Politecnico di Torino

EURECOM
Sophia Antipolis

Double Master's Degree in Data Science and Engineering

ARTIFACTS SEGMENTATION WITH CONVOLUTIONAL NEURAL NETWORKS FOR SMARTPHONE CAMERA IMAGE SIGNAL PROCESSOR

Supervisors
Prof. Andrea BOTTINO
Prof. Maria ZULUAGA
Dr. Abdelhadi TEMMAR

Candidate
Andrea GHIGLIONE

Academic Year 2022 - 2023

To my family, friends and teachers,
who always believed in me and were part of a journey I will never forget.

Ad maiora semper !

Abstract

In recent years, deep learning has garnered significant interest due to its ability to achieve state-of-the-art results across a wide range of applications, including image processing. By leveraging powerful algorithms and vast amounts of data, deep learning has produced remarkable results in tasks such as image classification, object detection and semantic segmentation, through Convolutional Neural Networks architectures. This is due to their capabilities in learning high-level representations of the data, capturing the underlying patterns and extracting features from the images. As a result, traditional image processing algorithms have been outperformed by Artificial Intelligence techniques in terms of accuracy and efficiency, and researchers have been able to tackle more challenging and complex tasks with greater ease.

At Huawei Nice Research Center the teams are working intensely on the smartphone camera image signal processor, developing efficient solutions which can be integrated into the next generations of smartphones. This thesis work aims to demonstrate how Convolutional Neural Networks can be used to detect image artifacts at the pixel level that may be created in the image signal processor. First, by focusing on the demosaicing and ghosting artifacts, which are known to have a significant impact on image quality, the study builds the datasets labels for the training of deep learning models capable of detecting these artifacts, in a weak-supervised scenario. Then, the models performances are rigorously assessed on a variety of images, ensuring their robustness and testing their generalization capabilities. The ultimate goal is to use the trained models to create a useful metric based on artifacts detection, which can be used to assess the quality of images and to aid in the development of more effective image processing tools. Achieved results are promising since the trained models are using a limited number of parameters and they are able to correctly detect common image artifacts with a low inference time. The models have also been successfully used to optimize the parameters of the well-known Malvar-He-Cutler demosaicing algorithm which is an important improvement that may be applied to smartphone camera image signal processor.

Résumé

Ces dernières années, le deep learning a suscité un intérêt considérable en raison de sa capacité à obtenir des résultats exceptionnels dans toutes sortes d'applications, y compris le traitement et l'analyse d'images. En tirant parti d'une base de données d'images, le deep learning a produit des résultats remarquables, en utilisant entre autres des réseaux neuronaux convolutionnels, dans des tâches telles que la classification d'images, la détection d'objets et la segmentation sémantique. Ceci est dû à leur capacité à saisir les modèles sous-jacents et à extraire les caractéristiques de haut niveau/bas niveau des images. En conséquence, les algorithmes traditionnels de traitement d'images ont été surpassés par les techniques d'intelligence artificielle en matière de résultats et d'efficacité, et les chercheurs ont été en mesure de s'attaquer à des tâches plus difficiles et plus complexes.
Au centre de recherche Huawei Nice, les équipes travaillent intensément sur le processeur de signal d'image de la caméra de smartphone, développant des solutions efficaces qui peuvent être intégrées dans les prochaines générations de smartphones. Ce travail de thèse vise à démontrer comment les réseaux neuronaux convolutifs peuvent être utilisés pour détecter des artefacts dans les images créées par certains algorithmes. Tout d'abord, en se concentrant sur les artefacts de demosaicing et de ghosting, qui sont connus pour avoir un impact significatif sur la qualité de l'image. Une première partie de cette thèse a consisté à construire automatiquement le dataset annoté pour l'entraînement de modèles de deep learning capables de détecter ces artefacts, dans un scénario weekly supervised. Ensuite, les performances des modèles sont rigoureusement évaluées sur une variété d'images, en assurant leurs robustesses et en testant leurs capacités de généralisation. Le but ultime est d'utiliser les modèles entrainés dans le but de créer une métrique pour évaluer la qualité des images et pour ainsi aider à la mise au point d'outils de traitement d'images plus efficaces. Les résultats obtenus sont prometteurs puisque les modèles entraines utilisent un nombre limité de paramètres et sont capables de détecter correctement des artefacts avec un temps d'inférence faible. Les modèles ont également été utilisés avec succès pour optimiser les paramètres de l'algorithme de demosaicing bien connu Malvar-He-Cutler. Ce qui est une amélioration importante et peut-être appliquée pour améliorer les algorithmes de caméra de smartphone.

Table of Contents

Table of Contents **I**

List of Figures **III**

List of Tables **V**

1 Introduction **1**
1.1 About Huawei . . . 1
1.2 Nice Research Center . . . 2
1.3 Thesis Overview . . . 3
1.4 Hardware, Software and Programming Tools . . . 4

2 Demosaicing artifacts **7**
2.1 Overview . . . 7
2.2 Data and Dataset creation . . . 7
2.2.1 Gamut conversion . . . 9
2.2.2 Mosaicing . . . 10
2.2.3 Demosaicing . . . 11
2.2.4 Labels creation . . . 12
2.2.5 Training-Validation-Test Split . . . 17
2.3 Segmentation network . . . 18
2.4 Training . . . 18
2.5 Results . . . 22

3 Ghosting artifacts **25**
3.1 Overview . . . 25
3.2 High Dynamic Range images . . . 25
3.3 Data and Dataset creation . . . 26
3.3.1 Merging exposures . . . 26
3.3.2 Labels creation . . . 28
3.4 Training . . . 31
3.4.1 Data Augmentation . . . 32
3.4.2 Network . . . 33
3.5 Results . . . 33

4 Joint network **38**

4.1 Architecture 38
4.2 Training 38
4.3 Results 39

5 Optimization **42**
5.1 Training 44
5.2 Results 46

6 Conclusions and future work **51**

Bibliography **55**

A Appendix **58**
A.1 Role of luminance in 3D sRGB Gamut 58
A.2 Additional equations in Malvar-He-Cutler algorithm 59

B Appendix **60**
B.1 From Tversky Index to Dice Score 60

C Appendix **62**
C.1 Malvar-He-Cutler initial kernel values 62

List of Figures

1 Map of Huawei globalized resource deployment 1
2 Evolution of economic fingerprint of Huawei France from 2012 to 2021. . . . 3
3 Example of DNG image and renditions from the five experts 8
4 Gamut of standard RGB 10
5 Gamut of ProPhoto RGB 10
6 Bayer filter 11
7 Profile and cross-section of photosensors 11
8 Examples of bilinear interpolation of red at a blue location, green at a red location, and blue at a red location respectively. Highlighted pixels are averaged in order to get the interpolated value. 12
9 The 5×5 linear filters. The coefficient values are scaled by 8 13
10 Application of mosaicing and demosaicing on input image 14
11 Example of difference between original image (left) and interpolated image (right) 14
12 First criteria that is used to label a pixel as demosaicing artifact 15
13 From left to right: original raw label, label after closing, label after opening . 16
14 Example of how using the adaptive threshold improves the results in noisy images 16
15 From left to right: original scene, artifacts generated, labels 17
16 Our U-Net architecture 19
17 First training crops sampling criteria we implemented 20
18 Second training crops sampling criteria we implemented 20
19 Examples of good demosaicing's artifacts segmentation 23
20 Examples of bad demosaicing artifacts segmentation. On the left, there are False Positives, and on the right there are False Negatives. 23
21 Examples of how the input quality of an image can fool the network. 24
22 Feature maps at third U-Net decoder for input image with and without artifacts. 24
23 Examples of merging exposures and tone mapping 26
24 Merging pair of exposures with movements across each other 27
25 How the difference between two exposures is computed 29
26 Low luminance mask computation and raw detection of ghosting in dark and medium brightness areas 29
27 Visual example of how the raw label is created by exploiting previous steps and luminance information 30
28 From left to right: merged image, approximation of raw label, improved raw label with the pre-trained network, closing and opening operations. 31

29 Examples of raw ghosting artifacts labels. 32
30 Data augmentation criteria applied for Kalantari's dataset. Each x_i is sampled from a uniform distribution with range $[0, 1]$. 34
31 Examples of ghosting prediction on clean images. 35
32 Examples of ghosting prediction on images with artifacts. 36
33 Feature maps at third U-Net decoder for input image with and without artifacts. 37
34 Joint U-Net architecture. 39
35 Effectiveness of joint network with respect to the demosaicing network. . . . 40
36 Effectiveness of joint network with respect to the ghosting network. 41
37 How, given a Bayer input, the RGB image is obtained through Malvar-He-Cutler. 42
38 Comparison between trained model result (left) and Ground Truth image (right) 47
39 Comparison between trained model result (left) and original MHC result (right) 47
40 Comparison between the trained model result (left) and the model trained with L1 loss term only (right) 48
41 Comparison between the demosaicing artifacts prediction with an RGB input which comes from the standard MHC kernels (left), and with an RGB input which comes from the optimized version of the MHC kernels (right). 49
42 Optimized kernel parameters for the MHC demosaicing algorithm 50
43 3D sRGB Gamut, with the role of luminance 58
44 Binary classification problem through Set Theory 60
45 Dice Score representation in terms of sets 61

List of Tables

1 Top 10 categories' sets in terms of number of images in MIT-Adobe FiveK Dataset 8
2 Set of hyperparameters tried in training the demosaicing segmentation network 21
3 Set of hyperparameters tried in training the ghosting segmentation network . 34
4 Set of hyperparameters tried in training the joint segmentation network . . . 38
5 Results in terms of Dice Score for the joint model in different settings. 41
6 Set of values tried in training the optimization network 46

Chapter 1

1 Introduction

1.1 About Huawei

Huawei Technologies Co. Ltd. is a Chinese private company founded the 15 September 1987 in Shenzhen in China. Its native name is 华为 which means *Chinese Achievement*. Huawei is a leading Information and Communications Technology (ICT) solutions provider and it operates in a variety of industries such as Telecommunications, Electronics, Semiconductors, Artificial Intelligence, Automation, and Cloud Computing. It is the second biggest global investor in Research and Development (R&D) and its products and services are used in more than 170 countries, serving over one-third of the world's population as reported in Figure 1.

Figure 1: Map of Huawei globalized resource deployment

In particular, Huawei has more than 80 laboratories and 15000 researchers across the world and it invests approximately 15% of the company's turnover in Research, with a peak of 22.4% in 2021. Huawei is also helping the most promising tech start-ups across the world. In France, they created a project called *Digital InPulse* in 2014 which consists of financial

help to the start-ups that performed better in public competitions organized every year, and it also consists of business trips in China in order to discover the Chinese technology ecosystem. Up to 2022, Huawei has helped almost 90 start-ups with a total of more than 2.6 million euros. This project is organized with local french organizations and its theme changes every year, and it may be IoT, Big Data, Smart City, Mobility, Well-being, and Green Tech. Huawei France is involved in other big projects such as *OpenLab* where they work on development and innovation with their partners on different topics (up to today, more than 50 projects have been realized). One of the most impactful examples is the work that Huawei did with *Orange* establishing a new world record for transmission of 157 Tbit/s on a fiber of 120km. Huawei is also working for the environment, making sure that future technologies will be fully-sustainable. They are working very hard on reducing the impacts of their infrastructures of communications, and they are also pushing their clients' companies to do the same. In particular, Huawei is designing infrastructures at high density to lower heat dissipation and they are reducing the data volume, which needs to be processed and requires a lot of energy. This process is already providing good results as Huawei has produced 84.2 kWh of green electricity, reducing carbon dioxide emissions by 23.8 million tons. Finally, Huawei is also including young and talented students through several projects such as *Talents Numériques* and *Huawei ICT Academy*.

1.2 Nice Research Center

In France, there are 5 Huawei Research and Development centers; *Aesthetics* in Paris, *Mathématiques, Algorithms and Software* and *Standardisation industrielle* in Boulogne Billancourt, *Capteurs and Santé* in Grenoble and the *Nice Research Center* (NRC). These centers have been experiencing a strongly positive evolution in recent years as shown in Figure 2.

Nice Research Center is the second biggest establishment of Huawei in France, after the Paris site. It is located in Mougins, Sophia-Antipolis, and it was founded in 2013 by Mr. Stephen Busch, who picked a team of 9 talented people whose goal was to develop a complete Image Signal Processor (ISP) which will be released in 2016. The ISP was integrated into the P9 smartphone, and this success allowed the center to grow into a team of 30 people, from 14 different countries. Employees at Huawei NRC contributed heavily to the progress in the ISP domain, which role has increasingly become crucial and complex. Image processing algorithms are updated constantly and the progress in research allows new possibilities for smartphone camera features. In recent years in Nice Center, a lot of work has been done in the Artificial Intelligence (AI) domain in order to achieve even better results. However, AI is a field very hungry for data and energy, which is a physical limitation to the

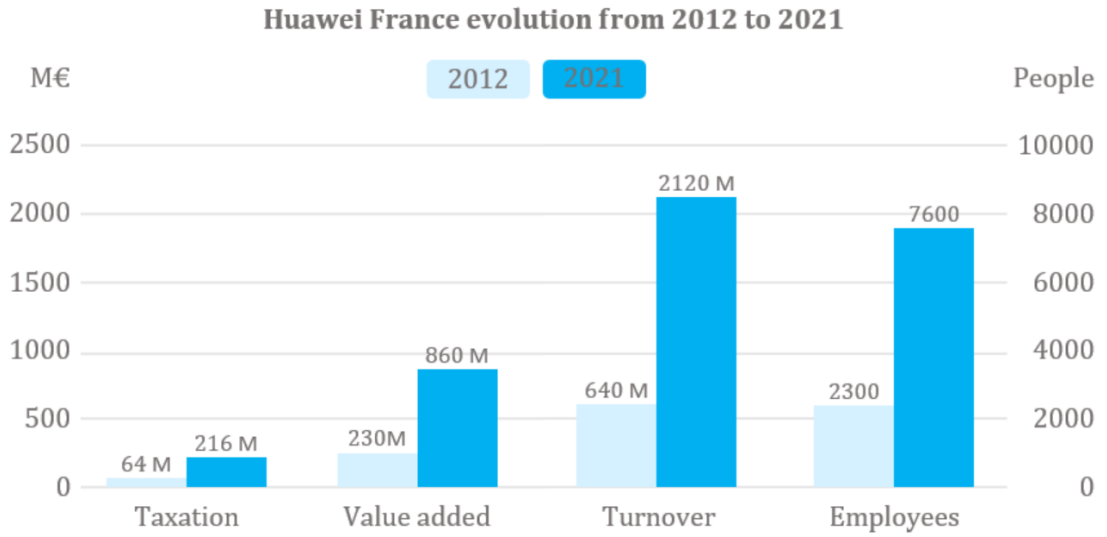

Figure 2: Evolution of economic fingerprint of Huawei France from 2012 to 2021.

hardware. Since progress is always more difficult to accomplish, the main focus in the last years was on developing and applying state-of-the-art Computer Vision algorithms in the ISP domain, but also to make these algorithms highly efficient in order to run on a smartphone without exceeding the thermal budget of 4 Watts. Overall, NRC's focus moved from full Development to a balance between Development and Research throughout the years, mixing business culture and university culture, with several opportunities as Master's Internships and Doctorate projects.

1.3 Thesis Overview

After Image Signal Processing (ISP) pipeline, it is very common to have images with *artifacts*, which are features that did not appear in the original scene. They are also known as errors in image representation or perception from the human visual system. These wrong areas in the image may be very small and difficult to detect at first sight, and it makes it more complicated to evaluate the performance of image processing algorithms automatically. The goal of this project is to create datasets of images with artifacts and to build a Convolutional Neural Network (CNN) which is able to segment different kinds of artifacts as demosaicing artifacts (false color, zippering, etc.) and ghosting artifacts automatically and quickly. Results of the CNN are then going to be used as a further metric to assess image

quality since methods like *Peak Signal-to-Noise Ratio* (PSNR) or *Structural Similarity Index Measure* (SSIM) are not extremely precise at estimating quality in small regions of an image. In addition, the trained demosaicing CNN will be used to optimize the well-known *Malvar-He-Cutler* demosaicing algorithm. To our knowledge, currently there is no competitor on this task, and for this reason we will not be able to benchmark our results against any available model. In particular, this project is more oriented to be a proof of concept in the image processing research domain.

The project can be divided into different steps: the first one consists of creating a dataset with *demosaicing* artifacts. In particular, the selected dataset is MIT-Adobe FiveK dataset [1]. First, the images are mapped into the Bayer domain, then demosaicing is applied to these images through the Malvar-He-Cutler algorithm, and the artifacts are automatically annotated. The following step is building a segmentation model to provide pixel-wise labels of the artifacts at full resolution since Deep Learning models have been proven to perform extremely well on Computer Vision tasks. Finally, the trained segmentation network is validated on the test set, which was obtained with a split on the Adobe FiveK dataset. Once these steps are completed, we introduced a new artifact: *ghosting*. This has been done by using the Kalantari dataset [2] which contains a set of exposures for different images. Again, labels are created automatically, a convolutional neural network is trained on them, and its performances are assessed. Then, the two networks are joined in order to have a unique model which is able to segment precisely both artifacts with a low inference time. Finally, the demosaicing network results are used as a further metric for optimizing the kernel parameters of the Malvar-He-Cutler demosaicing algorithm.

1.4 Hardware, Software and Programming Tools

All the work has been done either locally or on a remote machine in order to exploit an external Graphics Processing Unit. The local machine is a ThinkPad with an Intel(R) Core(TM) i7 processor. The RAM installed is 16.0 GB and the system type is characterized by a 64-bit operating system (Windows 10 Pro), x64-based processor. The remote machine has Windows operating system as well, and it provides a powerful GPU; Nvidia Tesla P100-PCIE, 16GB. The chosen programming language was Python 3.9 due to its usability in a Deep Learning context. A variety of libraries have been used, such as:

- **argparse**: It is a parser for command-line options, arguments, and sub-commands. It has been used since it allows running Python scripts with command line parameters through command line [3].

- **colour**: It is a library for manipulating common color representation such as RGB,

HSV, etc. in an easy and pythonic way [4].

- **cv2**: OpenCV is the most used library for Computer Vision tasks in Python. It allows extremely efficient ways to load, store and manipulate images in a variety of formats [5].
- **Matplotlib**: It is a well-known Python library for creating static, animated, and interactive visualizations in Python, conveying results in a compact way [6].
- **NumPy**: It stands for Numerical Python and it offers a comprehensive set of routines for handling numerical problems and complex data structures in an easy way [7].
- **os**: It provides a portable way of using operating system-dependent functionality, such as opening files, manipulating paths, etc [8].
- **pathlib**: It handles different operations related to the file system, allowing modification and creation of different files and folders in a specific path in Python [9].
- **PIL**: It is the Python Imaging Library and it includes a set of image processing capabilities, extensive file format support, and an efficient internal representation [10].
- **SciPy**: Scientific Python library that provides utility functions for integration, optimization, statistics, etc [11].
- **shutil**: It consists of a set of high-level efficient operations on files and collection of files [12].
- **skimage**: Scikit-Image is a collection of algorithms for image processing that often includes some uncommon models that may be useful in specific scenarios [13].
- **time**: It provides time-related functions and it is extremely helpful for comparing algorithms' speed or testing deep learning model's inference time [14].
- **torch**: It is the core library of this project. Torch is an open-source library for Deep Learning, which allows building Deep Neural Networks in a user-friendly way. It is extremely flexible since operations on each network layer are easy and more pythonic than other Deep Learning libraries. It was installed with **CUDA** [15] to allow GPU programming [16].
- **torchvision**: It is part of the PyTorch project and it consists of popular datasets and model architectures for Computer Vision [17].

- **urllib**: It is a collection of several modules for working with Uniform Resource Locators (URLs) [18].

Besides these libraries, we also used a software called **Beyond Compare 4** [19] which allows comparing two images at a pixel level, according to a certain threshold. It works for a variety of images format and it includes many useful features such as blending and swapping, that helps when looking at artifacts predictions.

Chapter 2

2 Demosaicing artifacts

2.1 Overview

The first artifact type that we worked on is the *demosaicing* artifact. It is an error that occurs at the pixel level when interpolating an image from the raw domain (i.e. how the light was captured from the camera's sensors) to the RGB domain. This process may introduce problems in the image such as zippering, false colors, etc. which will decrease the image quality. We were not provided with data containing these problems, hence we replicated locally the issue starting from clean RGB images available in an open-source dataset. Overall, demosaicing algorithms which interpolate raw inputs are working very well, however, they introduce small artifact regions whose detection will be our focus in this section.

2.2 Data and Dataset creation

The dataset we used is the MIT-Adobe FiveK Dataset. It contains 5000 different-shape photos taken from Single Lens Reflex (SLR) cameras, which are cameras using a system made of a mirror and a prism, allowing the possibility to see what will be captured through the lens. In particular, all the images are in RAW format, preserving all the information captured by the camera's sensors. The available data archive consists of the following:

- 5000 images in DNG format, which stands for *Digital Negative* and it is characterized by being *raw, open* and *lossless*. It is open since is licensed through an open license and its specification can be used and implemented by anyone. The raw format is instead well-known for containing data that has been minimally processed, allowing for almost full-information preservation. Finally, it is lossless since the data can be reconstructed from its compressed version without loss of detail.

- 5 renditions per image made by five experts from an art school. Each of them adjusted the photos through Adobe Lightroom which is a photo-adjustment software on which they had a lot of processing skills. They retouched each photo according to their visual pleasure, so the lack of an objective metric allowed us to have a good range of visual exposures per image. Experts' names are *Todd Carroll, David Mager, Jaime Permuth, LaNola Katheleen Stone* and *Damian Wampler*, which will be called experts A, B, C, D, and E for the sake of brevity.

- Semantic information about each photo, as reported in Table 1. In particular, there is information about the subject, light, location, time, and camera that took the picture. We did not report the camera information as the majority of the photos have been taken with a different camera setting. For each image, there are 5 high-quality renditions made by the five experts. We selected the first 2000 images for computational reasons, rendered from expert C since they are visually more similar to the DNG images as reported below (3).

Subject	**Light**	**Location**	**Time**	**Number of images**
Man-made object	Sun or sky	Outdoors	Day	1193
Nature	Sun or sky	Outdoors	Day	848
Person(s)	Sun or sky	Outdoors	Day	692
Person(s)	Artificial	Indoors	Unknown	360
Animal(s)	Sun or sky	Outdoors	Day	239
Man-made object	Artificial	Indoors	Unknown	210
Unknown	Mixed	Unknown	Unknown	112
Unknown	Mixed	Outdoors	Unknown	104
Man-made object	Sun or sky	Outdoors	Dawn or dusk	86
Nature	Sun or sky	Outdoors	Dawn or dusk	84

Table 1: Top 10 categories' sets in terms of number of images in MIT-Adobe FiveK Dataset

Figure 3: Example of DNG image and renditions from the five experts

2.2.1 Gamut conversion

The dataset images' renditions were in *ProPhoto RGB* format (also known as ROMM RGB) so we had to map them to *sRGB* (standard RGB) format in order to process them with well-known Python libraries such as NumPy and OpenCV. The *RGB* term stands for Red, Green, and Blue respectively, and every color we can see is made by taking into account different amounts of red, green, and blue channels. However, channel information is necessary but not sufficient in identifying a unique color; further information that we need is the *color space* in which we are operating. ProPhoto RGB and sRGB are both color spaces: a specific set of colors obtained by taking two paints (e.g. red and blue) plus a white canvas and mixing the two paints. The difference between sRGB and ProPhoto RGB is the dimension of the *gamut* which is the range area of what human's eyes can see among all the colors. For the sake of simplicity, if we take a 2-dimensional representation of every color humans can see (not accounting for the luminance dimension) we can visualize the difference between the two gamuts in Figure 4 and Figure 5, while the luminance contribution can be found in Appendix A.1. The sRGB color space is the smallest and it is widely used on smartphones, web browsers, etc. for reading and displaying colors, since there are hardware limitations on different devices and a standard is required. In particular, choosing a small color space allows consistency through a variety of devices, at the cost of introducing a bigger limitation in the number of different colors that can be displayed. ProPhoto RGB includes the so-called *imaginary colors* which are not visible to humans. We mapped values from ProPhoto RGB to sRGB reducing also the bit depth from 16 bits per pixel to 8 bits per pixel, hence we reduced the number of combinations of different RGB values. We did that because many applications and the majority of Python libraries assume we are using a default color space, and so ProPhoto RGB pixel coordinates would be read wrongly. However, when it is possible, it is better to use ProPhoto as the *working* space, and sRGB as the *output* space. In order to convert from ProPhoto RGB to sRGB, first we convert the image's pixels to floating-point representation in the range $[0, 1]$ dividing by $2^{16} - 1$ which is the largest number that can be represented on 16 bits. After that, we decode with *RIMM-ROMM RGB* decoding function and then from that gamut to sRGB gamut using CAT02 conversion matrix (CAT stands for Chromatic Adaption Transform, which is a function emulating human's capability to adjust to changes in illumination in order to preserve the appearance of object colors). Finally, we encode with sRGB inverse Electro-Optical Transfer Function and we convert to integer representation in the range $[0, 255]$.

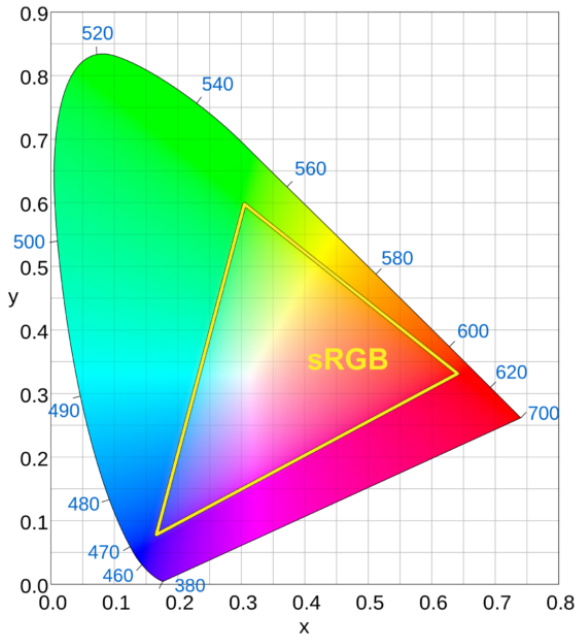

Figure 4: Gamut of standard RGB

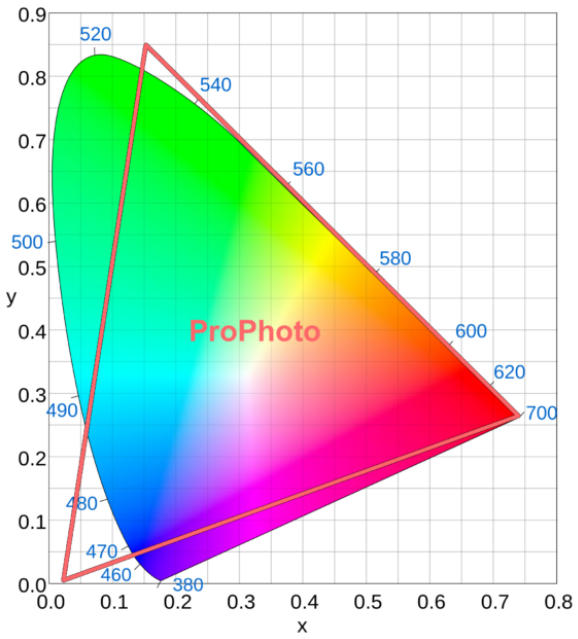

Figure 5: Gamut of ProPhoto RGB

2.2.2 Mosaicing

Mosaicing is a process that is applied to the RGB images in order to map them to the Bayer domain. In particular, this is the domain that is obtained after that a *Color Filter Array* (CFA) is applied on a grid of the camera's photosensors. There are different filters that can be used, however, we chose the well-known *Bayer* filter as shown in Figure 6 which is an RGB filter that allows capturing either red, green, or blue light's intensity for a specific pixel in the image (Figure 7). The photosensors are made of semiconductors and the camera's green photosensors are commonly called *luminance-sensitive* elements, while red and blue are called *chrominance-sensitive* elements.

The filter layer is hence composed of red, green, and blue filters which filter the light by the wavelength range, providing information about the light's intensity in the three different colors. The filter pattern is half green, one-quarter red, and one-quarter blue. Depending on the arrangement of these 4 color filters we can have BGGR, GBRG, GRBG, or RGGB Bayer filters. The Bayer pattern alternates *blue rows* (B, G, B, G, ...) and *red rows* (G, R, G, R, ...), using twice as many green as red or blue because the human eye is more sensitive to green light and the filters wish to mimic human eye physiology. The filter that we chose

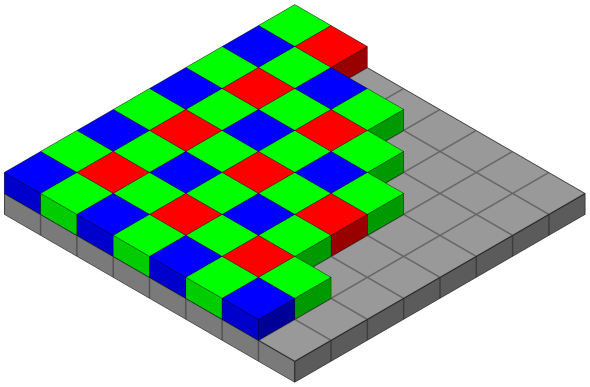

Figure 6: Bayer filter

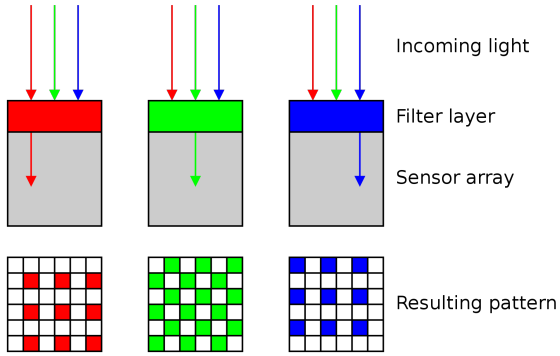

Figure 7: Profile and cross-section of photosensors

is BGGR which is also represented in Figure 7.

2.2.3 Demosaicing

The demosaicing process consists of interpolating the color value of the neighborhood's pixels in order to get the complete color information for a specific pixel, in an image that has been captured through a Bayer filter. There are different demosaicing algorithms, and we chose the well-known Malvar-He-Cutler gradient-corrected bilinear interpolation [20], proposed in 2004, which exploits a linear method with 5×5 filters. This method is a modification of the bilinear interpolation where, at a blue or red location, the green component of a pixel is derived by the average of its four axial neighbors (Equation 1), while red and blue components at a green location are derived similarly, but using the four diagonal components (Figure 8), due to the filter shape.

$$\widehat{G}^{bl}(i,j) = \frac{1}{4}(G(i-1,j) + G(i+1,j) + G(i,j-1) + G(i,j+1)) \tag{1}$$

To improve upon the quality of this simple method, Malvar, He, and Cutler add *Laplacian cross-channel corrections*. For example, the green component at a red pixel location is estimated as:

$$\begin{gathered} \widehat{G}(i,j) = \widehat{G}^{bl}(i,j) + \alpha\Delta_R(i,j) \\ \Delta_R(i,j) := R(i,j) - \frac{1}{4}(R(i-2,j) + R(i+2,j) + R(i,j-2) + R(i,j+2)) \end{gathered} \tag{2}$$

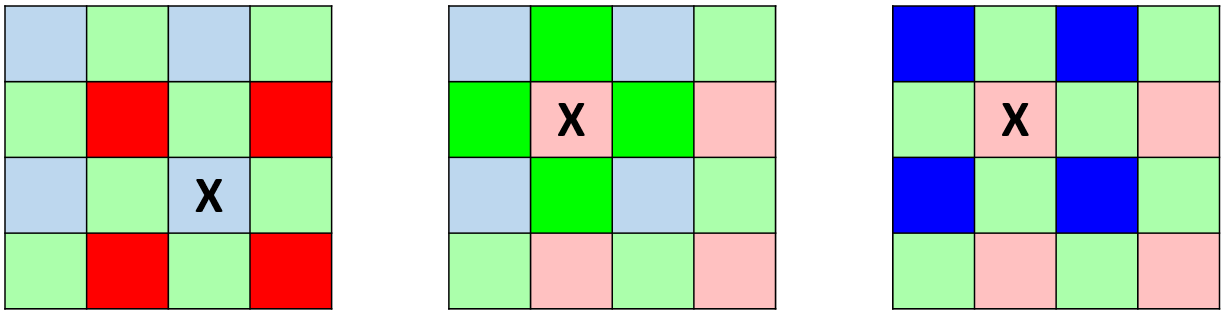

Figure 8: Examples of bilinear interpolation of red at a blue location, green at a red location, and blue at a red location respectively. Highlighted pixels are averaged in order to get the interpolated value.

where Δ_R is the discrete 5-point Laplacian of the red channel and α controls the weight of the Laplacian correction term. This term is about a different channel with respect to the one we are computing, and that is why the correction is called cross-channel. Further equations can be found in Appendix A.2. The 5×5 linear filters and their coefficient are represented in Figure 9.

In order to deal with the poor quality of interpolation on the borders of the images, we applied padding of size 5 with reflection. In this way, pixels on the border had better quality and after doing that, padding pixels were removed. An example of the results of the mosaicing and the demosaicing process is shown in Figure 10, where artifacts are particularly evident for visualization purposes.

2.2.4 Labels creation

Our goal at this point was to annotate automatically the artifacts. In Figure 11 it is evident that at first sight images obtained with demosaicing look identical to original scenes, however, zooming on the details we can clearly see that interpolation led to the introduction of artifacts in some image's areas.

In order to annotate automatically the artifacts the following steps are followed:

- Obtain a **raw approximation** of the labels: pixels of the original image are compared with the pixels of the image after the interpolation and if the value on at least one channel is higher than a default threshold, then the pixel is considered as part of an artifact (Figure 12). The comparison is pixel-wise and the chosen threshold was 20

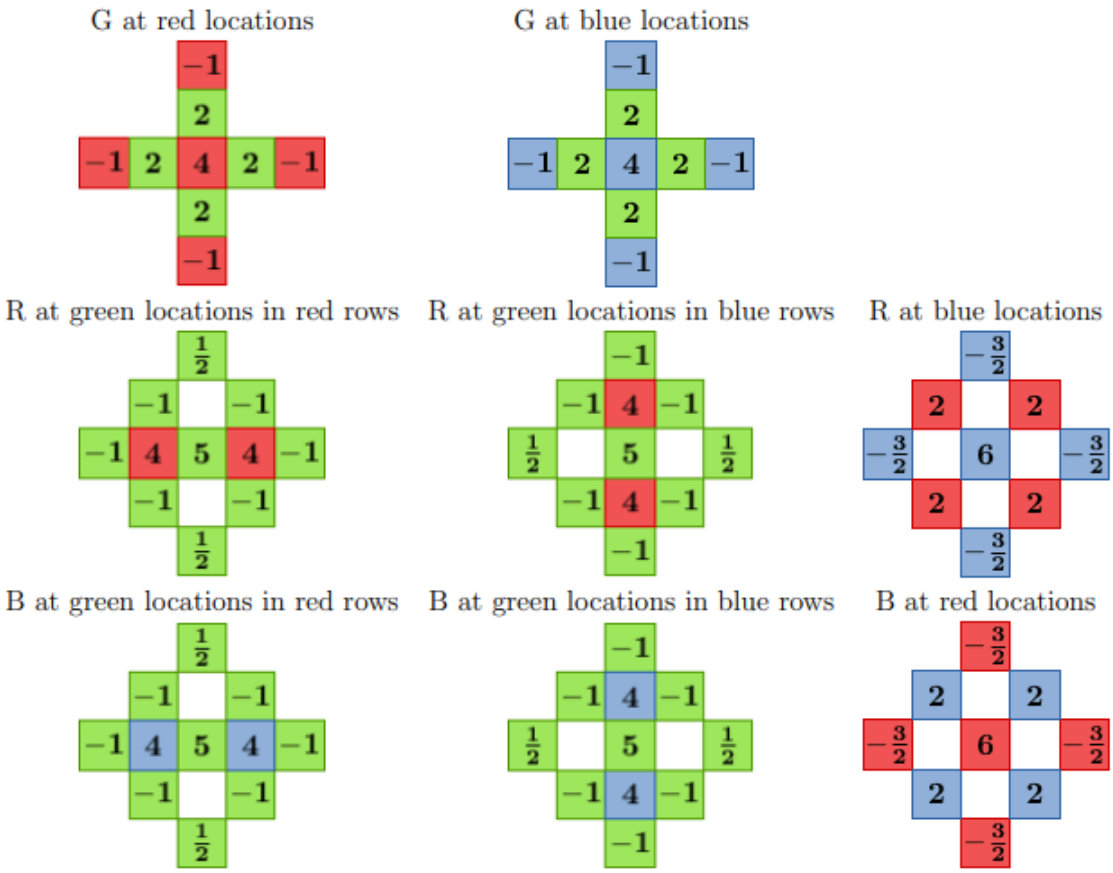

Figure 9: The 5×5 linear filters. The coefficient values are scaled by 8

since it was the value that allowed us to visually highlight the majority of the artifacts (difference between pixel values) in Beyond Compare 4.

- Apply **closing**: it consists of a *dilation* followed by an *erosion* and it is useful for having wider labeled areas since the first approximation is very noisy and not smooth. In particular, both dilation and erosion operations are computed through a kernel. Dilation consists of applying a convolution to an image through a kernel, and every pixel in the image region under the kernel is replaced by the maximum value of the same image's region. Since the image in our case is a binary label this means that if at least one pixel value in the label region under the kernel is 1, then all the area

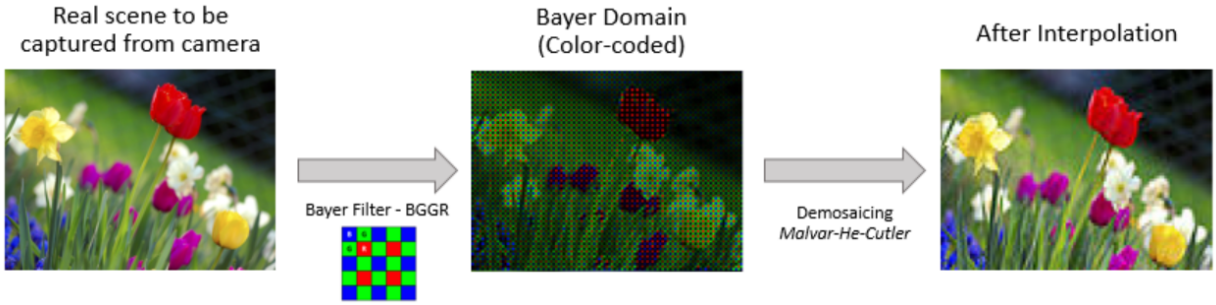

Figure 10: Application of mosaicing and demosaicing on input image

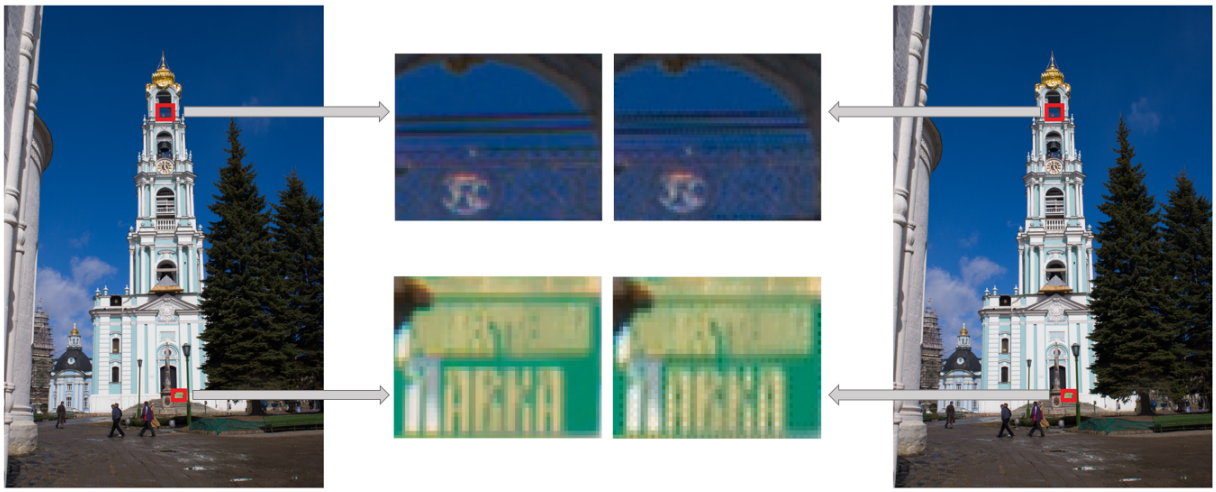

Figure 11: Example of difference between original image (left) and interpolated image (right)

under the kernel becomes 1. Erosion is similar to dilation, but it uses the minimum value instead of the maximum for the replacement of pixel values in the area under the kernel. Dilation and Erosion operations are reported respectively in the set of equations 3.

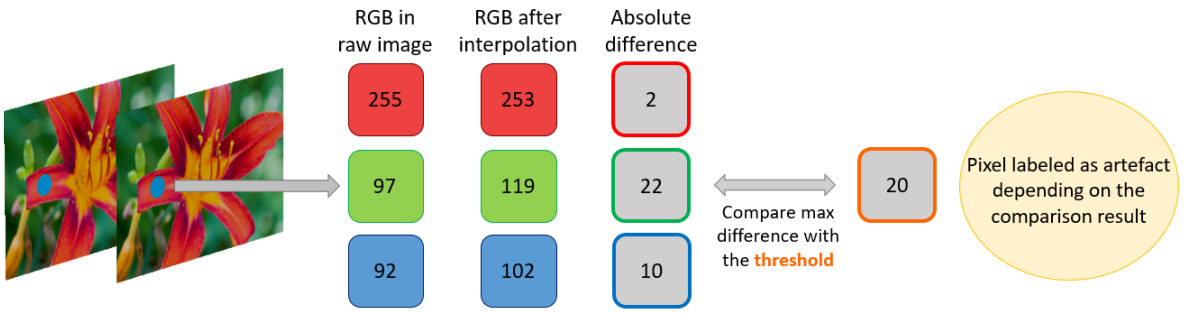

Figure 12: First criteria that is used to label a pixel as demosaicing artifact

$$dst(x,y) \overset{\text{dil}}{=} max_{(x',y'):element(x',y')\neq 0}\, src(x+x', y+y')$$
$$dst(x,y) \overset{\text{er}}{=} min_{(x',y'):element(x',y')\neq 0}\, src(x+x', y+y') \tag{3}$$

- **Remove** regions with few pixels: it is important to have labeled regions that do not contain too many sparse pixels since they carry little information alone and overall they are a contribution to noise in the label. We implemented two ways to do this: the first consists in applying *opening* which is the opposite of closing (erosion followed by dilation), and the second way is to apply the Region Proposals algorithm from *skimage* library and removing regions made of one pixel only. This algorithm simply clusters some possible regions of pixels, and we discarded the regions which were not relevant from an area point of view.

An example of this process is reported in Figure 13. These steps are done using a threshold of 20, a closing kernel with shape $(4, 4)$, and the Region Proposals algorithm, because it provided the best visual results. Once these operations are done, the percentage of the image which has been labeled is considered. This information is used for computing an adaptive threshold (Equation 4), adapted to the current image, in order to avoid labeling too many pixels in noisy images. Without doing that, noisy images would have had labels containing a lot of pixels labeled as artifacts, when in fact they are only noise. The adaptive threshold mitigated this problem and it allowed us to avoid having misleading label information. An example of the effectiveness of an adaptive threshold is reported in Figure 14.

Figure 13: From left to right: original raw label, label after closing, label after opening

$$th_{adaptive} = \left\lfloor th_{default} \cdot \left(1 + 10 \cdot \frac{pixels_{artifacts}}{pixels_{total}}\right)\right\rfloor \quad ; \quad th_{default} = 20 \tag{4}$$

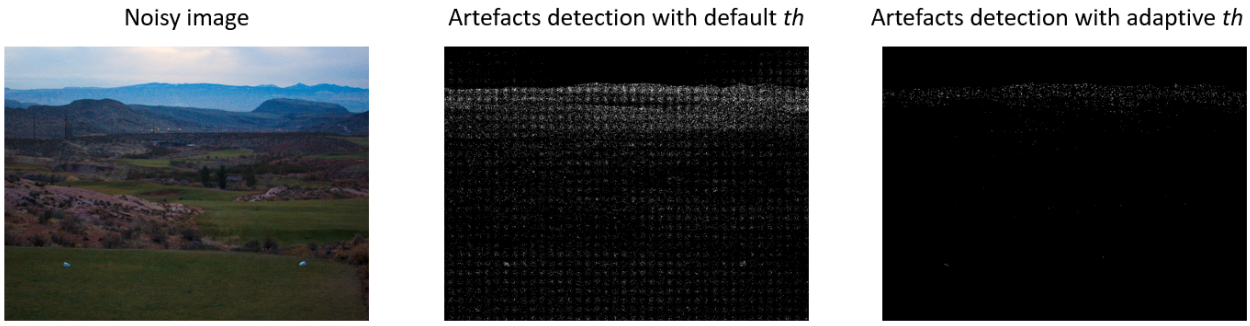

Figure 14: Example of how using the adaptive threshold improves the results in noisy images

The adaptive threshold has been computed with the aim of making the threshold more severe with an increasing ratio of pixels labeled as demosaicing artifacts in the overall image. The 10 factor in Equation 4 has been introduced because, on average, there was a small number of foreground pixels, and so we would have ended up having a range of thresholds that was very small (each threshold would have been very close to 20, which is the default threshold). Instead, we wanted a broad range that was able to represent the labels for different input images correctly. Then, the process explained above (applying closing and removing regions with few pixels) is repeated to obtain the final label image, which is however to be considered as *weak supervision*. Visual examples of artifacts and their labels are reported in Figure 15.

Figure 15: From left to right: original scene, artifacts generated, labels

2.2.5 Training-Validation-Test Split

For computational reasons, we worked with the first 2000 images of the MIT-Adobe FiveK dataset instead of the full 5000 images dataset and, in order to set up the data for a supervised deep learning model, (in this case, a segmentation Convolution Neural Network) we divided the images into 3 sets:

- **Training set**: This set contains the images which will be used for training the model, it should be large enough in order to be representative of the problem we would like to address. It is called training because it is the set that the model exploits for tuning its layers parameters which are part of the non-linear function which will provide a solution to our problem. In our case, this set consists of 1440 images.

- **Validation set**: It is fundamental that the model we will train will be able to learn, but also to *generalize* on a variety of data, without the risk of *overfitting* on the training set (i.e. achieving good performances on the training data, with poor results on new data). In order to do that we use a validation set that we generated by selecting 160 images.

- **Test set**: This is the set of images that will be used to assess the performance of our model; it consists of images that are not seen during the training process, in order to quantify the performance of our model on new data. For this set, we used 400 images.

The split was performed by setting a random seed that ensured the replicability of all the experiments. Further seeds should have been tried, but for computational and time reasons we worked with seed 42 only.

2.3 Segmentation network

The next step consists in building a segmentation network that allows labeling artifacts in a full *automatic* way. The network we chose is the U-Net [21]. Originally, this network was used for biomedical images and it became very popular for its promising results with a reasonable training time. The network architecture consists of a contractive path and an extractive path. The former is made of a sequence of two 3×3 convolutions, followed by batch normalization (not present in the original network, it is an addition we did in order to mitigate internal covariance shift), a ReLU activation function and a 2×2 max pooling operation, with a stride 2. In the original paper architecture, at each downsampling the number of feature maps' channels is doubled, however, we kept a 32-depth of feature maps since we prefer light models from a hardware perspective. On the other hand, the extractive path consists of a sequence of two 3×3 convolutions as well, also applying a 2×2 up-convolution that upscales the feature maps, but instead of halving the number of channels as in the original paper, it keeps them at a fixed value of 32 channels. In addition, in each block of the extractive path, it is also applied a concatenation with the features of the corresponding contractive path block. Finally, we obtain a segmentation mask on which we apply the sigmoid function to get the predicted labels. An overview of the U-Net architecture is reported in Figure 16.

2.4 Training

Since all the input images have different shapes, we cropped each input image, making sure that the crop contains some artifact pixels. In addition, to make the training more robust we sometimes feed training crops from original images, which do not contain artifacts. We tried two different ways to select training crops. The first one consists of taking 20 random crops in a random image among the training ones. Then, we pick the random crop with the highest number of foreground pixels (i.e. the crop with the highest number of artifacts in the current image). We feed this crop to the network, but we also feed the same crop from the original image, which is in theory without artifacts, but this depends on the original data's

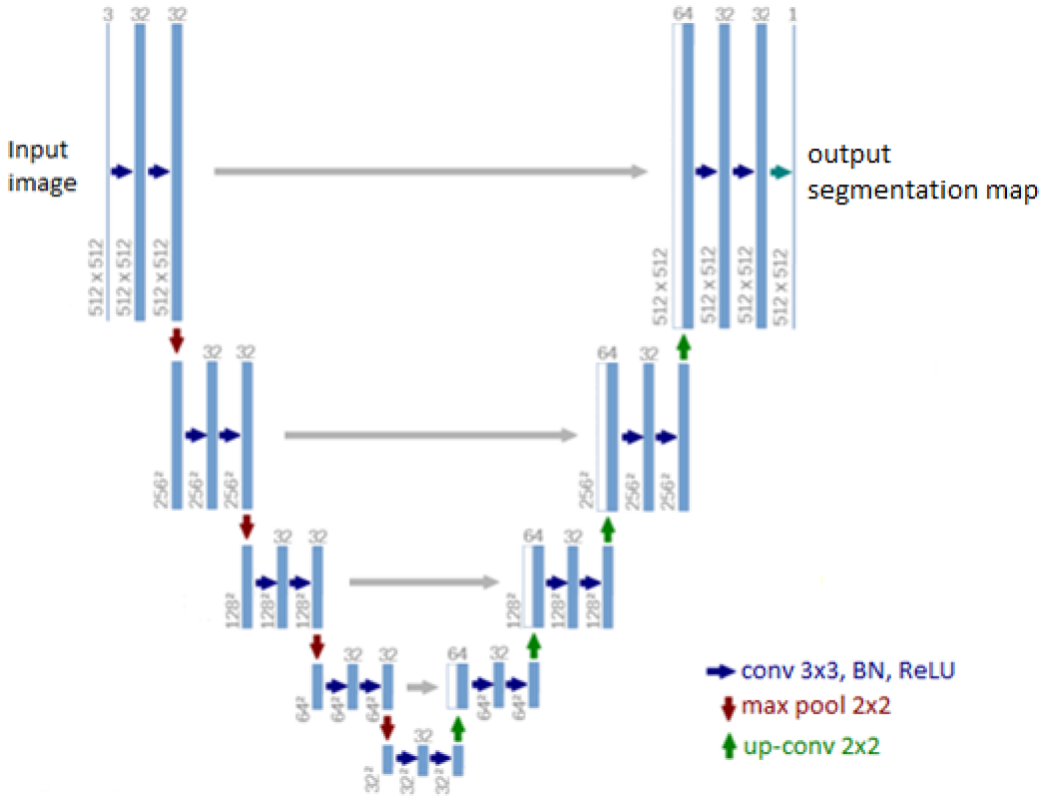

Figure 16: Our U-Net architecture

quality, which we assumed to be completely clean from artifacts. In this way, the network will have at its disposal the difference in appearance between a crop with artifacts and a crop without them. A scheme of this process is reported in Figure 17.

The second way to select training crops we tried consists in selecting a random image and then sampling a crop from either its version with artifacts or from the clean one. We used a Bernoulli random variable with probability $p = 0.4$ to sample a random crop from a training image with artifacts, otherwise, we pick a random crop from a random image without artifacts. As before, if the crop is sampled from an image with artifacts, we make sure to select the crop with the highest number of foreground pixels, among a set of 20 sampled crops. Eventually, we picked this latter training crop criteria, which process is reported in Figure 18.

We trained the network on a GPU (Nvidia Tesla P100-PCIE, 16GB), assessing the results

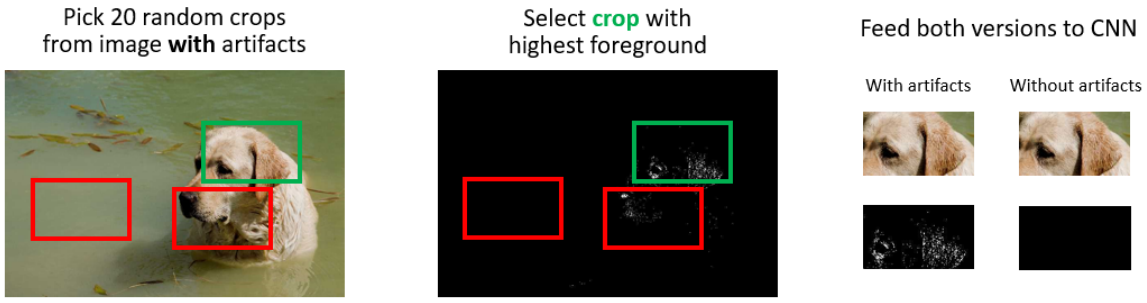

Figure 17: First training crops sampling criteria we implemented

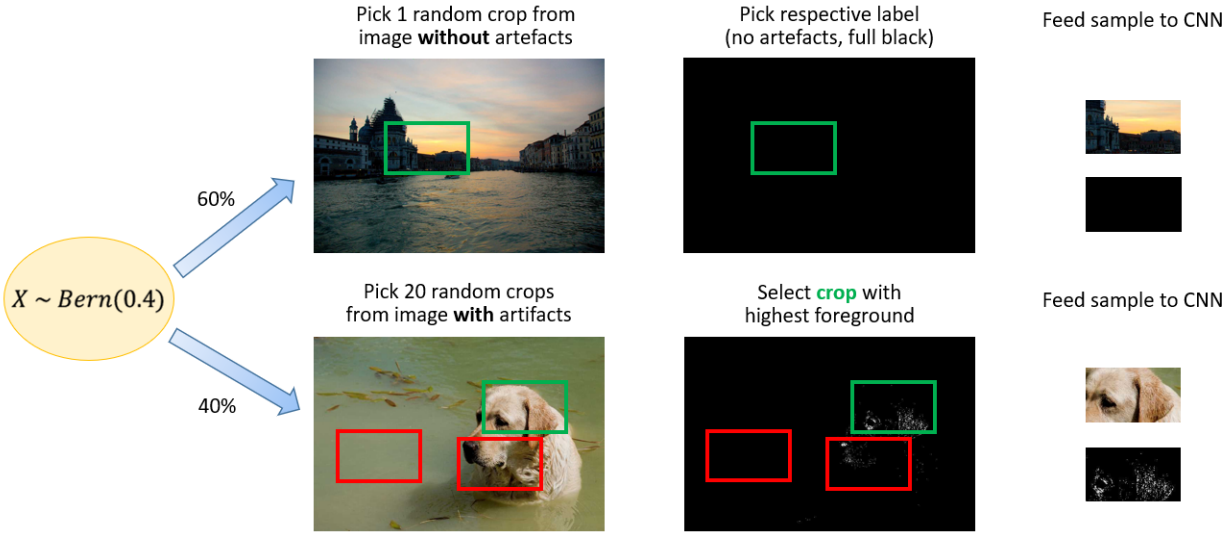

Figure 18: Second training crops sampling criteria we implemented

both visually and in terms of Dice Score which will provide partial information about the ability of the neural network to identify demosaicing artifacts. We also used visual quality as a metric because the Dice Score may be misleading sometimes since we have labels that are very small compared to the input image size and since the Ground Truth labels are weak. The set of parameters, losses, and optimizers we tried are reported in Table 2, and the best set has been highlighted. The training was 40 epochs long, which took approximately 35 hours.

We chose to work with *region-based* loss functions only, as this approach was more suitable

Batch size	**Crop size**	**Learning rate**	**Optimizer**	**Loss**
2	256	**0.001**	SGD	**Dice**
4	512	0.01	Adam	Tversky ($\beta = 0.25$)
8	**1024**	0.05	**RMSProp**	Tversky ($\beta = 0.75$)

Table 2: Set of hyperparameters tried in training the demosaicing segmentation network

for the type of data that we had as it considers loss information both locally and globally (*distribution-based* losses like the Binary Cross-Entropy Loss provides a lower performance). The Dice Loss [22] is derived from the *Dice Score* which is a well-known metric used in Computer Vision to compute the similarity between two images. From this metric, we can compute the Dice Loss as reported in the set of equations 5. From a Set Theory perspective, the Dice Score is a measure of overlap between two sets, in particular between two image labels in this case (y which is the Ground Truth label, and $\hat{y}$ which is the predicted label). The score ranges between 0 and 1, and we can use $1 - Dice_{score}$ to maximize the overlap between the two labels. The numerator of the Dice Score consists of 2 times the overlap, instead, the denominator is the union of the two sets. They both have an additional ϵ term (default is 1) to avoid edge scenarios in which the function is not defined (if $y = \hat{y} = 0$).

$$\begin{aligned} Dice_{Score}(y, \hat{y}) &= \frac{2y\hat{y} + \epsilon}{y + \hat{y} + \epsilon} \\ Dice_{Loss}(y, \hat{y}) &= 1 - Dice_{Score} \end{aligned} \tag{5}$$

The Tversky Loss [23] is instead derived from the *Tversky index*, which is a generalization of the Dice Score. The derivation of the loss is the same as Dice, and the Tversky Index introduces a weight for both False Positives (FP) and False Negatives (FN) through a β parameter which ranges from 0 to 1, as the index itself (6). In particular, a higher β would lead to a model more sensitive to FP, and the value of the index would decrease when the number of False Positives increases. When FP and FN have the same weight ($\beta = 0.5$) the Tversky Index corresponds to the Dice Coefficient; the proof can be found in Appendix B.1.

$$Tversky_{Index}(y, \hat{y}) = \frac{y\hat{y} + \epsilon}{y\hat{y} + \beta(1-y)\hat{y} + (1-\beta)y(1-\hat{y}) + \epsilon}$$
$$Tversky_{Loss}(y, \hat{y}) = 1 - Tversky_{Index} \tag{6}$$

2.5 Results

With the previously explained setting, we obtained an average Dice Score of 0.62, which was computed by considering the score on the demosaicing artifacts class (0.31) and the score on the background class (0.93) on all the test set images. This set consists of 400 images that were never seen during training. However, these quantitative results must be analyzed considering that the Ground Truth consisted of weak labels automatically generated, and since the artifacts are very small regions where the Ground Truth may not be fully reliable, we analyzed with more attention the visual results as they are more representative of the true performances of the model. In addition, we paid attention to the inference time as we wanted to build a fast model and the average prediction time is 34.2 ± 0.5 ms per image with a shape of $(1024, 1024)$ and it was computed again considering the test set only (even though this was not strictly necessary), making sure to apply a GPU warm-up, in order to avoid misleading results due to GPU initialization time. Examples of visual results of the inference step are reported in Figure 19 which consists of zoomed areas of the original images since demosaicing artifacts regions are tiny and difficult to be seen looking at the whole image.

As shown in Figure 20 sometimes there is a problem with False Positives (i.e. the pixels labeled as artifacts, which are instead correct pixels) and False Negatives, even though this issue is raising only in a small number of predictions. This problem is more likely to happen in images where there is a strong texture, as *grass, tree* and other *nature's* elements. However, sometimes False Positives are not actually false, but the bad quality of the input (an image that should be clean, but in reality, it is not) is fooling the network, which will predict some areas of pixels as artifacts (21).

Finally, we report an analysis over the *feature maps* which the trained neural network is producing at a specific layer. In particular, we want to visually assess that if we take the same image, with and without demosaicing artifacts, the network is able to extract different features. As shown in Figure 22, we can see that taking the third U-Net decoder, the feature maps are very different depending on the input presence of artifacts. The network is clearly able to focus automatically on the regions affected by the artifacts, which is what we wished

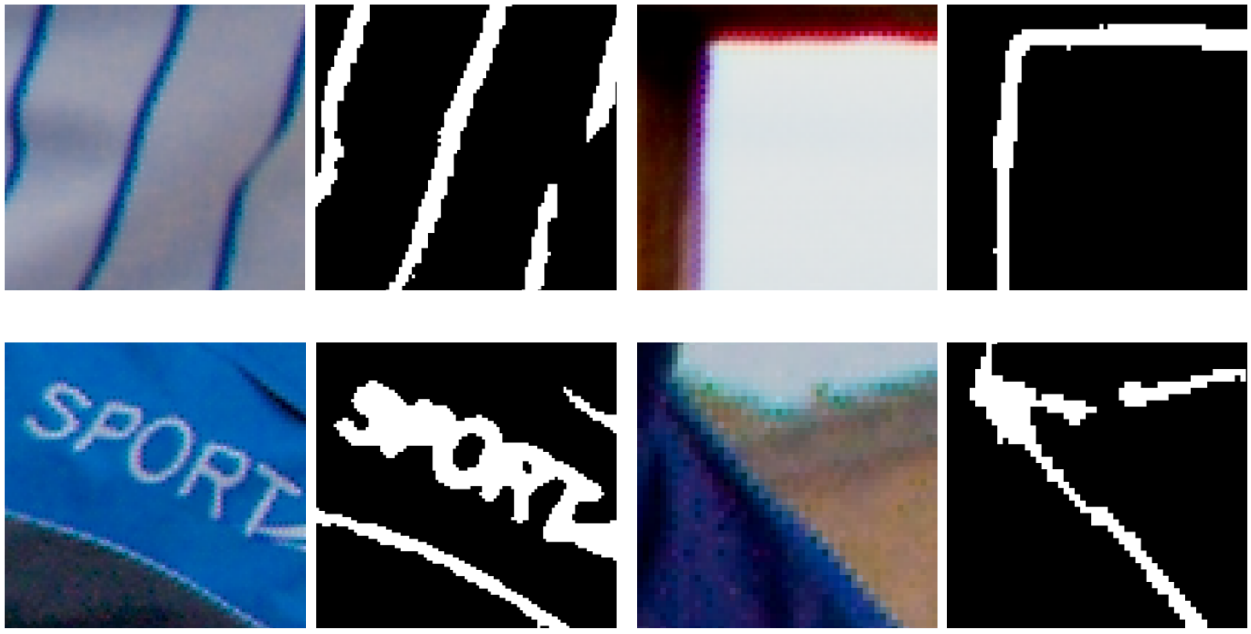

Figure 19: Examples of good demosaicing's artifacts segmentation

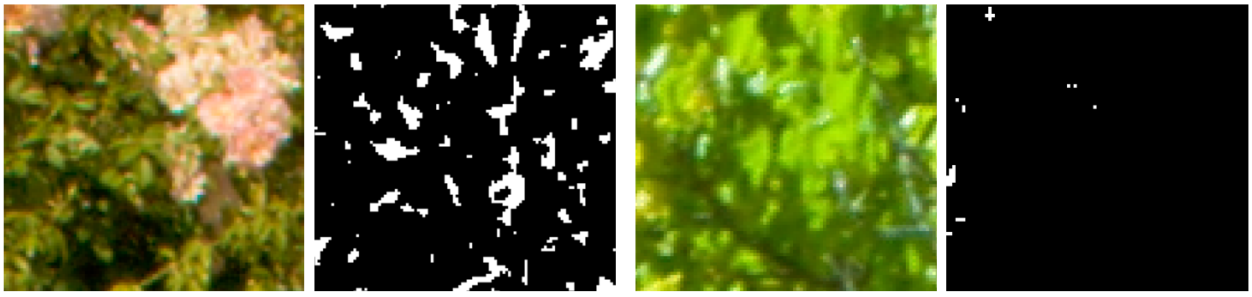

Figure 20: Examples of bad demosaicing artifacts segmentation. On the left, there are False Positives, and on the right there are False Negatives.

and expected.

Figure 21: Examples of how the input quality of an image can fool the network.

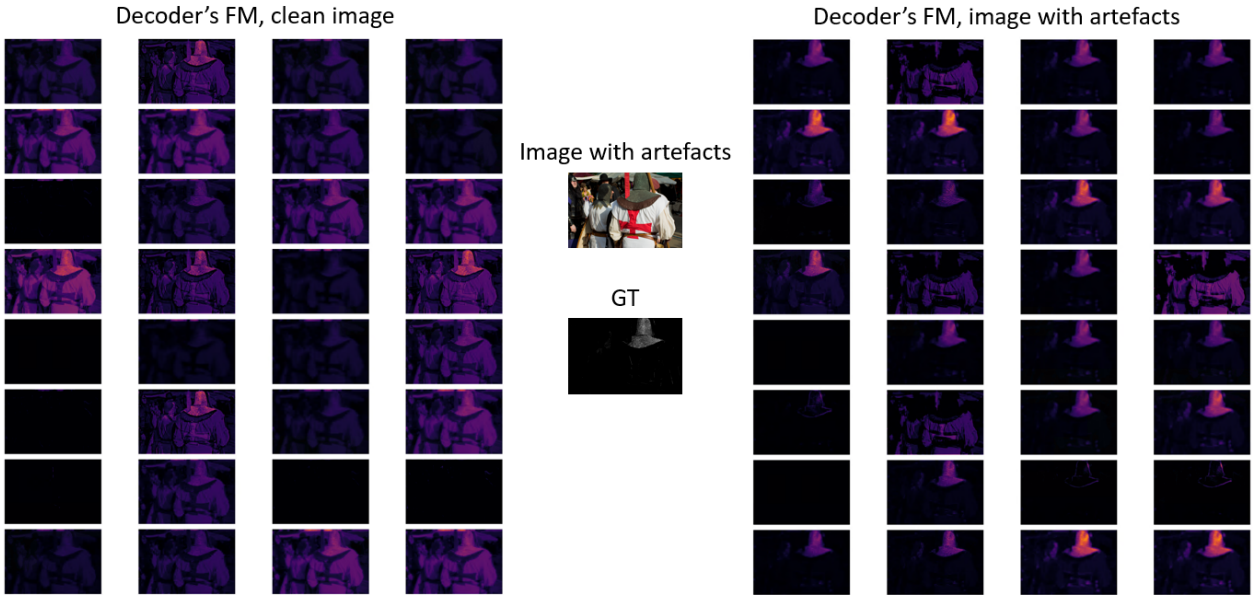

Figure 22: Feature maps at third U-Net decoder for input image with and without artifacts.

Chapter 3

3 Ghosting artifacts

3.1 Overview

At this point, we wanted to assess our model on a new type of artifact: the so-called *ghosting*. Ghosting artifacts are created when merging between multiple *exposures* happens and there has been some movement across them. In particular, an exposure is the amount of light that reaches a camera's sensor for a period of time. Movement across exposures can be either referred to objects in the scene, or to the camera. For this new artifact, we worked with the Kalantari dataset which paper was presented at SIGGRAPH conference of 2017, which is a well-known computer graphics conference organized by the Association for Computing Machinery's Special Interest Group on Computer Graphics and Interactive Techniques.

3.2 High Dynamic Range images

High Dynamic Range images, also known as HDR, are images obtained by combining different exposures (different amounts of light per unit area) of a certain subject. The *dynamic range* is quantified as the ratio between the maximum value of a quantity, and its minimum. In particular, in image context, the dynamic range is referred to as the quantity of *light*. A natural effect in camera images is that the human eye is only able to differentiate between luminosity in a certain range, outside of which brighter areas are perceived as pure white and darker areas as pure black. However, by taking different exposures and combining them we can obtain a higher dynamic range; that is where the name comes from.

The dynamic range in image processing is measured in *EVs* which stands for *Exposure Value* and it is a combination of exposure time (t) and f-number (N), as reported in Equation 7.

$$EV_{absolute} = log_2 \frac{N^2}{t} \tag{7}$$

This is the most common way to define the EV, in an absolute way. However, it is also possible to define it in a relative way by using a *reference* exposure time which is defined in Equation 8.

$$EV_{relative} = log_2 \frac{t_{exposure}}{t_{ref}} \tag{8}$$

After having merged the exposures, we have to apply *tone mapping*. Tone mapping is an operation performed with the aim to map a 32-bit image to another set of colors (typically 8-bits) giving the image the appearance of a High Dynamic Range image. An example of this process is reported in Figure 23.

Figure 23: Examples of merging exposures and tone mapping

3.3 Data and Dataset creation

For this artifact, we used the Kalantari dataset, which originally contains 74 training images and 15 test images. For each image, there are 3 different exposures, a text file providing information about the exposure values, and the merged HDR image. In order to create a functional dataset we had to create ghosting artifacts by merging the provided exposures for each image. Since among each exposure there has been movement, the final merged image will contain ghosting. We have to label automatically the ghosting first, and then do it through a segmentation network as we did for the demosaicing artifacts.

3.3.1 Merging exposures

The original dataset contains, for each image, a short, medium, and long exposure. In order to enlarge the dataset, we decided to merge each pair of exposures (i.e. short with medium, medium with long, and short with long) as reported in Figure 24.

Also, we kept the original exposures to have a dataset containing both images with ghosting and clean images. In this way, we managed to obtain a training dataset of 444 images and a test set of 90 images. There are many different algorithms to merge exposures; we implemented *Debevec* [24], *Robertson* [25] and *Mertens* [26] thanks to the OpenCV library. The first two algorithms use the exposure times and the tone mapping, while Mertens does not need them, and it is the one that we used. It does so because this allows a simplification of the acquisition pipeline (physically-based HDR assembly is skipped) and it is computationally efficient. The main idea is that the merging is guided by simple quality measures as

Figure 24: Merging pair of exposures with movements across each other

contrast, saturation and *well-exposedness*. Contrast is computed through a Laplacian filter to the grayscale version of each image (taking the absolute value of the filter response); in this way, the higher weights will be assigned to edges and texture areas. Saturation is computed as the standard deviation within the 3 channels at each pixel. Finally, well-exposedness is computed by looking at the raw intensity within a channel and multiplying them. The fusion is then computed through a weighted average along each pixel whose value is the multiplication of the three metrics we discussed above. The final step is to compute the fusion using a Laplacian pyramid decomposition and a Gaussian pyramid of the weight maps, which represents measures such as contrast and saturation. This algorithm expects the images to be aligned, however, we did not align them because we wanted to introduce the ghosting effect.

3.3.2 Labels creation

As we did for the artifacts coming from demosaicing in the MIT-Adobe FiveK Dataset, the first step is to create raw labels automatically. A naïve and simple solution would be to compare the provided HDR image with the merged image, and the difference would be the ghosting artifacts, however, in the Kalantari dataset the HDR images are referred to the medium exposure and so these images become useless in this context. In particular, Kalantari's HDR image for a certain set of exposures is derived from a merging operation between the exposures, after that they have been aligned to the middle exposure. For this reason, the HDR could not be used as a Ground Truth image for labeling ghosting artifacts automatically. So, the solution we implemented consists of the following:

- First, we **invert Gamma correction** by exponentiating each pixel value by the Gamma value, which is 2.2 in most of the cases. This is done for each image and for each set of exposures; for example first image and set *(short, medium)*, set *(medium, long)*, and so on. Thus, we would end with a set of exposure images whose colors are different in a linear way. Then we map each pair of exposures to the same colors by exploiting the EV information, which is provided for every image. For example, if an exposure has an EV=0 and another exposure has EV=2 we can map the first one to the colors of the second one by inverting gamma correction, and multiplying the image by a factor of $2^2/2^0$, assuming a reference exposure time of 1 second. Finally, we compute the absolute difference between these two exposures (which we will name D), taking the maximum difference along channels and normalizing the values in the interval $[0, 1]$. This first step is summarized in Figure 25.

- Despite being useful, the D difference is not sufficient information for labeling a pixel as a ghosting artifact. In fact, if we would simply select a threshold above which a pixel is considered a ghosting artifact, we would lose the ghosting artifacts in darker regions, where the difference from the previous point may be low, but still evident to the human eye and relevant from a ghosting perspective. For this reason, we compute a mask of **low luminance** of the image coming from the merging of the exposures. A pixel is considered as part of the low luminance area if its luminance value is below the average image's luminance, multiplied by a factor of $\frac{2}{3}$, which was a tuned parameter. This mask is used to compute another mask starting from D; it is a mask that aims to select possible ghosting pixels in dark regions and in regions of medium brightness. These pixels are selected through two bounds, LB and UB, which were tuned by looking at visual results. An example of this step is reported in Figure 26.

- At this point, we computed a mask for ghosting pixels which are in brighter areas,

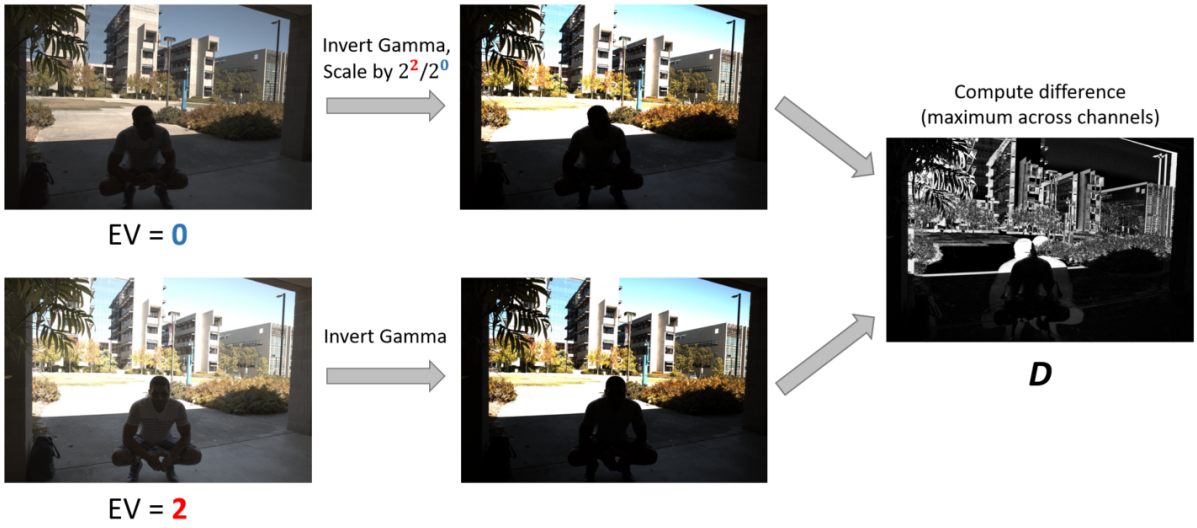

Figure 25: How the difference between two exposures is computed

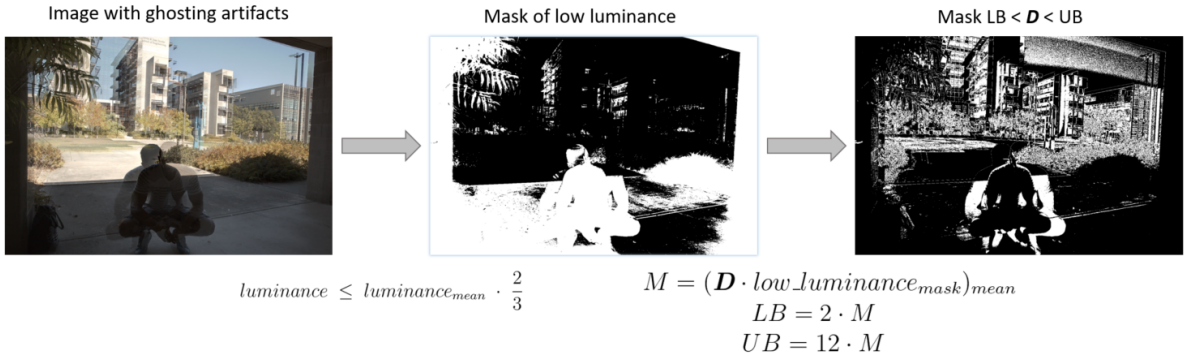

Figure 26: Low luminance mask computation and raw detection of ghosting in dark and medium brightness areas

simply by looking at pixels whose D value was above a certain threshold, which was tuned again by looking at visual results. Finally, a first approximation of the raw ghosting label is derived according to Equation 9, which exploits all the masks that we

computed.

$$label_{raw} = mask_{low\_luminance} \cdot mask_{LB < \boldsymbol{D} < UB} + mask_{\boldsymbol{D} > TH} \tag{9}$$

The mask that deals with ghosting pixels in dark and medium brightness areas is multiplied by the mask of low luminance because the bounds values we found through parameter tuning may not work for every image in the dataset, so this multiplication allowed us to make sure that the bounded region is consistent with the low luminance area of the current image. An example of how these masks are combined is reported in Figure 27.

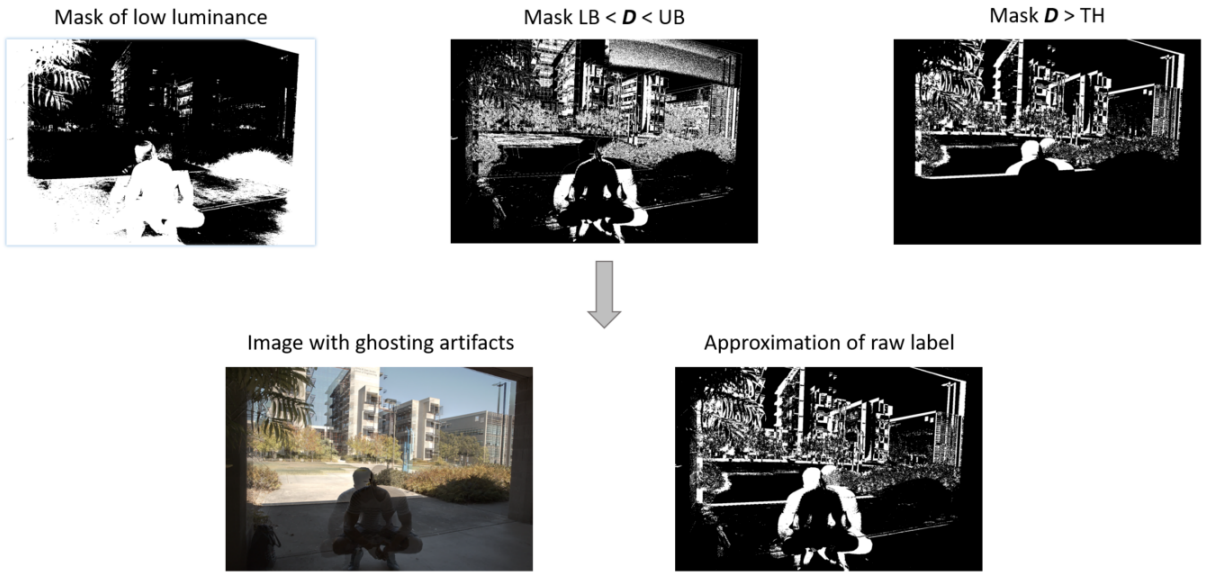

Figure 27: Visual example of how the raw label is created by exploiting previous steps and luminance information

- Finally, we exploited the dataset's information that the person is always moving across exposures. In particular, sometimes the person is in a very dark area of the image and some details of the ghosting are lost there. In order to partially recover them, we combine the previous step's raw label with the pixel segmentation of the person. The person's segmentation has been obtained for every merged image thanks to a

pre-trained semantic segmentation network. For this step, we used PyTorch pre-trained *DeepLabV3-ResNet50* [27] which provides very accurate results. Therefore, the final label is given by:

$$label_{final} = label_{raw} + mask_{person} \cdot (\boldsymbol{D} > (\boldsymbol{D} \cdot mask_{person})_{mean}) \tag{10}$$

and the result is improved through operations of dilation and erosion as it was done for the previous kind of artifact. An example of this last step is reported in Figure 28.

Figure 28: From left to right: merged image, approximation of raw label, improved raw label with the pre-trained network, closing and opening operations.

After these steps were done, we obtained some good quality raw labels, as reported in Figure 29.

3.4 Training

The training process is similar to the demosaicing model, however, this time we have to consider 3 aspects:

- The dataset is **balanced** because for every set of 3 exposures we generated 3 merged images containing ghosting artifacts, and we also kept the exposures as clean data. For this reason, at training time we simply picked a random image, and this process can be seen as randomly selecting from a Bernoulli random variable with $p = 0.5$, in the long run.
- Differently from the demosaicing artifacts, when dealing with ghosting in the Kalantari dataset, we have a **wider area with foreground pixels**. For this reason, we decided not to apply the *most representative foreground crop* criteria as we did for demosaicing (Figure 18), but to simply take a random crop, since selecting the most representative crop is an expensive operation that we should perform only if strictly necessary.

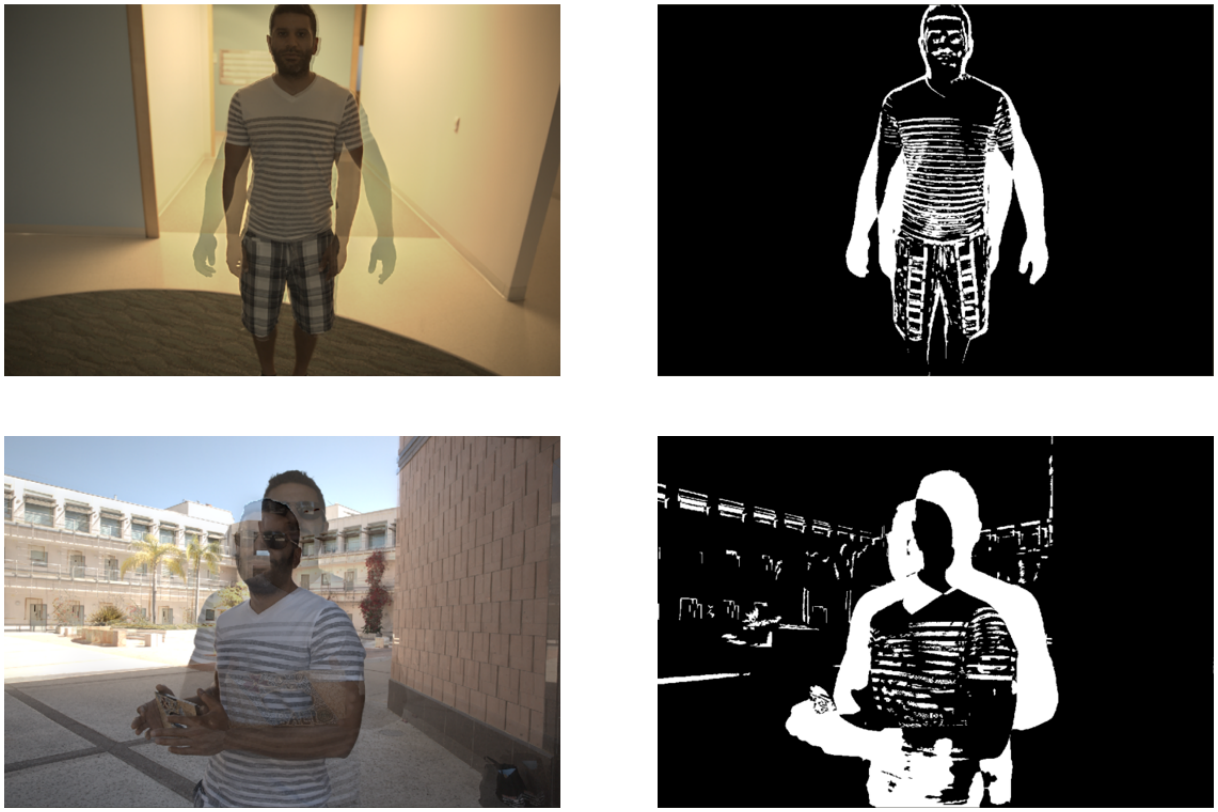

Figure 29: Examples of raw ghosting artifacts labels.

- The Kalantari dataset is much **smaller** than the MIT-Adobe FiveK Dataset, and for this reason, we performed an important *data augmentation* step, which will be described in detail in the next subsection.

Following the same approach that we used for the previous training, we made sure to have 3 datasets: training set, validation set, and test set since we wanted to avoid overfitting and we wanted to be able to test the model's generalization capabilities.

3.4.1 Data Augmentation

Data Augmentation is a process that aims to both enlarge small datasets, but also to introduce *variety* in the data we are dealing with. In fact, deep learning models need a huge amount of training data and a small dataset may not be enough to train a good model (even

though pre-trained models exist and practices like *fine-tuning* and *transfer learning* are very common and effective). Also, the variety of data is a crucial aspect when training a model because we want to avoid overfitting on a very limited domain of input images. For example, if a training dataset consists of images of winter scenes and at inference time we are provided with similar images, but in a different season, results may be poor. In the Kalantari dataset, we had to keep into account this effect because it is a dataset containing images created by a small number of people, with pretty similar scenes and so augmentation was necessary to allow a better generalization on this task. We performed data augmentation with the following criteria; with 60% probability, we made the image's crop darker by lowering its brightness, through a brightness factor that has a deterministic term and a random term x for better generalization, as reported in Equation 11. This factor was further clipped into the range $[0, 1]$, which is the same range value of the input image's crop.

$$brightness_{factor} = \frac{1 - image_{mean}}{3} + x \quad ; \quad x \in [0, 1) \tag{11}$$

After that, with a 30% probability, we entered in another set of augmentations, each of which was applied with a 50% probability (hence, we can apply multiple augmentations at the same time). These augmentations are *random horizontal flipping, color jittering* and *Gaussian noise* addition. We made sure to apply a small color jittering and Gaussian noise, not altering too much the images' domain. A scheme of this data augmentation model is reported in Figure 30 where augmentations are strongly intensified only for visualization purposes.

3.4.2 Network

For the ghosting segmentation task, we used the same network that we utilized in the Demosaicing Artifacts segmentation task: the U-Net, in a lighter version. Following a similar approach with respect to what we did in demosaicing problem, we tuned the U-Net by performing different training runs with different hyperparameter sets which are reported in Table 3, where we highlighted the best combination of them.

3.5 Results

From a visual point of view, after a training of 20 epochs (which took approximately 18 hours), the CNN yielded very good results on clean images, i.e. it predicted almost every pixel in the image as a background pixel, which means not being a ghosting artifact. An example of this model's quality is reported in Figure 31.

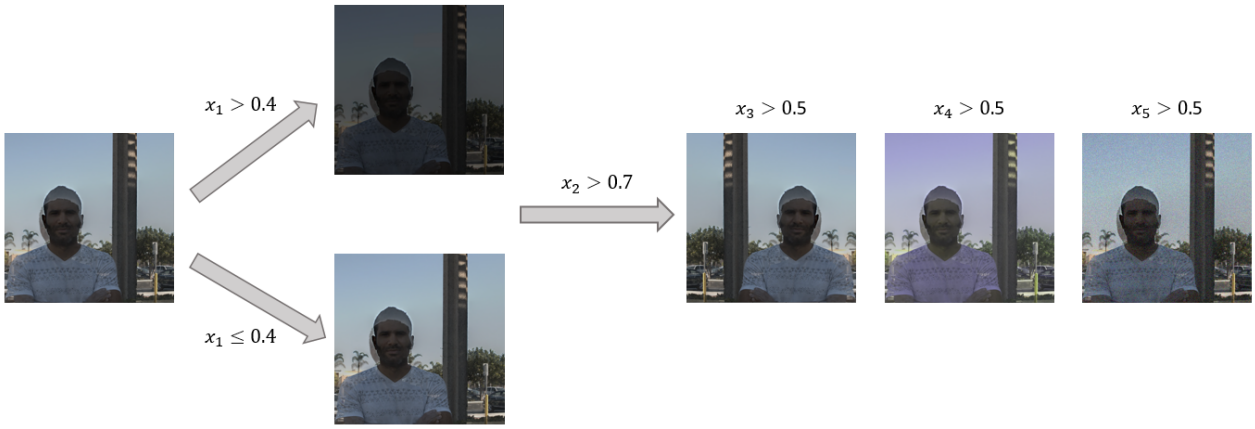

Figure 30: Data augmentation criteria applied for Kalantari's dataset. Each x_i is sampled from a uniform distribution with range $[0, 1]$.

Batch size	**Crop size**	**Learning rate**	**Optimizer**	**Loss**
2	256	**0.001**	SGD	**Dice**
4	**512**	0.01	Adam	Tversky ($\beta = 0.25$)
8	1024	0.05	**RMSProp**	Tversky ($\beta = 0.75$)

Table 3: Set of hyperparameters tried in training the ghosting segmentation network

Instead, predictions on images containing ghosting artifacts were not extremely well-performing. Overall, ghosting areas are detected, but label boundaries are a bit noisy and the label area is often slightly wide with respect to the ghosting area. This may be due to the implicit hard task we are trying to solve. In particular, detecting and labeling a ghosting area is difficult also for the human eye, because, as the name suggests, ghosting areas are regions where there is just a shadow of objects which were present in a certain exposure, and if this shadow is not particularly opaque it is very difficult to detect it. Also, in the Kalantari dataset, it often happens to have exposures in which objects have moved

Figure 31: Examples of ghosting prediction on clean images.

considerably, making it even harder to select which regions can be considered as real, and which areas as ghosted, in the final merged image. We report some visual results in Figure 32 where it is clearly visible that ghosting CNN performance is acceptable, but there is room for improvement, especially for what concerns the precision of label boundaries.

Probably, a different and deeper network may have helped in achieving better results, but we kept working with the light U-Net for hardware limitations and also because we wanted to fairly assess the CNN performance on a different range of artifacts. As we discussed in the Demosaicing Network, when we are dealing with artifacts and weak labels, numerical metrics such as the Dice Score can not be considered fully reliable. However, for the sake of completeness, we report that the average Dice Score of the model is 0.7, and the scores of ghosting artifacts class and background are respectively 0.46 and 0.94. These values have

Figure 32: Examples of ghosting prediction on images with artifacts.

been computed considering the predictions on the test set, which consists of 90 images never seen during the training step. Instead, the average inference time of the model is 36.9 ± 0.4 ms for an image with shape $(1024, 1024)$, and it was computed as we did previously for the demosaicing network.

Finally, we report an analysis of the feature maps which the trained neural network is producing at a specific layer, as we did previously for the demosaicing CNN. As shown in Figure 33 we can see that taking again the third U-Net decoder, the feature maps are very different depending on the input presence of the ghosting artifacts. Also this time, the network seems to understand correctly the meaning of ghosting artifacts and it automatically focuses on the regions of interest when getting closer to the predictions.

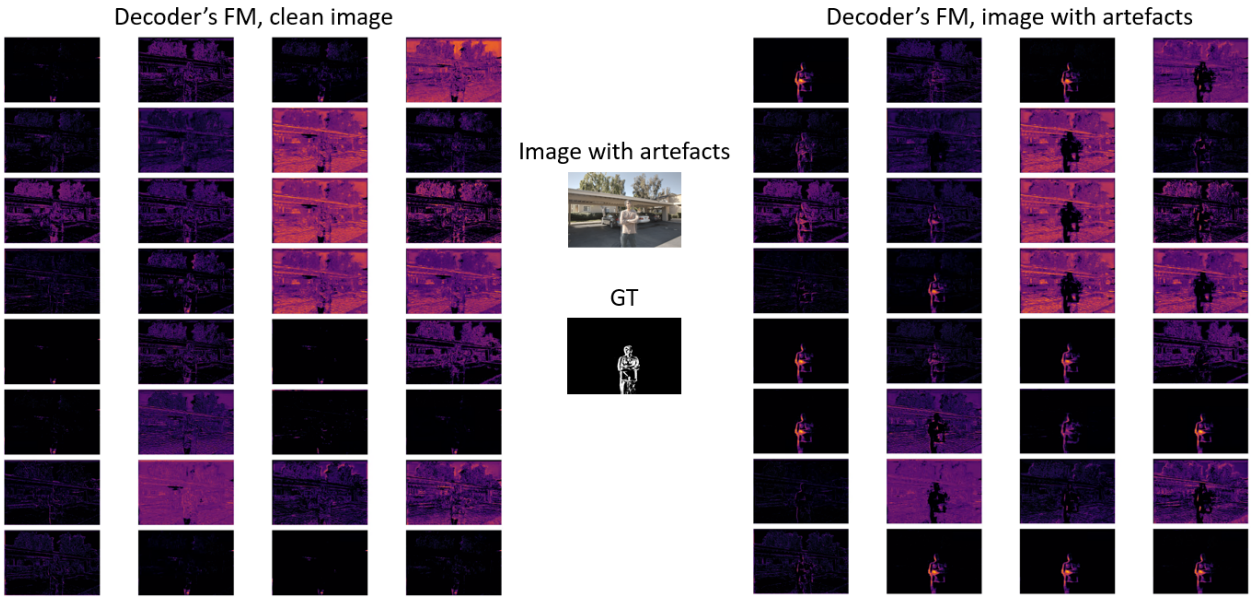

Figure 33: Feature maps at third U-Net decoder for input image with and without artifacts.

Chapter 4

4 Joint network

4.1 Architecture

Once the two artifacts segmentation networks have been successfully trained, we merged them in order to have a unique model being able to identify both demosaicing and ghosting artifacts. We kept the U-Net structure, however instead of moving towards a multi-class problem we decided to keep it binary and adding two branches at the end of the U-Net, one for each type of artifact. This was done because it allowed us to have a first part of shared parameters, and then a set of specific parameters for each artifact. Both branches consist of a convolution block which is made by two sets of convolutions with $(3, 3)$ kernels, a Batch Normalization layer, and a ReLU activation function. Finally, each branch has a mapping from the 32-channel feature map to the output segmentation map, through a convolution and a sigmoid activation function to map real values to probabilities. The joint network architecture is reported in Figure 34.

4.2 Training

Despite the two branches do not contain a big amount of blocks, they introduce a set of new parameters to be tuned. A clear consequence of this is a lower memory space when training the model. Indeed, we had to reduce the batch size in order to match CUDA memory requirements for the joint deep neural network. For this reason, the space of hyperparameters that we explored during the training runs had smaller batch size values, as summarized in Table 4, where we reported part of the hyperparameters that we tried. The model was trained for 40 epochs which took approximately 40 hours.

Batch size	**Crop size**	**Learning rate**	**Optimizer**	**Loss**
2	256	**0.001**	SGD	**Dice**
4	**512**	0.01	Adam	Tversky ($\beta = 0.25$)
8	1024	0.05	**RMSProp**	Tversky ($\beta = 0.75$)

Table 4: Set of hyperparameters tried in training the joint segmentation network

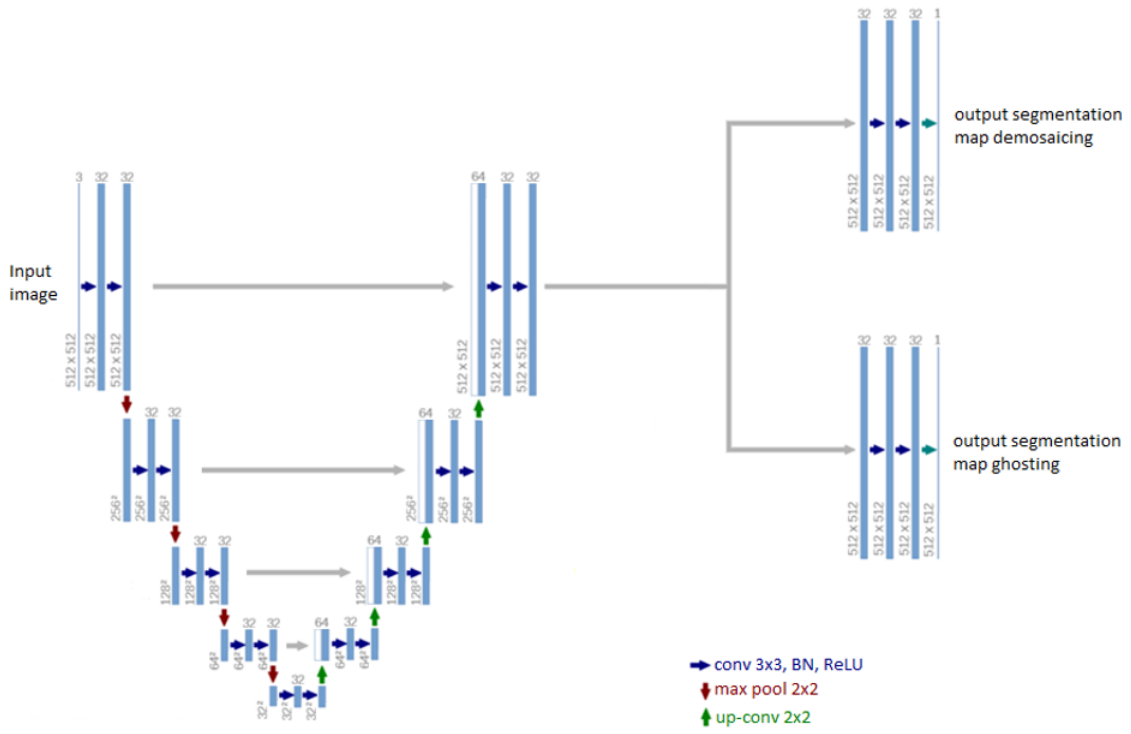

Figure 34: Joint U-Net architecture.

The training criteria did not change, as it just consisted of randomly picking images from either the demosaicing dataset or the ghosting dataset. Then, the local dataset training criteria were the same as the two single models that we described in previous sections. As we did and explained before, we used a training, validation, and test dataset split.

4.3 Results

The joint network had the goal to maximize performance on both kinds of artifacts at the same time. Overall, we noticed that this goal was achieved, even though a trade-off was necessary. In particular, convolutional neural networks trained for demosaicing artifacts detection and ghosting artifacts detection separately provided better results on the kind of artifacts they were trained on, but poor performance on the other type of artifact as it was never seen during training. The joint network instead yielded much better predictions on

the type of artifact on which the network was not trained before, at the cost of a slight decrease in performance on the artifacts on which the single-artifact network was trained. For example, the CNN trained only on ghosting artifacts is providing good performance on this kind of artifact, but poor results if the input contains demosaicing artifacts, while the joint network provides good results on both (even though the performance on ghosting decreases a bit as the network is trained on two artifact types instead of ghosting only). An example of this behavior is reported in Figure 35 and Figure 36.

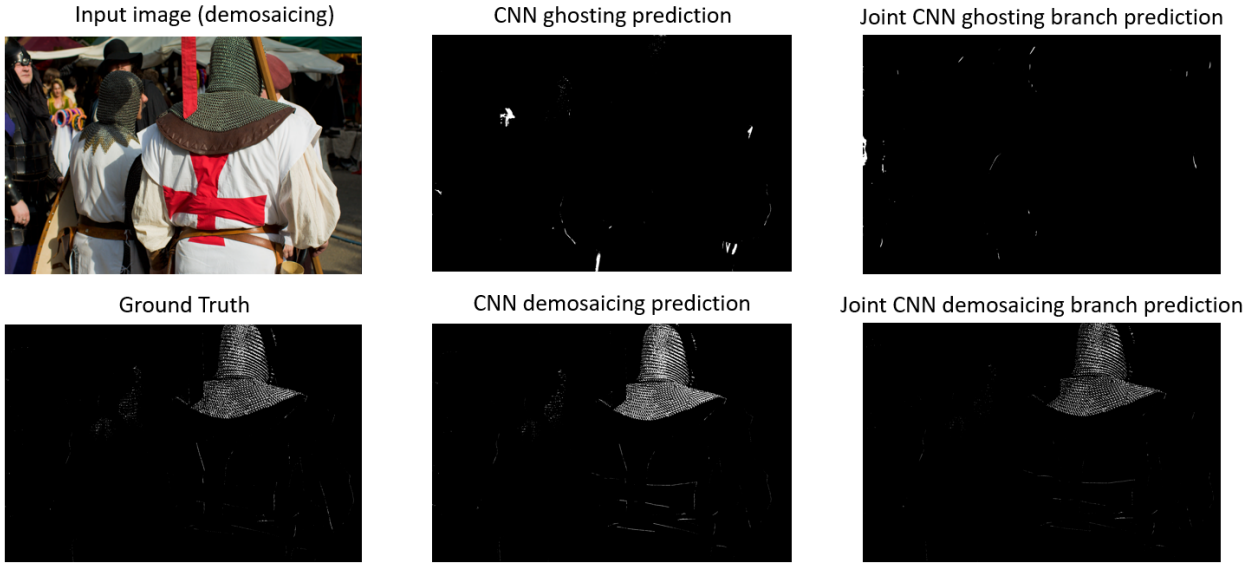

Figure 35: Effectiveness of joint network with respect to the demosaicing network.

As we did for the previous networks, for the sake of completeness we report some metric values for the joint artifacts segmentation network, which are not fully reliable as we discussed in previous sections. In particular, in Table 5 we report a scheme in terms of Dice Scores in different settings. It is interesting to notice that the joint model has learned to distinguish the two artifacts very well. In fact, when using the joint model with an input containing demosaicing artifacts, its predictions very rarely contain ghosting artifacts (and vice-versa). In particular, in these cases, the average percentage of False Positives was 0.05%, which was again computed and averaged on both the full test sets. The average inference time is instead 55.6 ± 0.9 ms for an image with shape $(1024, 1024)$, which is a bigger value with respect to the single-artifact models, and we could expect that since the joint network has more parameters than the previous networks.

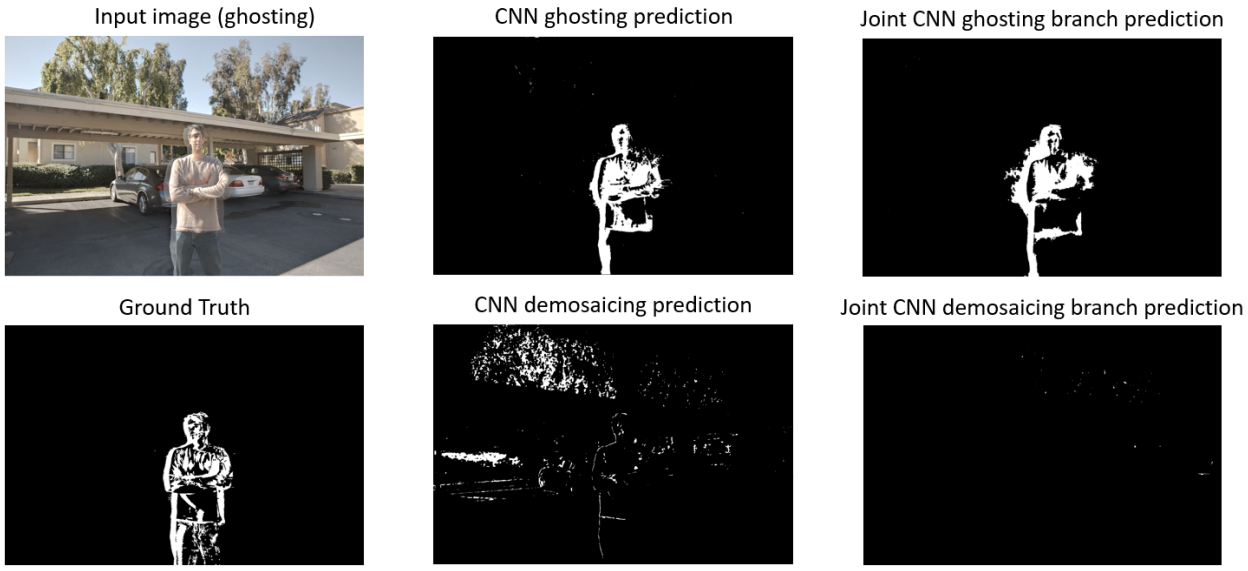

Figure 36: Effectiveness of joint network with respect to the ghosting network.

Dice Score	**Joint model on demosaicing test set Demosaicing Artifacts predictions**	**Joint model on ghosting test set Ghosting Artifacts predictions**
Artifacts Class	0.3	0.43
Background Class	0.94	0.95
Average	0.62	0.69

Table 5: Results in terms of Dice Score for the joint model in different settings.

Chapter 5

5 Optimization

The last goal of this work consists of optimizing well-known and used image processing algorithms by exploiting the Convolution Neural Networks that we trained in previous steps. The algorithm that we chose was the Malvar-He-Cutler (MHC) demosaicing method, which we used when creating the demosaicing dataset from MIT-Adobe FiveK images. In particular, our goal was to optimize the weights of the MHC filters in order to improve the quality of the images after the interpolation, making them more similar to the original RGB images. The Malvar-He-Cutler algorithm can be formulated through programming, by computing convolution operations through kernels. As reported in Figure 37 , given the raw information captured by the camera, the process is divided into two stages:

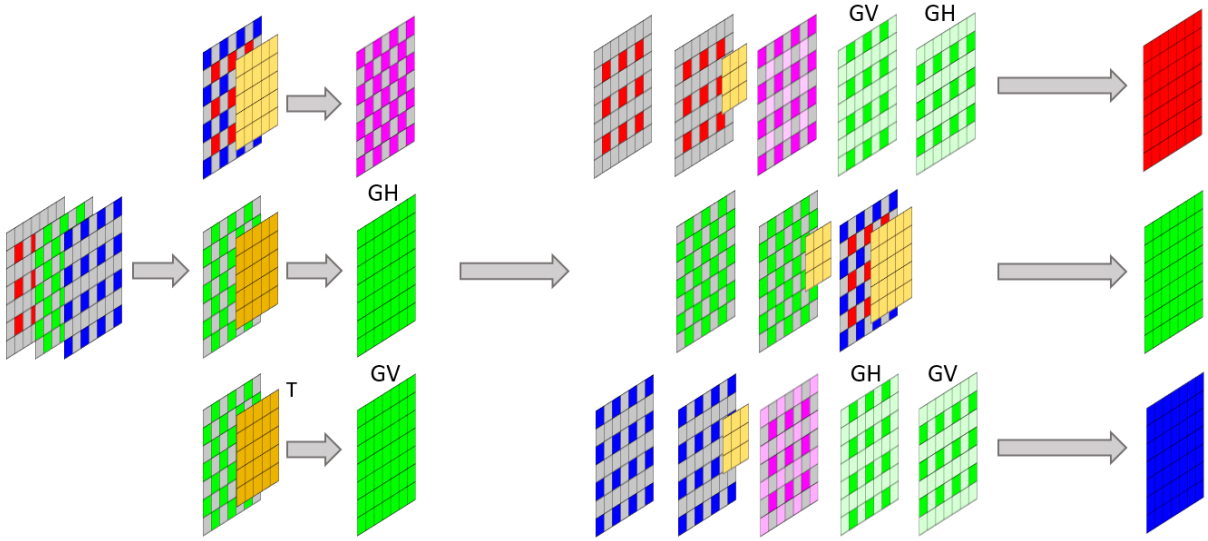

Figure 37: How, given a Bayer input, the RGB image is obtained through Malvar-He-Cutler.

- We apply a set of 3 convolutions; the first one consists of applying a kernel of shape $(5, 5)$ to the sum of the red and blue Bayer channels information. The output is going to be a weighted combination of these 2 colors, which we represented in purple. It will contain a grid of zero values according to the input MHC kernel. The other

convolutions provide two green maps, which we call Green Horizontal (GH) and Green Vertical (GV) as they are the horizontal and gradient corrections from green. These maps are computed from the same kernel, however, the kernel for GV is transposed.

- In the last stage, we compute the sum of a set of maps for each channel. Some of these maps come from a convolution operation with either a kernel of size $(3, 3)$ or size $(5, 5)$. In particular, the red channel information is interpolated through the sum of original red Bayer information, the same information convolved with a kernel $(3, 3)$, the blue locations' information from the purple map described before, and the green information in the blue rows and red rows from GV and GH respectively. The green information is instead computed through the sum of the original green Bayer information, the same information convolved with a kernel $(3, 3)$, and the sum of red and blue Bayer information convolved with a kernel $(5, 5)$. Finally, the blue information is obtained through the sum of the original blue Bayer information, the same information convolved with a kernel $(3, 3)$, the red locations' information from the purple map, and the green information in blue and red rows from GH and GV respectively. Overall, this step can be summarized as retrieving the channel information by the contribution of the Bayer filter values for that channel, and the interpolation of missing values through the other channels (exploiting the cross-channel information).

In this process, each convolution would lead to map shapes with a lower size with respect to the Bayer input, for this reason, proper zero-padding is applied when computing each convolution, allowing the preservation of the input shape throughout the algorithm. The optimization we performed was applied to the kernels in the MHC algorithm, which default values are reported in Appendix C.1. This algorithm can be mathematically formulated through the set of equations 12, on which we are going to optimize the kernels parameters.

$$
\begin{aligned}
RB &= (R_{Bayer} + B_{Bayer}) * K_1 \\
GH &= G_{Bayer} * K_2 \\
GV &= G_{Bayer} * K_2^T \\
R &= R_{Bayer} + R_{Bayer} * K_3 + RB_{blue\_locations} + GH_{red\_rows\_G} + GV_{blue\_rows\_G} \\
G &= G_{Bayer} + G_{Bayer} * K_4 + (R_{Bayer} + B_{Bayer}) * K_5 \\
B &= B_{Bayer} + B_{Bayer} * K_6 + RB_{red\_locations} + GH_{blue\_rows\_G} + GV_{red\_rows\_G}
\end{aligned}
\tag{12}
$$

5.1 Training

We implemented these equations as a simple Convolutional Neural Network which contains 7 convolutions operations through kernels that we optimize with backpropagation. In particular, we implemented the loss function (Equation 13) which consists of 2 terms.

$$
L_{optim}(y, \hat{y}) = L1(y, \hat{y}) + \frac{\lambda}{crop_{size}} \sum CNN_{demosaicing}(\hat{y}) \tag{13}
$$

The first is the L1 loss which is computed between the current RGB image produced by the MHC algorithm ($\hat{y}$) and the true RGB image (y, the original image we downloaded from the MIT-Adobe FiveK Dataset), this term is thought for allowing a prediction that overall looks similar to the true image. The second term is instead thought for allowing an improvement on small demosaicing artifacts region, and it consists of the sum of the pixels probabilities of $\hat{y}$ to be demosaicing artifacts, normalized by the cropping size (1024). Those pixel probabilities are obtained through the best demosaicing CNN that we trained and discussed in previous sections, which is kept frozen during the training (gradient flows back, but CNN parameters are not updated). These two terms are balanced through a hyper-parameter λ whose reference value was 0.001 and it has been obtained by looking at

the average ratio between the two loss terms. However, this value would give approximately the same weight to the two terms in the loss, but we want to give a higher weight to the L1 term because the term ruled by the CNN demosaicing network is related only to very small and specific image regions; the artifacts. In fact, if we would rely too much on the artifacts term we would lead the model toward a wrong solution from a color point of view because it would be optimal only for a very small amount of pixels in the image. We did several training processes with different λ values in order to get the best visual results, which were obtained with $\lambda = 1.5 \cdot 10^{-4}$. Also, we implemented different scheduler criteria, such as standard learning rate and *cosine annealing* learning rate. The latter consists in setting an initial learning rate value and progressively decreasing it (through a cosine function) to a minimum value that is set by the programmer. In addition, the number of epochs during which the annealing is performed must be specified. It is defined in Equation 14 where η_{min} and η_{max} are the final and initial learning rates respectively, T_{max} is the maximum number of iterations and T_{cur} represents the number of epochs since the last restart of the process. However, we implemented the cosine annealing process without the so-called *warm restarts*, hence the learning rate is decreasing in the specified number of epochs and once it reaches the minimum learning rate it does not change anymore (instead, with restarts it would have been updated to a higher value and the decaying process would have been restarted).

$$\eta_t = \eta_{min} + \frac{1}{2}(\eta_{max} - \eta_{min}) \left(1 + cos\left(\frac{T_{cur}}{T_{max}}\pi\right)\right) \tag{14}$$

Finally, we tried different kernels for the training process. In particular, we first trained initializing the kernel values to the original values of the MHC algorithm, however since they contain a limited number of parameters, and a high number of 0-values parameters, we also tried to use bigger kernels with respect to the original MHC demosaicing algorithm and we tried to initialize these kernels both full randomly and also in a partially random way (padding original MHC kernels with zeros and setting randomly only the 0-values). This was done in order to increase the number of parameters, at the cost of a heavier training process and a longer time to get final results. A summary of the parameters and combinations we tried during the training step is reported in Table 6, where we highlighted the combination which provided the best visual results.

Batch size	**Crop size**	**Learning rate**	**Optimizer**	**Kernels initialization**
2	256	0.001	SGD	**Standard MHC**
4	512	**Cosine Annealing** $\mathbf{T_{max} = 15}$ $\eta_{\mathbf{max}} = \mathbf{0.001}$ $\eta_{\mathbf{min}} = \mathbf{0.0001}$	Adam	Standard MHC Size = (11,11) Randomly initialized 0-values
8	**1024**	Cosine Annealing $T_{max} = 15$ $\eta_{max} = 0.01$ $\eta_{min} = 0.001$	**RMSProp**	Size = (11,11) Fully randomly initialized

Table 6: Set of values tried in training the optimization network

5.2 Results

In the end, we chose to use the standard MHC kernels as initialization of the parameters of our Convolutional Neural Network. We have done that because, as we reported in previous sections (Figure 37), it is important that the parameters of the kernels are organized in a certain pattern. In particular, we want to make sure that the raw input channels information is preserved as much as possible, as it is the true amount of light captured by the camera's photosensors. Instead, by selecting a larger kernel with full-random values, it would be more likely for the parameters to heavily alter the original camera raw information, getting an output with poor-quality colors. However, it is clear that a training process long enough would allow us to reach a good local minimum in the loss landscape (in general it is very hard to reach a global optimum), however, both for time constraints and computational power we went with the more effective and faster solution which consisted in using standard MHC kernels as initial parameters for the network. This solution required also adopting a very small learning rate because the initial point has been already proven to be good in the MHC paper. We chose to use cosine annealing as it could be difficult to escape local minima at the beginning and we wanted to make sure to find other possible local minima by starting with a higher learning rate and then decreasing it over epochs. Overall, the goal was achieved as the trained network parameters allowed us to obtain better quality images, closer to the Ground Truth (clean RGB image) and better than the original Malvar-He-Cutler algorithm, with a

training of 20 epochs (approximately 5 hours). An example of these results is reported in Figure 38 and Figure 39 respectively.

Figure 38: Comparison between trained model result (left) and Ground Truth image (right)

Figure 39: Comparison between trained model result (left) and original MHC result (right)

We also proved the effectiveness of the second term in the loss function, the one related to the demosaicing artifacts regions. In fact, as shown in Figure 40 we can see how the introduction of this term is beneficial with respect to a model trained considering only the L1 loss between the clean RGB image and the RGB predicted image. In this case, however, the difference may be difficult to notice for a non-expert eye.

The results of the optimization process are satisfactory as the visual quality of the image is overall better with respect to the quality obtained through the standard MHC algorithm.

Figure 40: Comparison between the trained model result (left) and the model trained with L1 loss term only (right)

The scores in terms of L1 loss between these two models and the true RGB image are very similar (L1 score is around 0.007 in both cases, considering the full demosaicing test set), however, it is well-known that global metrics are not always indicative of what is happening in very small regions of the image, as we explained in previous sections, and in the context of artifacts reduction we should rely more on the visual results rather than the numerical results. We also tested the performance of the best demosaicing network (which we trained in previous sections) on these new RGB images coming from the optimized version of the Malvar-He-Cutler kernels. We assessed that these new input images, with less demosaicing artifacts, yield a better result in terms of artifacts detected from the segmentation network, which confirms the success of the optimization process. In particular, the average percentage of artifact pixels detected as demosaicing artifacts in an image after the optimization process is 0.9%, while without the optimization process is 1.8%. Both values have been computed considering the 400 images of the test set that we created in the MIT-Adobe FiveK Dataset. Overall, we can say that the optimization process allowed to create RGB images whose quality halves the number of artifacts predicted by the demosaicing artifacts segmentation neural network. An example of this positive result is reported in Figure 41.

For the sake of completeness, we report the optimized Malvar-He-Cutler kernel parameters in Figure 42. Note that the kernels names correspond to the ones defined in the set of equations 12, but we renamed K_2^T as K_7 since the transpose condition is no longer true after the model's training.

Figure 41: Comparison between the demosaicing artifacts prediction with an RGB input which comes from the standard MHC kernels (left), and with an RGB input which comes from the optimized version of the MHC kernels (right).

K_1

0.0651	0	-0.1992	0	0.0627
0	0.2549	0	0.2497	0
-0.1905	0	0.5424	0	-0.1981
0	0.2522	0	0.2508	0
0.0654	0	-0.2028	0	0.0591

K_2

0.0411	0	0.0383	0	0.0372
0	-0.134	0	-0.1383	0
-0.1209	0.5	0.5558	0.5	-0.1245
0	-0.1375	0	-0.1398	0
0.0383	0	0.0395	0	0.0381

K_3

8.26e-4	0.503	-1.71e-3
0.509	-8.31e-5	0.492
4.88e-3	0.497	-2.92e-3

K_4

3.91e-3	0.249	1.28e-2
0.265	-3.41e-2	0.261
6.11e-3	0.257	7.59e-3

K_5

3.99e-2	8.91e-4	-0.1275	2.89e-3	3.72e-2
1.12e-3	-4.98e-3	1.19e-3	3.67e-3	1.87e-3
-0.137	-7.29e-3	0.336	-4.25e-3	-0.127
3.35e-4	3.24e-3	-1.82e-3	-1.06e-2	-1.43e-3
3.41e-2	2.12e-3	-0.121	5.66e-3	4.52e-2

K_6

2.12e-3	0.496	-2.19e-3
0.498	-3.84e-4	0.506
-3.08e-3	0.507	-8.39e-4

K_7

0.0428	0	-0.1202	0	0.0387
0	-0.1403	0.5	-0.1416	0
-0.0429	0	0.5583	0	0.0431
0	-0.1408	0.5	-0.1408	0
0.0373	0	-0.1239	0	0.0386

Figure 42: Optimized kernel parameters for the MHC demosaicing algorithm

Chapter 6

6 Conclusions and future work

The main goal of this thesis work was to perform a study on the possibility to apply Computer Vision algorithms to the difficult task of artifacts segmentation. The work was challenging as we had to create our own datasets with weak labels to be used as information to feed to the Convolutional Neural Networks. Then we had to use the trained models to optimize common image processing algorithms. Nevertheless, we can be satisfied with the results, since the trained models are using a limited number of parameters and they are able to correctly detect common image artifacts with a low inference time. The models have also been successfully used to optimize the parameters of the well-known Malvar-He-Cutler demosaicing algorithm which is an important improvement that may be applied to smartphone camera image signal processor. We could not benchmark our results against any available model since to our knowledge there is no current competitor, however, our work can be considered as a first step towards a new research topic in the image processing domain.

There is certainly room for improvement, as new techniques for weak labeling could be tried and the computational resources dedicated to this project in Huawei Nice Research Center were limited. However, research in the Computer Vision domain is reaching important milestones every year, and so new architectures could be used for improving artifacts segmentation, or new optimization techniques could be applied to the current networks to make them faster and even lighter. Unfortunately, very deep Neural Networks may not be able to fit on smartphones as there are strict hardware requirements, however, in this work we showed how the trained models may be useful to optimize standard image processing algorithms that are implemented in the smartphone camera image signal processor. In conclusion, Convolutional Neural Networks and other Computer Vision architectures are showing their ability to achieve state-of-the-art results across a wide range of applications, learning high-level representations of the data and capturing the underlying patterns, and image processing is certainly a field in which Deep Learning potential is going to radically change the domain.

Acknowledgments

I would like to thank my supervisor Abdelhadi Temmar for his constant support and guidance throughout the internship. His expertise and mentorship have been invaluable and they helped me to achieve my first important career goal at Huawei Nice Research Center, having the chance to work on a challenging project in a world-leading company. I would also like to thank Hadi and the Pixel Reconstruction team from a human perspective because I consider them not only excellent colleagues, but good friends of mine. I want to express my appreciation for the help and suggestions received from my Academic supervisors, Professor Andrea Bottino and Professor Maria Zuluaga, who embraced my project with enthusiasm. I am also deeply thankful to all my friends and the people I met, who taught me a lot and who shared amazing moments with me.

Finally, I want to dedicate special words to my family: my parents Antonella and Carlo, and my sister Noemi. They are my Home and my role models, who enriched my life with endless love and support. They have always been there for me, making me feel the luckiest person in the world and without them, I would not be the person I am today. Thank you with all my heart. I love you unconditionally.

Andrea

Bibliography

[1] Vladimir Bychkovsky, Sylvain Paris, Eric Chan, and Frédo Durand. Learning photographic global tonal adjustment with a database of input / output image pairs. In *The Twenty-Fourth IEEE Conference on Computer Vision and Pattern Recognition*, 2011.

[2] Nima Khademi Kalantari and Ravi Ramamoorthi. Deep high dynamic range imaging of dynamic scenes. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2017)*, 36(4), 2017.

[3] Argparse python library. *Parser for command-line options, arguments and sub-commands.*

[4] Colour python library. *Converts and manipulates common color representation.*

[5] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[6] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007.

[7] Charles R. Harris, K. Jarrod Millman, Stéfan J. van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, September 2020.

[8] Os python library. *Miscellaneous operating system interfaces.*

[9] Pathlib python library. *Offering a set of classes to handle filesystem paths.*

[10] Fredrik Lundh and Contributors. Pil python library. *Python Imaging Library.*

[11] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0

Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020.

[12] Shutil python library. *High-level operations on files and collection of files.*

[13] Stefan Van der Walt, Johannes L Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D Warner, Neil Yager, Emmanuelle Gouillart, and Tony Yu. scikit-image: image processing in python. *PeerJ*, 2:e453, 2014.

[14] Time python library. *Time access and conversions.*

[15] NVIDIA, Péter Vingelmann, and Frank H.P. Fitzek. Cuda, release: 10.2.89, 2020.

[16] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[17] Torchvision python library. *Popular datasets, architectures and common image transformations for Computer Vision.*

[18] Urllib python library. *Collecting several modules for working with URLs.*

[19] Beyond compare 4. *Multi-platform utility that combines directory compare and file compare functions in one package.*

[20] Pascal Getreuer. Malvar-he-cutler linear image demosaicking. *Image Processing On Line*, 1, 08 2011.

[21] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[22] Carole H. Sudre, Wenqi Li, Tom Vercauteren, Sebastien Ourselin, and M. Jorge Cardoso. Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations. In *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support*, pages 240–248. Springer International Publishing, 2017.

[23] Seyed Sadegh Mohseni Salehi, Deniz Erdogmus, and Ali Gholipour. Tversky loss function for image segmentation using 3d fully convolutional deep networks, 2017.

[24] Paul E. Debevec and Jitendra Malik. Recovering high dynamic range radiance maps from photographs. *SIGGRAPH 97*, August 1997.

[25] M.A. Robertson, S. Borman, and R.L. Stevenson. Dynamic range improvement through multiple exposures. In *Proceedings 1999 International Conference on Image Processing (Cat. 99CH36348)*, volume 3, pages 159–163 vol.3, 1999.

[26] Tom Mertens, Jan Kautz, and Frank Van Reeth. Exposure fusion. *15th Pacific Conference on Computer Graphics and Applications (PG'07)*, pages 382–390, 2007.

[27] Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation, 2017.

A Appendix

A.1 Role of luminance in 3D sRGB Gamut

We report the visualization of the sRGB gamut in 3D through the *colour* library. The third dimension is used to represent the luminance information.

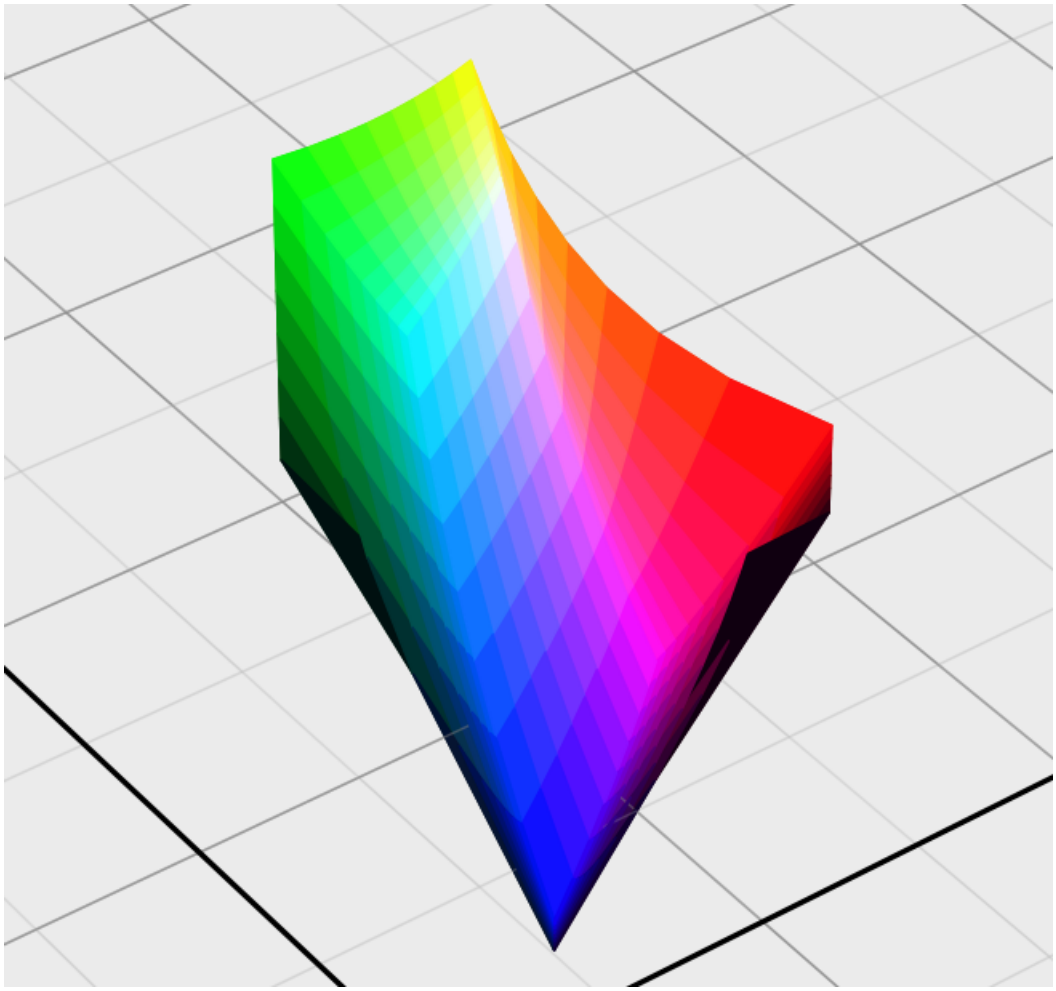

Figure 43: 3D sRGB Gamut, with the role of luminance

A.2 Additional equations in Malvar-He-Cutler algorithm

We report the equation to estimate a red component at a green pixel location (15), the equation to estimate a red component at a blue pixel location (16), and the optimal coefficients for the Malvar-He-Cutler algorithm (17).

$$\widehat{R}(i,j) = \widehat{R}^{bl} + \beta\Delta_G(i,j) \tag{15}$$

$$\widehat{R}(i,j) = \widehat{R}^{bl} + \gamma\Delta_B(i,j) \tag{16}$$

$$\alpha = \frac{1}{2}, \quad \beta = \frac{5}{8}, \quad \gamma = \frac{3}{4} \tag{17}$$

B Appendix

B.1 From Tversky Index to Dice Score

By exploiting Set Theory in binary classification problems (44) and in Dice formulation (45), we can mathematically prove that the Dice Score is equal to the Tversky Index when β is 0.5, as reported below:

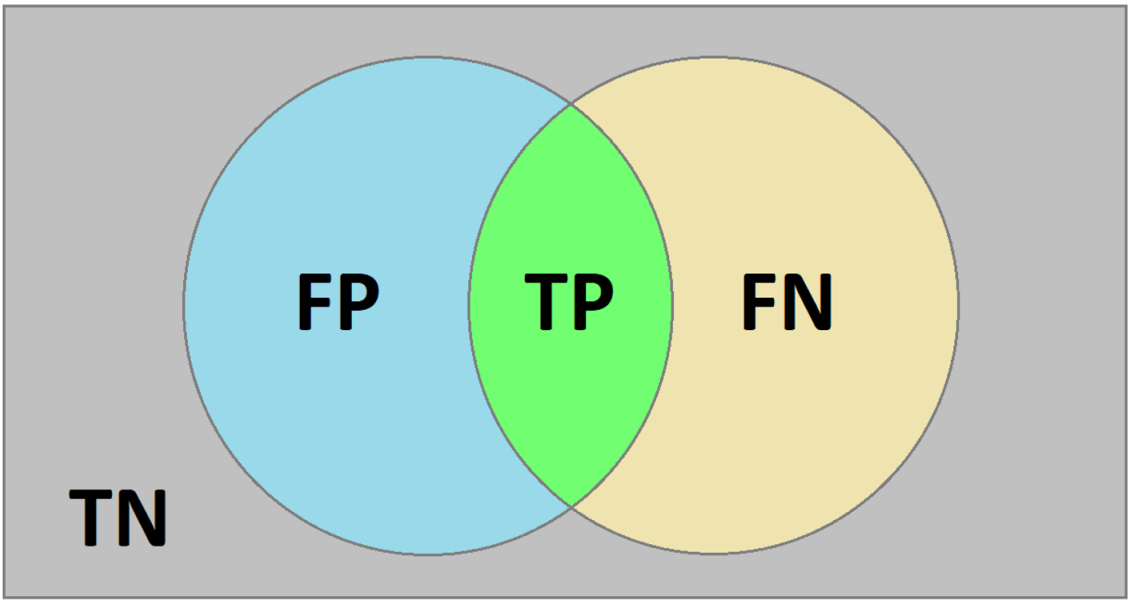

Figure 44: Binary classification problem through Set Theory

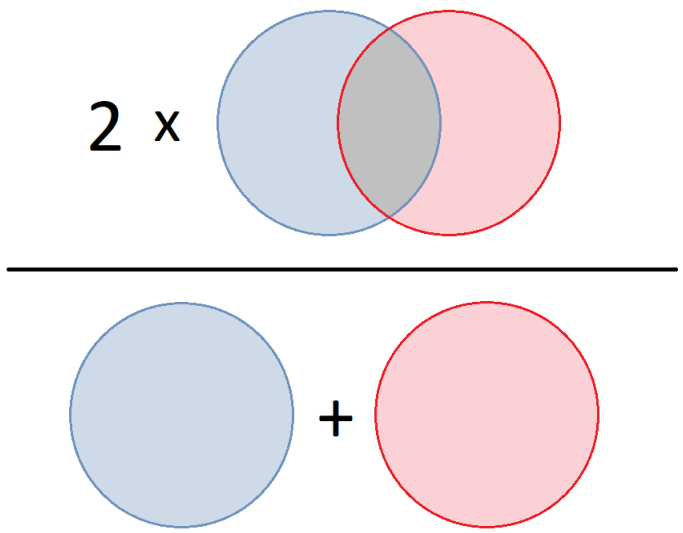

Figure 45: Dice Score representation in terms of sets

$$
\begin{aligned}
Dice_{Score}(y, \hat{y}) &= \frac{2y\hat{y} + \epsilon}{y + \hat{y} + \epsilon} \\
&= \frac{\mathbf{2TP}}{\mathbf{2TP + FP + FN}}
\end{aligned}
$$

$$
\begin{aligned}
Tversky_{Index}(y, \hat{y}) &= \frac{y\hat{y} + \epsilon}{y\hat{y} + \beta(1-y)\hat{y} + (1-\beta)y(1-\hat{y}) + \epsilon} \\
&= \frac{TP}{TP + \beta FP + (1-\beta)FN} \\
&= \frac{TP}{TP + 0.5FP + 0.5FN} \\
&= \frac{\mathbf{2TP}}{\mathbf{2TP + FP + FN}}
\end{aligned}
$$

C Appendix

C.1 Malvar-He-Cutler initial kernel values

We report the original kernel values which are used to replicate the Malvar-He-Cutler algorithm at a programming level. They have been used as initialization for the optimization model. Note that the kernel names correspond to the ones defined in the set of equations 12.

K_1

0	0	-0.1875	0	0
0	0.25	0	0.25	0
-0.1875	0	0.75	0	-0.1875
0	0.25	0	0.25	0
0	0	-0.1875	0	0

K_2

0	0	0.0625	0	0
0	-0.125	0	-0.125	0
-0.125	0.5	0.625	0.5	-0.125
0	-0.125	0	-0.125	0
0	0	0.0625	0	0

K_3

0	0.5	0
0.5	0	0.5
0	0.5	0

K_4

0	0.25	0
0.25	0	0.25
0	0.25	0

K_2^T

0	0	-0.125	0	0
0	-0.125	0.5	-0.125	0
0.0625	0	0.625	0	0.0625
0	-0.125	0.5	-0.125	0
0	0	-0.125	0	0

K_5

0	0	-0.125	0	0
0	0	0	0	0
-0.125	0	0.5	0	-0.125
0	0	0	0	0
0	0	-0.125	0	0

K_6

0	0.5	0
0.5	0	0.5
0	0.5	0