MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITÉ ABDELHAMID IBN BADIS – MOSTAGANEM

Faculté des Sciences Exactes et d'Informatique

Département de Mathématiques et informatique

Filière: Informatique

MÉMOIRE DE FIN D'ÉTUDES

Pour l'Obtention du Diplôme de Master en Informatique

Option: **Ingénierie des Systèmes d'Information**

Présenté par:

Benkedadra Mohamed

THÈME:

Image Enhancement Based On Supervised Learning

Soutenu le : 10 / 09 / 2020

Devant le jury composé de :

Hassain Farida	MAA	Université de Mostaganem	Président
Hocine Nadia	MCB	Université de Mostaganem	Examinateur
Moumene Mohammed El Amine	MCB	Université de Mostaganem	Encadrant

Année Universitaire 2019-2020

Abstract

Dynamic range increase methods are used for creating more detailed close to real-life representations of captured scenes, with challenging lighting conditions. These methods vary in quality, speed, and processing power. In this research project, we designed and implemented a novel solution for dynamic range increase. Using machine learning techniques, we created EFNN, a neural network that does exposure fusion at higher speeds than the traditional methods, while also not compromising on quality. Our method takes half the time that traditional exposure fusion takes, while keeping the same level of detail. Hence, it allows further research into real-time dynamic range increase. while facilitating the creation of production-level solutions that use dynamic range increase in order to solve pressing hard to solve problems, like autonomous vehicle crashes due to poor computer vision in extreme lighting conditions.

Résumé

Les méthodes d'augmentation de la plage dynamique sont utilisées pour créer des représentations plus détaillées et plus proches de la vie réelle des scènes capturées dans des conditions d'éclairage difficiles. Ces méthodes varient en qualité, en vitesse et en puissance de traitement. Dans ce projet de recherche, nous avons conçu et mis en œuvre une nouvelle solution pour l'augmentation de la plage dynamique. En utilisant des techniques d'apprentissage automatique, nous avons créé EFNN, un réseau de neurones qui effectue une fusion d'exposition à des vitesses plus élevées que les méthodes traditionnelles , sans compromettre la qualité. Notre méthode prend deux fois moins de temps que la fusion d'exposition traditionnelle, tout en conservant le même niveau de détail. Cela ouvre également la porte à la création de solutions aux niveaux industriel. Comme la résolution des problèmes d'accidents de véhicules autonomes causés par une mauvaise vision par ordinateur dans des conditions d'éclairage difficiles.

ملخص

تُستخدم طرق زيادة المدى الديناميكي لإنتاج صور ذات تفاصيل واقعية للمشاهد التي يتم التقاطها في ظروف إضاءة صعبة. تختلف هذه الأساليب في الجودة والسرعة وقوة المعالجة الضرورية. في هذا المشروع البحثي، قمنا بتصميم وتنفيذ حل جديد لزيادة مدى النطاق الديناميكي. باستخدام تقنيات التعلم الآلي، أنشأنا شبكة عصبية اصطناعية تنتج صور ذات نطاق ديناميكي واسع بسرعات أعلى من الطرق التقليدية، مع ضمان حسن الجودة . حيث تستغرق طريقتنا نصف الوقت الذي تستغرقه الطرق التقليدية مع الحفاظ على نفس المستوى من التفاصيل. كما أنها تفتح المجال لإنشاء حلول على المستوى الصناعي والإنتاجي. مثل حل مشاكل حوادث المركبات الآلية بدون سائق التي تنجم عن ضعف رؤية الكمبيوتر في ظروف الإضاءة القاسية.

Dedications

I dedicate this to anyone who's reading this and is too tired of whatever life has thrown at them. May Allah bless you with happier days.

Acknowledgements

Praise be to ALLAH, his Majesty for his uncountable blessings, and best prayers and peace be unto his best messenger Mohammed peace be upon him.

I would like to begin by remembering my father. It does not sadden me that he's gone as Allah Knows best. It saddens me that I can't brag to him about becoming an adult. as I often wonder if i'm capable enough to replace his presence.

I would like to state my love and admiration to my second role model after our teacher Mohammed (pbuh). My second teacher ... My mother. As words cannot describe my appreciation for what you do I can only say that I love you.

I would like to thank my supervisor, Mr Moumene Mohammed El Amine for being such an awesome humble person. I would like to thank him for giving me space to work as I like, while guiding me when i'm stuck or lost.

I Would like to thank "Ammi Kadda" and "Tata Saliha" my second father and mother. It's rare for one to have a good support system in today's society. So, I thank god everyday that they are in my life, my allah bless them with happiness.

I would like to thank my sisters, may allah guide them in their work and studies into becoming strong, independent, and happy.

I would like to thank Mr. Roukh Amine for always being my teacher. Mrs. Hocine Nadia for being such an inspiration. Mr. Hallam Amine for guiding me in the industry. and for everyone who has ever been nice to me, Not having your name here isn't a sign of my ingratitude, excuse my forgetfulness as doing this research project has tired my mind and soul.

List of Figures

List of Tables

Abbreviations

AE Autoencoder

AI Artificial Intelligence

AV Autonomous Vehicule

CNN Convolutional Neural Network

CRF Camera Response Function

EB Exposure Bracketing

EF Exposure Fusion

EFNN Exposure Fusion Neural Network

EV Exposure Value

HDR High Dynamic Range

HDRI High Dynamic Range Imaging

LDR Low Dynamic Range

ML Machine Learning

NN Neural Network

RNN Recurrent Neural Network

Contents

Introduction

Nowadays, vehicle manufacturing companies are starting to get into the Artificial Intelligence (AI) movement. As a result, self driving cars are no longer a part of our imagination. Companies like GOOGLE and TESLA are leading the race for full autonomy, and open source projects like COMMA.AI [1] are gaining a lot of attention.

The Autonomous Vehicule (AV) needs to be able to safely function in extreme conditions, in order for it to reach a level of full autonomy. This can not be achieved, unless the AI can detect the vehicle's surroundings with high accuracy at all time. Unfortunately, that is not always the case, since the different sensors on the vehicle, could find some difficulties in some scenarios. In that event, the AI will not have sufficient data for correct decision making, which will decrease the safety level of the AV as it becomes prone to making mistakes.

The camera is one of the most important sensors on the AV. It is usually used for the detection of lines on the sides of road lanes, which are hugely responsible for the decision making that the AI does, while steering the vehicle. But, cameras are tedious to use when light conditions are constantly changing. Because they are light capturing devices that capture light and turn it into an image, the changes in the lighting intensity can effect the camera's ability to capture a scene.

For example, when the camera is facing the sun, its flare will cause an increase in the scene's dynamic range, which is the difference between the darkest and the brightest lights in that scene. This High Dynamic Range (HDR) is usually not supported by the camera's light capturing sensor, causing it to focus on a sub dynamic range. As a result, areas with lighting intensities outside of that sub-range are not going to be captured correctly, which causes a loss of details. It should be noted that this issue is camera centred and does not only touch lane lines detection systems, but is present in all camera dependant fields and methods, like photography.

In this research project, we will explore the difficulties of correctly exposing images when faced with extreme lighting conditions. Starting from a case study of road lane lines detection systems, to understand how extreme lighting and HDR affect cameras in real production environments. We will generalize the discovered problem and study it on a broader scale, which will require an introduction to basic photography concepts, like bracketing, exposure and dynamic range.

After getting a grasp on the idea of dynamic range and what problems it could cause to image capturing devices, we will take a look at currently used High Dynamic Range Imaging (HDRI) solutions and try to understand them. This will be followed by an introduction to Machine Learning (ML) and Neural Networks (NNs). An approach that will help us create a better more optimized solution for HDR problems, by eliminating many of the resources heavy operations that are currently used in the HDRI pipeline. As we will replace this set of operations by a single fast and reliable NN which we call Exposure Fusion Neural Network (EFNN).

Chapter 1

Case study : The Effects Of High Dynamic Range On Road Lane Lines Detection

1.1 Introduction

In order to understand the problem at hand, we chose to take road lane lines detection systems as a case study, since they are dependent on cameras for computer vision.

While taking into consideration that there are many ways to detect objects, shapes, and in our case, lines. We are going to be taking a look at only one of them [2], as the main focus of this project isn't on the actual lane lines detection in Autonomous Vehicules (AVs), but rather on the problem that cameras face in extreme lighting conditions.

1.2 Road lane lines detection

The camera on the AV captures video at X number of frames per second. Each frame is preprocessed by converting it into a grayscale image, then darkened in order to have more space between the darkest and lightest parts of the scene (Figure 1.1).

(A) Camera captured frame (B) Grayscale image (C) Darkened grayscale image

FIGURE 1.1 – Image preprocessing in a simple lane lines detection system

The preprocessing phase lightens the lane lines and darkens the rest of the image, in order to be able to isolate the lane lines using a color mask for each line color, mainly white and yellow (Figure 1.2). These masks are extracted from the original colored image, combined with each other using a bit-wise OR operation and applied to the darkened image using a bit-wise AND operation.

FIGURE 1.2 – Lines isolation using color masks

After the isolation phase, we run an edge detection algorithm like the "Canny Edge Detector" [3] that allows for the detection of lane lines' edges (Figure 1.3). This is where the actual lane recognition starts taking place, as the previous steps were all preprocessing techniques, that needed to be applied to facilitate the detection process.

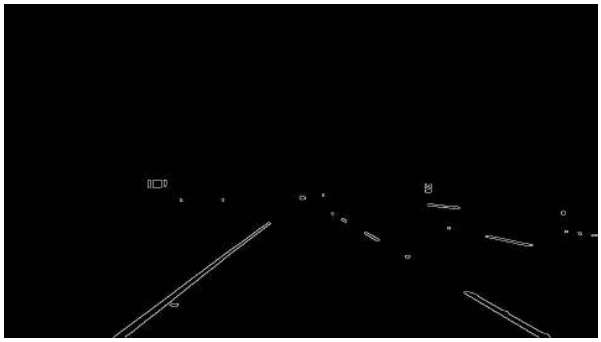


FIGURE 1.3 – Result of running the canny edge detector

The camera might capture other lanes or objects that are similar in color to the lane lines. Hence, the computer vision system needs to focus its attention on the lane that the AV is on. This processes is called "Selection of the Region of Interest" (Figure 1.4) [2]. Simply put, we are going to select a polygon that is decided upon in advance, according to the way in which the camera is setup. This polygon will define what region of the image we need. Consequently, the rest of the pixels in the image are ignored.

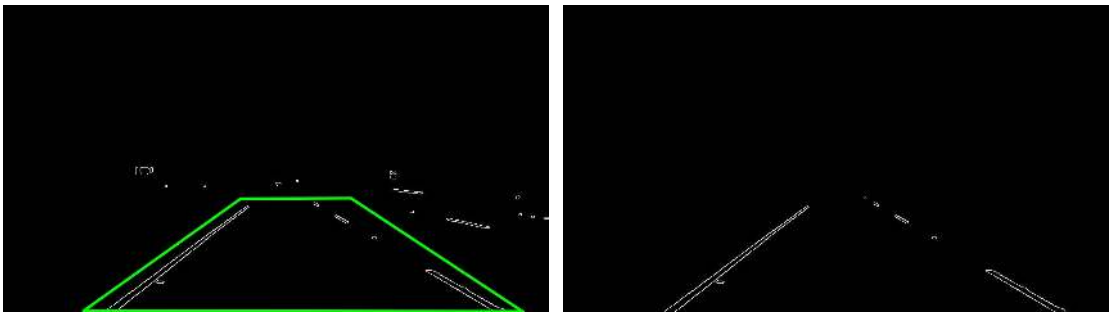


FIGURE 1.4 – Selection of the region of interest

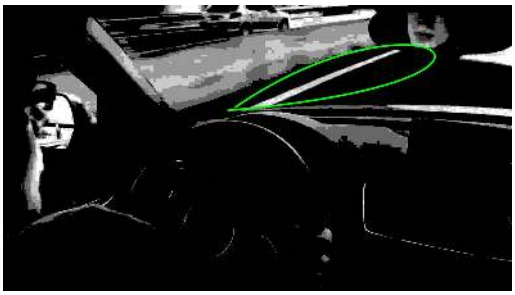
This is how the first steps of the detection process are done. In fact, other steps will follow, like the need to run other algorithms to unify the detected edges into two lane lines, which are used to define the AV's lane. The main approach being the "Hough Transform" [4], where after generating the "Hough Space" which is the space of the lines parameters, lines having a greater accumulation of edge points are returned. "Radon transform" [5] is also widely used to detect lines on gray scale images.

FIGURE 1.5 – Hough transformation for lane lines detection

1.2.1 Extreme lighting conditions' effects on road lane lines detection

Now that we understand how road lane lines detection is done, we can easily comprehend the problems that the detection pipeline might face in extreme lighting conditions. To demonstrate the issue at hand, we took two different frames from a recording of a moving vehicle. These two frames, represent two possible lighting conditions.

As shown in Figure 1.6, when the previously seen detection steps are applied to the first frame, the lane line is easily recognized. On the other hand, the second frame doesn't allow for the lane line detection. Actually, it is impossible to see the lane line even in the original captured image.



(A) Frame 1 - Normal lighting conditions

(B) Frame 2 - Extreme lighting conditions

FIGURE 1.6 – Lane lines detection in two different lighting conditions

This issue is a result of the camera receiving more light than it expects, causing the image to be "Over Exposed". This means that the camera is not able to capture details in areas with a lot of light. But is able in return, to capture more details in areas with little or dim lighting.

1.3 Understanding dynamic range and exposure

Cameras construct images from light. If there is no light, there is no image. This is also the same way the humain eye works on a basic level. When a picture is taken, the camera's sensor which is a light sensitive surface, captures light particles that are coming in into the camera's lense and uses them to create an image.

Even with the human eye, we cannot always see everything in front of us. When two areas in the same scene have two different lightings, we may be able to see one area and not be able to see another, depending on which area we direct our attention to. This phenomenon is also present in cameras, where it is impossible to capture a single shot of a scene, with two drastically different lighting conditions. These scenes are called High Dynamic Range (HDR) scenes.

Dynamic range is the difference between the least and the most intense lights in a scene. When the dynamic range is too wide for the camera's sensor, a sub-range is captured causing details to be lost in the areas where lighting intensity is outside of that captured sub-range. Consequently, forcing the camera to produce a Low Dynamic Range (LDR) image (Figure 1.7), which is a term that denotes that the captured image doesn't contain all the correct scene information, because of inconsistencies between that scene's real life dynamic range and the camera's light capturing capabilities.

FIGURE 1.7 – LDR image of an HDR scene that shows failure in capturing sky details

The setting that defines what sub-range of light intensity the camera should focus on, is the Exposure Value (EV). It defines how much light the camera is exposed to. The more light the camera should expect, the less light it should be exposed to.

(A) 3 Stops under exposed (-3 EV)

(B) 3 Stops over exposed (+3 EV)

FIGURE 1.8 – Changes in captured details when changing EV for the same scene

As shown in Figure 1.8, decreasing the EV means that the camera should expect intense lighting, which decreases the camera's opening to under expose its sensor to light. As a consequence, allowing it to capture areas with intense lighting. This on the other hand, makes it unable to capture areas with low or dim lighting. As we increase the EV, the camera's opening widens. Hence, allowing more light to come in. This will provide the sensor with the ability to capture areas with low lighting, but makes it unable to capture areas with intense lighting.

Now that we understand what exposure is, we can easily see how it could create a problem. Modern cameras often come with a built-in auto exposure feature, that helps beginners expose scenes correctly. But, the automatic exposure process is often slow and sometimes inaccurate. In addition, it requires a good amount of hardware and software to be useful in a real time system. And, even if we choose to use advanced camera equipment, the lighting conditions might change faster than the process that automatically calculates and sets the correct EVs.

1.4 Conclusion

As you can easily guess, correctly exposing images isn't an issue that touches AVs alone, but is present in every field that deals with LDR cameras. Mainly, the field of photography. Therefore, we aren't going to be focusing our research on solving exposure based problems in lane lines detection alone. Instead, we have chosen to broaden our research, where we are going to be focusing on dynamic range and exposure issues in photography in general. As a way to generalize both the problem and the solution, and try to touch and improve as many domains as possible.

In the next chapter, we are going to be learning about the currently used techniques and solutions, and try to understand them on a basic level.

Chapter 2

State of the Art : High Dynamic Range Imaging (HDRI) Techniques

2.1 Introduction

From what we went through in the previous chapter, we understand that exposure and dynamic range create many challenges in the field of photography. That's mainly because of inconsistencies between possible luminance ranges in scenes, and their support in capture devices and in display technologies (Figure 2.1).

FIGURE 2.1 – Dynamic range in the world and its support

In order to compress the huge variations of light into a displayable image, and make sure that what we see in our displays, and what we capture with our cameras is the same as what exists in the real world, many technologies and solutions have appeared. In this chapter, we will take a look at some of them.

2.2 High Dynamic Range Imaging (HDRI)

It's a set of techniques and methods used in order to correctly capture HDR scenes with minimum information loss, into either HDR or LDR image formats. These techniques allow us

to capture, process and store the huge amount of information that exists in a real life scene, into an image using an LDR camera. They cover the entire imaging pipeline, from the capturing of the scene to the display of the image (Figure 2.2).

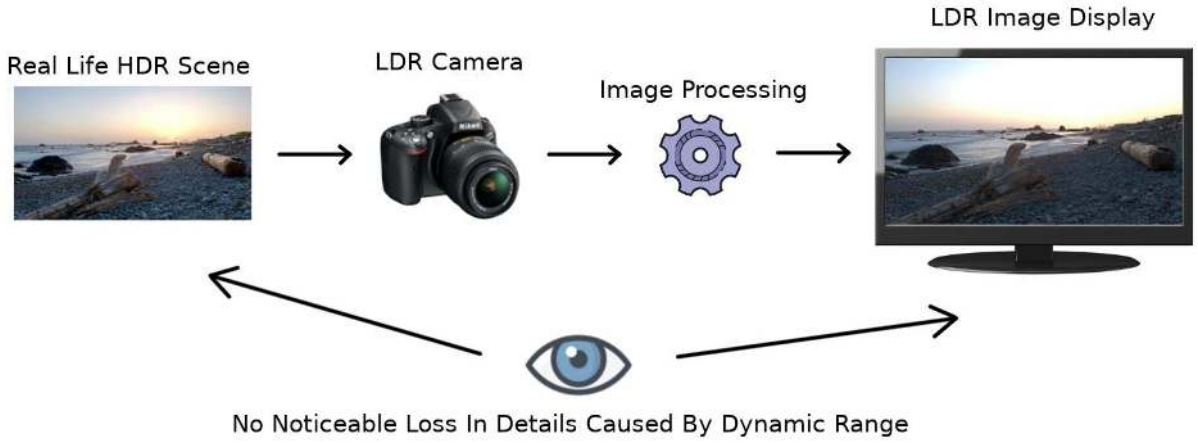


FIGURE 2.2 – Simplified HDRI pipeline for displaying HDR images in an LDR format

The HDRI pipeline starts from the scene capturing phase. Exposure Bracketing (EB) [6, 7] is the process of taking multiple pictures of the same scene with different Exposure Values (EVs). It is used to acquire multiple exposures of the same scene. This allows for the capturing of different details in different areas, with different lighting conditions (Figure 2.3).

FIGURE 2.3 – Exposure Bracketing (EB) from -3EV to +3EV (+1EV per picture left to right)

The captured images need to be aligned, meaning that movement needs to be reduced to a minimum when doing manual EB, which consists of manually changing the EV and taking different pictures of the scene. Consequently, camera manufacturers added Auto EB as a setting that allows photographers to set the “Stops Spacing” (the difference between EVs) and the number of images to take, and with only one click, multiple images are produced with instant

continuous capturing at different stops (EVs).

What follows is the creation of an image that contains a close to true representation of the scene's real dynamic range, using the captured set of images. For this, many methods can be used. But, most if not all, start with an optional but recommended alignment process, in order to be able to merge the different images into a single one, that contains the most important details from each one of those images, without having any issues like image ghosting.

Many algorithms can be used to do High Dynamic Range Imaging (HDRI). We are going to be taking a look at two of them, since they are the ones that are used the most, and the ones that inspired the solution that we propose at the end of this report.

The first method is simply known as HDR [8, 9, 10, 11]. It starts with a Camera Response Function (CRF) calibration, followed by an image fusion process to create a 32bit HDR image. That image will be later converted using a tone mapping operator [12], into an 8bit more detailed LDR image, that is easier to work with and display.

The second solution is Exposure Fusion (EF) [13]. It is the easier one to do, since it simply merges multiple LDR images with different EVs into a single LDR image, containing as much details as possible. This is done in a single process, without going through the inconvenience of CRF estimation, HDR image creation and tone mapping operator application.

In order to facilitate the creation and manipulation of HDR images, special image formats like "OpenEXR" [14] have appeared. These formats empower the HDR pipeline by permitting the storage of more information per pixel. In fact, they can store up to 32bit float values per color channel, allowing for an extremely wide range of possible brightness levels. As opposed to LDR image formats like JPG, that are only able to store up to 8-bit per channel, limiting the representation to only 256 different levels of brightness, which is not enough to display all the variations of brightness in an HDR scene.

The amount of information that can be stored per pixel using these HDR image formats, makes the representation of many different levels of details throughout different levels of brightness possible. In addition, it allows for more control over the scene's final representation.

Changing the brightness of an HDR image, will reveal details in areas and hide details in others. As a result, it looks as if we were fading in and out and transitioning between the different LDR images used to create the HDR one.

2.2.1 Camera Response Function (CRF) and Tone Mapping

The HDR method starts with image alignment, followed by a Camera Response Function (CRF) calibration. A process that provides a lot of information about the captured scene. Information that is required for the LDR to HDR reconstruction using HDR image fusion.

CRF calibration or estimation consists of analysing different images of the same scene with different EVs, in order to extract the Camera Response Function (CRF). This function is a form of signature that provides a natural basis for the captured scene. And, most importantly, can be used to find a relation between the different exposure intensities in the different images and the real life light intensities in the the different areas of the scene.

CRF is followed by an image fusion operation, that reconstructs the different LDR images into a single 32bit HDR image containing much more information per pixel. In hopes of bringing us closer to the scene's true dynamic range.

After the HDR image is created, a tone mapping operator is used to produce an 8 or 16bit LDR image. That image provides a better representation of the scene, while needing the same amount of space for each pixel as a normal single photographe.

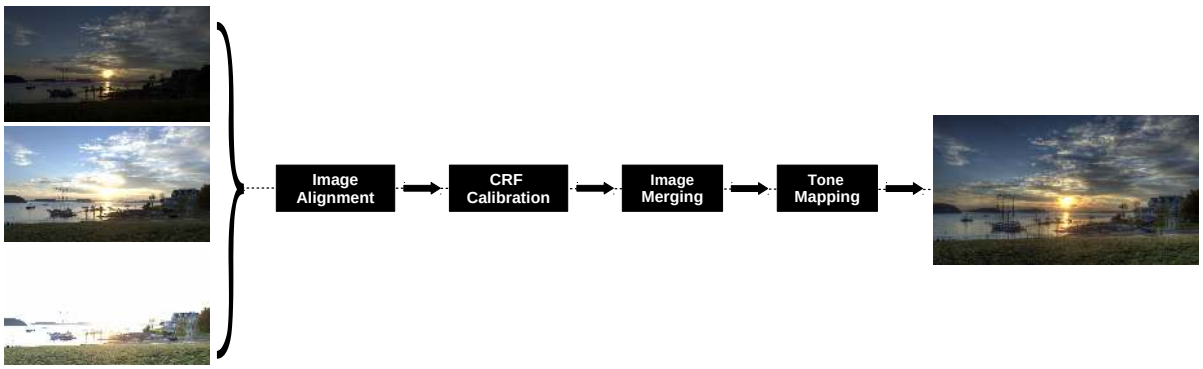


FIGURE 2.4 – HDRI using image fusion and tone mapping

CRF, image fusion, and tone mapping, are what defines the HDR method from Exposure Fusion (EF), as they allow for far more control over the result, which may come with its downsides. While the usage of different tone mapping operators provides the ability to create different possible LDR images from the HDR one (Figure 2.5), the resulted images may look unrealistic (BAD HDR) [12, 15], if the tone mapping operation was poorly done.

FIGURE 2.5 – Tone mapping examples using Photomatix 5.0 Presets

left to right : Natural, Balanced, Painterly (BAD HDR)

2.3 Exposure Fusion (EF)

In comparison to the HDR method, Exposure Fusion (EF) which was improved upon and made popular by the "Exposure Fusion" paper by Tom Mertens, Frank Van Reeth and Jan Kautz [13], takes less time. As it bypasses the HDR image creation process that requires both CRF calibration and image fusion. In addition, it doesn't include the usage of a tone mapping operator, since it creates an LDR image that contains more scene details, directly from other LDR images with different EVs.

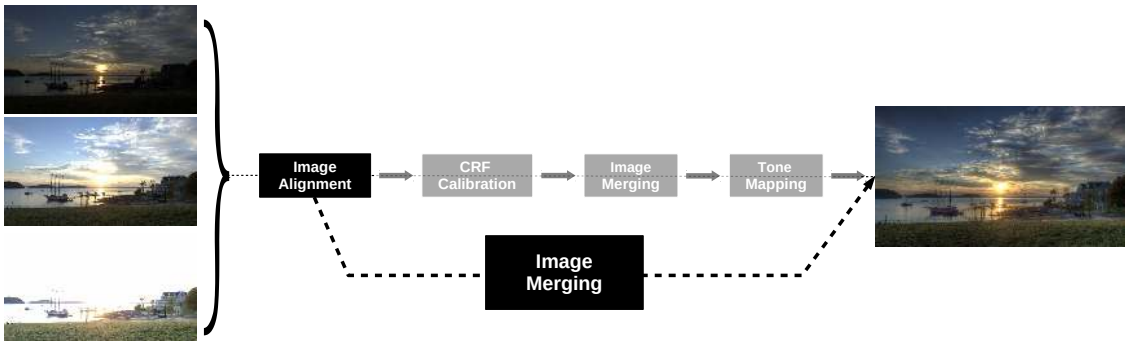


FIGURE 2.6 – High Dynamic Range Imaging (HDRI) using Exposure Fusion (EF)

This is done by calculating different quality mesures for each pixel across the different LDR images, to produce weight maps that represent what pixels in which bracketed image, contribute more to the overall quality of the final representation of the captured scene. The weight maps are later blended with the different images to construct a single more detailed image, containing the best details from each one of those images (Figure 3.3). The calculated quality mesures which were mentioned in the EF paper are :

- Contrast: Emphasises on edges, shapes, shadows, textures ..etc

- Saturation: Makes sure that vividly colored areas are prioritized.
- Well-Exposedness: How well an area has been exposed can be analysed from each pixel and used in order to decide on what areas were well exposed and what areas were not.

FIGURE 2.7 – Exposure Fusion (EF) guided by weight maps calculated using quality mesures for each input image

This approach provides less control over the result in comparison with the previously seen HDRI solution. As there is no usage of tone mapping operators, which permitted us to achieve different results. In return, the restriction that EF puts on the process by simplifying it, guarantees a more realistic result. Furthermore, it requires less processing power than the HDR method, since no actual HDR image is created which bypasses the need for many operations and algorithms.

We must keep in mind that post processing can be applied to the result to achieve different HDR effects. But, standalone EF gives a constant output for the same input, as it is simply cherry-picking the best quality set of pixels from each LDR image and using them to create a new more detailed one.

2.4 Image ghosting

Both High Dynamic Range (HDR) and Exposure Fusion (EF) depend on merging different pictures with different EVs, which can cause issues like Image Ghosting (Figure 2.8). This phenomenon happens when the scene is changed by movement of either the camera or the subjects. Hence, it is crucial that we use a tripod and preferably align the different pictures before merging them. Image alignment can be done through different techniques, from simple edge detection to ML based solutions. Additionally, deghosting algorithms [16] can be used to avoid this phenomenon.

FIGURE 2.8 – Severe image ghosting example

2.5 Popular HDRI solutions

Many libraries offer implementations of HDRI algorithms. The most popular one being “OpenCV” [17] that provides implementations for both the “Debevec” [9] and the “Robertson” [10, 11] algorithms for CRF calibration, radiance map extraction and image fusion. Furthermore, it provides many algorithms for tone mapping. like the “DRAGO” tone mapping operator, which is an implementation of the “Adaptive Logarithmic Mapping For Displaying High Contrast Scenes” paper [18]. Or, the “Durand” tone mapping operator, which is an implementation of the “Fast Bilateral Filtering for the Display of High-Dynamic-Range Images” paper [19]. Additionally, Exposure Fusion (EF) can also be done in “OpenCV” using its implementation of the popular previously seen, “Mertens” EF Algorithm [13].

Other solutions are more user friendly. The most popular one being “Photomatix” [20], a proprietary piece of software used to do both HDR and EF. There is also “Luminance HDR”, a more open source solution, and “Aurora HDR” which uses Artificial Intelligence (AI) for tone mapping, image fusion, and other parts of the HDRI pipeline.

2.6 Conclusion

As we do not need to comprehend every single detail about every single HDRI technique to understand the subject at hand, we will not be going more in depth into other methods. Instead, having a general idea, allowed us to understand what High Dynamic Range Imaging (HDRI) is, and how it’s done. This is more than enough to understand the proposed solution in the next chapters.

Chapter 3
Contribution : HDRI Using Neural Networks (NNs)

3.1 Introduction

As seen in the previous chapter, HDRI consists of many resources hungry algorithms, methods and operations. HDR image reconstruction and tone mapping for example are both taxing on system resources and tedious to execute on not so powerful machines. They also need to be preceded by CRF calibration, an additional computational step that slows down the pipeline.

EF might seem like a better and faster approach. But, it also depends on many time consuming steps, from calculating quality measures for each exposure, to generating weight maps and doing image fusion. As a consequence, we have chosen to create a lighter solution by eliminating all of these operations and algorithms, and replacing the entire pipeline with a single ML fusion model.

This new method will take in multiple exposures as an input and produce a more detailed fusion as an output, without the need for additional computational steps. As a result, we hope to guarantee a consistent high quality output, across different types of scenes, and hopefully achieve real time HDR.

In this chapter, we are going to take a look at Supervised Machine Learning (ML) and Neural Networks (NNs) [21, 22] which are the basis for our HDR ML based solution.

After getting an understanding over some basic ML concepts, we will dive into our proposed solution and get to know the Exposure Fusion Neural Network (EFNN) pipeline.

3.2 Supervised Machine Learning (ML)

Children often learn by example and not by orders or instructions. When a child is taught what a pen is, a child is shown examples of pens, in different shapes, sizes and styles, giving the child the ability to recognize pens. This is a better approach in comparison with teaching the child by defining what a pen looks like.

Machine Learning (ML) follows the same principle, where typed machine instructions like statements and conditions are replaced by algorithms, often called "Models". These models can be trained using previously known examples, and later used to predict, recognize, classify, create or do other possible operations, on newer never before seen examples. Therefore, ML algorithms are just a way for machines to learn and reason like humans do, using a blend of statistics, probability, mathematics and computer science.

The process of teaching the machine using older samples or examples is the training phase. An iterative process where examples are passed through the model over and over again, hoping that it would learn patterns from those examples' inputs and outputs. In fact, a Supervised Machine Learning (ML) model is just a series of mathematical formulas. The goal of the training phase is to "FIT" those formulas, which is another way of saying that it tries to find the formulas' coefficients or weights. These weights are the deciding factor, on the amount of impact that each part of the input also called feature, is going to have on the output.

In the training phase we use a special mathematical function called the "COST" function, which mesures the error on each training iteration by calculating the difference between the samples' ground truth outputs and the outputs that the model produces. The iterative training process aims to lower that measured error by changing the model's weights in order to have an output that is close to the ground truth. Some of the most popular used functions in the training phase are the "Mean Squared Error" for cost and the "Adam Optimizer" or "Gradient Descent" for weight fitting and optimization.

The training process is followed by a testing phase, where newer examples are used to evaluate the quality of the model, using various mesures to test its efficiency, since it's prone to making mistakes just like humans do. After which the model is deployed to be utilized in a generalized fashion in a production environnement.

If we were to relate ML to the way students learn, we would say that the training phase

would be the cours and home exercises, while the testing phase would be something like a test or an exam. Where students can test how good they are in a specific subject, allowing them to know what kind of value they can bring to the real world.

3.2.1 Supervised vs unsupervised Machine Learning (ML)

Supervised Machine Learning (ML) algorithms take in previously labeled examples. These examples can later be used for "Regression", where the model learns how to do predictions on newer ones, or "Classification" where the examples' different labels are treated as classes and the model learns how to label newer examples and classify them into those classes correctly.

Unsupervised ML algorithms on the other hand take in unlabeled samples, and can be used to cluster them into different groups also called clusters, by finding relations between them.

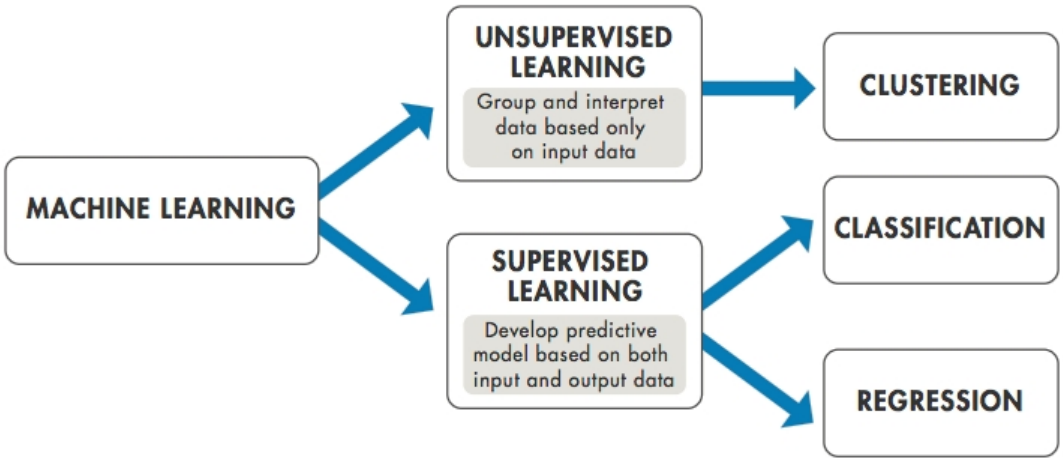


FIGURE 3.1 – Supervised vs unsupervised Machine Learning (ML)

3.2.2 Regression problems

An example of a regression problem would be the pricing of a house, according to the current real estate market. For that, we could create a house price prediction model, by providing samples of different houses that were recently sold. Each sample is a set of different features of a sold house, such as the age of the house, the size, the number of rooms ... etc, and each sample has a label, which in this case is the price of the house. The model will be trained using these samples, in order to be later used for predicting the best possible price of a new unpriced house according to the market.

This is the same approach that we are going to be taking in our proposed Supervised ML HDRI solution, where we will be feeding a ML model different samples of HDR scenes. Each sample is a set of different LDR exposures, taken using automatic EB, and is labeled with an already created Exposure Fusion (EF) image. As a consequence, we hope that the model will be able to learn how to produce a more detailed LDR scene representation, using a set of different LDR exposures of that scene. We also hope that it will be able to output a consistante quality throughout different scenes and different lighting conditions or different dynamic ranges.

3.3 Neural Networks (NNs)

In our search for the best ML methods and algorithms that would suit the task at hand, we have come to see Neural Networks (NNs) as a great way to take HDRI to the next level. As they are a very powerful tool, that appeared as an attempt to simulate the human's brain learning process, which makes it the perfect ML method, when you take into account the idea that ML is a way for machines to mimic human intelligence and reasoning.

The human brain contains a web of interconnected neurons. A new born baby's brain is like an empty canvas. As we grow up, we start learning to move, to speak and to reason. Scientists believe that the way this learning process happens, is that the brain programs it self to know these new things, by strengthening the connections between sets of neurons, and specifying a series of neural activations, that define which neurons are activated in what sequence, for what task. This is approximately the same way that NNs work.

The most basic part of a NN is the "Artificial Neuron", which could be defined by the same definition that we used for ML algorithms in general, where it's a mathematical formula that takes an input, does some calculations, and returns an output. A NN is a series of interconnected artificial neurons, grouped in a form of layers. The number of layers and their neurons and the way these neurons are connected, has the biggest affect on the performance of the NN at a specific task. This means, that there are several types of neural network architectures, some are more suitable for computer vision like Convolutional Neural Networks (CNNs), while other are better with text and audio like Recurrent Neural Networks (RNNs).

There are many types of NN layers. These types can be grouped into three main ones. The input layer is used to feed data into the network. The output layer on the other hand, is responsible for producing the NN's output. In between, we find hidden layers. These hidden

layers are what makes NNs so powerful.

As we have seen before, supervised ML algorithms will predict new information from already fitted mathematical formulas using older similar information. Hidden layers in NNs allow for the deduction of intermediate information, that allow the NN to learn more variations of data and still have a high accuracy. If we take the previously seen house pricing example, a regular supervised ML model may output a prediction from a set of given house features or qualities. But, a NN will actually try to deduce other qualities and features about the house in its hidden layers before predicting. This gives NNs a huge advantage over most ML algorithms.

As our solution is image dependant, we can't go over NNs without giving an example of an image centered problem. Figure 3.2 demonstrates how pixels of an image could be passed as features to a NN, in order to classify that image into a specific class.

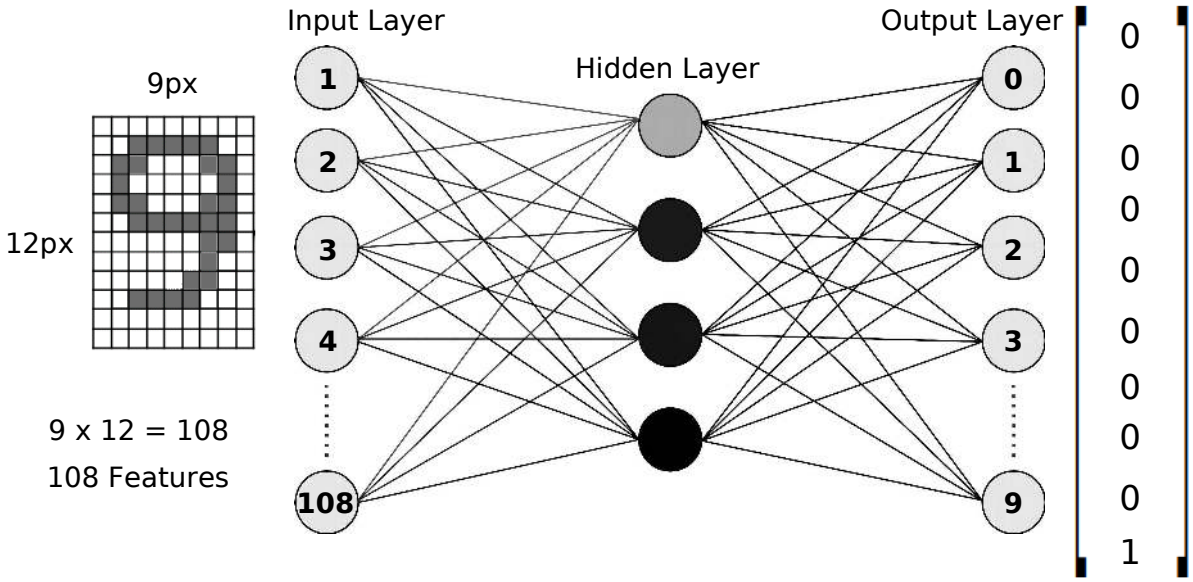


FIGURE 3.2 – Digit recognition (classification problem) using a fully connected Neural Network (NN)

3.4 Exposure Fusion (EF) Using Machine Learning (ML)

3.4.1 Introduction

We're trying to create a model that is able to replace traditional EF methods. To do so, We will train a Neural Network (NN) to create EFs from new never before seen exposures, using a dataset containing samples of scenes, captured at multiple Exposure Values (EVs), using

Exposure Bracketing (EB). Each sample of scene exposures is labeled with its EF, created using the Mertens EF algorithm.

We hope that by feeding the NN enough examples of captured exposures, with their fusions, it will learn how to successfully and efficiently do EF. We also hope for this new EF ML based solution to be lightweight and fast, in order to achieve both mobile compatibility and real time performance.

When we use an algorithmic approach to do EF, like the Mertens EF algorithm. The different exposures are given as inputs. They are analysed and used to create weight maps, using different quality mesures (Figure 3.3). The weight maps are later used to fuse the different exposures, into a more detailed representation of the scene. This fusion requires many calculations to be done, on each and every pixel of the exposures. The huge number of calculations and the numerous steps that need to be done to create the fusion take a lot of time. In addition, it may not always give the best quality output.

FIGURE 3.3 – EF using the Mertens algorithm

The method that we propose is much different than the algorithmic, statically typed approach. In order to understand it, we begin by taking a deep look at how the data (exposures, pixels, fusion . . . etc) is handled. Because, data is at the center of any NN ML based solution.

3.4.2 The Data Representation

We will be working in the RGB color space, where each image is represented as a $WIDTH HEIGHT NUMBER OF CHANNELS$ 3D Matrix. The last dimension is a vector containing the RGB values (3 values) for a pixel, at a specific position $[X, Y]$ (Figure 3.4).

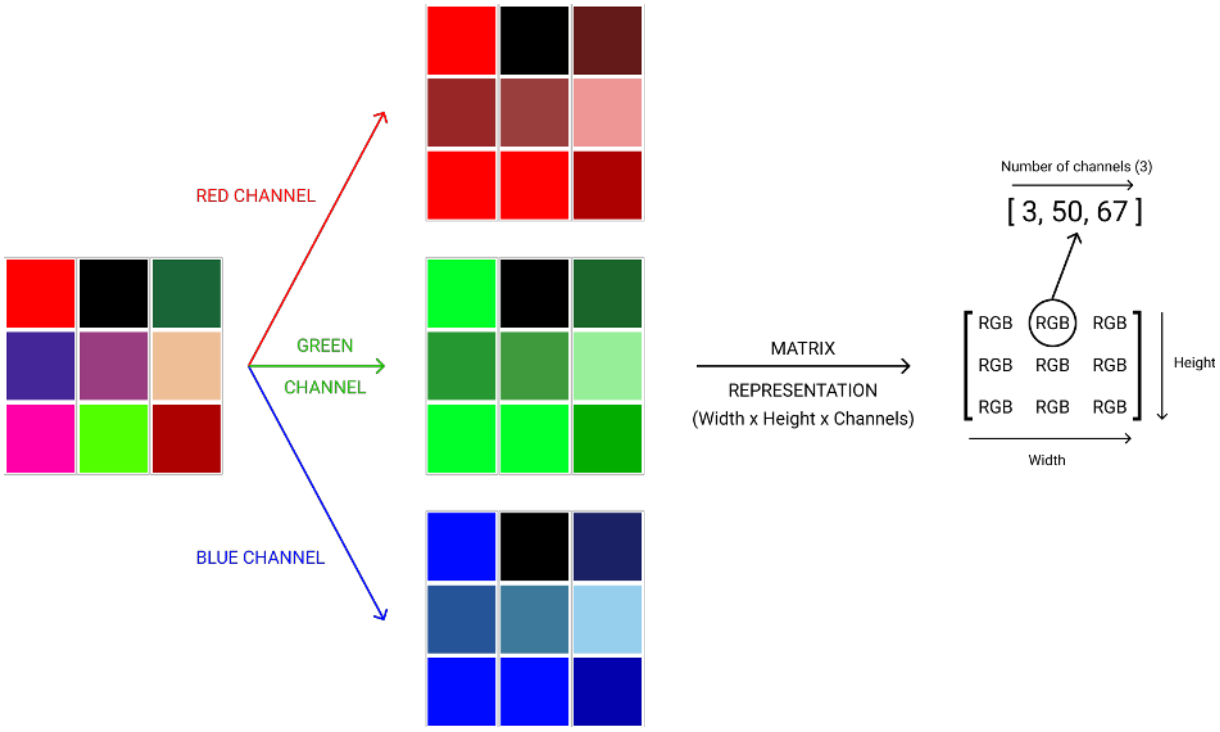


FIGURE 3.4 – Representation of an image using a matrix in the RGB color space

When using the Mertens EF algorithmic approach, we are able to input any number of different exposures. Unfortunately, the nature of NNs forces us to have a specific constant number of inputs. We chose to use three different exposures as inputs (Over-exposed, Normal exposure, Under-exposed).

Having three exposures means that we need to represent the data of three different images, instead of one. Consequently, We chose to represent each position (pixel) as a vector, just like before. But, instead of it containing three values, representing the Red, Green and Blue (RGB) of one pixel. The vector contains nine values, where each three are the RGB values of three different pixels at a specific position, across the three different exposures (Under, Normal, Over) (Figure 3.5).

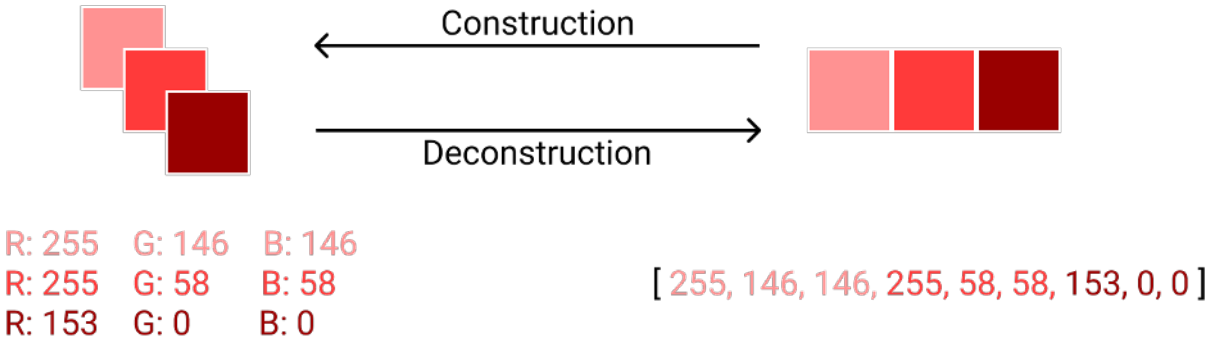


FIGURE 3.5 – Representation of a pixel position across multiple exposures

3.4.3 The Pipeline

Our method is as simple as calculating the RGB values for each pixel in the fusion, by using the RGB values of that pixel position across the different exposures. So, to get the RGB values for the pixel at position $[30, 64]$ in the fusion, we would only need the RGB values of the pixels at that same position, in the different exposures (Figure 3.6).

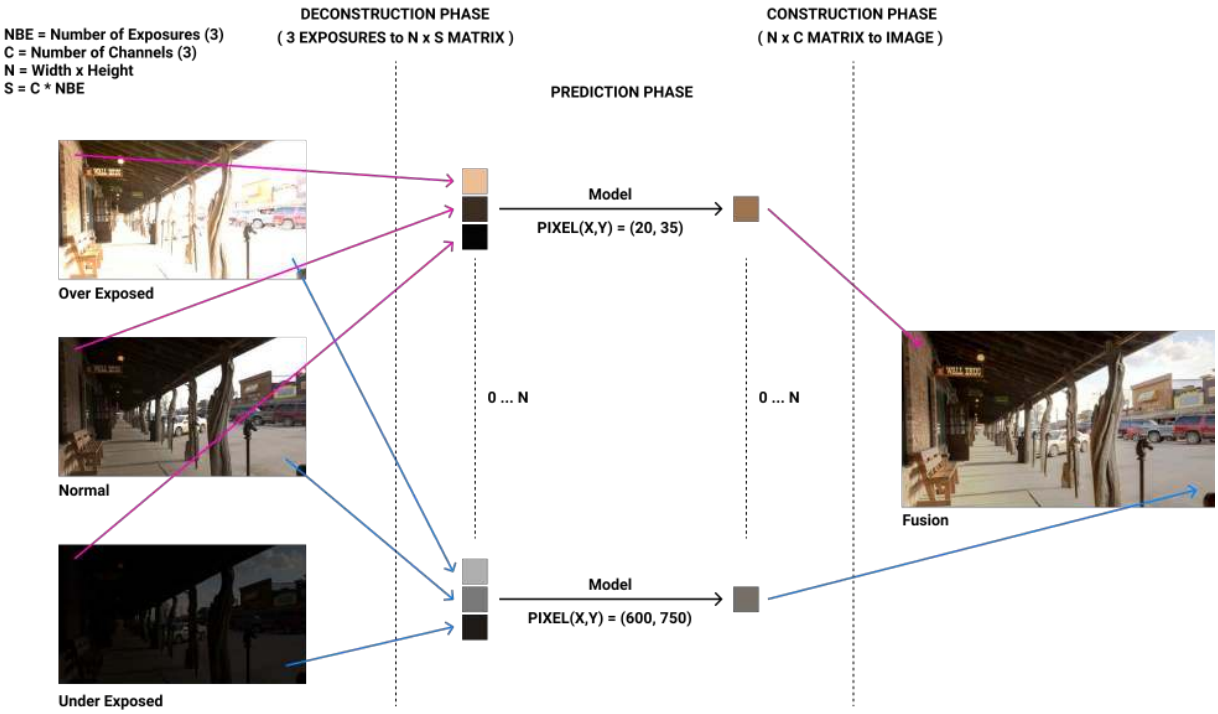


FIGURE 3.6 – EFNN solution pipeline

Specifically, we could give the model a 91 vector, representing the different RGB values, in the three different exposures, at a specific pixel position (ex: $[255, 255, 255, 255, 197, 63, 100, 16, 100]$). Consequently, It would return a 31 vector containing

the RGB values of the pixel at the same position in the fusion (ex: $[190, 100, 120]$). Therefore, we would need to run the model N number of times. Where N is the number of pixels in the fusion, or in a single exposure.

According to what was previously explained about the data and the pipeline, we begin by reading three different exposures of a scene. To simplify the explanation, let's say that we're working with 33 images. As a consequence, these exposures are represented as 333 matrices. Where the first dimension is the height of the image. The second is the width, and the third is the number of channels (3 for R, G and B).

The three 333 matrices are combined on the third dimension (the channels) into a 339 single matrix. Where the third dimension contains nine values as seen in Figure 3.5.

Since we are running the model for each pixel, we transform the 339 matrix into 99. Where the first dimension is a list of all pixel positions, and the second is the one that was previously the third dimension.

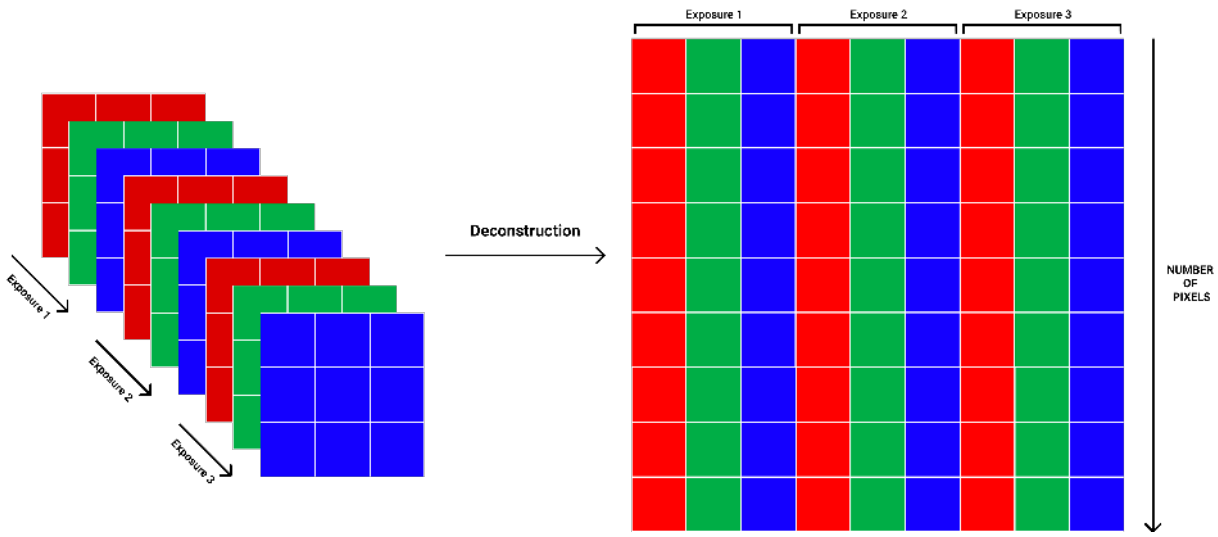


FIGURE 3.7 – Preparing the different exposures before feeding the data to the pre trained fusion model

This whole process is called "The Deconstruction Phase", where the three exposures are read and transformed into clean data that can be fed to the model. This phase is necessary in order to be able to easily use multiprocessing methods to speed up the prediction process. It should be noted that transforming the 339 matrix into a 99 one, is as simple as grabbing the pixels one by one from the first two dimensions, from the top left to the bottom right and putting them into a vector (single dimension) (Figure 3.8).

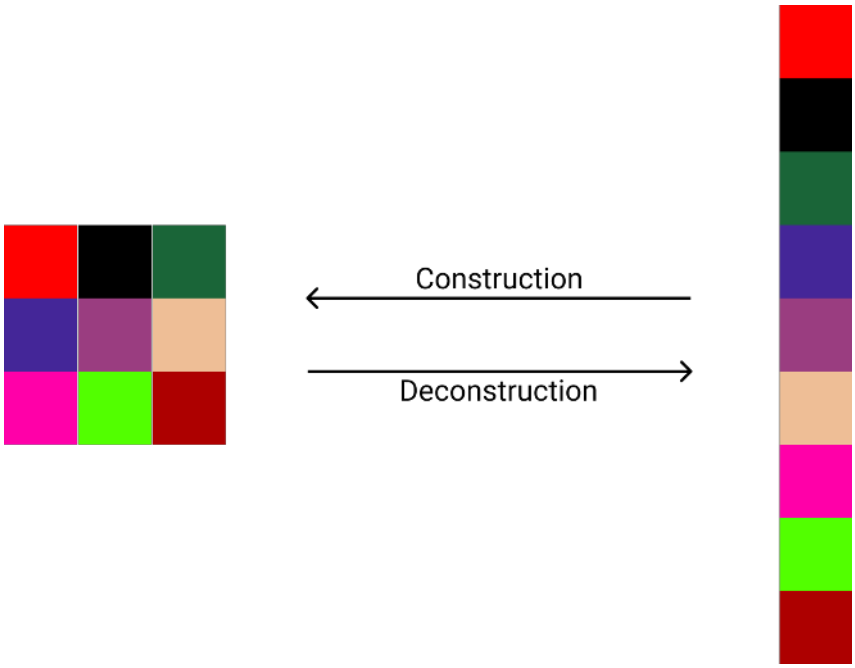


FIGURE 3.8 – How to flatten and de-flatten the data in the construction and the deconstruction phases

After running each row (each pixel position) in the 99 matrix through the model, we get a 93 matrix. Each row in this matrix, contains RGB values of a specific pixel in the fusion.

”The Construction Phase” is where this 93 result is transformed into an actual 333 fusion image (Figure 3.9).

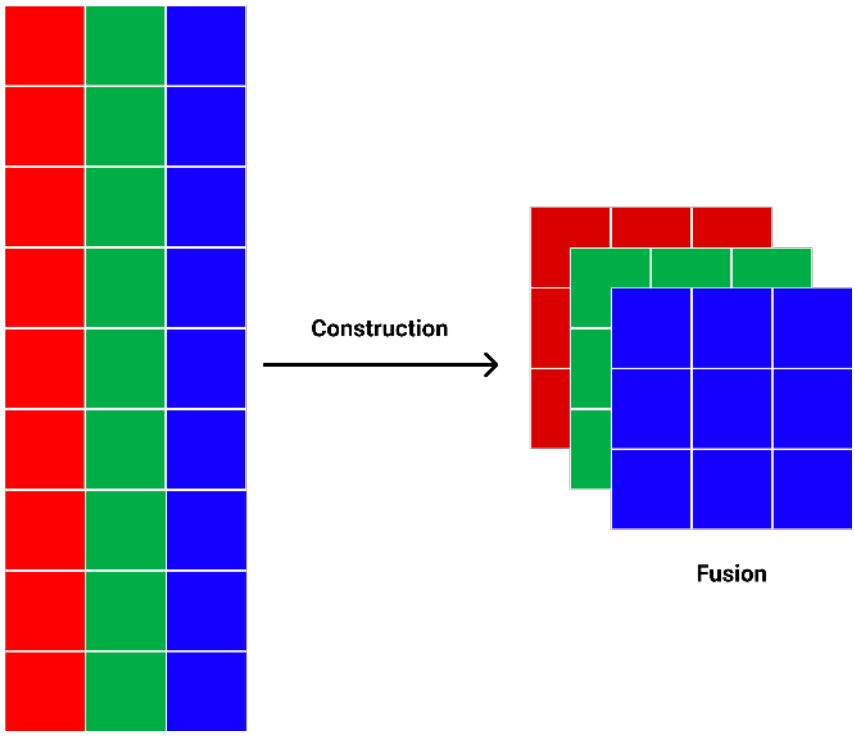


FIGURE 3.9 – Constructing the model’s result into an image

3.5 Exposure Fusion Neural Network (EFNN)

Now that we have a general understanding of how the data is read and processed in the EFNN pipeline from input to output, we will dive deeper into the actual model.

As we have previously mentioned, the model input is a 91 vector. The vector contains three different Red, Green and Blue values in three different exposures (Under, Normal, Over). The output is a 31 vector, containing three values for the Red, Green and Blue in the fusion.

All these values across the input and the output are related to a single pixel position, in the scene's representations (different exposures and fusion).

It should be noted that many Neural Network (NN) designs have been tested, with different layers and different numbers of nodes using different functions. Figure 3.10 represents the NN design that outputs the best result.

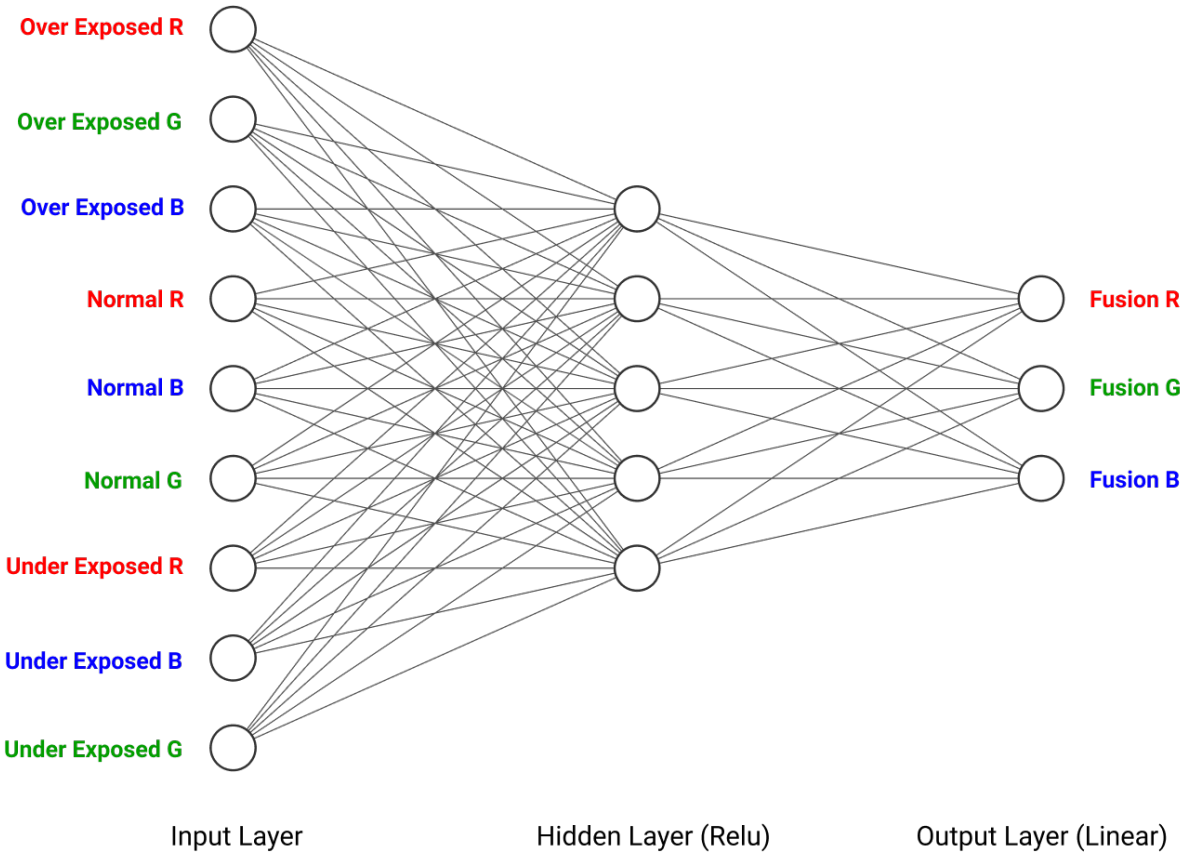


FIGURE 3.10 – Exposure Fusion Neural Network (EFNN)

3.5.1 Input Layer

The NN takes in nine values as an input. Since we are working in the RGB color space, the values are in the $[0-255]$ range. But, in order to have a normalised stable solution, we chose to normalise the values by using MINAMX Normalisation. This transforms all values to the $[0-1]$ range.

MINMAX Normalisation

$$Xi = \frac{Xi - Min}{Max - Min} = \frac{Xi - 0}{255 - 0} = \frac{Xi}{255}$$

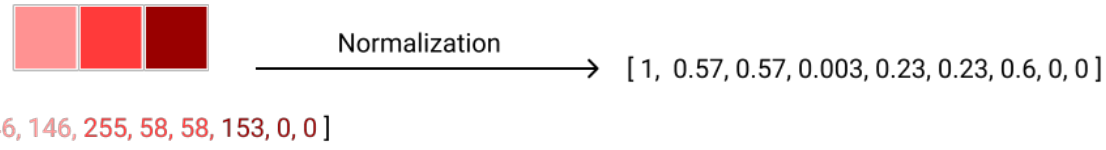


FIGURE 3.11 – Normalization Example

3.5.2 Hidden Layers

With regular mathematical solutions, we would have simple mathematical formulas. Where all the different aspects of the calculations (inputs, outputs, formulas, factors . . . etc) are clear.

Hidden layers in NNs add a level of complexity and details, that allow for extracting and transforming the input information, into data that increases the probability of the production of a much better quality output. This extraction process is often referred to as Feature Extraction. Therefore, these layers allow NNs to solve more complexe problems, with special edge cases.

We chose very simple functions as activations for the hidden layers' neurons. One function being a normal linear activation function. Another one being the popular ReLU (rectified linear unit) activation function [23]. A function that outputs only positive values, which is quite useful. As RGB values are always positive.

We have also tried sigmoid(1) activation, since we are working in the $[0-1]$ interval but it didn't return acceptable results.

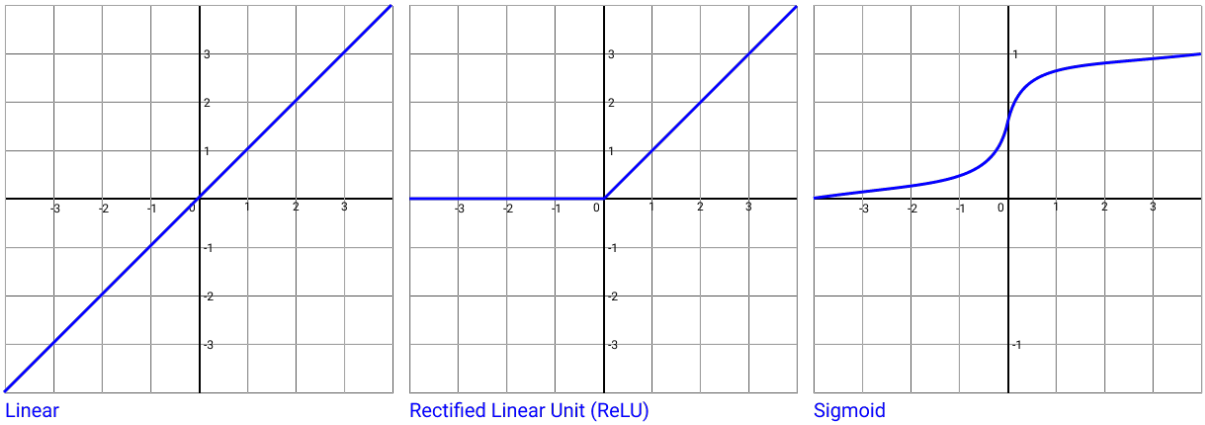


FIGURE 3.12 – Hidden layer Activation Functions

3.5.3 Output Layer

Since we chose to normalise the input values to a $[0-1]$ interval, we went with sigmoid(1) activation for the output nodes, in order to have an output that is also in the $[0-1]$ interval. This would allow us to simply multiply the result by 255 in order to get the real RGB values.

Later, it turned out that using sigmoid wasn't the best idea, as the $[0-1]$ limitation produced unacceptable result in certain extreme lighting conditions. Therefore, we went with a normal linear function. We still multiplied the result by 255 and limited it after multiplication to the $[0-255]$ interval.

Doing the limitation on integer values after the multiplication instead of doing it using sigmoid in the output layer, turned out to be the better approach. Because of floating point inaccuracy.

3.6 Why Neural Networks (NNs) ?

Some may still argue that using an algorithmic approach is still the better solution. Others agree that going with an ML method is superior, but using NNs is too extreme and we are better off with simple regression, or a one layered perceptron.

Let's begin with the first layer of our proposed solution (EFNN), which is the usage of a Machine Learning (ML) method. Solving problems starts with identification. We look at what we would like to solve. We understand it in a general manner and identify its nature, its causes

and its effects. Then, we try to look at the details, the special edge cases, the sub problems. We try to have an extremely deep understanding of everything that is related to the problem. Because, understanding the problem as an entity, isn't enough, since external actors and effects may change the dynamics of it. Therefore, using an algorthmic approach for modeling real world problems, is not an acceptable production ready solution anymore.

The inconsistency of nature and the world, often causes a problem to transform from a case to another. As a consequence, traditional algorithmic solutions can't be performante enough to get a hold of all the possible scenarios, classify them by priority and study them as a part of the problem itself. This shows that using an example based approach like ML methods, allows for a better grip over the problem, its nature and its edge cases.

Now, that we have given our reasoning for why an ML approach is much superior in data driven problems, let's take a look at why we chose NNs instead of simpler ML method.

The choice to use NNs is driven by one simple thing, which is the complexity of the problem. Regression solutions are like a water dam, where the amount of water to be released is controlled by a number of keys. Saying that we would like to predict the weather of a certain location based on sun intensity, entails the water flow is controlled by a single key, which is sun intensity. Saying that we would like to predict house prices using the age of the house and the number of rooms, implies that the water flow in controlled by a combination of 2 different keys.

So, when we look at a problem that involves image processing, and especially one that involves multiple images, with many different attributes and characteristics, we clearly see that the problem is affected by so many factors, that it is impossible to try and cover it as a whole with a simple one layered solution, like normal regression. Going back to the water dam example, when a problem is as complicated and as varied, as the one we're intending to solve. The number of keys at play and the number combinations to take intro consideration is so big, that it is impossible to have a general performante solution, without something that introduces a layer of complexity and care for details like NNs.

The depth of NNs is not the same as increasing the number of nodes in a single layered simple ML solution, where the more nodes means the more the possibilities and cases covered. The depth aspect of NNs allows for an exponential increase in the solution's reach, over the different combinations and possibilities of the problem. Meaning, we can create more performante models that solve extremely complexe problems, with far less calculations.

3.7 Conclusion

Now that we introduced our new proposed EFNN pipeline for HDRI. We will go over how this pipeline was created, take a look at the implementation steps and see the produced results.

It should be taken into account that this project requires a lot of processing power for training a huge number of models and handling large amount of data. It also requires a lot of experience with image processing, ML and algebra. Taking into account our limited time and experience, we believe that with more time and more research we could design and implement a better model. Nonetheless, that doesn't mean that we didn't get a good result. In fact, we're quite happy with the results of the model which we will be seeing.

Chapter 4

Implementation I : The Dataset

4.1 Introduction

The data acquisition and cleaning and the dataset creation, represent half of the overall work that needs to be done. As Machine Learning (ML) methods are data centered, the quality of the training data controls or has major effect on the quality of the trained model. Therefore, we have spent a lot of time acquiring and creating a good, diverse, quality driven dataset.

As previously mentioned, the model requires a dataset where each sample is a set of two vectors. The first being an input example represented as a 91 vector containing three RGB values of the same pixel position across three different exposures. The second one being the correct output for that input example, represented as a 31 vector containing the RGB value for that pixel position, in the fusion (Figure 4.1).

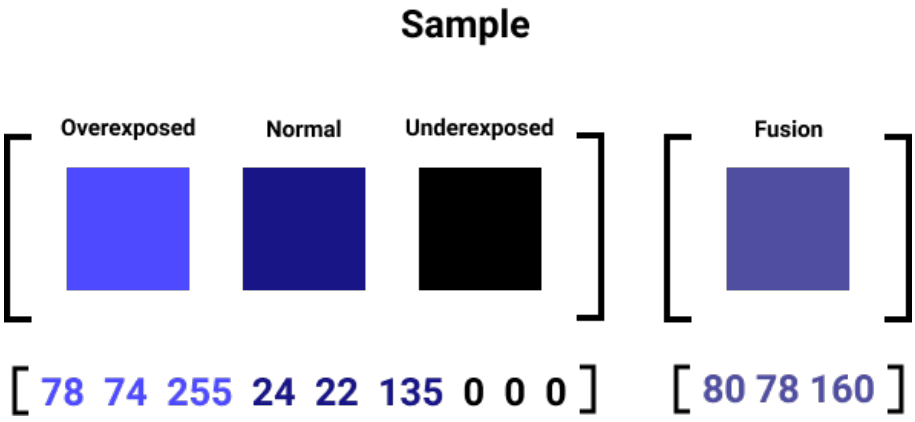


FIGURE 4.1 – Example of a data sample

The dataset (Figure 4.2) would hopefully contain hundreds of millions of samples, with pixels from different types of scenes, with different dynamic ranges. From indoor to landscape and from day to night. This extreme data variation that we wish to achieve, would assure that

the model wouldn't be biased towards specific kinds of scenes, or specific sub dynamic ranges. Instead, it would hopefully be able to produce beautiful Exposure Fusions (EFs) for any kind of scene.

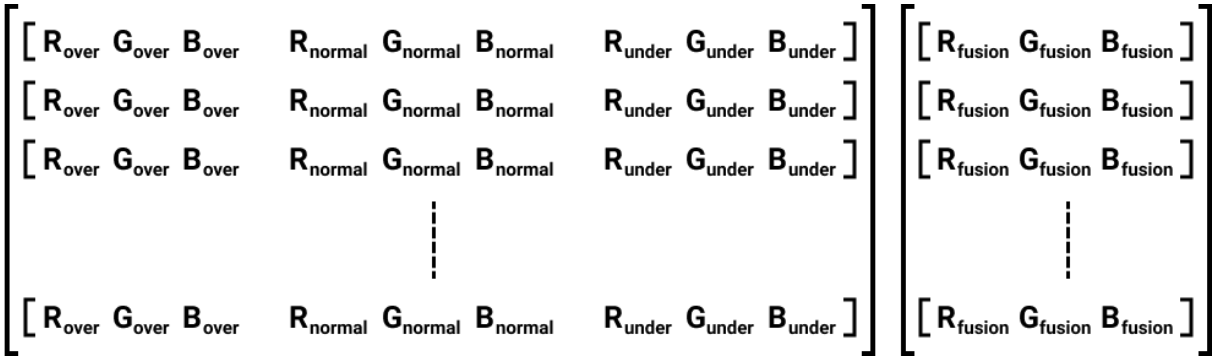


FIGURE 4.2 – Required dataset to train the Exposure Fusion Neural Network (EFNN)

The problem is that this dataset doesn't exist. Creating it from scratch isn't something that we can easily do either. It would require many weeks if not many months of work, taking hundreds if not thousands of pictures of different types of scenes, at different EVs.

Acquiring the exposures is only the beginning. In order to increase the chance of the model gaining the ability to produce high quality fusions, we must use the best possible fusions for each scene, in the training samples. Hence, We would need to create different fusions, using different combinations of EVs, in order to find the best possible one for each captured scene. Only then can we use those captured images of those hundreds of scenes, along with the chosen created fusions, to create the actual RGB dataset that is needed for training the model.

Another approach would be to find already captured scenes at different EVs, to eliminate the process of capturing them ourselves. Unfortunately, even though there are samples on the web of scenes captured using Exposure Bracketing (EB), these samples exist in order to explain what EF is, or to give people a way to test it, which makes them very poor quality and very few in quantity. In fact, we may be able to collect only up to 10 or 20 acceptable samples in total. Of course, these samples are very different in quality and resolution. So, it's impossible to use them for learning.

4.2 HDR Photographic Survey Dataset

Our research on dynamic range increase had lead us to read many papers about High Dynamic Range (HDR) and Exposure Fusion (EF). While doing so, we fell upon "The HDR

Photographic Survey" research project and paper, which aims to provide high quality HDR images for researchers. This dataset is provided by Prof. Mark D. Fairchild for the purpose of enriching future research on color grading, dynamic range increase, and image enhancement.

Even though this dataset isn't something that is usable in our ML project, we have found that it contains HDR images in the OpenEXR HDR file format. We know that HDR files are created by using HDR algorithms, that fuse different scene representations (exposures), at different EVs, into a more detailed scene representation.

This knowledge about HDR file formats had pushed us to think about reversing the HDR process, in order to extract the different exposures that we need from the OpenEXR HDR files. By doing so, we are able to acquire different exposures of different scenes, without having to do any EB and real life scene capturing. As a consequence we move one step closer to creating the required RGB dataset needed for training the model.

4.3 Acquisition

Upon contacting Prof. Fairchild, we acquired access to the dataset which is hosted on his personal website [24] (Figure 4.3). We found that all samples were listed in a single page. Each sample had a separate page of its own, where the link to download it as an OpenEXR file was available.

FIGURE 4.3 – HDRPS dataset website

In order not to go over the samples' pages and download them one at a time, we created a very simple crawler in python, using the `Requests` module and `BeautifulSoup4`. The crawler went over the pages and extracted all the OpenEXR download links, for all the samples in the dataset. Then, we used wget in order to download them (Figure 4.4).

```
http://rit-mcsl.org/fairchild/HDRPS/EXRs/507.exr
http://rit-mcsl.org/fairchild/HDRPS/EXRs/AhwahneeGreatLounge.exr
http://rit-mcsl.org/fairchild/HDRPS/EXRs/AirBellowsGap.exr
http://rit-mcsl.org/fairchild/HDRPS/EXRs/AmikeusBeaverDamPM1.exr
...etc
```

(A) Text file (crawler output) containing OpenEXR HDRPS dataset samples download links

```
wget --show-progress --timeout 10 --tries 0 --continue -i links.txt
```

(B) Downloading the dataset using wget and the extracted OpenEXR links

FIGURE 4.4 – Acquisition of HDRPS dataset samples

4.4 Data Inspection and exposures extraction

At this point, we had acquired all the samples of the HDRPS dataset as OpenEXR HDR files. Now, we had to open these files to inspect them and find a way to reverse the HDR fusion process. In order for us to acquire multiple exposures from each HDR file. For this, we used Darktable [25]. A software used mostly by photographers to view and manipulate RAW images, which also seemed to be quite useful and capable of creating, reading and editing HDR files.

We have found that Darktable has a very useful `exposure` setting (Figure 4.5), which allows the user to change the level of brightness of the scene.

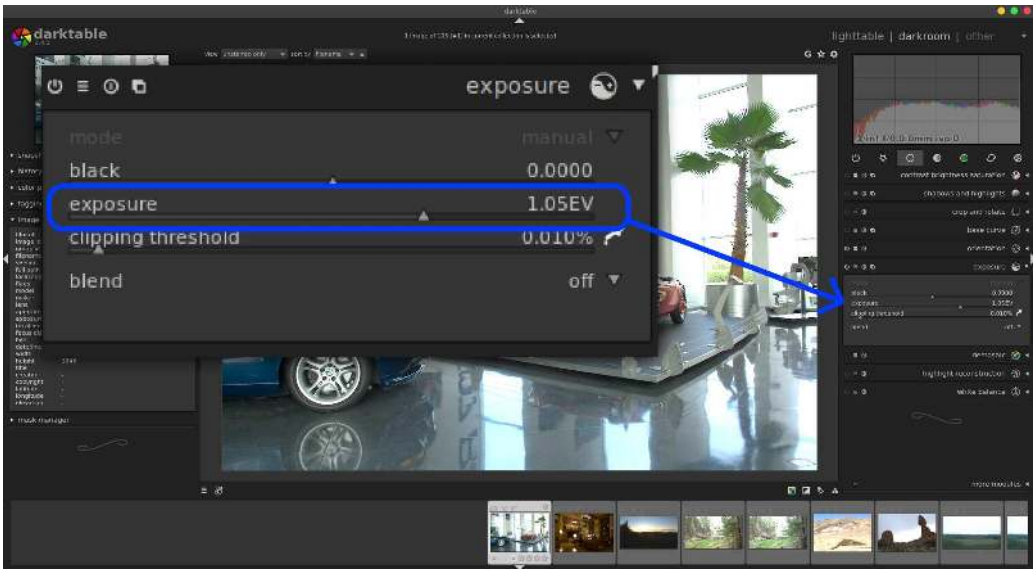


FIGURE 4.5 – Exposure setting in darktable

By changing the exposure of the HDR image, we are simulating the same process of EB scene capturing. Therefore, we have successfully found a way to extract different LDR exposures, at different EVs, from the OpenEXR HDR files. By simply changing the `exposure` setting (Figure 4.6) and exporting into a LDR file format.

(A) result of lowering exposure

(B) result of increasing the exposure

FIGURE 4.6 – Changing EVs of an OpenEXR image in Darktable to see different details

This process is very tiring and tedious. Fortunately, Darktable has a CLI tool that allows for automating tasks and applying effects, on multiple images. This is done using XMP files that contain ready to use configurations. So, we only had to create an XMP file for each EV that we would like to extract. Then, create a BASH script that would apply those XMP files using Darktable CLI tool, to each one of the OpenEXR files (Figure 4.7). In order to extract different exposures from each sample.

```
1    darktable-cli file.exr ev-1.xmp file-ev-1.png
2    darktable-cli file.exr ev+0.xmp file-ev+0.png
3    darktable-cli file.exr ev+1.xmp file-ev+1.png
4    ...etc
```

FIGURE 4.7 – Extracting exposures from an HDR image using Darktable

Even though Darktable is a great tool that allowed us to be familiar with HDR file formats. The need for creating an XMP config file for each EV had pushed us away from using it. Because, we had found another tool that allows for the extraction of the exposures, with a single command. That tool being an image processing CLI tool called ImageMagick 7 [26]. One of the operations that it provides is the `evaluate` command, which allows its user to preform arithmetic, relational, or logical operations on an image.

HDR file formats allow for the storage of more details per scene, by using more bits per pixel. This allows for the storage of more levels of brightness per pixel. Hence, more details

per pixel. Since LDR file formats don't provide the same space per pixel, when a HDR file is exported to a LDR format, without any tone mapping, only a sub-range of the data in each pixel is exported. Consequently, we are exporting the scene's representation at a certain Exposure Value (EV), which is exactly what we need. Except that we need to export multiple exposures at multiple EVs, instead of a single one.

After close inspection, we found that by multiplying the HDR image while exporting, by 2^n using the `evaluate` command, we are able to export different exposures at different EVs, where n is the Exposure Value (EV) for the needed exposure (Figure 4.8). For example, if we needed the -2 EV exposure, we would multiply by $2^{-2} = 0.25$.

```
1    sample.exr -evaluate multiply 0.25 sample-ev-2.jpg
2    sample.exr -evaluate multiply 0.5 sample-ev-1.jpg
3    sample.exr -evaluate multiply 1 sample-ev+0.jpg
4    sample.exr -evaluate multiply 2 sample-ev+1.jpg
5    sample.exr -evaluate multiply 4 sample-ev+2.jpg
6    ...etc
```

FIGURE 4.8 – Extracting exposures from an HDR image using ImageMagick 7

We created a BASH script to export 12 different exposures from each OpenEXR file (Figure 4.9), from -8 to +3 at 1 stop each (1 EV difference between each of the exposures).

FIGURE 4.9 – Extracting 12 exposures [-8 to +3] from an OpenEXR sample.

The different extracted exposures for each sample were saved into a single folder. Therefore, we would have a folder for each OpenEXR sample (Figure 4.10).

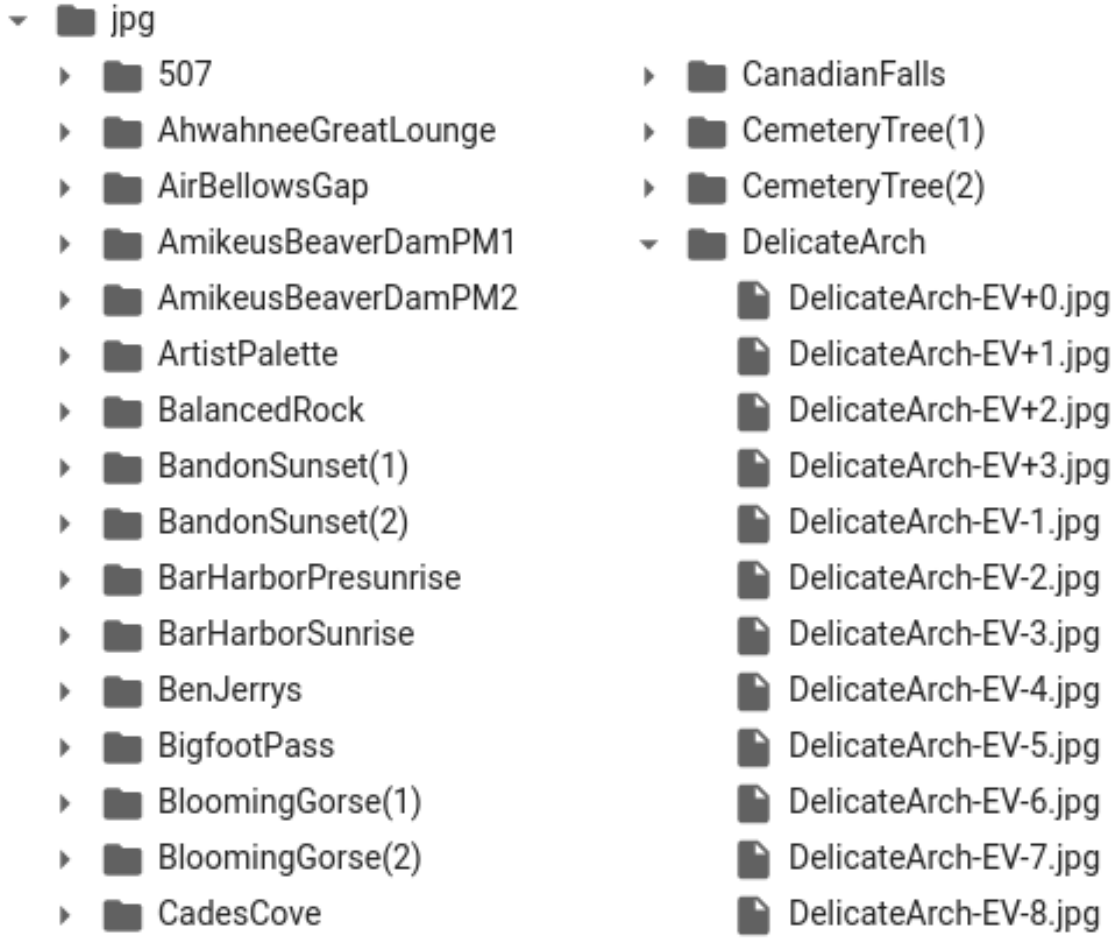


FIGURE 4.10 – Dataset directory structure

4.5 EF generation

Now that we acquired different high quality exposures of different scenes. We need to create the best possible fusions from those exposures.

The difference between the types and the dynamic ranges of the scenes, creates an issue. As using the same combination of three EVs, wouldn't create the best possible fusion for all the scenes. In fact, using some exposures in certain types of scenes, may create a fusion less detailed than a normal LDR scene representation, captured with a LDR camera (Figure 4.11). This is what drove us to extract 12 different exposures, instead of only three. Which is what is necessary for our model.

(A) Higher exposures are disregarded in intense lighting scenes

(B) More exposures are required for extremly high dynamic range scenes

FIGURE 4.11 – Difference in exposure variance requirement from scene type to another

The wide range of scene types and the number of available exposures, pushed us to create different fusions for each scene, using different combinations from the 12 different extracted exposures. Then, we chose the best fusion for each scene and 3 exposures, from the combination that was used to create such fusion.

Since we are constrained by both time and ressources, We chose six different combinations, that we think can largely englobe the different types of scenes in the HDRPS dataset and the real world. As seen in the third column of Tableau 4.1, the fact that we have to work with three exposures in the model's input, forces us to not be able to use all the combinations, that were used to create certain fusions. Therefore, we had to chose a subset of those combinations instead.

Fusion	Creation Combination	Chosen Sub Combination
0	-8, -4, +1	-8, -4, +1
1	-8, -7, -6, -5, -4, -3, -2, -1, +0	-8,-3,+0
2	-6, -2, +1	-6, -2, +1
3	-7, -3, +1	-7, -3, +1
4	-8, -7, -6, -5, -4, -3, -2, -1, +0, +1, +2, +3	-8,-2,+3
5	-8,-5,-2	-8,-5,-2

TABLE 4.1 – The different exposures' combinations used to create the different fusions

This inconsistency between the combination used to create the fusion and the combination used in the actual dataset, is caused by the fact that a good fusion for a certain sample may not be able to be created, using the same number of exposures as the number of

inputs that the model receives (three). This is a grey area that makes our work even harder.

From this point on, we had to find a more readable and more organized solution, to implement the previously mentioned steps. Consequently, we wrote some object oriented code (Figure 4.12) that would hopefully make this entire process more easy to handle and follow.

We started by choosing OpenCV as the main technology for the image processing that would need to be done. Like the reading and viewing of the images or the creation of the fusions using the Mertens EF algorithm. The Numpy library was used for matrix/vector operations to create and manipulate the base and final datasets.

We wrote the code that we needed on top of these technologies, in Python. Hence, the `Exposures` class.

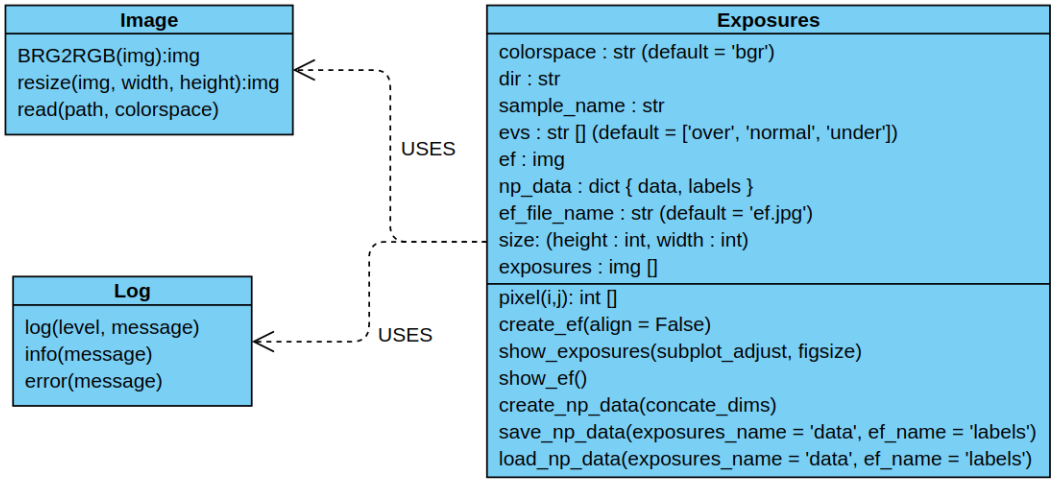


FIGURE 4.12 – Classes created to handle the dataset

After executing the previously mentioned extraction steps using ImageMagick 7 and acquiring the different exposures, a new object of the `Exposures` class is created for each sample folder (Figure 4.10). The class constructor is provided with a sample's folder path through the `dir` attribute. As well as the exposures combination for creating the fusion, through the `evs` attribute. In addition, the fusion image file name is required and passed to the `ef_file_name` attribute.

On object creation, the different exposure files are read, stored and can be accessed through the `exposures` attribute. If the EF file exists in the sample's directory, it is read and can be accessed through the `ef` attribute. Otherwise, the `create_ef` method can be used to have the file created and saved for future use.

The `Exposures` class was written to facilitate the creation of multiple fusions for each sample. In order to be able to compare those fusions and choose the best possible one.

For each sample, six different `Exposures` objects are created. Each one has a different `ef_file_name` value. As well as a different combination of exposures in `evs` attribute, according to Tableau 4.1. Then, the `create_ef` method is called to create different fusions and save them under different names to the sample's folder (Figure 4.13).

```
for folder_path in samples_folders:
    # creating fusion ef0 for sample
    Exposures(
        dir = folder_path,
        ef_file_name = 'ef0.jpg'
        evs = ['SAMPLE-EV-8', 'SAMPLE-EV-4', 'SAMPLE-EV+1']
    ).create_ef(check_exist = True)

    # creating fusion ef1 for sample
    Exposures(
        dir = folder_path,
        ef_file_name = 'ef1.jpg'
        evs = [
            'SAMPLE-EV-8', 'SAMPLE-EV-7', 'SAMPLE-EV-6',
            'SAMPLE-EV-5', 'SAMPLE-EV-4', 'SAMPLE-EV-3',
            'SAMPLE-EV-2', 'SAMPLE-EV-1', 'SAMPLE-EV+0'
        ]
    ).create_ef(check_exist = True)
    ... etc
```

FIGURE 4.13 – Reading a sample and creating its fusions using the Exposures class

The six different created fusions are compared (Figure 4.14). This comparison is supposed to be done using various measurement algorithms, that measure color, saturation, overall quality ...etc. We have chosen to do this comparison by eye, because the fusions were quite easy to compare as some of them were extremely unrealistic, too dark, too light, or just generally of bad quality.

FIGURE 4.14 – Different combinations create different fusions (ef0 to ef5)

The best fusion is chosen for each sample, along with the three exposures sub combination for that chosen fusion. This metadata is saved into a CSV file (Tableau 4.2) and is later used to extract the RGB/pixels dataset, which is needed to train the model.

Sample	Under Exposure	Normal Exposure	Over Exposure	Fusion
AhwahneeGreatLounge	-7	-3	+1	ef3
AirBellowsGap	-8	-2	+3	ef4
AmikeusBeaverDamPM1	-8	-5	-2	ef5
AmikeusBeaverDamPM2	-8	-5	-2	ef5
ArtistPalette	-8	-3	+0	ef1
... etc				

TABLE 4.2 – Base dataset metadata

Now, We have successfully created a high quality dataset, containing different types of scenes with different dynamic ranges. Each scene, or each sample, is defined by three exposures (Over, Normal, Under) and labeled with a fusion. But, this is not the dataset that we need to train the model. this is the base dataset from which we extract the actual training dataset.

To extract the RGB dataset, the `Exposure` class is used along with the previously created metadata. An `Exposure` object is created for each row from the base dataset samples table. Then, the `create_np_data()` function is used to transform the read data into RGB samples. This function creates a two keys dictionary (data and labels),which can be accessed through the

`np_data` attribute.

`np_data['data']` is a matrix, where each row is a vector containing three RGB values (nine integer values), representing the RGB values of a pixel position across the three different exposures of the sample. `np_data['labels']` is also a matrix, where each row represents the RGB value (three integer values) of a pixel position in the sample's fusion.

The `create_np_data()` function is executed for each row. The resulted data matrices are merged into a single matrix (Figure 4.15), containing all the exposures' pixels for all the samples. The same is done to the labels matrices.

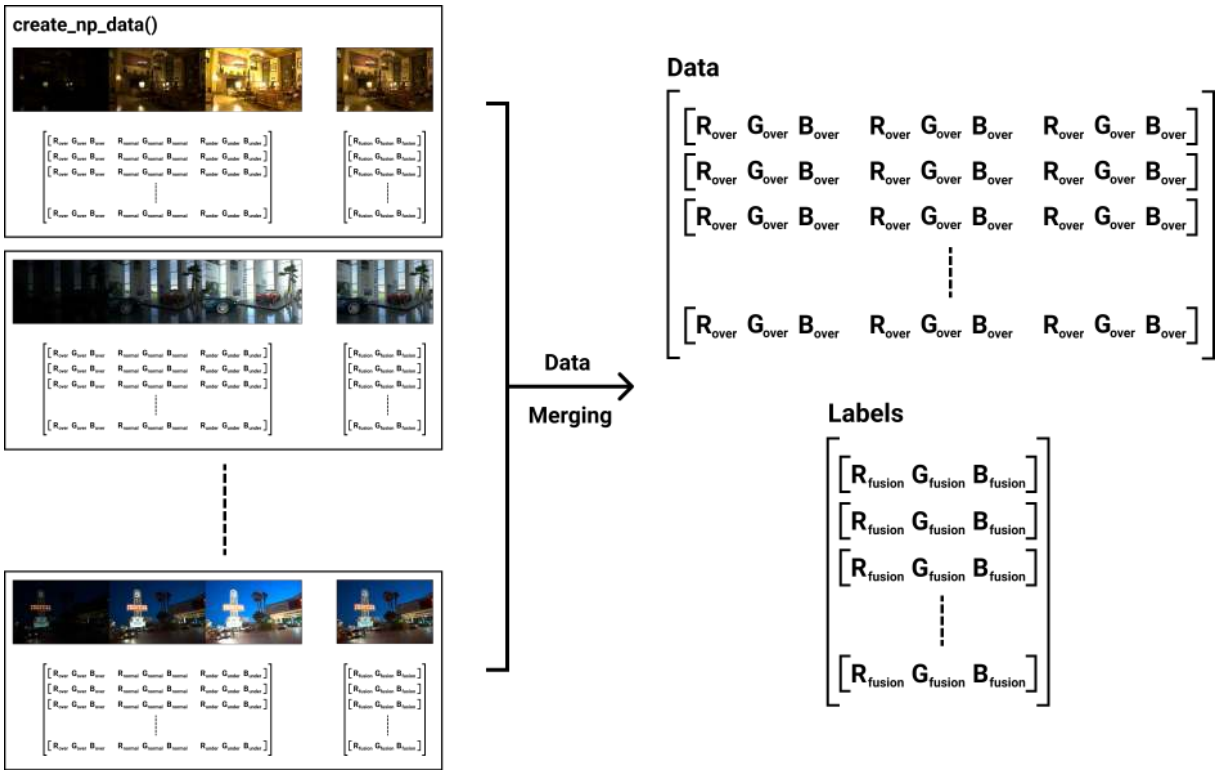


FIGURE 4.15 – RGB dataset creation from the base images dataset

The final data and labels matrices are shuffled, in order to have an unbiased variate dataset. The shuffle is done in harmony between the two matrices, since the row positions are crucial for the validity of the dataset. Where row *i* in the data matrix is labeled by row *i* in the labels matrix. This dataset of over 200 million samples is saved into npy files, so that it can be later used for training the model.

4.6 Conclusion

In this chapter we learned about the required dataset for training the EFNN. By manually creating such dataset we learned a lot about data and image processing tools. We also got to play with and reverse engineer HDR formats.

In the next chapter, we will see how we can manipulate and use this dataset in order to train the Exposure Fusion Neural Network (EFNN).

Chapter 5
Implementation II : Training & Result

5.1 Introduction

Creating a ML model is a tedious process. Because it requires training many different models with different parameters, in order to hopefully find the best possible one.

In this chapter we go over the technologies used in the learning or training process. We will take a look at the different parameters that control the quality of the resulted model. By comparing different models and their results, We will gain some insight on what affects the learning process and how it can be optimized.

5.2 Model implementation

For this, we went with Tensorflow 2.0 [27], a very popular library used by ML engineers. Using Keras, a high level API for the Tensorflow library, We were able to implement, test, and modify different models very quickly.

We began by reading the RGB dataset that was created in the previous chapter (Figure 5.1). The dataset contains over 200 million samples, with each sample containing 9 + 3 = 12 RGB values. In Python, each RGB value [0-255] takes at most 2 bytes of space [28]. Therefore, We would need at least 2.4 GB of RAM in order to load the dataset. Because :

$$200\ million\ samples 12\ RGB\ values = 2.4\ billion\ RGB\ values$$

$$2.4\ billion\ RGB\ values 2\ bytes = 2.4\ GB$$

This would be fine if we were using Integer values for our model. Unfortunately, in order the have a better learning experience it is better to normalize values to the [0 - 1] range. Doing this normalization requires us to use float point values, which take a lot of space as Python uses

8 bytes for float number. Therefore :

$$200\ million\ samples 12\ RGB\ values = 2.4\ billion\ RGB\ values$$

$$2.4\ billion\ RGB\ values 8\ bytes = 19.2\ GB$$

19.2 GB can actually be loaded into memory because the machine that we used in the learning process has 25 GB of RAM. But, we wouldn't be able to do much else, As the learning process requires a lot of ram when calculating the predictions and fitting the weights.

In order to solve this issue we took a subset of the dataset. Since the images used to create the RGB dataset are very high in resolution, taking one pixel in every X pixels wouldn't hurt. Because adjacent pixels have a very high probability of having the same or very close values.

```
data = np.load("data.npy")
labels = np.load("labels.npy")
```

FIGURE 5.1 – Reading the data and labels as npy files

We first divided the dataset into 2 parts. 80% of samples are used in the training phase, while 20% are later used to test the model. This approach is the standard for training models. But, it was quite useless in our case. Because we are doing predictions on single pixels independently, measuring the quality of the produced images using regular measurements like accuracy or precision is impossible. Therefore, it is necessary for us to measure the quality of the model by testing it each and every time on real images and use the naked eye to assess if the learning is heading towards the right direction.

So, we chose to not give much importance to the automatic testing phase and do it manually with the naked eye instead. This allowed us to use the entire data for training. This data is normalized to [0-1] values (Figure 5.2).

```
data = data / 255
labels = labels / 255
```

FIGURE 5.2 – Dataset normalization

Using Keras, we are able to implement the previously seen Exposure Fusion Neural Network (EFNN) in just a few lines of code (Figure 5.3).

```
model = tf.keras.Sequential()

model.add(
  tf.keras.layers.Dense( 5, input_shape = (9,), activation="relu" )
)

model.add(
  tf.keras.layers.Dense( 3, activation = "linear" )
)
```

FIGURE 5.3 – Creating the EFNN in Keras

The `Sequential()` class is a model base class that we're using to declare an empty NN. Then, the layers are added using the `model.add()` method. The first layer is the hidden five nodes layer, which is taking a 91 vector as an input layer, and using ReLU as its activation. Connected to the hidden layer is the output's three nodes layer, which uses a linear activation.

We know from previous chapters that in the learning phase, the model makes a series of predictions, then calculates a loss or cost function in order to see how far those predictions are, from the desired ground truth. Then, it uses an optimizer to update the formulas used for the prediction, in a way where the loss or the error is reduced. This process is done for many iterations until the model is able to make good predictions. Otherwise, the different parameters that control the learning process are changed and it's restarted again. These parameters are called "Hyper Parameters" and changing one of them can make or break you model. Finding the right parameters is the challenge that we take on when training ML models.

We use the `model.compile()` method in order to specify some of the hyper parameters (optimizer, learning rate, loss ..etc) (Figure 5.4).

```
  model.compile(
      loss="mean_squared_error",
      optimizer= tf.keras.optimizers.Adam(lr=0.0001),
      metrics = ["accuracy"]
  )
```

FIGURE 5.4 – Specifying the model's optimization algorithm and loss function in Keras

The way the learning happens is that the entire dataset is passed forward (from input to output) through the model. Predictions are made using certain weights. Then, a backward pass is done from output to input where the weight are recalculated and updated. An “Epoch” is a term that denotes one iteration of this process.

During an epoch the data cannot be passed all at once. So, we divide the data into batches of samples. During a single epoch, these batches are passed one by one through the model, where they do one forward and backward pass each.

Using keras we setup a checkpoint to save the model on each epoch. This way, we can see and test how the model changes as it learns during multiples epochs. Each saved model is identified by its epoch number, its batch number, and its loss and accuracy. Identifying the model by these parameters allowed us to train hundreds and hundreds of models, while being able to easily compare them to find the best possible one. In addition, training so many models using so many different parameters and comparing them, allowed us to have a deeper understanding over the learning process and see how different parameters affect it.

```
checkpoint = tf.keras.callbacks.ModelCheckpoint(
  filepath = "model-{epoch:02d}-{accuracy:.2f}-{loss:.5f}.hdf5",
  save_freq = 'epoch'
  monitor = 'loss',
  verbose = 1, save_best_only=False, mode='auto',
)
```

FIGURE 5.5 – Setting up a model checkpoint in Keras

To start the training, we pass the normalized data, the size of the batches and the epoch number that we would like the learning to run for.

```
model.fit(
  train_data, train_labels,
  epochs = 10,
  batch_size = 256,
  callbacks=[checkpoint]
)
```

FIGURE 5.6 – Fitting the model

Since we trained many models, and sometimes many at once when the ressources were available, we would leave them training and find good feedback information when we get back, thanks to Keras and Tensorflow. So, we can debug in case there were any issues.

```
Epoch 1/20
20479/20502 [====>.] - ETA: 0s - loss: 0.0073 - accuracy: 0.8191
Epoch 00001: saving model to model-256-01-0.82-0.00734.hdf5
20502/20502 [======] - 35s 2ms/step - loss: 0.0073 - accuracy: 0.8192

Epoch 2/20
20497/20502 [====>.] - ETA: 0s - loss: 0.0027 - accuracy: 0.9125
Epoch 00002: saving model to model-256-02-0.91-0.00274.hdf5
20502/20502 [======] - 35s 2ms/step - loss: 0.0027 - accuracy: 0.9125

Epoch 3/20
20500/20502 [====>.] - ETA: 0s - loss: 0.0027 - accuracy: 0.9171
Epoch 00003: saving model to model-256-03-0.92-0.00268.hdf5
20502/20502 [======] - 35s 2ms/step - loss: 0.0027 - accuracy: 0.9171
...etc
```

FIGURE 5.7 – Fitting's output and feedback

5.3 Model Usage

To execute the previously seen EFNN pipeline using a trained model, we start by reading three exposures of a scene. Then, we run the previously explained deconstruction phase using the `create_np_data()` method.

We can use either `model.predict()` or `model.predict_on_batch()` in order to run the model. The `model.predict()` method splits the inputted data into batches. Therefore, it is better for when the inputted data is bigger than what the system's ressources can handle.

Unfortunately, the splitting and manipulating of the input data before the prediction slows down the process. As a consequence, using `model.predict_on_batch()` is the preferred solution as it pass the inputted data directly into the model, and uses multiprocessing if possible to speed up the prediction.

In the construction phase, we reshape the predictions vector into an image using the

dimensions of one of the exposures. Since the fusion has the same number of pixels as a single exposure, it has the same dimension as well.

```
# Reading input data
sample = Exposures(
    dir = folder_path,
    ef_file_name = 'ef.jpg'
    evs = ['SAMPLE-EV-8', 'SAMPLE-EV-4', 'SAMPLE-EV+1']
  )

# Deconstruction phase
sample.create_np_data()

# Prediction phase
normalized_data = sample.np_data['data'] / 255
prediction = model.predict_on_batch(normalized_data)
prediction = prediction * 255

# Construction phase
fusion_shape = sample.exposures[ev[0]].shape
efnn_fusion = prediction.reshape(fusion_shape)

# View fusion image using OpenCV
cv2.imshow(efnn_fusion)
```

FIGURE 5.8 – EFNN execution

5.4 EFNN vs Mertens Exposure Fusion (EF)

This entire project has began as a way to find a faster EF method. So, let's take a look at how our EFNN pipeline compares to the most used traditional EF method.

We are actually very happy with the results that we are getting using EFNN in all sorts of scenes and lighting conditions. The quality is quite good. There are certain differences in tone, color, saturation ... etc, between the two methods. But, different doesn't mean bad.

In certains scenes we notice that EFNN has a higher level of detail and more correct color representation than the traditional method.

EFNN

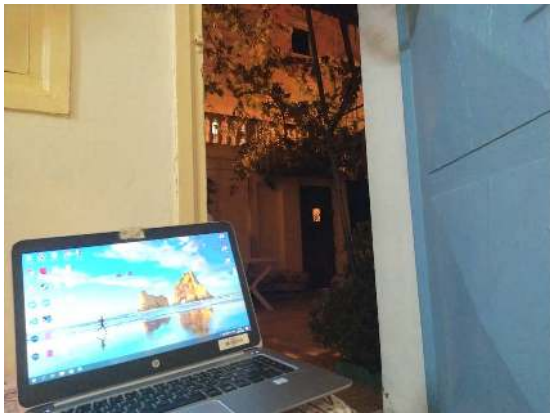
Mertens EF

(A) HALL OF FAME (4288x2848)

EFNN

Mertens EF

(B) MARK HDR (4286x2848)



EFNN

Mertens EF

(C) MOSTAGANEM (1000x750)

FIGURE 5.9 – EFNN vs EF quality

In other scenes, EFNN falls a bit short, but not to the point where the result isn't acceptable. Looking at the different samples, we can clearly see that we have definitely reached

an acceptable or even good quality.

But, quality wasn't our main goal. In fact, we have started this entire project in order to speed up traditional methods. Since we want to find a real time capable HDR method to solve problems caused by extreme lighting conditions, like the case study seen in chapter one, where the car's camera's inability to do computer vision in extreme lighting conditions, might lead to accidents and possible causalities.

Looking at EFNN from a speed point of view, we can certainly see how much faster it is than traditional methods. In fact, it takes half the time needed by the widely used Mertens EF algorithm. This improvement is even more impactful at high resolutions, since the needed time difference becomes more noticeable.

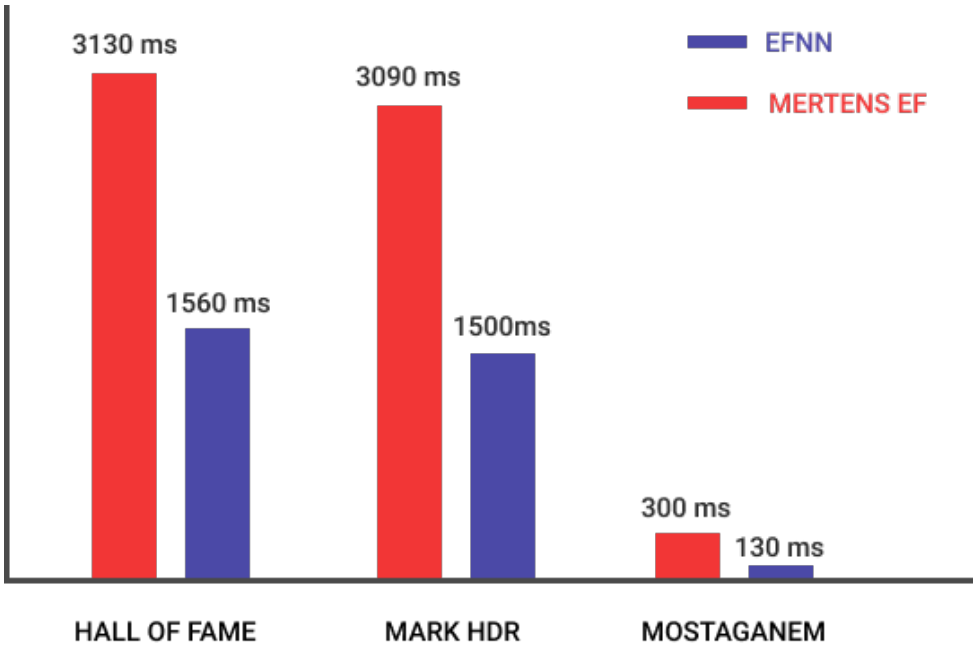


FIGURE 5.10 – EFNN vs EF speed

5.5 Hyperparameters & learning insights

Finally, we would like to share how we selected by trial and error the correct parameters in order to get this EFNN model that produces such great results.

5.5.1 Layers and nodes

While learning about Neural Networks (NNs) and ML, we learned about Autoencoders (AEs) [29, 30] which are a special kind of NNs used for encoding data, or transforming data

representations from a form to another. Autoencoders (AEs) contain two parts. The first part of the NN is the “Encoder” which encodes the input by doing a dimension reduction. The second part is the “Decoder”, which decodes the data into an output.

In a way, what EFNN does is encode a lot of data (multiple images) into less but more useful data (single fusion). Hence, the Encoder part of Autoencoders (AEs) inspired the shape and design of the EFNN.

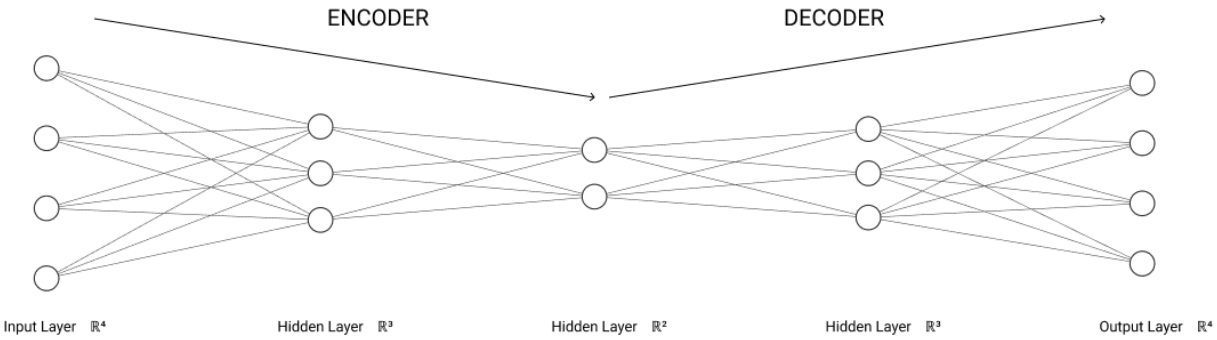
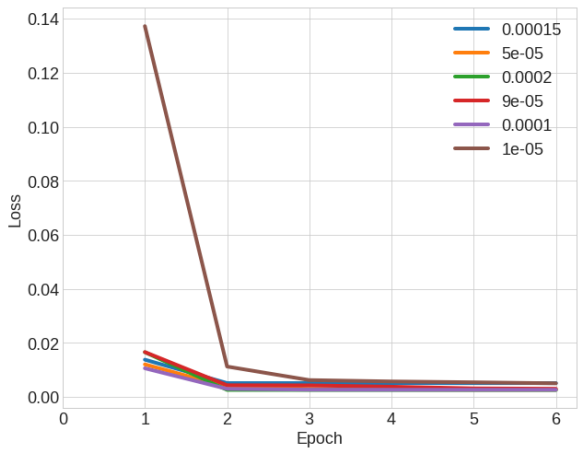


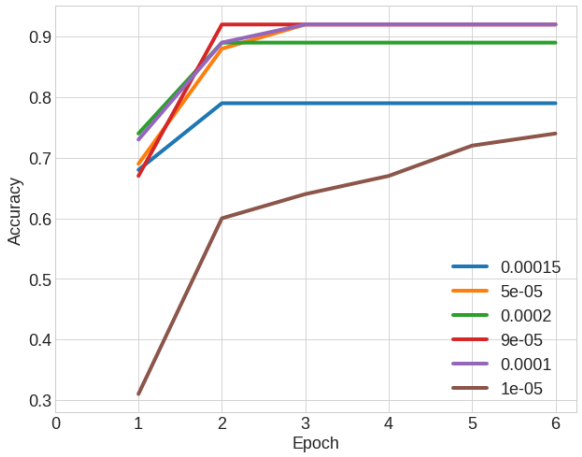
FIGURE 5.11 – Autoencoder (AE) Architecture

Stating from the input layer with nine nodes, each layer would have one less node than the layer before it. So we had seven layers starting from nine nodes (input layer) to three nodes (output layer). Then, we started removing each layers or two and testing until we started getting some improvement. By the end we settled on the previously seen (9-5-3) architecture.

5.5.2 Learning rate



(A) Loss



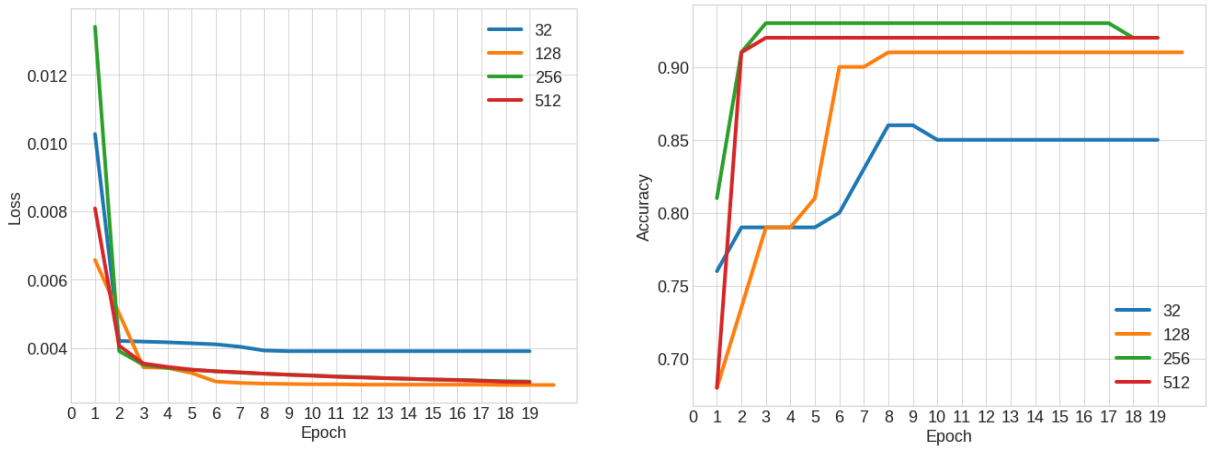
(B) Accuracy

FIGURE 5.12 – Effects of different learning rates on loss and accuracy

The learning rate controls how much the weights are changed on each update. After training many models with different rate, we settled with a learning rate between 0.00005 and 0.0001. As you can see in Figure 5.12 learning rates in that interval are much better from a metrics point of view.

5.5.3 Batch size

A general rule for choosing the batch number is going as low as you could think. Then, increasing the number as you go. Hence, we started with 32, then increased it to 128, 256 ...etc. We later settled for a batch number of 256 because it returns the best accuracy while decreasing the loss.



(A) Loss

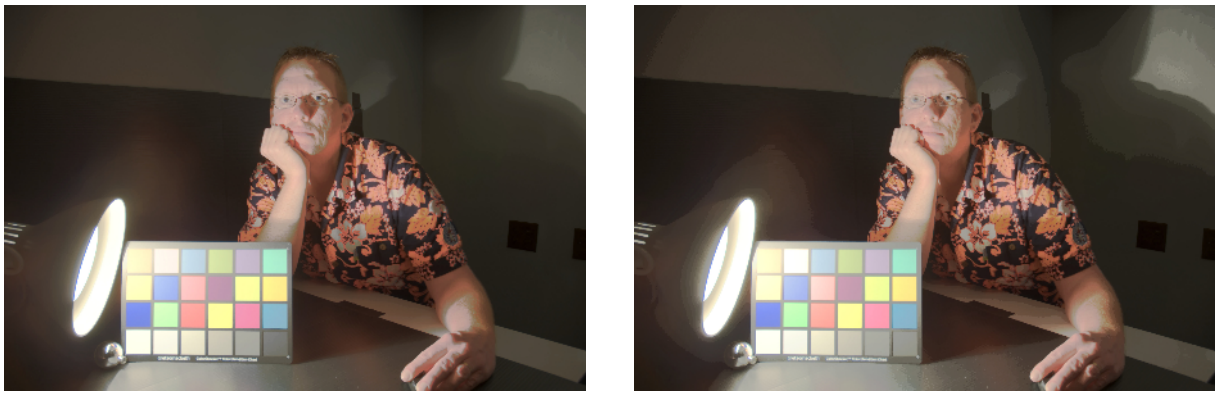
(B) Accuracy

FIGURE 5.13 – Effects of different batch sizes on loss and accuracy

5.5.4 Epochs

Many ignore the number of epochs when training and simply put a big number, in order to insure that the model would have a higher chance at converging, even if it took a longer time. This approach if very wrong. Training the model for longer times increases the chances of overfitting.

In fact you may create this great model that works perfectly on both the training and the test sets. But, it will fail the moment it's tested for generalization in real production environnements. Figure 5.14 demonstrated how training for longer times caused poor quality in lighting and shadows.



(A) 4th Epoch (Good)

(B) 20th Epoch (Overfitted)

FIGURE 5.14 – Overfitting caused by too many iterations over the dataset

Using the previously seen checkpoint callback, we got to save copies of the model at each epoch. Later, we would test each model and get the one with the best result, instead of testing the one saved after all the epochs are done, which may be prone to overfitting.

5.6 Conclusion

Training a ML neural network is like raising a baby without the fun parts. It's a tedious and tiring process that may take days, weeks, or in our case months. But its result is quite impressive as it provides a great balance between speed and quality. Thankfully, the availability of modern ML libraries has facilitated our entrance into the field of ML, and gave us a lot of power to test quickly and make more informed decisions about the model and the training.

Conclusion

We have went through many different topics during this research project. For a start, we took road lane lines detection as a case study, by explaining how the camera used to capture the road, can find it difficult, if not impossible to capture detailed images when faced with extreme lighting conditions, like being exposed to too much sunlight. This example has hopefully given us a solid understanding of dynamic range and the issues that might be caused because of its increase.

Our newly acquired knowledge about HDR problems, has led us into a journey of discovery. Where we learned about state of the art algorithms, that are both basis for current cutting edge scientific research and for currently used production ready solutions. Our understanding of these algorithms has inspired us to find better approaches, fuelled by our believe in the power of ML and NNs.

As a result, We proposed a novel solution where we got rid of numerous required steps and operations. From CRF calibration, HDR image reconstruction and tone mapping in the HDRI process, to quality measure calculation, weight maps generation an image fusion in the Exposure Fusion (EF) process. We replaced these many time consuming, data quality dependent and ressources heavy steps by a single ML model that takes in pixel values of bracketed exposures of a scene, and rapidly produces pixel values of their fusion.

As we conclude this research project, We leave you with an opportunity. EFNN was proposed and it works. But, no method is ever perfect. So, take the chance to test it and improve upon it. Use it in production environments and see how it works. Not only will you be contributing to a very critical field, but you'll also be gaining so much knowledge and experience about many other fields of study.

Bibliography

[1] CommaAI, "`https://comma.ai/`," 2019.

[2] M. Li, Y. Li, and M. Jiang, "Lane detection based on connection of various feature extraction methods," *Advances in Multimedia*, vol. 2018, pp. 1–13, Aug. 2018.

[3] X. Yan and Y. Li, "A method of lane edge detection based on canny algorithm," in *2017 Chinese Automation Congress (CAC)*, pp. 2120–2124, Oct 2017.

[4] He Mao and Mei Xie, "Lane detection based on hough transform and endpoints classification," in *2012 International Conference on Wavelet Active Media Technology and Information Processing (ICWAMTIP)*, pp. 125–127, Dec 2012.

[5] R. Nourine, M. E. Boudihir, and S. F. Khelifi, "Application of radon transform to lane boundaries tracking," in *Lecture Notes in Computer Science*, pp. 563–571, Springer Berlin Heidelberg, 2004.

[6] K. Hirakawa, "Iterative exposure bracketing," in *Imaging and Applied Optics Congress*, p. DMC4, Optical Society of America, 2010.

[7] M. Gupta, D. Iso, and S. K. Nayar, "Fibonacci exposure bracketing for high dynamic range imaging," in *2013 IEEE International Conference on Computer Vision*, pp. 1473–1480, Dec 2013.

[8] C. Aguerrebere, J. Delon, Y. Gousseau, and P. Musé, "Best algorithms for hdr image generation. a study of performance bounds," *SIAM Journal on Imaging Sciences*, vol. 7, no. 1, pp. 1–34, 2014.

[9] P. E. Debevec and J. Malik, "Recovering high dynamic range radiance maps from photographs," in *ACM SIGGRAPH 2008 classes*, pp. 1–10, 2008.

[10] 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)*, vol. 3, pp. 159–163, IEEE, 1999.

[11] M. A. Robertson, S. Borman, and R. L. Stevenson, "Estimation-theoretic approach to dynamic range enhancement using multiple exposures," *Journal of Electronic Imaging*, vol. 12, no. 2, pp. 219–229, 2003.

[12] Y. Salih, W. bt. Md-Esa, A. S. Malik, and N. Saad, "Tone mapping of hdr images: A review," in *2012 4th International Conference on Intelligent and Advanced Systems (ICIAS2012)*, vol. 1, pp. 368–373, June 2012.

[13] T. Mertens, J. Kautz, and F. Van Reeth, "Exposure fusion: A simple and practical alternative to high dynamic range photography," in *Computer graphics forum*, vol. 28, pp. 161–171, Wiley Online Library, 2009.

[14] OpenEXR, "`https://www.openexr.com/`," 2019.

[15] M. Čadík, M. Wimmer, L. Neumann, and A. Artusi, "Evaluation of hdr tone mapping methods using essential perceptual attributes," *Computers & Graphics*, vol. 32, no. 3, pp. 330–349, 2008.

[16] S. Silk and J. Lang, "High dynamic range image deghosting by fast approximate background modelling," *Computers & Graphics*, vol. 36, no. 8, pp. 1060–1071, 2012.

[17] OpenCV, "`https://docs.opencv.org/4.2.0/`," 2019.

[18] F. Drago, K. Myszkowski, T. Annen, and N. Chiba, "Adaptive logarithmic mapping for displaying high contrast scenes," in *Computer graphics forum*, vol. 22, pp. 419–426, Wiley Online Library, 2003.

[19] F. Durand and J. Dorsey, "Fast bilateral filtering for the display of high-dynamic-range images," in *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pp. 257–266, 2002.

[20] P. HDRSOFT, "`https://www.hdrsoft.com/resources/`," 2020.

[21] I. J. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. Cambridge, MA, USA: MIT Press, 2016. `http://www.deeplearningbook.org`.

[22] S. Haykin, *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.

[23] A. F. Agarap, "Deep learning using rectified linear units (relu)," *arXiv preprint arXiv:1803.08375*, 2018.

[24] M. D. Fairchild, "The hdr photographic survey dataset (hdrps)," 2017. `http://markfairchild.org/HDR.html`.

[25] Darktable, "`https://www.darktable.org/usermanual/en/`," 2019.

[26] ImageMagick, "`https://imagemagick.org/`," 2020.

[27] T. docs, "`https://www.tensorflow.org/guide`," 2019.

[28] ProjectPython, "`http://projectpython.net/chapter02/`," 2018.

[29] K. G. Lore, A. Akintayo, and S. Sarkar, "Llnet: A deep autoencoder approach to natural low-light image enhancement," *Pattern Recognition*, vol. 61, pp. 650–662, 2017.

[30] K. Zeng, J. Yu, R. Wang, C. Li, and D. Tao, "Coupled deep autoencoder for single image super-resolution," *IEEE Transactions on Cybernetics*, vol. 47, pp. 27–37, Jan 2017.