Sentinel-2 datasets with high and low-resolution image pairs. These datasets are designated as *degraded* in the graph.

3. Use the single-resolution Sentinel-2 and bicubic interpolation to create another version of the Sentinel-2 dataset with high and low-resolution pairs.

4. The generated Sentinel-2 datasets should contain subsets for training and testing super-resolution networks.

5. Use all the generated datasets to train the HighRes-net super-resolution network.

6. Evaluate the super-resolution networks using various datasets:

   (a) Evaluate networks on the test subsets associated with the training sets. Test each network on the data created in the same way as its training set.

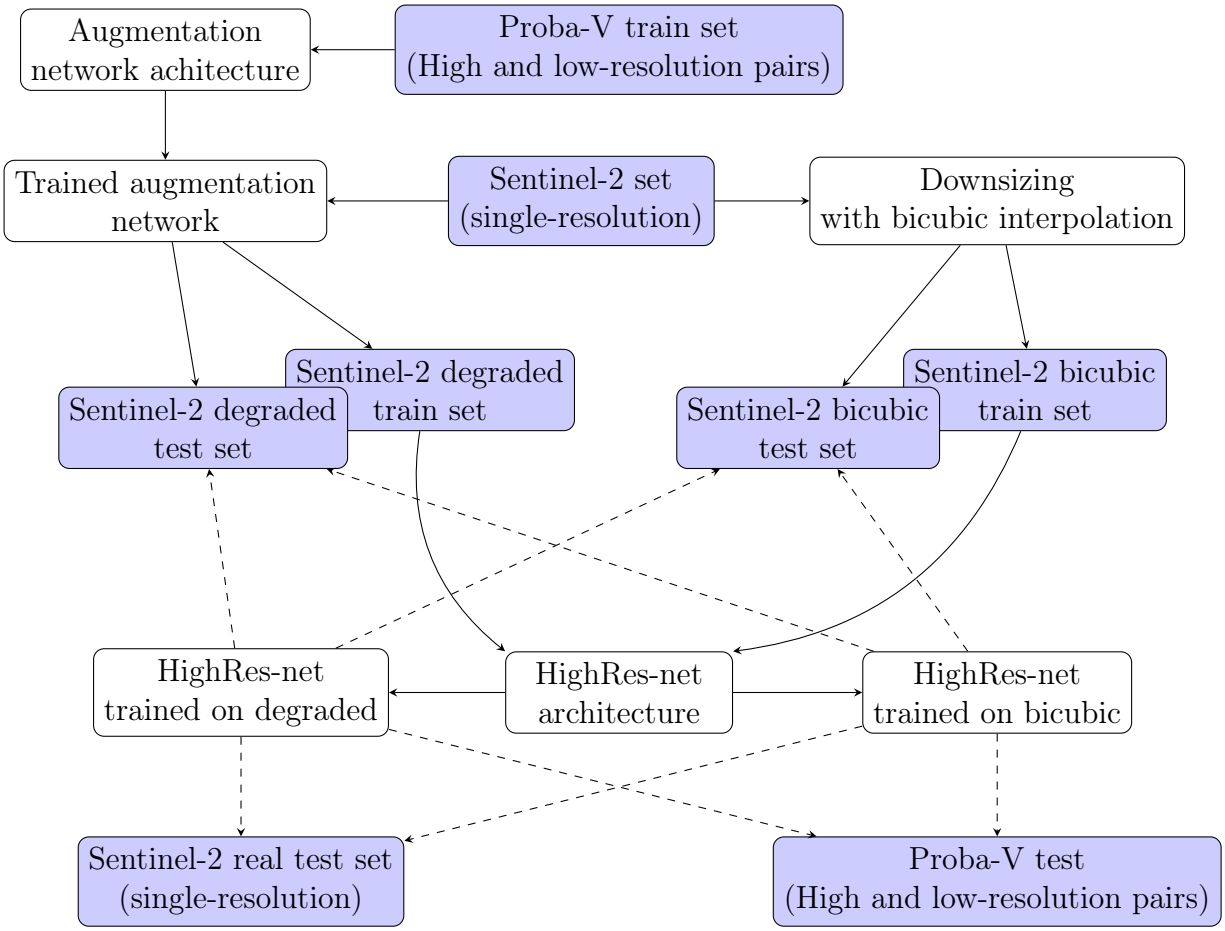


Figure 3.5: Graph of the experiment flow (solid lines indicate trainings, dashed arrows designate evaluations, blue stands for datasets, white is for models and algorithms)

(b) Use the generated test subsets to perform the cross-type evaluation. Test the networks on the image sets created differently from the training data.

(c) Use the Proba-V test subset to evaluate results of training HighRes-net on different datasets.

(d) Use the separate real-life single-image Sentinel-2 data to perform visual checks.

3.3 Selection of tools and technologies

The implementation of the experiment plan requires a selection of programming tools. Subsequent sections provide a rationale for choosing a set of necessary technologies and libraries to accomplish the project's goals.

3.3.1 Language and libraries

Python, which is an industry-standard for neural network research and scientific computing has been chosen as the main language for the implementation. It combines the ease of prototyping new code with the 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 which consists in writing neural network architectures and training them was written using the *Tensorflow* library. Tensorflow has been a leading deep learning library in recent years and is widely used in the industry and academia. Version 2.5 of the library was chosen for the project. From the second version, Tensorflow includes *Keras* neural network prototyping interface natively as default and recommended way for model building. This work utilizes the Keras API as well as some underlying Tensorflow functions 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 the *SciKit-image* library which offers a variety of utilities, transformations, and analysis tools. *Matplotlib*, another wide-spread Python library was used to create plots and heatmaps in the work. Alongside SciKit-image, *Pillow* was used for some image-related operations (mainly for resizing images with specific interpolation modes).

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 a contiguous memory layout, which enables the 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, as their size is predetermined. This constrain is a prerequisite to the contiguous memory layout of the arrays, which enables the 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 and prevents the integration of 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 the *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 data and popular Git server providers such as *GitHub* offer limited storage for repositories. For this reason, *Data Version Control (DVC)*—a supplementary tool was used. DVC consists of two main components: data versioning and experiment tracking. The data versioning part is similar 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 the command `dvc pull`. Since 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 results 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. Like 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; however, scripts in the project can

also be run independently of DVC (more details about reproducing experiments can be found in Appendix B).

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 the current commit.

dvc.yaml DVC experiment description.

params.yaml models and experiment configuration.

requirements.txt libraries requirements.

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

Chapter 4

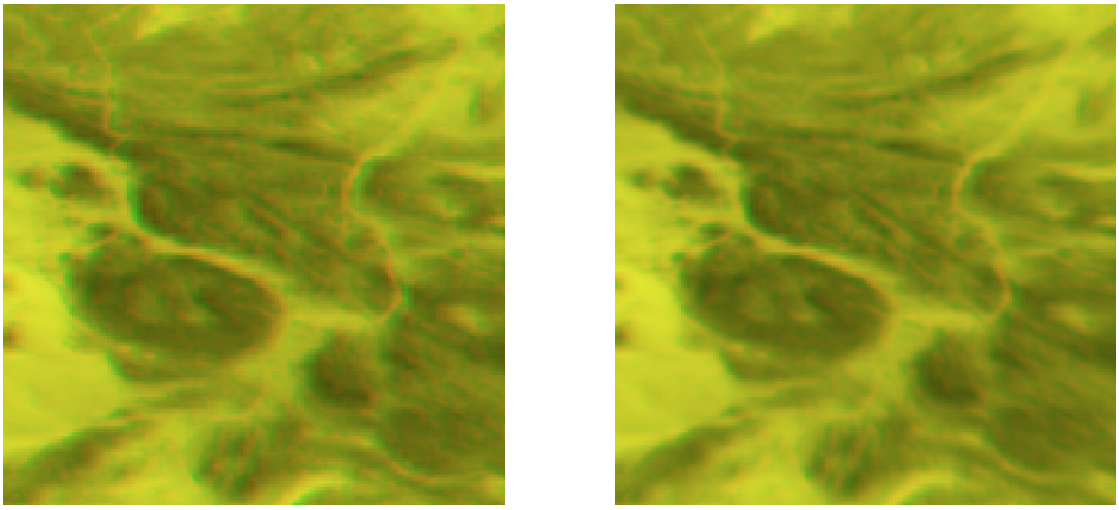
Data augmentation with the usage of deep learning

This chapter includes the main part of the work—creation and training of deep neural networks for data augmentation (generation of training images for super-resolution networks). This process includes preprocessing the training data, deciding on network architectures, implementing models, and conducting the training process. The chapter ends with a summary of intermediate results (before training the super-resolution network).

4.1 Proba-V preprocessing

As justified in Chapter 3, 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 alignment can be implemented. The Proba-V dataset features a single high-resolution image per many low-resolution ones for the same scene. The preparation of the dataset requires turning the single high-resolution and multiple low-resolution images into high and low-resolution pairs. This can be done by multiplying high-resolution images and aligning each copy with the corresponding low-resolution one.

Among the image registration algorithms presented in Section 2.2.5, phase-correlation was chosen for aligning image pairs in the Proba-V dataset. This method is suitable for the preprocessing task and is implemented as a part of popular image processing libraries. Since pictures to be aligned in Proba-V are of different resolutions, they should be resized to a common shape before registration.



(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*

It was decided to perform registration in the high-resolution image domain (because during registration images are upscaled anyway to calculate subpixel translations). Shifting images will create blank columns and rows on their borders. This problem was minimized by cropping the images. The low-resolution images were trimmed with a one-pixel border. High-resolution images are three times larger, consequently, a margin of three pixels was removed from them. After this step of preprocessing high-res images are 378×378 pixels and low-resolution photos are 126×126 pixels. It is valuable to visualize the effects of the registration process to ensure its correctness. Because Proba-V images are monochromatic, they can be visualized as different color channels. This technique is utilized in Figure 4.1. In this picture, each of the channels—red and blue contains one of the monochromatic images. If the pictures are aligned perfectly, the image should be yellow because red and blue channels overlay perfectly (red and blue channels mixed in equal proportions give yellow color). The more unaligned the pictures are, the more red and blue are 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 16-bit images; however, only 14-bit values are utilized. This should be noted to scale the images properly. Using the Proba-V dataset with real-life high and low-resolution images poses an additional challenge. As stated, the images contained in the dataset are taken at different

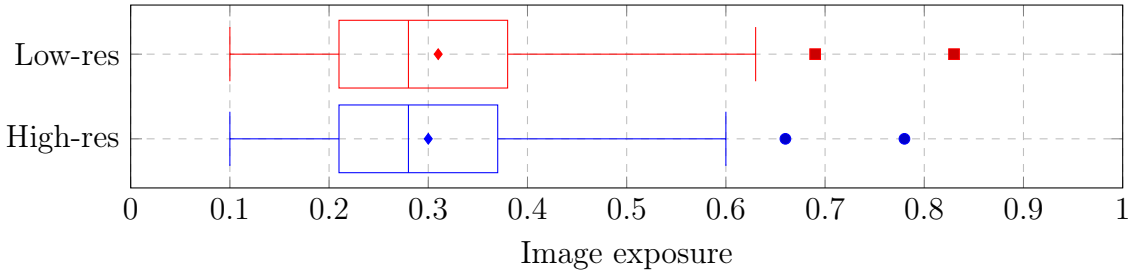


Figure 4.2: Exposure (mean-pixel value per image in 14-bit range) distributions in the RED Proba-V dataset

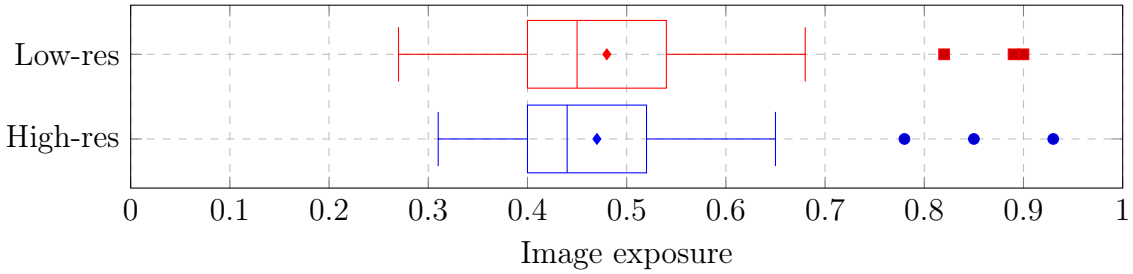


Figure 4.3: Exposure (mean-pixel value per image in 14-bit range) distributions in the NIR Proba-V dataset

moments, thus they differ in exposure and contrast. Images in high and low-resolution 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 Figures 4.2 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 in Section 3.2.1, Proba-V includes binary masks denoting areas of images that are suitable for training. Some images include noticeable large areas of unusable pixels. For this reason, nine images with the least percentage of invalid pixels were chosen per scene as training data. This minimizes the number of incorrect pixels in the training set.

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

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

4.2 Augmentation network architectures

The augmentation network is to perform a rather simple task of reducing the size of input pictures. Contrary to the traditional image downsampling, the networks should learn to shrink images from pairs of real-life images as stated in Chapter 1. Three architectures that perform this task have been created, they are presented with increasing complexity:

1. Simple convolutional network
2. Encoder-decoder network
3. GAN

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 3×3. The midmost convolution slides the filter window with a stride of three to decrease the size of input three times. The last convolutional layer features one filter to reduce the output depth to a single channel. The Rectified Linear Unit (ReLu) activation function was used for each applicable layer. A precise description of the architecture is presented in Table 4.1.

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

Table 4.2: Encoder-decoder network architecture for data augmentation

Layer type	Output Shape	Number of parameters
$Input$	$378 \times 378 \times 1$	0
$Conv2D(filters = 64, stride = 1)$	$378 \times 378 \times 64$	640
$ReLU$	$378 \times 378 \times 64$	0
$Conv2D(filters = 64, stride = 3)$	$126 \times 126 \times 64$	36928
$ReLU$	$126 \times 126 \times 64$	0
$Conv2D(filters = 64, stride = 2)$	$63 \times 63 \times 64$	36928
$ReLU$	$63 \times 63 \times 64$	0
$UpSampling2D(stride = 2)$	$126 \times 126 \times 64$	0
$Conv2D(filters = 1, stride = 1)$	$126 \times 126 \times 1$	577
$Sigmoid$	$126 \times 126 \times 1$	0

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 latter 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 [**?**]. 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 Table 4.2.

4.2.3 Generative Adversarial Network

The general idea of an adversarial network was covered in Section 2.2.3. As mentioned, models with adversarial loss can be used in numerous data generation scenarios. Usually, the GAN generator creates output from random values. However, adversarial networks can also work in a 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 judgments. An algorithm for training such a network is provided in the pseudocode as Algorithm 2. One further optimization can be applied to GAN—often better results are achieved if small noise is applied to the labels [**?**].

The final Tensorflow 2 implementation of the custom training loop step is shown in Listing 1. Since the network is trained in a supervised way each step requires high-resolution input images x and ground truth low-resolution pictures y_{gt}.

Algorithm 2 GAN training flow (single train step)

Require: x, y_{gt}
$y_{fake} = generator(x)$
$y_{pred} = discriminator(cat(y_{fake}, y_{gt}))$
$loss_d = lossfn_d(y_{pred}, cat(\mathbb{0}, \mathbb{1}))$
$optimize_d(loss_d)$
$freeze\_learning(discriminator)$ {During the generator learning the optimizer should not be trained}
$y_{pred} = discriminator(generator(x))$
$loss_g = lossfn_g(y_{pred}, \mathbb{1})$ {Notice the misleading labels matrix}
$optimize_g(loss_g)$

The discriminator architecture resembles a common binary classifier with convolutional and pooling layers. A *leaky ReLu* 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 trainings is the 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, the stride of size 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 Table 4.3. The generator is structured similarly to the previously presented simple fully-convolutional network, as it performs a similar task. The full architecture is the same as in Table 4.1.

4.3 Training details

The NIR subset of the Proba-V dataset was used to train the augmentation networks. As mentioned before, Proba-V 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 calculating the loss. This feature works only with pixel-wise losses such as MAE and MSE. Because SSIM works in a more

Listing 1 Custom Tensorflow 2 train step method for training a GAN network

```
def train_step(self, data):
        x, y, y_mask = data
        batch_size = tf.shape(x)[0]

        y_fake = self.generator(x)

        # Discriminator training
        discriminator_input = tf.concat([y_fake, y], axis=0)
        fake_labels = tf.zeros((batch_size, 1))
        true_labels = tf.ones((batch_size, 1))
        labels = tf.concat([fake_labels, true_labels], axis=0)
        labels += 0.15 * tf.random.uniform(tf.shape(labels))

        with tf.GradientTape() as tape:
            y_pred = self.discriminator(discriminator_input)
            d_loss = self.loss_fn(labels, y_pred)

        grads = tape.gradient(d_loss, self.discriminator.trainable_weights)
        self.d_optimizer.apply_gradients(
            zip(grads, self.discriminator.trainable_weights))

        # Generator training
        misleading_labels = tf.ones((batch_size, 1))

        with tf.GradientTape() as tape:
            y_pred = self.discriminator(self.generator(x))
            g_loss = self.loss_fn(misleading_labels, y_pred)

        grads = tape.gradient(g_loss, self.generator.trainable_weights)
        self.g_optimizer.apply_gradients(
            zip(grads, self.generator.trainable_weights))

        return {'d_loss': d_loss, 'g_loss': g_loss}
```

Table 4.3: Discriminator architecture in the proposed GAN for data augmentation

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

structural way, it is harder to remove some pixels from the evaluation when using this metric. For this reason, using masks during loss calculation is not supported when using SSIM.

Both simple and encoder-decoder networks use *Adaptive Moment Estimation (ADAM)* [**?**] optimizer for gradient descent. This kind of optimization algorithm combines the advantages of *momentum* technique and *Root Mean Square Propagation (RMSprop)* [**?**]. The momentum component uses a moving average of gradients from consecutive steps to update weights, instead of a single gradient value, as traditional algorithms would use. This modification helps the gradient descent to avoid slowdowns on plateaus in the search space. The RMSprop-alike part of the ADAM is responsible for adaptive scaling of the learning rate based on the 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 does not 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 fitting 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

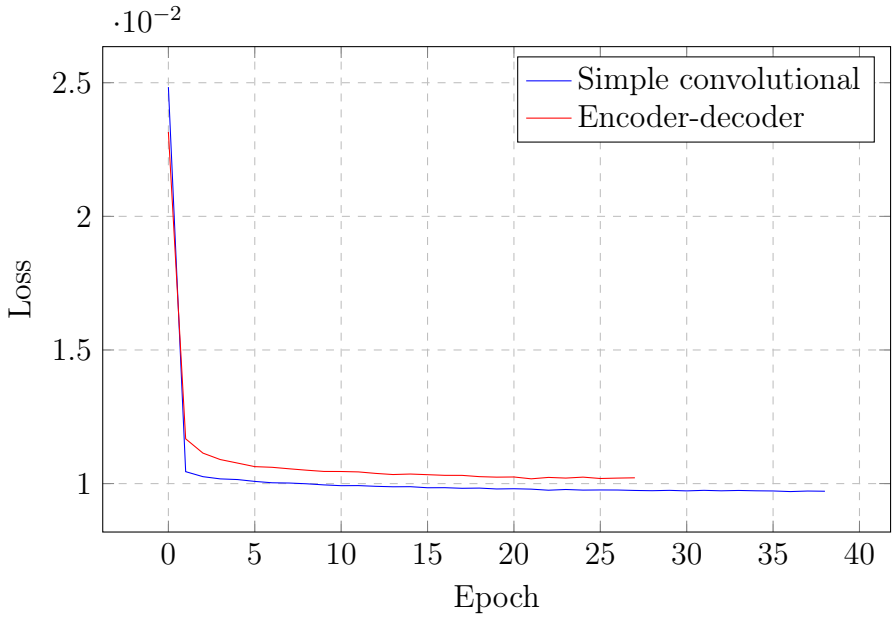
validation loss value. Early stopping helps to avoid overfitting and provides a more rational stop condition than training for an arbitrary number of epochs. The training histories of the simple convolutional and the encoder-decoder network are presented in Figure 4.4. The subfigures show the simultaneous decrease of training and validation loss. The decreasing loss curves resemble an exponential decay function, which is a sign of a proper training.

Training a GAN can pose a significant challenge when the stop 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 a 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 relative to the discriminator's ability to differentiate real and generated images. This does not 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 the SSIM value of the generator output and ground truth. This approach is not applicable to all GANs 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. The history of the training loss of the discriminator and generator parts of the GAN are presented in Figure 4.5. The model from 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.

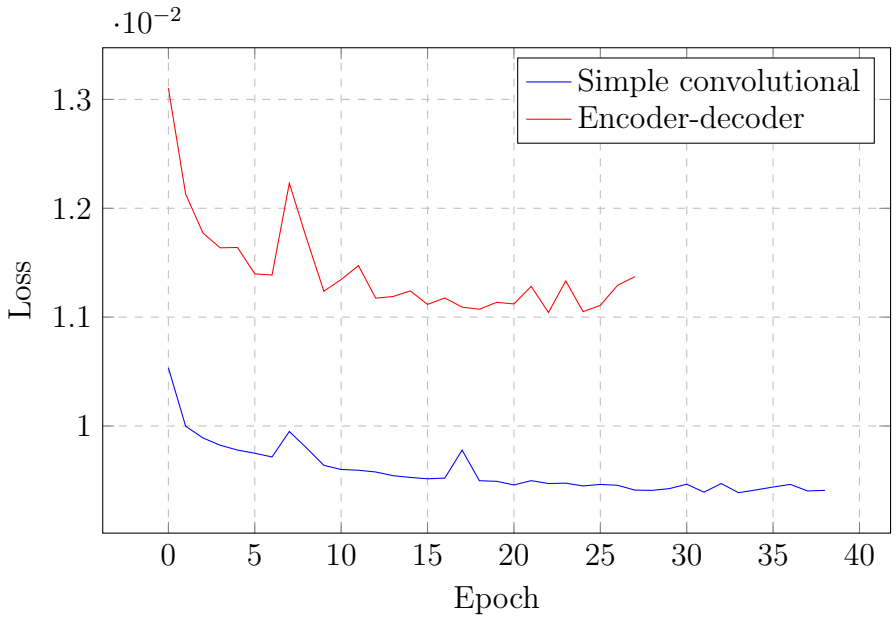
The contents of the *params.yaml* file used for training the augmentation networks are included in the appendix, in Chapter A.

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 evaluate 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 the augmented pictures resemble the real-life low-resolution ones. 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 traditional image-resizing algorithms. The results are presented



(a) Training loss history



(b) Validation loss history

Figure 4.4: Training history of the simple convolutional and encoder-decoder networks

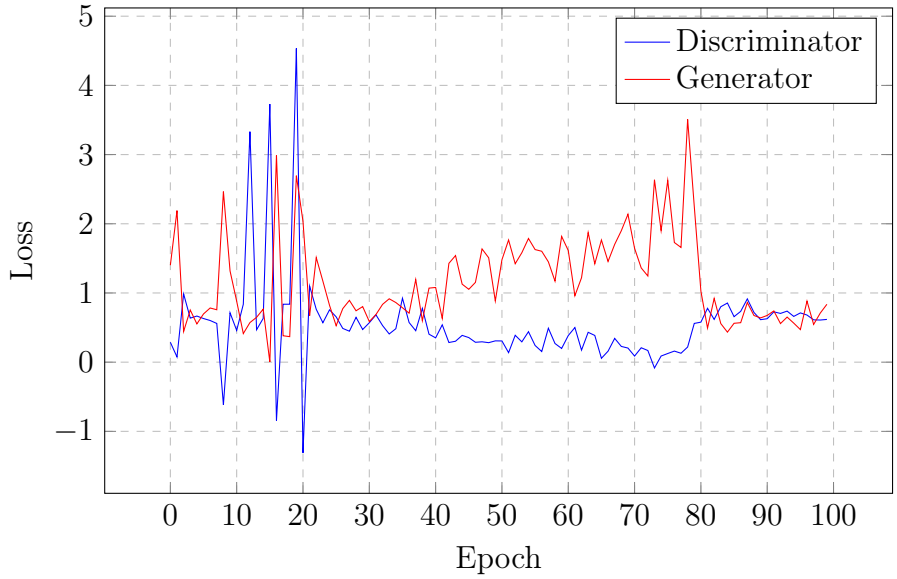


Figure 4.5: Training history of GAN generator and discriminator subnetworks

in Table 4.4. The images generated by the neural networks and most of the interpolation algorithms resemble the real-life low-resolution images with SSIM with over 0.9. This indicates that the networks can create images that are fairly similar to real-file images from the Proba-V dataset which is crucial for training super-resolution images in the future. The results can also be examined visually by displaying images side-by-side. Figure 4.6 presents an example of image shrinking using various methods, Figure 4.7 presents an enlarged 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.

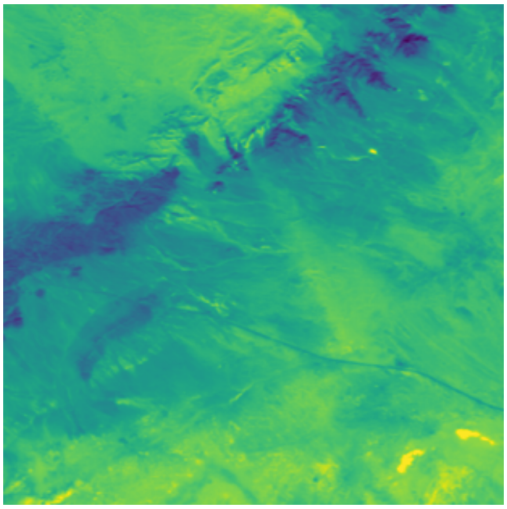
The intermediate results indicate that the augmentation process works reasonably well and can be used for creating new datasets. The numeric metrics look fairly similar for all given methods, their usefulness is to be determined during super-resolution training and evaluation. The visual results also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

4.5 Implementation details

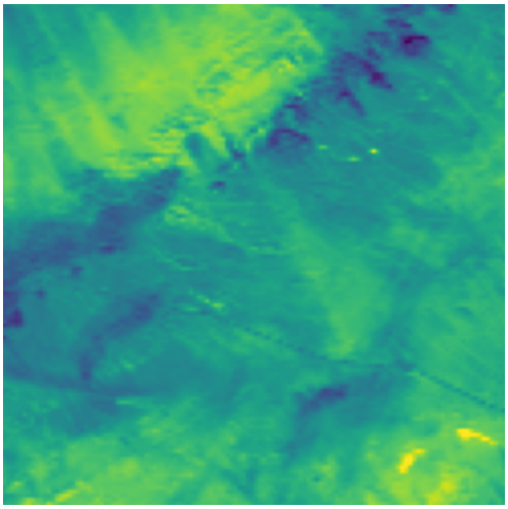
As stated before, all networks were implemented in *Python* using *Tensorflow 2* library. The built-in Keras interface offers several ways of building models; the most flexible one is called *model subclassing*. This approach enables using binary masks during training thanks to the possibility to overwrite the fitting method. A

Table 4.4: Intermediate results of evaluation on Proba-V test dataset (SSIM metric, the larger, the better)

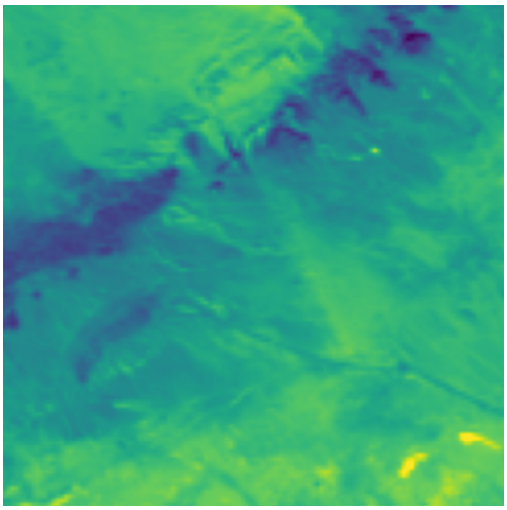
SSIM	Real	Simple conv	Encoder-decoder	GAN	Nearest	Bilinear	Bicubic	Lanczos
Real	1.0	0.946	0.935	0.923	0.887	0.947	0.945	0.94
Simple conv	0.946	1.0	0.987	0.963	0.938	0.995	0.996	0.992
Encoder-decoder	0.935	0.987	1.0	0.958	0.91	0.982	0.982	0.978
GAN	0.923	0.963	0.958	1.0	0.892	0.967	0.966	0.962
Nearest	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



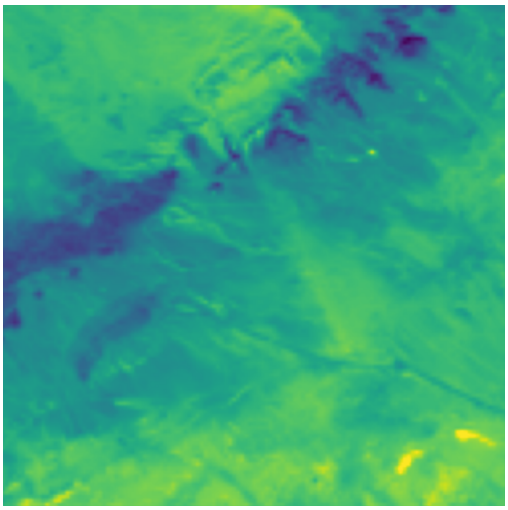
(a) Real-life high-resolution image



(b) Real-life low-resolution image

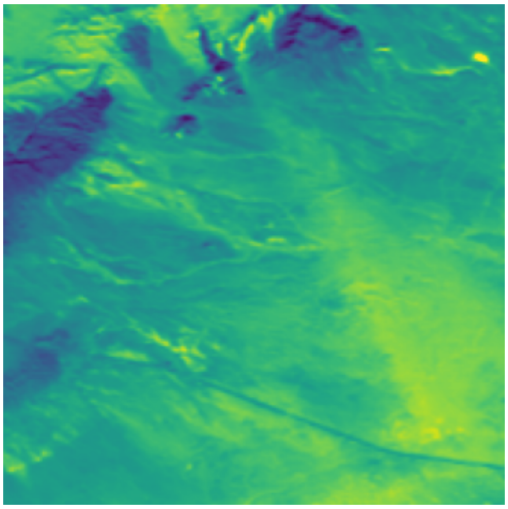


(c) Low-resolution image created by downsampling the high-resolution one with the augmentation networks

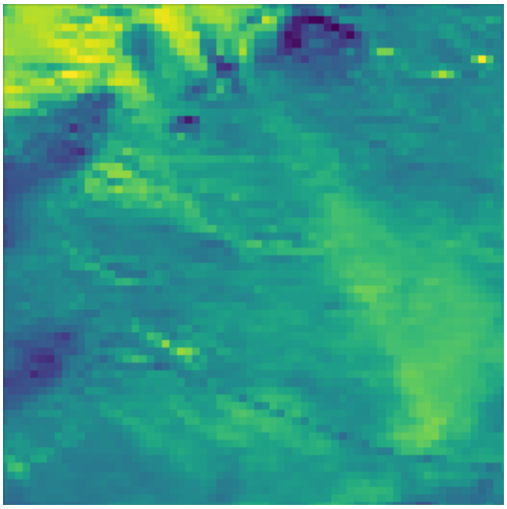


(d) Low-resolution image created by resizing the high-resolution one with bicubic interpolation

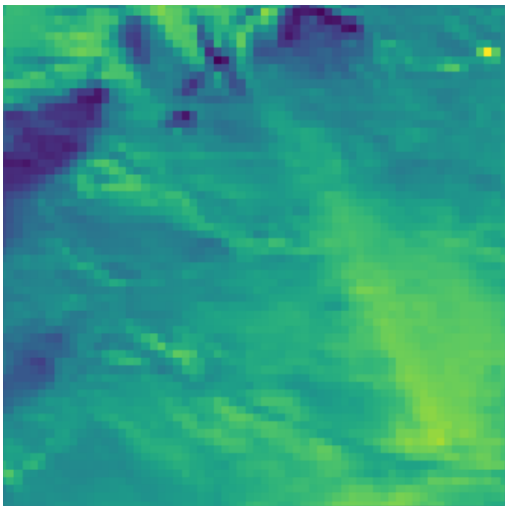
Figure 4.6: Intermediate results of evaluation on Proba-V test dataset for the simple convolutional augmentation network



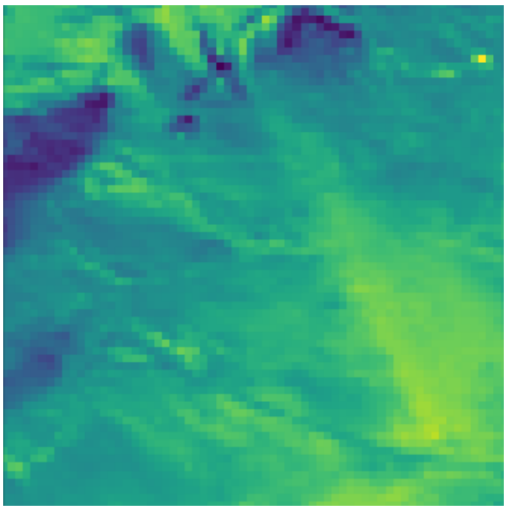
(a) Real-life high-resolution image



(b) Real-life low-resolution image



(c) Low-resolution image created by downsampling the high-resolution one with the augmentation networks



(d) Low-resolution image created by resizing the high-resolution one with bicubic interpolation

Figure 4.7: Zoomed intermediate results of evaluation on Proba-V test dataset for simple convolutional augmentation network

Listing 2 Custom train step method with masking capabilites

```
def train_step(self, data):
    x, y, y_mask = data

    with tf.GradientTape() as tape:
        y_pred = self(x, training=True)
        if self._use_lr_masks:
            y_pred = tf.boolean_mask(y_pred, y_mask)
            y = tf.boolean_mask(y, y_mask)
        loss = self.compiled_loss(
            y, y_pred, regularization_losses=self.losses)

    trainable_vars = self.trainable_variables
    gradients = tape.gradient(loss, trainable_vars)
    self.optimizer.apply_gradients(zip(gradients, trainable_vars))
    self.compiled_metrics.update_state(y, y_pred)
        return {m.name: m.result() for m in self.metrics}
```

custom train step with masking capability is shown in Listing 2.

Model subclassing involves inheritance to implement custom training loops for Tensorflow models. The simple convolutional and encoder-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. A custom inference preview callback was implemented to enhance the training process. Every training epoch a preview of inference on validation data is saved to a file.

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. These plots usually contain the history of training and validation loss values throughout the fitting process. 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 the 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.

Chapter 5

Super-resolution training and evaluation

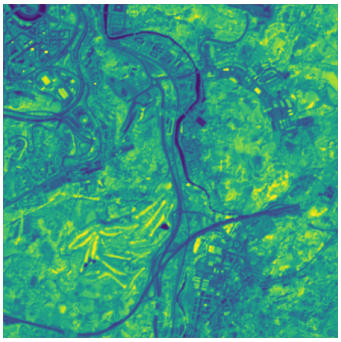
This chapter covers the training and evaluation of the HighRes-net super-resolution network trained with different datasets. Various training possibilities are explored in regard to the network generalization problem. The end of the chapter presents and discusses the summarized results.

5.1 Training HighRes-net

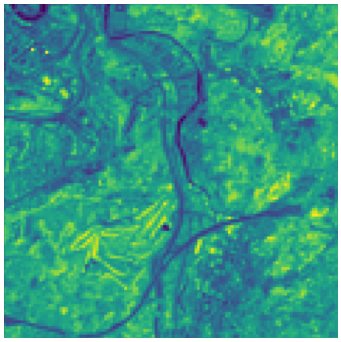
As stated in the experiment layout, the final step consists in training HighRes-net super-resolution model with different data. One final step before conducting the training is to examine visually the results of augmentation on a Sentinel-2 image. These are presented for the all the augmentation network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

It should be kept in mind, that the training dataset created by resizing with bicubic interpolation was subject to additional transformations of contrast, noise, and blur as stated in Chapter 3.

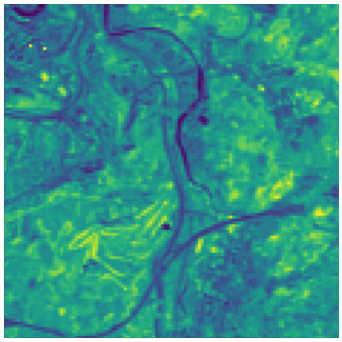
As planned, the super-resolution is to be trained with multi-image data in a single-band (band eight) mode. Training is configured in such a way that only weights for the best validation score are saved. However, automatic stopping is not enabled. For this reason, fitting was interrupted manually around epoch 100 when all training curves plateaued. The training history has been plotted to visualize the fitting progress. Figure 5.3 shows the loss history for all discussed HighRes-net trainings, Figure 5.4 presents the validation loss improvement. Section 5.1.1 discusses some parts of these figures in a more detailed way.



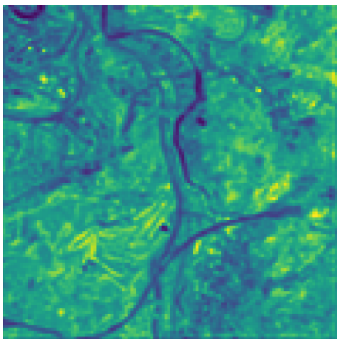
(a) High-resolution



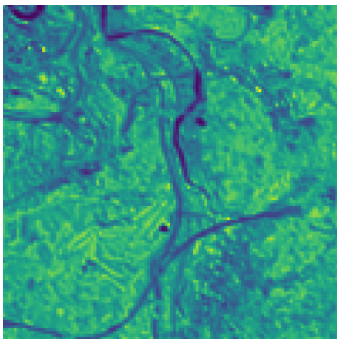
(b) Low-resolution image created from the high-resolution one with bicubic interpolation



(c) Low-resolution image created from the high-resolution one by the simple convolutional network

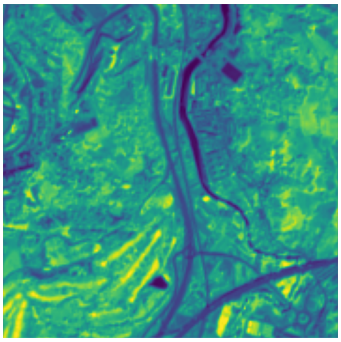


(d) Low-resolution image created from the high-resolution one by the encoder-decoder network

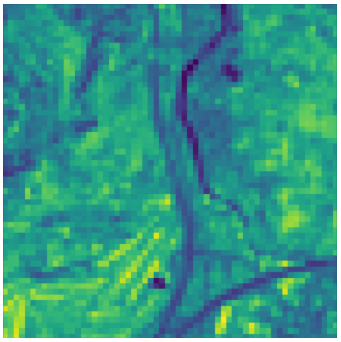


(e) Low-resolution image created from the high-resolution one by the GAN network

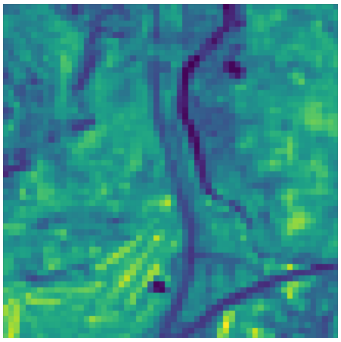
Figure 5.1: Example of Sentinel-2 training dataset images created in different ways



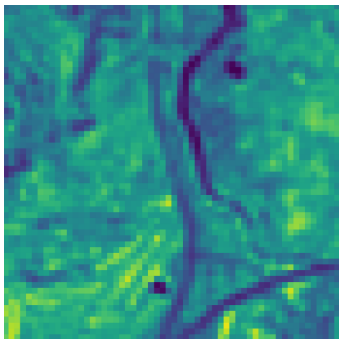
(a) High-resolution



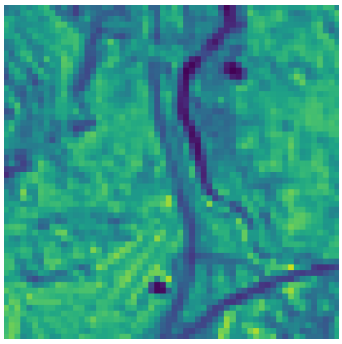
(b) Low-resolution image created from the high-resolution one with bicubic interpolation



(c) Low-resolution image created from the high-resolution one by the simple convolutional network



(d) Low-resolution image created from the high-resolution one by the encoder-decoder network



(e) Low-resolution image created from the high-resolution one by the GAN network

Figure 5.2: Zoomed examples of Sentinel-2 training dataset images created in different ways

5.1.1 Stop condition based on cross-data-type validation

Having multiple datasets generated in different ways enables an alternative way of validating the super-resolution 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. This kind of validation is available thanks to the multiplicity of the generated datasets and is unique to this work. The dataset created using the simple convolutional network and resizing algorithm were used for the cros-type-validation. In this scheme the training and validation sets are created in different ways which enables broader look on evaluation during training. It should be noticed, that the validation subsets were taken from the training datasets and not the evaluation test sets to prevent data leaks. In Figures 5.3 and 5.4, in the case of the cross-data-type validation scheme, the letter t denotes the training dataset and the letter v denotes the validation dataset. When using the cross-data-type evaluation the validation loss stops to decrease significantly earlier, as seen in Figure 5.4. A small raise in validation loss is visible for over ten epochs before the training was stopped. In this case, the best weights were saved nearly three times earlier than during the non-cross-type-validated training. This may indicate that the vast part of the fitting process does not contribute to the overall generalization capability of the super-resolution network. The cross-validated training is stopped earlier; since only weights for the epoch with the best validation score are saved, it is pointless to run longer fittings.

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.
- Real-life Sentinel-2 images that were not used in the training process and do not include high and low-resolution pairs, so only visual examination can be performed. These are original images that were not downsampled. This means that they have different resolution and GSD than the low-resolution images in the artificial datasets used for training the super-resolution networks.
- Proba-V original real-life test dataset that can be used for calculating numerical metrics.

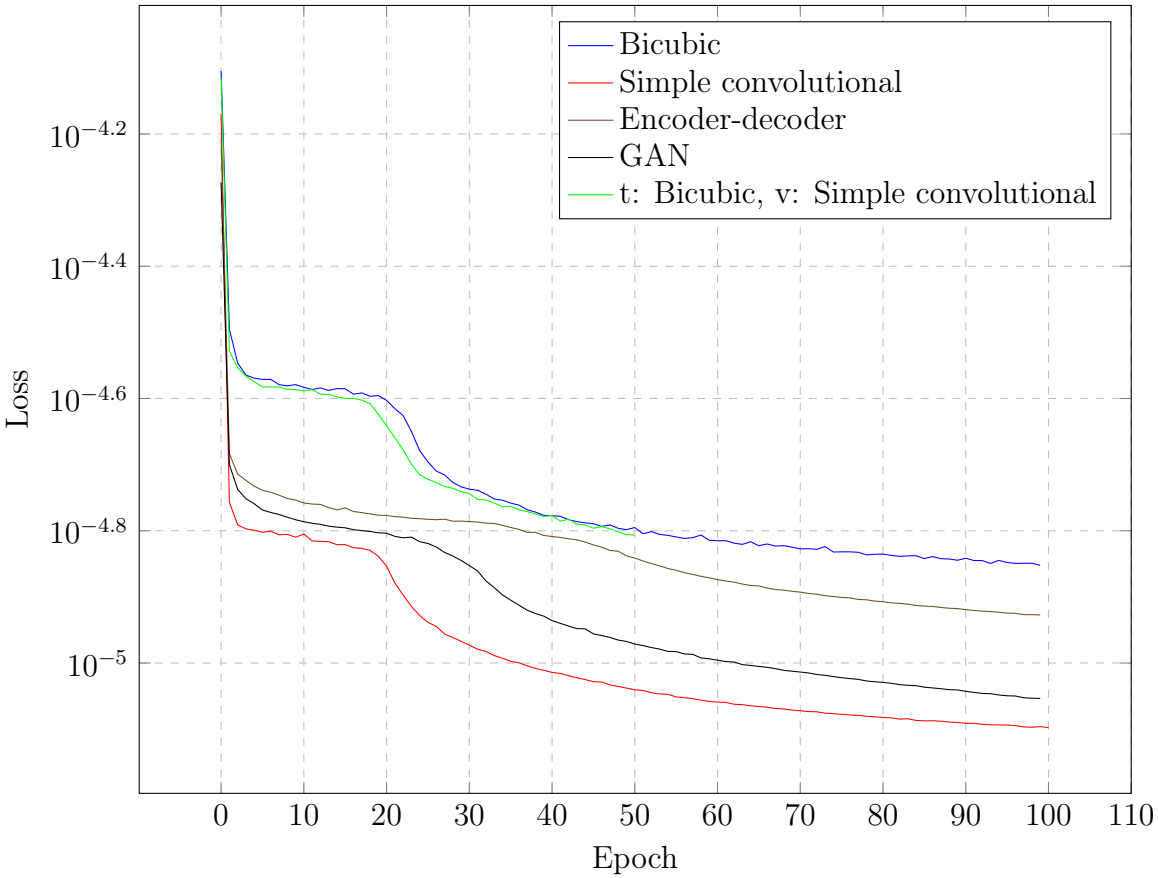


Figure 5.3: HighRes-net training loss history with training sets created in different ways

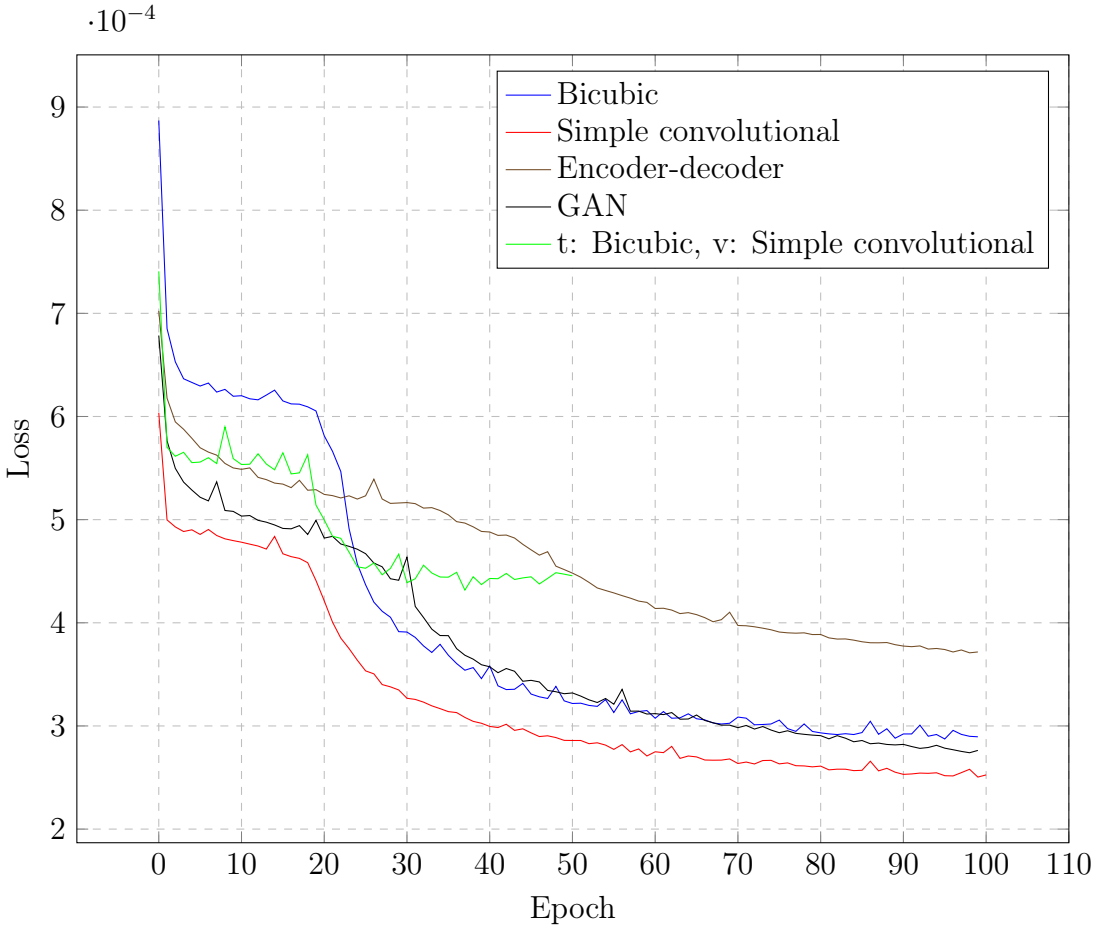


Figure 5.4: HighRes-net validation loss history with training sets created in different ways

Table 5.1: Evaluation of super-resolution training on different test sets (cPSNR metric, the larger, the better)

		Training data				
		Bicubic	Simple conv	Encoder-decoder	GAN	t: Bicubic v: Simple conv
Test data	Bicubic	**35.78**	28.96	24.55	25.80	35.26
	Simple conv	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
	Proba-V	43.36	42.02	40.43	40.91	**43.51**

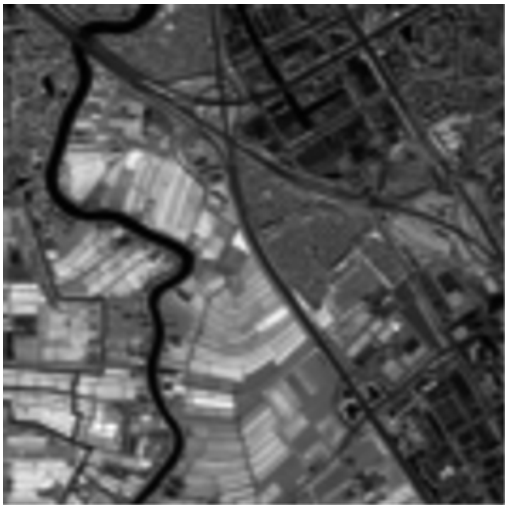
It should be noticed, that the latter two test sets differ in the GSD parameter from the synthetic low-resolution images in the Sentinel-2 training datasets. This data is still valid for evaluation and investigation because it is desirable that super-resolution and machine learning algorithms in general work on objects of any scale and size. Limiting the tests to a single GSD would draw an incomplete picture of the results. The results of the evaluation are presented in 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 the training and validation datasets respectively. The cPSNR score was used as the super-resolution evaluation metric.

The best results are observed along the diagonal of the table, which means that the super-resolution model works best on test data created in the same way as training data for a given model. A visual demonstration of such an evaluation can be seen in Figure 5.5. In this case, HighRes-net was trained on the dataset created by the simple convolutional network and evaluated on the test subpart of the same dataset. An example of image upscaling with bicubic interpolation was included in the figure as a reference point. The best results along the diagonal are expected and indicate that no gross mistakes were made in the course of the work. Each network works best on the data created in the same way as its training set. The super-resolution network trained using data generated by the simple convolutional network recreates the original high-resolution images best (with cPSNR of 36.15).

As stated in Chapter 3, an additional set of original real-life Sentinel-2 images is available. This set includes single-resolution images with different GSD than the synthetic training datasets. Since this set does not feature high and low-resolution image pairs, it can be used only for visual examination. Samples for such visual evaluation are shown in Figure 5.6. Some mosaic or checkerboard-like artifacts can be observed in the super-resolved images. As seen in the samples, some form

(a) One of nine low-resolution images

(b) Super-resolution reconstruction of the scene



(c) Upscaling to high-resolution with bicubic interpolation

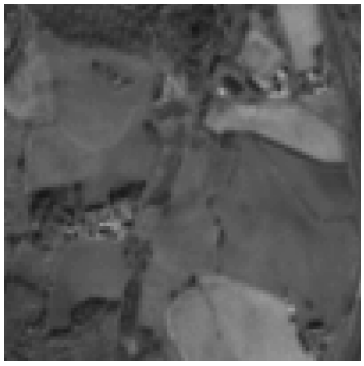
(d) Real-life high-resolution picture of the scene

Figure 5.5: A visual example of evaluation of an augmented dataset; both the training and test datasets were generated using the simple convolutional network

of artifacts occurred for all the networks on real-life Sentinel-2 data (with higher GSD than the training low-resolution images).

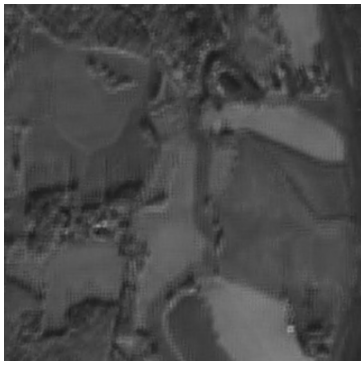
The results in Table 5.1 and Figure 5.6 can be summarized in the following statements:

- 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 (achieves best results for the real-life Proba-V test subset).
- At the moment, the bicubic interpolation gives slightly better results for the real-life Proba-V test subset. Possible reasons for that and suggestions of improvement are included in the summary.
- The cross-data-type-validation proves that all trainings are prone to overfitting to the data generation techniques. Early stopping based on different datasets can lead to this conclusion. Using cross-data-type-validation resulted in a network that was trained for fewer epochs, yet performed slightly better on the real-life Proba-V test subset.
- The generalization capabilities of the networks trained on synthetic data pose a problem. Artifacts on real data can be observed.



(a) One of the low-resolution images of the test scene

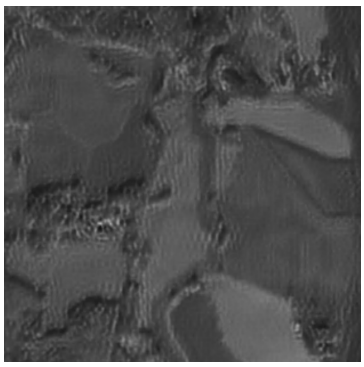
(b) Upscaling to high-resolution with bicubic interpolation



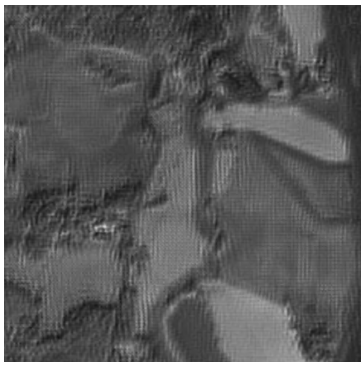
(c) HigRes-net trained on data created with bicubic interpolation

(d) HigRes-net trained on data created with bicubic interpolation; training done with cross-data-type validation

(e) HigRes-net trained on data created by the simple conv network



(f) HigRes-net trained on data created by the encoder-decoder network



(g) HigRes-net trained on data created by the GAN network

Figure 5.6: A visual evaluation of super-resolution results on real-life Sentinel-2 test data. Images are super-resolved by HighRes-net trained on datasets created in different ways. One of the original low-resolution images and a high-resolution one created by upscaling with bicubic interpolation are added for reference.

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 generate new datasets to fit the super-resolution algorithm. The mid-process evaluation step was performed to investigate levels of similarity between synthetic and real-life data. The generated data resembled real Proba-V images with the SSIM score over 0.9. The artificial datasets were used to train the HighRes-net neural network model. Then, the super-resolution models were evaluated on various synthetic and real-life images; both in a numerical and visual way. The cross-data-type-based validation approach to training supervision was confirmed to be valuable and possible thanks to the variety of the generated data. The network trained with this kind of validation mechanism was stopped earlier, yet scored better on the real-life Proba-V dataset. After visual investigation on the separate Sentinel-2 data, presence of checkerboard-like artifacts was noted. The deep learning based-methods proved to be of utility, yet slightly below what would be expected, since they did not surpass the bicubic interpolation approach for the real-life Proba-V test subset. There may be a variety of reasons for that to explore. However, the deep learning-based datasets were used to perform the cross-data-type validation scheme.

The field of data augmentation and generation for super-resolution is yet to be further investigated in future research. Many possibilities were not explored in a full way; translations between low-resolution images may be investigated in greater detail. Modeling them after the 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. Since the generalization capabilities remain an issue, more emphasis may be put on regularization. Adding more noise to augmented images may lead to less overfitting and artifacts generation. More

trainings may be conducted on mixed datasets. The possibility to create datasets that include images created in different ways remains open. The real-life images may be mixed with the synthetic ones (this approach would be similar to the traditional data augmentation approach). Better results could also be achieved by connecting augmentation and super-resolution networks into one. Such a network would generate data on the fly during the training process like in [**?**].

The observations connected with the cross-data-type-validation approach are worth being noticed. Early stopping training based on this approach leads to slightly better results compared to the normal trainings. 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 have been made.

There is an opportunity for further development of the work and a more detailed investigation. Achieving robustness and greater generalization of super-resolution techniques through better data augmentation is an important step towards the productization of this technique. A super-resolution algorithm independent of satellite-data type would be useful in image processing pipelines 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 Listing 3.

```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 not supported for GAN
  earlystopping: false
  stopping_delta: 0.0
  stopping_patience: 0
  preview: true
```

Listing 3: Contents of the *params.yaml* file used for training

Appendix B

Technical documentation

As mentioned in Section 3.3.2 trainings are done either done with the aid of the DVC system or by running scripts manually. The trainings done automatically by the DVC system contain the word *dvc* as the experiment name. The status of these trainings is supervised by DVC, current progress can be checked with the `dvc status` command. Trainings are automatically rerun or recached based on configuration files after running the `dvc repro` instruction. Manual trainings are run by invoking Python directly. This can be done with the 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.

Validation on Proba-V dataset can be run with the 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-2 datasets can be created by running the 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).

Appendix C

Supplementary media

The supplementary media include:

- This document.
- Source code files.
- The datasets can be fetched from the cloud using DVC; to gain access to the data contact the author for the passcodes.

Acronyms

ADAM Adaptive Moment Estimation. 42

DeepSUM Deep Neural Network for Super-Resolution of Unregistered Multitemporal Images. 15

DVC Data Version Control. 33, 34, 65, 69, 71

GAN Generative Adversarial Network. 12, 14, 38–43, 45, 46, 49, 52, 53, 55–57, 60, 73, 75, 77, 79

GPU Graphics Processing Unit. 10, 32

GSD Ground Sampling Distance. 8, 27, 29, 54, 57, 59

MAE Mean Absolute Error. 12, 40

MFSR Multi-Frame Super-Resolution. 15, 16

MSE Mean Squared Error. 12, 40

NIR Near-Infrared Light Spectrum. 27, 28, 36, 37, 40, 73

PNG Portable Network Graphics. 8

PReLu Parametric Rectified Linear Unit. 16, 17

PSNR Peak Signal-to-Noise Ratio. 12, 13, 50

RAMS Residual Attention Multi-Image Super-Resolution. 15, 18

RED Red Light Spectrum. 27, 37, 73

ReLu Rectified Linear Unit. 38, 40

RMSprop Root Mean Square Propagation. 42

SIFT Scale-Invariant Feature Transform. 14

SSIM Structural Similarity Index Measure. 13, 40, 42, 43, 45, 46, 50, 75

SURF Speeded-Up Robust Features. 14

TIFF Tagged Image File Format. 8

List of Figures

List of Tables

List of Algorithms

List of Listings