National University of Science and Technology POLITEHNICA Bucharest
Faculty of Electronics, Telecommunications and Information Technology

Degraded colour images inpainting system using deep learning techniques

Diploma Thesis

Submitted in partial fulfilment of the requirements
for the degree of *Engineer*
in the domain of *Computer Science and Information Technology*
bachelor study programme *Information Engineering*

Thesis advisors
Asst. Prof. PhD. Eng. Ana Neacșu
Prof. PhD. Eng. Corneliu Burileanu

Student
Cristian CRISTEA

July 2023

Statement of Academic Honesty

I hereby declare that the thesis titled *Degraded colour images inpainting system using deep learning techniques*, submitted to the Faculty of Electronics, Telecommunications and Information Technology, National University of Science and Technology POLITEHNICA Bucharest, in partial fulfilment of the requirements for the degree of *Engineer* in the domain of *Computer Science and Information Technology*, bachelor study programme *Information Engineering*, is written by myself and was never before submitted to any faculty or any higher learning institution in Romania or any other country.

I declare that all information sources I used, including the ones I found on the Internet, are properly cited in the thesis as bibliographical references. Text fragments cited "as is" or translated from other languages are written between quotes and are referenced to the source. Reformulation using different words from certain texts is also properly referenced. I understand that plagiarism constitutes an offence punishable by law.

I declare that all the results I present as coming from simulations, experiments, and measurements I performed, together with the procedures used to obtain them, are real and indeed come from respective simulations, experiments, or measurements. I understand that faking data or results is punishable per the university's regulations.

Bucharest, July 2023

Cristian CRISTEA

.....

"Per aspera ad astra"

Lucius Annaeus Seneca

Table of Contents

List of Figures

List of Tables

List of Acronyms

ANN – Artificial Neural Network

CIFAR – Canadian Institute For Advanced Research

CNN – Convolutional Neural Network

COCO – Common Objects in Context

DNN – Dense Neural Network

DRI – Degradation Ratio Interval

ELU – Exponential Linear Unit

FCNN – Fully-Connected Neural Network

FMM – Fast Marching Method

GAN – Generative Adversarial Network

GD – Gradient Descent

LPIPS – Learned Perceptual Image Patch Similarity

MAE – Mean Absolute Error

MBGD – Mini-Batch Gradient Descent

MSE – Mean Squared Error

NN – Neural Network

PSNR – Peak Signal-to-Noise Ratio

RAM – Random Access Memory

RELU – Rectified Linear Unit

RGB – Red, Green and Blue

SELU – Scaled Exponential Linear Unit

SGD – Stochastic Gradient Descent

SNR – Signal-to-Noise Ratio

SSIM – Structural Similarity Index Measure

VAE – Variational Autoencoder

Introduction

Motivation and Applicability

The advent of digital imagery has fundamentally transformed the way we communicate, share and interpret visual data. Images have become an indispensable part of human interaction, facilitating various applications ranging from social media and telecommunications to complex domains such as medicine, astronomy and remote sensing. However, these images are frequently compromised by degradations such as noise, blur and missing parts, limiting their utility and the accuracy of the inferences drawn from them. This provides the impetus for research into the domain of image inpainting, which seeks to restore these degraded colour images to their original, uncorrupted state.

Image inpainting, essentially, is a technique of filling in missing or corrupted parts of an image in a visually plausible manner. It is a subject of increasing significance within the fields of computer vision and machine learning due to its intricate complexity and vast potential applications. The problem of image restoration, specifically image inpainting, is indeed a complex one. Traditionally, methods such as texture synthesis and patch-based techniques were employed. However, these methods often struggled to reconstruct the degraded sections of complex images accurately. They faced challenges in preserving the global structure and coherence of the image, leading to subpar results, especially for large corrupted regions or intricate textures. The advancements in machine learning, particularly deep learning, have ushered in a promising era for image inpainting. Deep learning techniques, owing to their capability to model high-level abstractions, have shown immense potential in understanding the underlying patterns in image data.

The motivation to engage in this study is multilayered, encompassing both the practical application of the technique and the scientific challenges it presents. Image inpainting offers vast potential for real-world application. The technique is not only employed in the restoration of degraded regions, such as motion blur and missing parts but also for the purpose of removing unwanted elements from an image. The ability to remove irrelevant components while maintaining the image's integrity offers immense value across a broad spectrum of industries and fields.

Historically, the practice of image inpainting was conducted manually, a task requiring the expertise of graphic designers using sophisticated software tools like Adobe Photoshop.

These conventional methods often necessitated a considerable investment of time and resources, especially in the context of high-resolution, complex images. The advent of automatic image inpainting techniques in recent years has sparked a transformation in the field. Leveraging machine learning algorithms, these methods can infer the missing or degraded parts of an image based on the contextual information of the surrounding regions, resulting in a largely streamlined and efficient process.

However, the automatic image inpainting process is not without its challenges. Algorithms capable of reconstructing intricate image structures while preserving natural textures and colours are yet to reach their pinnacle of development. Moreover, the performance of these algorithms can be heavily contingent on the image's specific characteristics and the nature of the degradation. These considerations underscore the necessity for ongoing exploration and enhancement of image inpainting methodologies.

Objectives

The primary goal of this project is to delve into the realm of image inpainting, focusing on the application of deep learning techniques. By constructing a system dedicated to the restoration of degraded colour images, this study is poised to present an exploration of the current landscape of the field. The emphasis of the investigation does not reside in pushing the boundaries of the discipline or aiming for an innovative leap forward. Instead, it seeks to illuminate the intricacies of the domain, presenting an understanding of some existing methodologies and challenges associated with image inpainting. Thus, the motivation for this project is rooted in the pursuit of knowledge and the aspiration to establish a solid foundation in the realm of image processing and machine learning, rather than in the drive for advancements in the field.

The objectives of this thesis are set to explore image inpainting using deep learning techniques. This project is committed to understanding and documenting the process of creating an inpainting system. The objectives form a cohesive research journey, each adding to our overall understanding of this field:

- **Acquisition of online datasets** – The first objective is to identify and acquire suitable online datasets. This is of foremost importance as the quality and relevance of the dataset fundamentally determine the potential accuracy and applicability of the developed model. Careful consideration will be given to ensure the selected datasets are representative of a wide array of image conditions.

- **Development of pre-processing workflow and pipeline** – Next, the focus will be on the design and implementation of a robust pre-processing workflow. The importance of this stage lies in its ability to condition the data into a format suitable for subsequent processing by the neural network. The pre-processing pipeline will involve operations such

as resizing, normalization and the application of synthetic degradation to better prepare the model for diverse image conditions.

- **Designing the neural network architectures** – A vital objective of this thesis is the design of suitable neural network architectures for the task at hand. This will entail a careful evaluation of various Convolutional Neural Networks architectures. The choice of architecture will be guided by a detailed review of the literature, the nature of the datasets and the specific requirements of the image inpainting task.

- **Establishment of training and testing workflow** – Henceforth, a workflow will be developed for the training and testing of the designed models. This involves the creation of a systematic procedure for feeding the data into the model, adjusting model parameters and validating the model's performance.

- **Assessment of model performance** – The final objective is an assessment of the models' performance and thus various evaluation metrics will be employed. The outcomes will not only indicate the effectiveness of the models in the context of the chosen datasets but also provide insight into areas of potential improvement or further exploration.

Chapter 1
Theoretical Aspects

1.1 Visual Representations

1.1.1 The digital image

Images are visual representations of data that are captured, processed and stored in digital format. They consist of individual pixels (i.e. picture element), which are the smallest unit of an image. Images may be seen from the perspective of digital signal processing as $2D$ signals that are represented by a matrix of pixel values. The pixels are organized in a grid and each pixel in the picture corresponds to a discrete value that indicates the intensity at that position. The term grey level is often used to refer to the intensity of monochrome (grayscale) images. Individual monochrome images are combined to create colour images (e.g. in the RGB colour system a colour image consists of three individual monochrome images, referred to as the red, green and blue components or channels).[1]

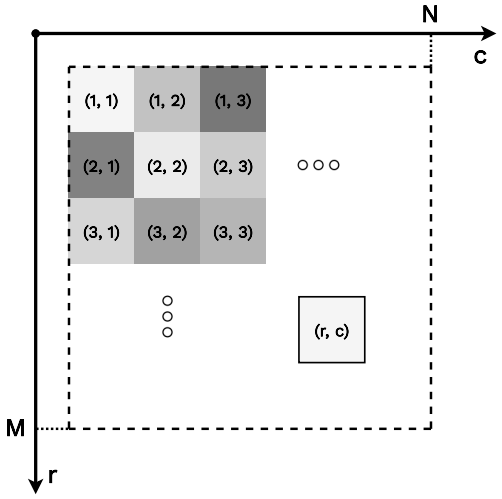


Figure 1.1: Matrix representation of an image

Analogue visual representations are continuous in terms of both the spatial domain and intensity. To be able to store and manipulate them in digital form, the captured images need to be digitized. In the spatial domain, sampling is used to obtain discrete regions, each of which contains a discrete intensity value that was acquired through quantisation. Hence, an

image can alternatively be thought of as a two-dimensional function, $f(r, c)$, where r and c are spatial plane coordinates and the amplitude of f at any pair of coordinates (r, c) is the aforementioned intensity. Conventionally, the image's upper-left corner serves as the origin of the coordinate system. From top to bottom, the row's coordinate advances, whereas the column's coordinate advances from left to right. An image has a size of $M \times N$, where M is the number of rows and N is the number of columns[2]. Images with many channels can be viewed as a vector comprising of their corresponding components, so it can be said that the total size of this type of image is $M \times N \times L$, where L is the number of channels. As a result, the two-dimensional function is now $f(r, c, l)$, where l represents the current channel and r and c retain their original meanings. For example, the indexing scheme for an RGB image can be seen in Figure 1.2.

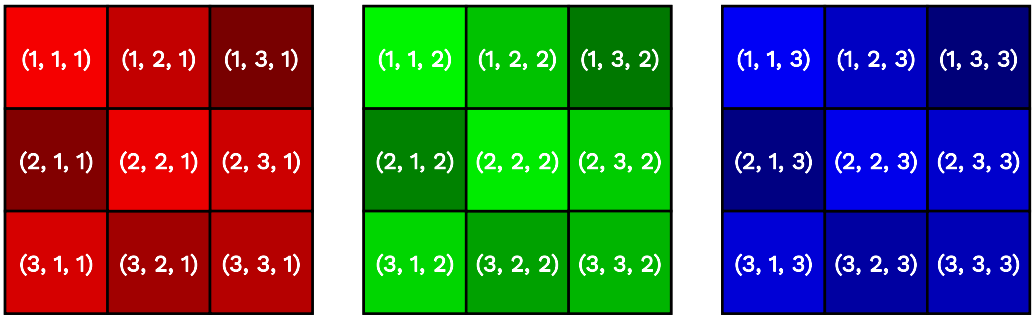


Figure 1.2: Simple 3×3 components of an RGB image

1.1.2 Traditional and perceptual metrics

Image metrics are used to evaluate the quality of images in various applications, such as image compression, image restoration and image enhancement. Traditional image metrics are based on mathematical measures of distance or similarity between two images, while perceptual image metrics take into account the human perception and semantic aspects of images.

Traditional image metrics include Mean Absolute Error, Mean Squared Error, Signal-to-Noise Ratio and Peak Signal-to-Noise Ratio. These metrics are based on the comparison of pixel values or statistical properties of two images and provide a quantitative measure of image quality. However, they may not always correlate well with human perception of image quality and may not capture certain aspects of image quality, such as texture or colour.

Perceptual image metrics, on the other hand, aim to capture the perceptual and semantic aspects of images. These metrics are based on models of human visual perception and can take into account factors such as contrast, texture and colour appearance. Some examples of perceptual image metrics include the Structural Similarity Index Measure and the Learned Perceptual Image Patch Similarity.

In the upcoming sections, definitions and other details will be provided for both traditional and perceptual metrics that are commonly used for evaluating image quality. The functioning of each metric will be explored, while its limitations will also be discussed.

In the following, we will denote g as the original or ground truth image, while the variable f refers to the processed or reconstructed image.

Mean Absolute Error (MAE)

The Mean Absolute Error metric is based on the ℓ_1 norm where its value is scaled by the total number of pixels in the image. Thus, the metric is in fact the normed Manhattan distance between the two images, as defined by the following equation:

$$\mathrm{MAE}(f,g) = \frac{1}{MNL}\sum_{r=1}^{M}\sum_{c=1}^{N}\sum_{l=1}^{L}|f(r,c,l)-g(r,c,l)| \ . \tag{1.1}$$

Mean Squared Error (MSE)

The Mean Squared Error metric is also based on a norm where its value is scaled by the total number of pixels in the image. The used function is ℓ_2, so the metric is actually the normed Euclidean distance between the two images and is defined as follows:

$$\mathrm{MSE}(f,g) = \frac{1}{MNL}\sum_{r=1}^{M}\sum_{c=1}^{N}\sum_{l=1}^{L}\left(f(r,c,l)-g(r,c,l)\right)^2 \ . \tag{1.2}$$

Signal-to-Noise Ratio (SNR)

The Signal-to-Noise Ratio is a metric that measures the quality of an image by comparing the power of the signal (the image) to the power of the noise in the image. The SNR is typically computed as the ratio of the mean pixel value of the signal to the standard deviation of the noise in the image and is described by the equation below:

$$\mathrm{SNR}(f,g) = 10\log_{10}\frac{\frac{1}{MNL}\sum_{r=1}^{M}\sum_{c=1}^{N}\sum_{l=1}^{L}\left(g(r,c,l)\right)^2}{\frac{1}{MNL}\sum_{r=1}^{M}\sum_{c=1}^{N}\sum_{l=1}^{L}\left(f(r,c,l)-g(r,c,l)\right)^2}\ [dB]\,. \tag{1.3}$$

The SNR is measured in decibels (dB), so a higher value indicates a stronger signal relative to the noise, and therefore a higher-quality image with less noise. The SNR may not always be the most appropriate metric for evaluating image quality, as it assumes that the noise in the image is independent and identically distributed, but in many cases, the noise in images can be correlated and exhibit spatial structure.

Peak Signal-to-Noise Ratio (PSNR)

The Peak Signal-to-Noise Ratio is a metric that measures the quality of an image by comparing the maximum possible pixel value of the signal (the image) to the power of the noise in the image. The PSNR is typically computed as the ratio of the peak signal value of the signal (usually 255 for an 8-bit image) to the standard deviation of the noise in the image and is defined as follows:

$$\mathrm{PSNR}(f,g) = 10\log_{10}\frac{\max\limits_{(r,c,l)}\left(g(r,c,l)\right)^2}{\frac{1}{MNL}\sum\limits_{r=1}^{M}\sum\limits_{c=1}^{N}\sum\limits_{l=1}^{L}\left(f(r,c,l)-g(r,c,l)\right)^2}\,[dB]\,. \tag{1.4}$$

The PSNR has the same limitations as the SNR regarding the statistical proprieties of the signal when used to measure the quality of images.

Structural Similarity Index Measure (SSIM)

Natural digital images possess significant structural characteristics: their pixel values demonstrate robust interdependencies, particularly in spatially adjacent regions, and these interdependencies convey essential information concerning the organization of the objects within the visual scene[3]. Therefore, the main problem with the above metrics is that they are all based solely on mathematical measures of distance or similarity between two images, without taking into account the perceptual or semantic aspects of the images. For example, two images that have similar MAE or MSE may still look different to a human observer due to differences in texture, contrast, or other visual features. Additionally, a high SNR or PSNR value may not necessarily correspond to a high subjective quality of the image. Therefore, while these metrics are useful for comparing and optimizing algorithms based on mathematical measures, they should be used in conjunction with other metrics that include perceptual information about the signals or images.

The Structural Similarity Index Measure is a widely used perceptual metric in image processing that measures the similarity between two images, by aggregating local and adjacent image characteristics. The SSIM ranges between 0 and 1, where a value of 1 indicates perfect similarity between the two images. This metric has proven to be effective in assessing image quality in a variety of applications, including image compression, denoising, reconstruction and super-resolution[3]–[5].

As can be seen in Figure 1.3, the SSIM proposes three comparison functions: *luminance* measures the overall brightness or intensity of an image, *contrast* calculates the difference between the brightest and darkest parts of an image and *structure* measures the patterns and edges in an image. They are defined as follows:

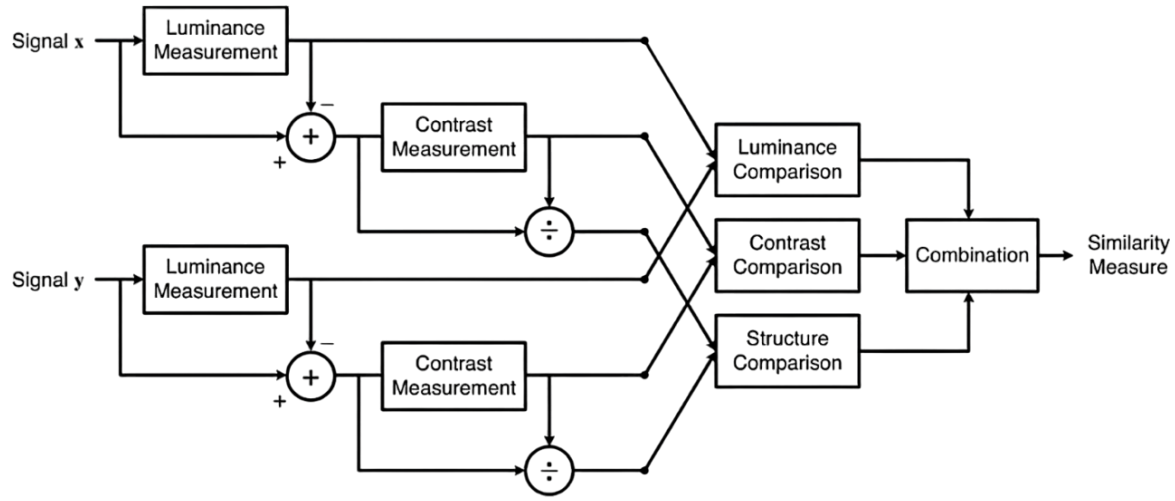


Figure 1.3: Diagram of the structural similarity (SSIM) measurement system[3]

- Luminance

$$l(\mathbf{x},\mathbf{y}) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1}\,, \tag{1.5}$$

- Contrast

$$c(\mathbf{x},\mathbf{y}) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2}\,, \tag{1.6}$$

- Structure

$$s(\mathbf{x},\mathbf{y}) = \frac{\sigma_{xy} + C_3}{\sigma_x + \sigma_y + C_3}\,, \tag{1.7}$$

Here-above, the three constants (i.e. C_1, C_2 and $C_3 \in \mathbb{R}$) are present to avoid instability or division by zero when any of the denominators is very close to zero. They are systematically set using the following definitions:

$$C_1 = (K_1 D)^2\,, C_2 = (K_2 D)^2\,, C_3 = \frac{C_2}{2}\,, \tag{1.8}$$

where D signifies the dynamic range of the pixel values (typically 255 for 8-bit images) and K_1 and K_2 are very small (i.e. $0 < K_1, K_2 \ll 1$) arbitrarily chosen constants. In the original implementation, their values are 0.01 and 0.03 respectively[3].

Moreover, it is crucial to take spatiality into account, as, in the context of image quality assessment, employing the SSIM index on a local basis proves more beneficial than applying it globally. Firstly, the statistical attributes of an image tend to exhibit substantial spatial non-stationarity. Secondly, image distortions, which may be either dependent or independent of local image statistics, can also display spatial variability. Lastly, at customary viewing distances, a confined area within the image can be discerned with high resolution by the human observer at a single instance. The computation of local statistics for the current pixel will be performed employing an 11×11 circular-symmetric Gaussian weighting function $\mathbf{w} = (w_i)_{1 \le i \le W}$, exhibiting a standard deviation of 1.5 samples and normalized to achieve a

unit sum[3]. The process of convolution that forms the basis for how this kernel is applied to every pixel in the image will be further discussed in section 1.3. So the estimates for the local means, dispersions and cross-correlation are computed using the following equations:

$$\mu_x = \sum_{i=1}^{W} w_i x_i \quad and \quad \mu_y = \sum_{i=1}^{W} w_i y_i \,, \tag{1.9}$$

$$\sigma_x = \sqrt{\sum_{i=1}^{W} w_i (x_i - \mu_x)^2} \quad and \quad \sigma_y = \sqrt{\sum_{i=1}^{W} w_i (y_i - \mu_y)^2} \,, \tag{1.10}$$

$$\sigma_{xy} = \sum_{i=1}^{W} w_i (x_i - \mu_x)(y_i - \mu_y) \,. \tag{1.11}$$

In the above computations, the two signals $\mathbf{x}$ and $\mathbf{y}$ correspond to the 11×11 patches centred on the currently processed pixel taken from both images and x_i and y_i respectively are the pixels contained in them. Finally, the three comparison functions from equations (1.5), (1.6) and (1.7) are combined to form the similarity measure called the SSIM index:

$$\mathrm{SSIM}(\mathbf{x}, \mathbf{y}) = [l(\mathbf{x}, \mathbf{y})]^{\alpha} \cdot [c(\mathbf{x}, \mathbf{y})]^{\beta} \cdot [s(\mathbf{x}, \mathbf{y})]^{\gamma} \,, \tag{1.12}$$

where $\alpha > 0, \beta > 0$ and $\gamma > 0$ are parameters used to alter the relative relevance of the three components. To simplify the expression, the original implementation sets $\alpha = \beta = \gamma = 1$, therefore equation (1.12) becomes:

$$\mathrm{SSIM}(\mathbf{x}, \mathbf{y}) = \frac{\left(2\mu_x\mu_y + C_1\right)\left(2\sigma_{xy} + C_2\right)}{\left(\mu_x^2 + \mu_y^2 + C_1\right)\left(\sigma_x^2 + \sigma_y^2 + C_2\right)} \,. \tag{1.13}$$

In practical applications, it is typically necessary to obtain a singular, comprehensive quality measure for the entire image. The mean SSIM (MSSIM) index serves to assess the overall image quality as follows:

$$\mathrm{MSSIM}(\mathbf{X}, \mathbf{Y}) = \frac{1}{P} \sum_{k=1}^{P} \mathrm{SSIM}(\mathbf{x}_k, \mathbf{y}_k) \,, \tag{1.14}$$

where $\mathbf{X}$ and $\mathbf{Y}$ represent the reference and distorted images, respectively; $\mathbf{x}_k$ and $\mathbf{y}_k$ denote the image contents at the k-th local window and P signifies the total number of local windows in the image[3].

However, the Structural Similarity Index Measure has some rather essential limitations. For instance, it does not consider colour information. It, therefore, may not be suitable for applications where colour accuracy is essential, but it can be computed for multiple channels images by using various techniques. For example, for RGB images we can employ three methods. Firstly, the colour image is converted to grey, then the SSIM is computed. Secondly, the image's RGB space is transformed to $\mathrm{YC_bC_r}$ and a weighted average of each

channel's SSIM calculated, where the luma component is often given greater weight. Lastly, one can compute the SSIM for every constituent layer and take the average to obtain the final metric[6]. Throughout this project, this last methodology will be considered and used in the actual implementation.

Learned Perceptual Image Patch Similarity (LPIPS)

The major issue with the heretofore-presented image metrics is that they assess the quality of an image with respect to a reference one from a strictly analytical standpoint, so one may try to come up with a more human-like method. However, although humans can effortlessly evaluate the perceptual similarity between two images with relative ease, the underlying processes are believed to be considerably intricate. Nevertheless, prevalent traditional metrics, including PSNR and SSIM, are elementary, superficial functions and neglect numerous subtleties of human perception. The distinct challenge presented by computer vision lies in the fact that even the seemingly uncomplicated task of comparing visual patterns persists as an unresolved problem. Visual patterns are not only high-dimensional and highly correlated, but the concept of visual similarity itself is frequently subjective, aiming to emulate human visual perception[7].

As defined above, MAE, MSE, SNR and PSNR behave more like a distance and are mostly unsuitable for evaluating structured outputs like colour images because they presume pixel-by-pixel independence. For example, blurring an image causes a small pixel value change, but a noticeable one from a human perception point of view. There have been multiple attempts at solving this issue, such as the aforementioned SSIM, but the main problem is that it only considers structural information in the images, such as edges and texture, and does not take into account colour information or other visual features. This means that SSIM may not accurately capture the perceived similarity of two images in certain cases, such as when one image is brighter or has more saturated colours than the other. So, is there any way of implementing a perceptual similarity? As it turns out, it has been observed that the internal activations of deep convolutional networks, which are trained for high-level image classification tasks, often exhibit an unexpected versatility as a representational space for a substantially broader array of tasks within the computer vision community[7].

Therefore, [7] defines what is called the Learned Perceptual Image Patch Similarity (LPIPS), which uses different convolutional neural networks, such as VGGNet[8] or AlexNet[9], to compute a value in the interval $[0, 1]$ to assess the similitude between the two images. A value approaching 0 means that the images are highly similar (i.e. behaves like a loss function), but for the purposes of this project and to obtain a similar meaning to the SSIM, these values were inverted, so the closer the value to 1 the better. The metric is calculated as follows: given a convolutional neural network $\mathcal{F}$, deep embeddings are computed (after some numbers of layers, depending on the architecture used), the activations are normalized per channel, each one is scaled by a vector $\mathbf{w}$ and the ℓ_2 distance is computed; then the

average across spatial dimension and the sum across all layers are taken. Those distances are then fed to a small fully-connected network with a sigmoid output, resulting in the expected metric. All the mentioned neural network-related terms and notions will be further discussed in section 1.3. (*Note:* AlexNet has been used in this project for calculating the LPIPS metric and VGGNet when training.)

1.2 Neural Networks

Artificial Neural Networks (ANNs), also simply known as "Neural Networks" (NNs), are a class of machine learning models that draw their inspiration from the structure and operation of biological neural networks and are often meant to depict the mapping function of various input variables to matching outputs. Biological neural networks are made up of various assemblies of neurons that are connected by links known as synapses. Sequences of action potentials (i.e. nerve impulses or spikes) are used to encode the information that is transmitted between biological neurons. When a neuron spikes, it releases a neurotransmitter, a chemical that travels a small distance across a synapse before reaching other neurons. A spike is released when the producing neuron gets enough input current to have a membrane potential that is greater than a certain threshold value. This threshold determines whether a neuron participates in the ongoing process by transmitting information to the next members of the current group[10].

A **Fully-Connected Neural Network** (FCNN), also referred to as a **Dense Neural Network** (DNN), is one of the most basic neural network types. The basic building block of this model is the neuron, which is modelled as a mathematical function that takes one or more inputs and produces an output. The inputs to a neuron are weighted, meaning that some inputs are more important than others, and the neuron applies an activation function to the weighted sum of the inputs to produce an output. As can be seen in Figure 1.4, neurons are organized into layers, with each layer consisting of a set of neurons that process the inputs from the previous layer, and each connection between two adjacent layers is represented by a weight matrix. The first layer is called the input layer and serves as a buffer for the input data, and the last layer is called the output layer. There may be one or more hidden layers between the input and output layers and each layer may have a different number of neurons.

Activation functions are an essential component of neural networks. They are used to introduce non-linearity into the output of each neuron, allowing the network to learn and model complex nonlinear relationships in the data. An activation function takes the weighted sum of the inputs to a neuron and applies a mathematical function to produce the output of the neuron. The activation function's output is subsequently passed on to the network's next layer. In the scenario of employing a linear activation function, the output generated by a fully-connected neural network can solely be expressed as a linear composition of the input variables. As such, the neural network would essentially be reduced to a composition of merely

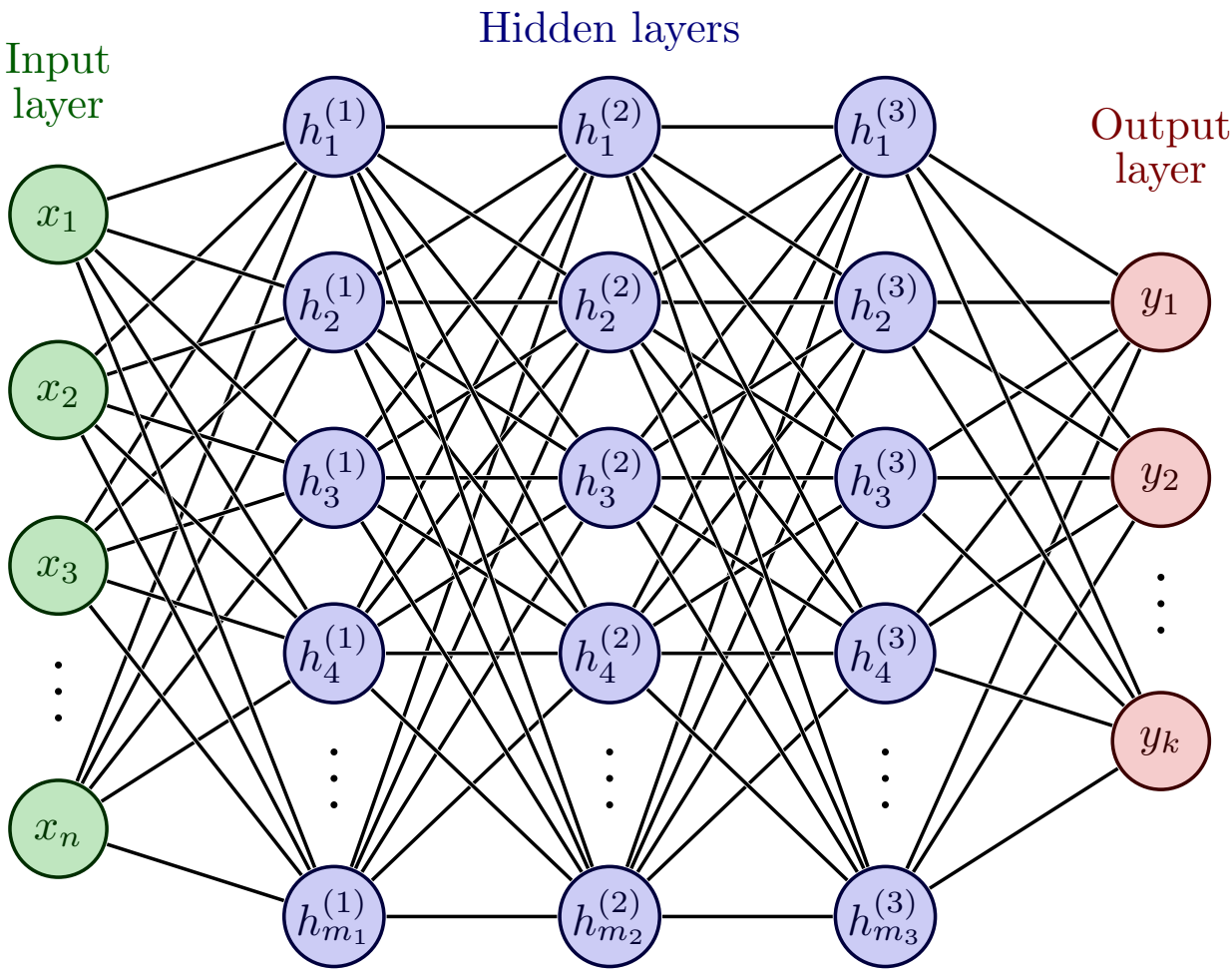


Figure 1.4: Graph representation of a fully connected neural network[11]

the input and output layers, with the weight matrix encapsulating the coefficients pertaining to the aforementioned linear combination. This eliminates the necessity for any auxiliary hidden layers within the network architecture, rendering it a simple linear mapping from the input space to the output space. Unfortunately, linear activation functions have the big disadvantage of not being able to model nonlinear relationships between inputs and outputs and since most problems cannot be adequately modelled with only a linear combination, almost all neural networks are created using non-linear activation functions. Furthermore, activation functions help guarantee that the network's output is bounded, preventing it from becoming arbitrarily large or small, which can lead to numerical instability.

Let us revisit the example presented in Figure 1.4, a fully-connected neural network comprised of five layers: an input layer containing n neurons, three intermediate or hidden layers composed of m_1, m_2 and m_3 neurons, respectively and an output layer with k neurons. In the context of this model, let us consider $\mathbf{x} \in \mathbb{R}^n$ as an input vector and $\mathbf{W}^{(1)} \in \mathbb{R}^{m_1 \times n}$ as the weight matrix facilitating the connection from the input layer to the first hidden layer. The vectors $\mathbf{z}^{(1)}$, $\mathbf{h}^{(1)} \in \mathbb{R}^{m_1}$ represent the input and output vectors of the first hidden layer, respectively, while the function $\Phi_1 : \mathbb{R} \to \mathbb{R}$ represents the activation function applied to each

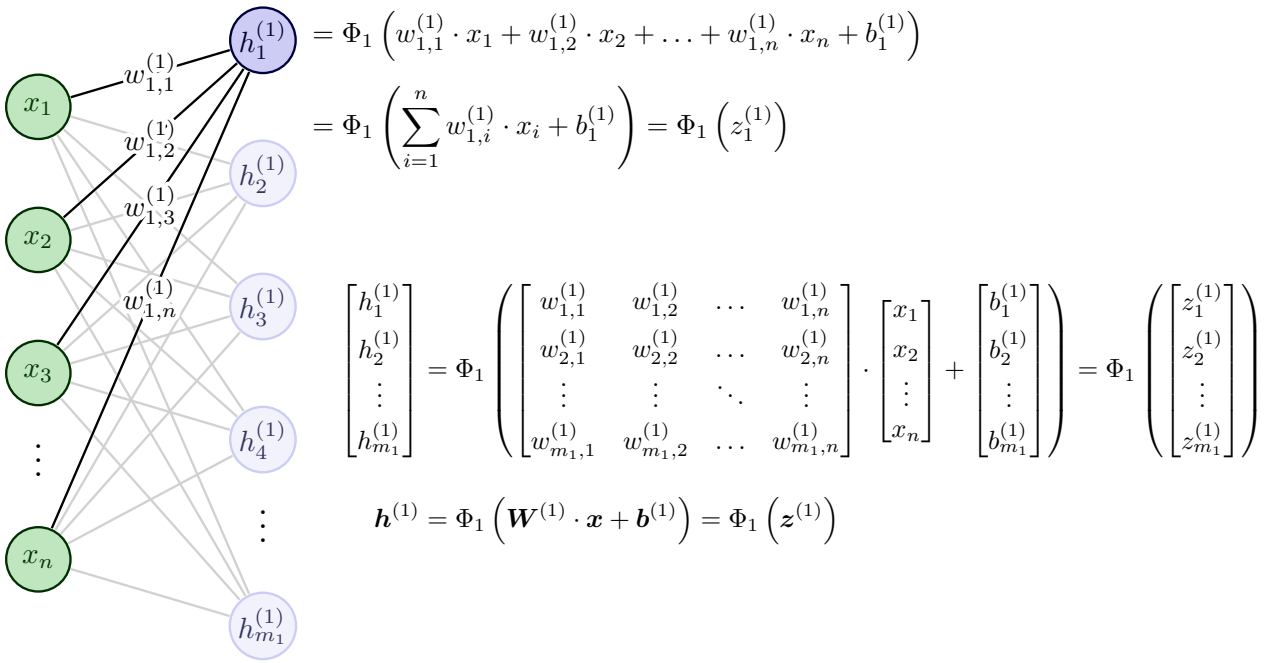


Figure 1.5: Derived computation for the outputs of the first hidden layer[11]

neuron in the first hidden layer. The computations involved in determining the output of the first neuron in the initial hidden layer and the collective output of the first hidden layer are explained in Figure 1.5. In this figure, $\mathbf{b}^{(1)} \in \mathbb{R}^{m_1}$ signifies the vector of biases and applying a function to a vector means applying it element-wise. Given that the outputs of one layer function as the inputs for the following layer, similar computations are employed for each successive hidden layer. This propagative process continues until the final output of the network is generated, using the corresponding parameters and activation functions for each layer.

For us humans, the main goal of our biological neural networks is to attain knowledge through the process called learning. This entails gathering new information to construct new representations of various events and creating links between new information and previously learned patterns. From the perspective of artificial neural networks, **learning** involves changing its internal state by updating its parameters to accomplish a goal within the constraints set by a special function, called the objective function. This corresponds to decreasing the dissimilarity between the neural network's output representations and some ideal target representations. With the introduction of NNs for tackling specific tasks, many learning paradigms have been developed, the two most common being supervised learning and unsupervised learning.

Supervised learning entails that the network has access to labelled data, where the expected result for each input is known. The purpose of supervised learning is to develop a function that, given labelled data, can map inputs to outputs. This function may then be used to forecast or categorize previously unknown data. Regression, where the objective

is to predict a continuous output variable, and classification, where the goal is to predict a categorical output variable, are two common instances of supervised learning.

Unsupervised learning, on the other hand, is concerned with neural networks that uses unlabeled data, where the desired outcome for each input is unknown. The purpose of unsupervised learning is to discover patterns and correlations in input data without the use of tagged data. Unsupervised learning examples include clustering, which aims to group similar data points, and dimensionality reduction, which strives to minimize the number of features in the input data while maintaining as much information as possible. The process for learning such representations entails imposing external constraints on the output such that similitude is preserved after the network's transformation.

We will concentrate our attention on supervised methodology, thus our main goal is to update the network's weights to better approximate the mapping function between the input and output variables. To be able to do so, we need to define a quantifiable measure that will tell us the discrepancy between the predicted output and the true target values. This distance-like set of measures bears the name of **loss** functions (or simply losses), also known as cost or objective functions. Let $\mathbf{x}$ be the input vector, $\mathbf{y}$ the ground truth output vector and $\hat{\mathbf{y}}$ the neural network's output vector, then we denote the loss function as the expression $\mathcal{L} = \mathcal{L}(\mathbf{y}, \hat{\mathbf{y}})$. Furthermore, we can compute $\hat{\mathbf{y}}$ by passing the input data $\mathbf{x}$ through the network's layers and applying the associated activation functions, an operation called **forward propagation**. In a feedforward neural network, the output of each layer serves as the input for the next layer, so the forward propagation process can be summarised using the following equations:

$$\begin{cases} \mathbf{h}^{(i)} = \mathbf{x} & \text{if } i = 0 \\ \mathbf{h}^{(i)} = \Phi^{(i)}(\mathbf{W}^{(i)} \cdot \mathbf{h}^{(i-1)} + \mathbf{b}^{(i)}) & \text{if } i \in [1, L-1] \\ \hat{\mathbf{y}} = \mathbf{h}^{(i)} = \Phi^{(i)}(\mathbf{W}^{(i)} \cdot \mathbf{h}^{(i-1)} + \mathbf{b}^{(i)}) & \text{if } i = L\,, \end{cases} \tag{1.15}$$

where L is the number of functional layers (i.e. the input layer is not considered), the output layer being the L^{th} one, and $\mathbf{W}^{(i)}$, $\mathbf{b}^{(i)}$, $\mathbf{h}^{(i)}$ and $\Phi^{(i)}$ is the weight matrix, biases vector, output and activation function of the i^{th} layer, respectively. Thus, the loss can be rewritten as a function of the network's internal parameters $\mathbf{W}^{(i)}$ and $\mathbf{b}^{(i)}$, given the expression of each activation functions of each layer.

The network's weights must be optimised through a process known as **training** that entails minimising the overall loss in order to achieve a more suitable mapping function. The training process of a neural network iteratively updates the model's parameters. To optimise the model's weights, a technique called **backpropagation** is employed, which calculates the gradients of the loss function with respect to each weight by applying the chain rule for derivatives. Backpropagation essentially computes the error signal for each neuron in the network, starting from the output layer and propagating backwards to the input layer. This

method has been the de facto standard algorithm for some time when it comes to training neural networks[12],[13]. To be able to encompass all the available data pairs $(\mathbf{y}_k, \hat{\mathbf{y}}_k)$ in the training process, we define the total loss function

$$\mathcal{L}_{\text{total}} = \frac{1}{T} \sum_{k=1}^{T} \mathcal{L}_k \,, \tag{1.16}$$

where $\mathcal{L}_k = \mathcal{L}\left(\mathbf{y}_k, \hat{\mathbf{y}}_k\right)$ and T is the number of training examples. Once the gradients of this loss are obtained, an optimisation algorithm updates the weights and biases based on these gradients. Moving forward, we define the following: θ represents the model's parameter (i.e. the weight $w_{j,p}^{(i)}$ or the bias $b_j^{(i)}$), t denotes the iteration, $\eta \in (0, 1]$ is the **hyperparameter** (non-trainable) **learning rate** and ε is a minuscule positive constant (roughly $\varepsilon \in [10^{-8}, 10^{-6}]$) to ensure numerical stability and prevent issues arising from division by very small numbers or even zero. Among the most popular **optimisation algorithms** are:

- **Gradient Descent** (GD) is a first-order optimisation algorithm and works iteratively by updating the model's parameters in the direction of the negative gradient of the loss function with respect to the parameters. It moves in the direction of the steepest descent by employing the partial derivative of the loss function calculated over the entire dataset. The update rule for each parameter can be described as follows:

$$\theta_{t+1} = \theta_t - \eta \cdot \frac{\partial \mathcal{L}_{\text{total}}}{\partial \theta_t} \,. \tag{1.17}$$

  The advantages of GD include its simplicity, however, it has some disadvantages. First, the algorithm can be slow to converge, particularly when the loss function is ill-conditioned, meaning that the curvature of the function is not uniform in all directions, leading to local minima instead of global ones. Second, the computational complexity is high for large datasets, as the gradient must be computed for all samples at each iteration.

- **Stochastic Gradient Descent** (SGD) is a variant of Gradient Descent that addresses the computational complexity issue by updating the model's parameters based on a single training sample at each iteration. The update rule for SGD is similar to that of Gradient Descent, but the gradient is computed using only one randomly chosen sample:

$$\theta_{t+1} = \theta_t - \eta \cdot \frac{\partial \mathcal{L}_k}{\partial \theta_t} \,, \quad \forall k \in \{1, 2, \ldots, T\} \,. \tag{1.18}$$

  By using only one sample, the computational cost is reduced and the algorithm can converge faster. However, the trade-off is that the convergence is noisier, with the parameter updates being more erratic. This can, on the other hand, help escape local minima in non-convex optimisation problems. The main disadvantage is the noisy convergence, which

can make it challenging to find an exact minimum, often requiring a decreasing learning rate schedule to improve convergence.

- **Mini-Batch Gradient Descent** (MBGD) is a compromise between Gradient Descent and Stochastic Gradient Descent. It updates the model's parameters using a small batch of training samples instead of a single sample or the entire dataset. The update rule for Mini-Batch Gradient Descent is as follows:

$$\theta_{t+1} = \theta_t - \eta \cdot \frac{\partial \mathcal{L}^{[b]}}{\partial \theta_t}, \quad \forall b \in \{1, 2, \dots, B\}, \tag{1.19}$$

where B is a mini-batch containing a subset of training samples. MBGD combines the benefits of both GD and SGD, providing a balance between computational efficiency and convergence stability. It leverages the power of vectorised operations on modern hardware, such as GPUs, to process multiple samples simultaneously, leading to faster convergence. The main disadvantage is that the choice of the mini-batch size can be a critical hyperparameter, affecting both convergence speed and stability. This approach of computing the gradients will be employed for all following similar types of algorithms.

Furthermore, MBGD can be considered to be the general case of the two previously described algorithms, because when the mini-batch size is equal to the entire dataset, it becomes equivalent to Gradient Descent and when the mini-batch size is equal to one, it turns into Stochastic Gradient Descent.

- **Gradient Descent with Momentum** is an extension of the standard Gradient Descent algorithm that aims to accelerate convergence and mitigate oscillations in the parameter updates. The key idea behind momentum-based methods is to incorporate a momentum term, which is a fraction of the previous weight update, into the current update. By doing so, the algorithm can accumulate velocity in consistent gradient directions while dampening oscillations in directions with alternating gradients. The algorithm has different implementations, so the most used update rule for Gradient Descent with Momentum can be described as follows:

$$\theta_{t+1} = \theta_t - \eta \cdot v_{\theta_{t+1}}, \tag{1.20}$$

$$v_{\theta_{t+1}} = \beta \cdot v_{\theta_t} + (1 - \beta) \cdot \frac{\partial \mathcal{L}^{[b]}}{\partial \theta_t}, \tag{1.21}$$

where v_θ is the velocity term and β is the momentum coefficient (typically $\beta \in [0.8, 0.9]$). Advantages of Gradient Descent with Momentum include faster convergence and reduced oscillations compared to standard Gradient Descent, particularly in situations where the loss function has an ill-conditioned curvature or exhibits elongated valleys. By accumulating velocity in consistent gradient directions, the algorithm can traverse these valleys more efficiently and reach the optimal solution more quickly. Moreover, the momentum term can help the algorithm escape shallow local minima and saddle points, which can be

particularly beneficial for non-convex optimisation problems encountered in deep learning.

Disadvantages of Gradient Descent with Momentum include the introduction of an additional hyperparameter, the momentum coefficient, which requires tuning for optimal performance. Furthermore, although the momentum term can help escape shallow local minima, it may also cause the algorithm to overshoot the optimal solution, especially if the learning rate is too high or the momentum coefficient is too large. Despite these disadvantages, it is often a preferred choice over standard Gradient Descent due to its improved convergence properties and robustness to various loss function shapes.

- **AdaGrad**[14] (Adaptive Gradient Algorithm) is an adaptive learning rate optimisation algorithm designed to improve the convergence of gradient-based methods. The key idea behind AdaGrad is to adapt the learning rate for each parameter based on the history of gradients, allowing different learning rates for different parameters. The algorithm achieves this by maintaining a per-parameter sum of squared gradients and updating the learning rate for each parameter accordingly. The update rule for AdaGrad is as follows:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\mathcal{S}_{\theta_{t+1}}} + \varepsilon} \cdot \frac{\partial \mathcal{L}^{[b]}}{\partial \theta_t}, \tag{1.22}$$

$$\mathcal{S}_{\theta_{t+1}} = \mathcal{S}_{\theta_t} + \left(\frac{\partial \mathcal{L}^{[b]}}{\partial \theta_t} \right)^2, \tag{1.23}$$

  where $\mathcal{S}_\theta$ represents the sum of squared gradients and is initialized with 0 for each parameter at the beginning of training. The main advantage of AdaGrad is its ability to handle sparse features and data with different scales effectively. By adapting the learning rate for each parameter, it can provide faster convergence for frequently occurring features and more conservative updates for less frequent features. This can be particularly useful in natural language processing and other tasks involving high-dimensional, sparse data.

  Nevertheless, there are some disadvantages to AdaGrad. One of the main issues is the accumulation of squared gradients in the denominator, which can cause the learning rate to decrease too rapidly, leading to slow convergence or premature stopping before reaching an optimal solution. This effect is particularly problematic for deep learning tasks, where the number of training iterations can be quite large.

- **Adam**[15] (Adaptive Moment Estimation) is an adaptive learning rate optimisation algorithm that combines the benefits of momentum-based optimisation and adaptive learning rate methods. The key idea behind Adam is to estimate both the first-order moment (mean) and the second-order moment (uncentered variance) of the gradients to adapt the learning rate for each parameter. This combination of momentum and adaptive learning rates allows Adam to achieve faster convergence and better performance than many other

optimisation algorithms. The update rule for Adam can be described as follows:

$$\theta_{t+1} = \theta_t - \eta \cdot \frac{\hat{m}_{\theta_{t+1}}}{\sqrt{\hat{v}_{\theta_{t+1}}} + \varepsilon}\,, \tag{1.24}$$

$$\hat{m}_{\theta_{t+1}} = \frac{m_{\theta_{t+1}}}{1-\beta_1^2} \quad \text{and} \quad m_{\theta_{t+1}} = \beta_1 \cdot m_{\theta_t} + (1-\beta_1) \cdot \frac{\partial \mathcal{L}^{[b]}}{\partial \theta_t}\,, \tag{1.25}$$

$$\hat{v}_{\theta_{t+1}} = \frac{v_{\theta_{t+1}}}{1-\beta_2^2} \quad \text{and} \quad v_{\theta_{t+1}} = \beta_2 \cdot v_{\theta_t} + (1-\beta_2) \cdot \left(\frac{\partial \mathcal{L}^{[b]}}{\partial \theta_t}\right)^2\,, \tag{1.26}$$

where m and v represent the first-order and second-order moment estimates, respectively and β_1 and β_2 are the decay rates for the moment estimates. Usually, β_1 is initialized to 0.9 and β_2 to 0.999, while the estimates are always initialized to 0 for every parameter. The advantages of Adam include its fast convergence and robustness to various types of data and model architectures. Additionally, it requires minimal tuning of its hyperparameters.

However, one disadvantage of Adam is that it can sometimes exhibit undesirable convergence properties, such as overshooting the optimal solution or displaying high variance in parameter updates. Consequently, AMSGrad[16] is an updated version of the Adam optimisation algorithm that aims to address the convergence issues found in the original Adam algorithm by introducing a new update rule for the second-order moment estimate that ensures the maximum of all past squared gradients is always used. This modification prevents the learning rate from increasing and provides more stable convergence. The update rule for AMSGrad is as follows:

$$\hat{v}_{\theta_{\max}} = \max\left(\hat{v}_{\theta_{t+1}},\ \hat{v}_{\theta_{\max}}\right), \tag{1.27}$$

$$\theta_{t+1} = \theta_t - \eta \cdot \frac{\hat{m}_{\theta_{t+1}}}{\sqrt{\hat{v}_{\theta_{\max}}} + \varepsilon}\,. \tag{1.28}$$

The main advantage of AMSGrad is that it provides improved convergence properties compared to the original Adam algorithm, particularly for non-convex optimisation problems. By using the maximum of past squared gradients, AMSGrad ensures more stable parameter updates and reduces the risk of overshooting the optimal solution. However, AMSGrad may exhibit slower convergence in certain situations compared to Adam.

- **Other** gradient-based optimisation techniques include Adadelta[17], NAdam[18], AdamW[19], RMSProp[20] and even more recent ones like Evolved Sign Momentum (Lion)[21]. Apart from these, there are numerous other optimisation algorithms that have been developed to address specific challenges or offer alternative approaches to model training. These include but are not limited to conjugate gradient methods, quasi-Newton methods (e.g. Broyden–Fletcher–Goldfarb–Shanno (BFGS) and Limited-memory BFGS (L-BFGS)[22] and evolutionary algorithms (e.g. Genetic Algorithms, Particle Swarm Optimisation and

Differential Evolution)[23]. Some of these algorithms are gradient-based, leveraging gradient information to guide the search for optimal solutions, while others are gradient-free, relying on heuristics or sampling techniques to explore the search space.

In the realm of deep learning, a multitude of activation functions exist, each with its unique properties and characteristics. The chosen function depends on the specific problem, the model architecture and the desired trade-offs in terms of computational complexity, learning speed and nonlinearity. While the range of available activation functions is vast, we will focus on the most commonly used and well-established activation functions as they are presented in Table 1.1. The binary step is a simple threshold-based function that activates the neuron if the input exceeds zero, providing a basic on-off switch. The sigmoid function is a smooth, S-shaped curve that maps input values to the range $(0, 1)$, enabling probabilistic interpretation for binary classification and gradient-based learning. The hyperbolic tangent function is similar to the sigmoid function, but it maps input values to the range $(-1, 1)$, providing a more balanced output with stronger gradients for learning. Rectified Linear Unit (ReLU)[24] is a piecewise linear function that outputs the input value if it is positive and zero otherwise, offering computational efficiency and mitigating the vanishing gradient problem. Leaky ReLU[25] addresses the *dying ReLU*[26] issue, where some neurons may become inactive, by introducing a small slope for negative input values, thus maintaining a non-zero gradient. Exponential Linear Unit (ELU)[27] improves upon ReLU and Leaky ReLU by providing a smooth, non-linear curve for negative input values, ensuring a non-zero gradient and reducing the vanishing gradient problem. Scaled Exponential Linear Unit (SELU)[28] is a self-normalising activation function that combines the benefits of ELU with an automatic normalisation of activations, leading to improved convergence and generalisation. There is also the Softmax activation function which typically is used in the output layer of a neural network for multi-class classification problems, extending the sigmoid function. It maps a vector of input values (logits) to a probability distribution over the possible classes. The function is defined as follows:

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}}, \quad \forall i \in \{1, 2, \ldots, k\}, \tag{1.29}$$

where x_i is the i^{th} component of the input vector and k is the number of classes. Softmax ensures that the sum of the probabilities for all classes is equal to 1, allowing for interpretable class probabilities and easier decision-making.

Training a neural network involves numerous challenges and potential pitfalls that can affect the performance and generalisation capabilities of the model. Among the most common issues encountered during the training process are overfitting, underfitting, vanishing gradients and exploding gradients. In the following paragraphs, we delve deeper into each of these issues, elaborating on their origins, consequences and mitigation strategies.

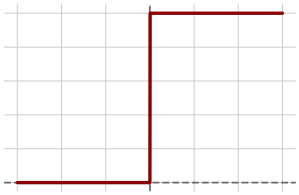
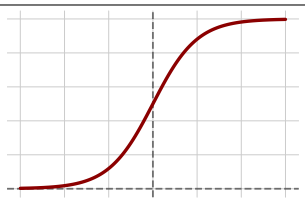
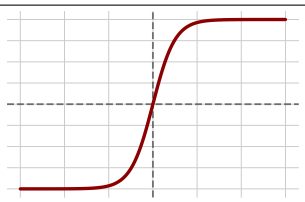
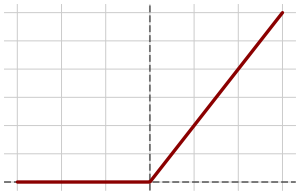
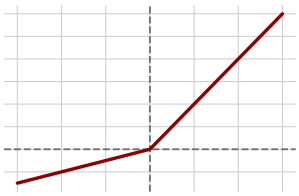
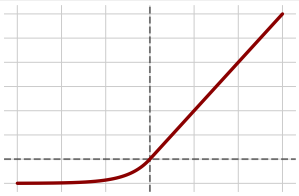
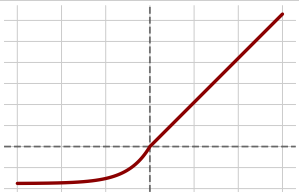
Name	Function	Graph
Binary step	$\Phi\colon \mathbb{R} \to \{0,1\}$ $\Phi(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x \geq 0 \end{cases}$	
Sigmoid (Logistic)	$\Phi\colon \mathbb{R} \to (0,1)$ $\Phi(x) = \dfrac{1}{1+e^{-x}}$	
Hyperbolic Tangent	$\Phi\colon \mathbb{R} \to (-1,1)$ $\Phi(x) = \dfrac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$	
Rectified Linear Unit (ReLU)	$\Phi\colon \mathbb{R} \to [0,\infty)$ $\Phi(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$	
Leaky Rectified Linear Unit (Leaky ReLU)	$\Phi\colon \mathbb{R} \to (-\infty,\infty)$ $\Phi(x) = \begin{cases} \alpha x & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$ $\alpha \in (0,1)$	
Exponential Linear Unit (ELU)	$\Phi\colon \mathbb{R} \to (-\alpha,\infty)$ $\Phi(x) = \begin{cases} \alpha\,(e^{x}-1) & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases}$ $\alpha \in \mathbb{R}$	
Scaled Exponential Linear Unit (SELU)	$\Phi\colon \mathbb{R} \to (-\lambda\alpha,\infty)$ $\Phi(x) = \begin{cases} \lambda\alpha\,(e^{x}-1) & \text{if } x < 0 \\ \lambda x & \text{if } x \geq 0 \end{cases}$ $\lambda\,,\alpha \in \mathbb{R}$	

Table 1.1: Representative activation functions

Overfitting is a prevalent issue in machine learning, particularly in deep learning, where models with a large number of parameters are prone to capturing noise or idiosyncrasies in the training data. This leads to excellent performance on the training data but poor generalisation of unseen data. Several factors contribute to overfitting, such as insufficient or unrepresentative training data, a complex model architecture with excessive parameters, or inadequate regularisation. To prevent overfitting, regularisation techniques can be applied or another used popular strategy is dropout[29]. Increasing the size and diversity of the training

data, employing data augmentation techniques[30], or simplifying the model architecture by reducing the number of layers or neurons can also help combat overfitting and improve the model's generalisation capabilities.

Underfitting arises when a neural network is unable to capture the true underlying patterns in the data, resulting in subpar performance on both training and validation datasets. The primary causes of underfitting include an overly simplistic model architecture, inadequate training time, or improper selection of hyperparameters such as the learning rate, batch size, or optimisation algorithm. To address underfitting, one can increase the model complexity by adding more layers or neurons, allowing the model to learn more expressive features. Training the model for a longer duration or with more advanced optimisation algorithms can also help improve learning. Tuning the hyperparameters, such as using grid search, can assist in finding the optimal combination that yields the best performance.

The **vanishing gradient** problem is a common challenge in deep learning, particularly in deep feedforward neural networks. As the gradients are backpropagated through the layers, they can become exceedingly small, resulting in minimal weight updates and slow or stalled learning. Activation functions, such as the sigmoid or tanh, which have saturating regions producing small gradients for large input values, are especially prone to vanishing gradients. To mitigate this issue, alternative activation functions, such as ReLU, Leaky ReLU or ELU, can be used, as they provide non-saturating non-linear transformations and maintain larger gradients for learning. Additionally, employing techniques like batch normalization[31]–[33], which normalises the input distribution at each layer, can help maintain a healthy gradient flow. Proper weight initialization schemes, such as Glorot[34] or He[35] initialization, can also aid in avoiding vanishing gradients by providing an appropriate starting point for the training process. Finally, introducing skip connections, as seen in residual networks[36],[37], can allow gradients to bypass certain layers, preserving the gradient magnitude and enabling more efficient learning.

Exploding gradients pose a significant challenge during the training of neural networks, especially in recurrent architectures dealing with long sequences. When gradients become excessively large during backpropagation, they can cause weight updates to be highly unstable, leading to oscillations or divergence in the training process. In some cases, exploding gradients can also result in numerical instability, causing the learning algorithm to fail completely. To tackle the exploding gradient issue, one widely adopted approach is gradient clipping, which involves limiting the magnitude of the gradients to a predefined threshold before updating the weights. This prevents excessively large weight updates and maintains stability during training. Proper weight initialization can also play a role in controlling the magnitude of gradients, as it can help prevent extreme weight values and gradient accumulation in the early stages of training. Regularisation techniques, such as weight decay or dropout, can further contribute to reducing the risk of exploding gradients by encouraging smoother and more stable weight updates throughout the training process.

Regularisation techniques play a crucial role in preventing overfitting and improving generalisation capabilities. By adding constraints or penalties to the model's complexity, regularisation encourages the learning algorithm to find simpler and more robust solutions, striking a balance between fitting the training data and maintaining adaptability to unseen data. Prevalent regularisation methods include:

- **Lasso (ℓ_1) Regularisation** is a technique used to prevent overfitting by adding a penalty term to the loss function based on the absolute values of the model weights. This penalty term encourages the model to learn a sparse set of features, effectively setting some of the less important weights to zero. The modified loss function with ℓ_1 regularisation can be written as:

$$\mathcal{L}_{\ell_1} = \mathcal{L} + \lambda \sum_{i=1}^{L} \left\| \mathbf{W}^{(i)} \right\|_1 + \lambda \sum_{i=1}^{L} \left\| \mathbf{b}^{(i)} \right\|_1 , \tag{1.30}$$

$$\left\| \mathbf{W}^{(i)} \right\|_1 = \frac{1}{R_i \cdot C_i} \sum_{j=1}^{R_i} \sum_{p=1}^{C_i} \left| w_{j,p}^{(i)} \right| \quad \text{and} \quad \left\| \mathbf{b}^{(i)} \right\|_1 = \frac{1}{R_i} \sum_{j=1}^{R_i} \left| b_j^{(i)} \right| , \tag{1.31}$$

  where λ is the regularisation coefficient hyperparameter and $R_i \times C_i$ is the size of the weight matrix $\mathbf{W}^{(i)}$. By adjusting the value of λ, one can balance the trade-off between fitting the data and enforcing sparsity in the model weights. Lasso regularisation can be particularly beneficial in situations where the input features are highly correlated or when there are more features than training examples, as it promotes feature selection and simplifies the model, thus improving generalisation.

- **Ridge (ℓ_2) Regularisation** or weight decay is another technique used to prevent overfitting by adding a penalty term to the loss function based on the squared values of the model weights. Unlike ℓ_1 regularisation, which enforces sparsity, ℓ_2 regularisation encourages the model to distribute the weights more evenly across the features, resulting in smaller weight values and a smoother model. The modified loss function with ℓ_2 regularisation can be written as:

$$\mathcal{L}_{\ell_2} = \mathcal{L} + \lambda \sum_{i=1}^{L} \left\| \mathbf{W}^{(i)} \right\|_2 + \lambda \sum_{i=1}^{L} \left\| \mathbf{b}^{(i)} \right\|_2 , \tag{1.32}$$

$$\left\| \mathbf{W}^{(i)} \right\|_2 = \frac{1}{R_i \cdot C_i} \sum_{j=1}^{R_i} \sum_{p=1}^{C_i} \left(w_{j,p}^{(i)} \right)^2 \quad \text{and} \quad \left\| \mathbf{b}^{(i)} \right\|_2 = \frac{1}{R_i} \sum_{j=1}^{R_i} \left(b_j^{(i)} \right)^2 . \tag{1.33}$$

  By varying the value of λ, one can balance the trade-off between fitting the data and enforcing smoothness in the model weights. Ridge regularisation is particularly effective in situations where there is a risk of multicollinearity or when the model is prone to overfitting, as it penalises large weights and helps to reduce the complexity of the model, thus enhancing its generalisation capabilities.

- **Dropout** is a powerful regularisation technique introduced to address overfitting in deep neural networks. Dropout works by randomly deactivating a subset of neurons (along

with their connections) during each training iteration, with the probability of deactivation controlled by a hyperparameter, usually denoted as the dropout rate. By doing so, dropout effectively creates an ensemble of smaller, varied sub-networks, each of which learns different aspects of the data. Meanwhile, during the testing or inference phase, all neurons are active.

Dropout provides several key advantages for training deep learning models. First, it prevents the model from relying too heavily on individual neurons or features, thus promoting better generalisation and robustness. Second, by creating an ensemble of sub-networks, dropout implicitly performs model averaging, which has been shown to improve generalisation performance in various tasks. Third, dropout serves as a computationally efficient form of regularisation, as it requires minimal additional computational resources during training and does not introduce any additional parameters to the model.

One potential drawback of dropout is that it may increase the training time, as the model must learn to compensate for the random deactivation of neurons. However, this additional training time is often offset by the improved generalisation performance and robustness that dropout brings to the model, making it a widely adopted and valuable regularisation technique in deep learning.

- **Early stopping** involves monitoring the model's performance on a validation dataset during the training process and terminating the training when the validation performance ceases to improve or starts to degrade. By stopping the training early, one can prevent the model from overfitting the training data and fine-tuning to the noise present in the training samples, thus preserving the model's generalisation capabilities. Early stopping effectively balances the trade-off between underfitting and overfitting by finding the optimal point at which the model performs well on both the training and validation datasets. In practice, early stopping is often combined with other regularisation techniques.

In the domain of neural networks, various **performance metrics** have been developed to evaluate the effectiveness of models in classification tasks, such as accuracy, precision, recall and F1 score. These metrics provide insights into the model's ability to correctly predict class labels and discern between different classes. However, our current focus is on a regression problem, specifically, the task of inpainting degraded colour images, which requires estimating continuous values rather than predicting discrete class labels. Therefore, in the following section, we will shift our attention to Convolutional Neural Networks (CNNs), a powerful class of deep learning models particularly suited for handling image data.

1.3 Convolutional Neural Networks

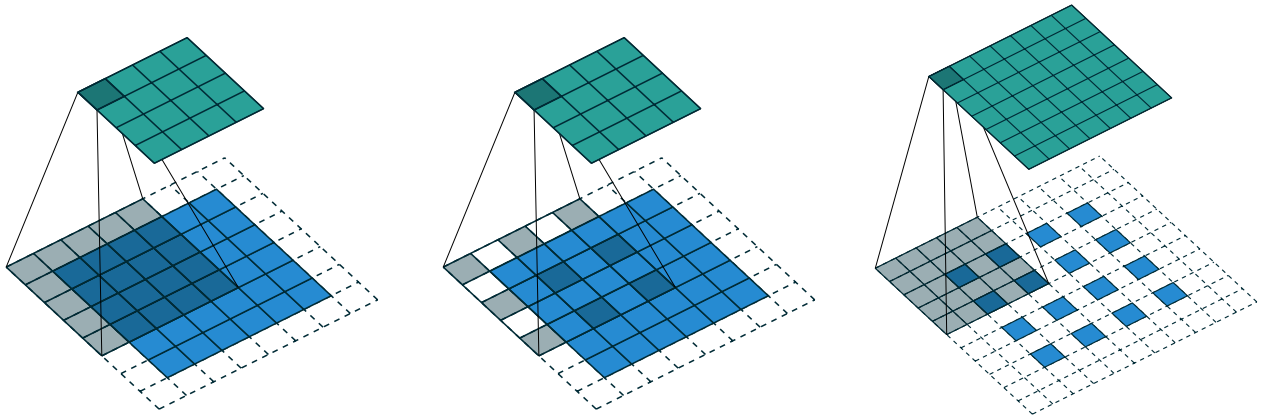
Convolutional Neural Networks (CNNs) are a class of deep learning models that are specifically designed to process grid-like data structures, such as images, by capturing and exploiting the spatial hierarchies and local patterns intrinsic in such data. CNNs have shown remarkable success in various applications, including image classification, object detection, semantic segmentation and image inpainting, among others. The architecture of a Convolutional Neural Network is characterised by a series of layers, each responsible for learning different levels of abstraction from the input data. The layers typically include convolutional layers, pooling layers and fully connected layers, arranged in a sequential manner.

Convolutional layers are the core building blocks of CNNs, designed to detect local patterns or features in input data, such as images. These layers perform the operation of convolution, which involves applying a set of trainable filters or convolutional kernels to the input data in a sliding window fashion. Each filter in a convolutional layer is typically a small matrix with fixed dimensions, commonly equal to each other (e.g. 3×3, 5×5), and is applied to a local receptive field of the input. By sliding the filter across the input spatially in both width and height dimensions, Hadamard product (i.e. element-wise multiplication) and summation are performed between the filter and the receptive field. The resulting sum is stored in a corresponding location in the output feature map. This process is repeated for all possible positions of the filter in the input, generating the complete output feature map for that specific filter, as can be seen in Figure 1.6a. Convolutional layers typically contain multiple filters, enabling them to learn a variety of features or patterns from the input data. Each filter produces its own output feature map, which, when combined, forms a multi-channel output, encoding different aspects of the input data's spatial and structural information.

Several key properties, such as padding and stride, can be adjusted to control the behaviour of the convolutional layer. **Padding** refers to the process of augmenting the input data's borders with additional values, which allows for control over the output feature map's spatial dimensions and better preservation of information at the edges. The values added during padding can be determined by various methods (e.g. adding zeros, repeating or reflecting existing values), depending on the specific requirements of the task at hand. **Stride** determines the step size by which the filter moves across the input, with larger strides resulting in smaller output feature maps.

Dilated convolutional layers, also known as atrous convolutions, are a variant of standard convolutional layers that incorporate a dilation factor to control the spacing between the filter's elements. This modification allows the receptive field to be larger without increasing the number of parameters in the filter, enabling the network to capture and integrate information from a wider spatial context without a significant increase in computational complexity, as observed in Figure 1.6b. In a dilated convolutional layer, each element of the

filter is separated by a gap of size $d-1$, where $d \in \mathbb{N}^*$ is the dilation factor. A dilation factor of 1 corresponds to a standard convolution, while a dilation factor greater than 1 results in an expanded receptive field. Dilated convolutions are particularly useful in tasks that require the integration of information from larger spatial contexts, such as semantic segmentation or image inpainting. They enable the network to capture multi-scale information without the need for pooling layers or multiple parallel convolutional pathways. Moreover, dilated convolutions can be employed in conjunction with standard convolutions in deep architectures, allowing the network to learn features at different scales and resolutions. However, one potential drawback of dilated convolutions is the increase in computational complexity for larger dilation factors, as the expanded receptive field may require additional memory and processing resources.



(a) *Normal convolution* with a 5×5 kernel at a stride of 1 on an 1-padded matrix

(b) *Dilated convolution* with a 2-dilated 3×3 kernel at a stride of 1 on an 1-padded matrix

(c) *Transposed convolution* with a 5x5 kernel at a stride of 2 on an 2-padded matrix

Figure 1.6: Comparison of convolutional operations[38]

Now let us consider the case of multiple-layer images as the input $\mathcal{I} \in \mathbb{R}^{H_{\text{in}} \times W_{\text{in}} \times L_{\text{in}}}$ for a convolutional layer, where H_{in}, W_{in} and L_{in} represent the height, width and number of layers, respectively. For a 2D convolution (i.e. computed along the first two dimensions), we define the kernel $\mathcal{K} \in \mathbb{R}^{H_{\text{ker}} \times W_{\text{ker}} \times L_{\text{ker}}}$, where H_{ker}, W_{ker} and L_{ker} are its width, height and number of filters, respectively, and in most cases $H_{\text{ker}} \ll H_{\text{in}}$ and $W_{\text{ker}} \ll W_{\text{in}}$. Likewise, let the pairs of constants (s_r, s_c) and (d_r, d_c) represent the strides and dilation factors for the two axes or directions, respectively. Therefore, the general convolution operation can be represented mathematically as follows:

$$\mathcal{O}(r,c) = (\mathcal{K} * \mathcal{I})(r,c) = \sum_{i=-\left\lfloor \frac{H_{\text{ker}}}{2} \right\rfloor}^{\left\lfloor \frac{H_{\text{ker}}}{2} \right\rfloor} \sum_{j=-\left\lfloor \frac{W_{\text{ker}}}{2} \right\rfloor}^{\left\lfloor \frac{W_{\text{ker}}}{2} \right\rfloor} \sum_{k=1}^{L_{\text{in}}} \mathcal{K}(i,j,k) \cdot \mathcal{I}(s_r \cdot r + d_r \cdot i, s_c \cdot c + d_c \cdot j, k)\,, \quad (1.34)$$

$$\forall r \in \left\{1, 2, \dots, \left\lceil \frac{H_{\text{in}}}{s_r} \right\rceil \right\} \quad \text{and} \quad \forall c \in \left\{1, 2, \dots, \left\lceil \frac{W_{\text{in}}}{s_c} \right\rceil \right\}. \quad (1.35)$$

On top of that, the aforementioned padding, defined by the pair of constants (p_r, p_c) for the two axes, is employed to supplement the image borders with additional values, ensuring that the convolutional filter can be applied on the pixels situated along the image's boundary. Consequently, after the convolution has been applied, the output $\mathcal{O} \in \mathbb{R}^{H_{\text{out}} \times W_{\text{out}} \times L_{\text{out}}}$ contains a number of $L_{\text{out}} = L_{\text{ker}}$ layers and the shape $H_{\text{out}} \times W_{\text{out}}$ is computed by the ensuing equations:

$$H_{\text{out}} = \left\lfloor \frac{H_{\text{in}} + 2 \cdot p_r - (H_{\text{ker}} - 1) \cdot d_r - 1}{s_r} \right\rfloor + 1 \,, \tag{1.36}$$

$$W_{\text{out}} = \left\lfloor \frac{W_{\text{in}} + 2 \cdot p_c - (W_{\text{ker}} - 1) \cdot d_c - 1}{s_c} \right\rfloor + 1 \,. \tag{1.37}$$

Transposed convolution or **fractionally-strided convolution** is a crucial operation in deep learning for up-sampling or expanding feature maps. The term *deconvolution* is also used to refer to this operation but is somewhat of a misnomer because is not a true deconvolution in the mathematical sense, as it does not reverse the process of convolution. This operation essentially reverses the forward and backward passes of a standard convolution process to reconstruct high-resolution feature maps from lower-resolution representations and aids in recovering spatial information lost during convolution and pooling operations, as depicted in Figure 1.6c[39]. One of the primary benefits of transposed convolution is spatial expansion. This characteristic enables the expansion of feature maps, which is beneficial for tasks that require high-resolution outputs. Additionally, transposed convolution layers can restore spatial information lost during the compression process in convolutional neural networks, resulting in more accurate and detailed output images or feature maps.

Gated convolutional layers are a variant of standard convolutional layers that incorporate gating mechanisms to control the flow of information through the network. These gating mechanisms enable the model to selectively learn and propagate relevant features, improving the network's ability to capture complex dependencies and context in the input data. A gated convolutional layer typically consists of two parallel convolution operations: one for the standard feature map generation and another for learning the gating mask. The feature map is computed using any activation function Φ applied after the standard convolution operation, while the gating mask is obtained by applying a sigmoid activation function σ to a separate convolution operation[40], expressed mathematically as follows:

$$\mathcal{O} = \Phi\left(\mathcal{K} * \mathcal{I}\right) \odot \sigma\left(\mathcal{G} * \mathcal{I}\right) \,, \tag{1.38}$$

where $\mathcal{K}$ is any convolution kernel and $\mathcal{G}$ is a different gating kernel, while $\odot$ denotes the Hadamard product.

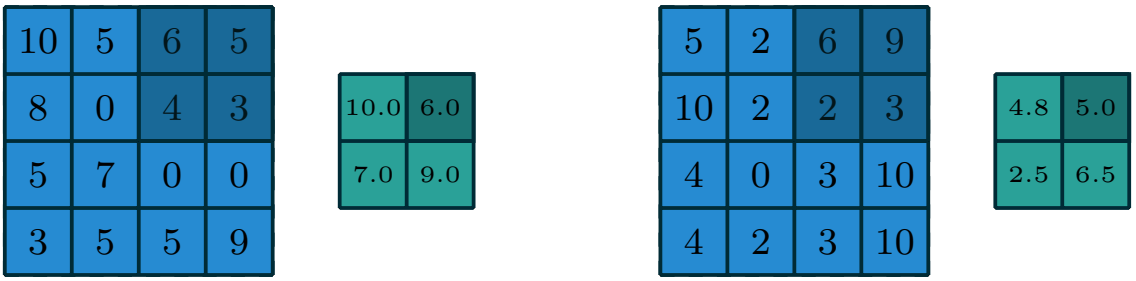
Pooling layers are an essential component of CNNs, playing a crucial role in reducing the spatial dimensions of the feature maps, which helps improve computational efficiency, control model complexity and enhance the model's ability to capture translation-invariant

features. Pooling layers perform a downsampling operation, aggregating information within (non-)overlapping local regions of the input feature map. The most common types of pooling operations are maximum pooling and average pooling and are illustrated in Figure 1.7. **Maximum pooling** selects the max value within the local region, while **average pooling** computes the mean value, as outlined by the following mathematical formulae:

$$\mathcal{P}_{\max}(r,c) = \max_{0\le i<H_{\mathrm{p}}} \max_{0\le j<W_{\mathrm{p}}} \mathcal{I}(s_r \cdot r + i, s_c \cdot c + j)\,, \tag{1.39}$$

$$\mathcal{P}_{\mathrm{avg}}(r,c) = \frac{1}{H_{\mathrm{p}} \cdot W_{\mathrm{p}}} \sum_{i=0}^{H_{\mathrm{p}}-1} \sum_{j=0}^{W_{\mathrm{p}}-1} \mathcal{I}(s_r \cdot r + i, s_c \cdot c + j)\,, \tag{1.40}$$

where $H_{\mathrm{p}} \times W_{\mathrm{p}}$ is the size of the filter and the pair (r, c) takes values such as previously described in equation (1.35). By summarizing the local information and reducing the spatial dimensions of the feature maps, pooling layers contribute to making the model more robust to small translations or deformations in the input data. This reduction in spatial dimensions also leads to a decrease in the number of parameters in succeeding layers, resulting in a more computationally efficient model with a lower risk of overfitting. However, one potential drawback of pooling layers is the loss of some spatial information due to the downsampling process.



(a) *Maximum pooling* with a 2×2 sliding window at a stride of 2

(b) *Average pooling* with a 2×2 sliding window at a stride of 2

Figure 1.7: Comparison of pooling operations[38]

Convolutional Neural Networks exhibit a range of unique properties that enable them to excel in various applications, particularly those related to image and signal processing. These key attributes comprise local connectivity, parameter sharing and spatial or shift-invariance. The **local connectivity** property ensures that the network's neurons are connected to specific, limited regions within the input data. This approach reduces the total number of parameters and allows the network to extract local features more effectively. Furthermore, **parameter sharing** is a crucial characteristic of CNNs, as it involves using the same set

of weights across multiple spatial locations in the input data. This technique not only contributes to a more efficient training process but also reduces the risk of overfitting. Lastly, the **spatial (shift) invariance** property instils the network with the ability to maintain robustness against translations in the input data. Consequently, it can recognise and generalise patterns irrespective of their position within the input space, thereby enhancing their performance in tasks such as object recognition and classification.

1.4 Image Inpainting

Image inpainting, an essential and widely researched topic in the field of computer vision and image processing, refers to the technique of reconstructing missing or damaged regions within an image to preserve visual coherence and aesthetics. The primary goal of image inpainting is to restore an image in a manner that renders the filled regions indistinguishable from the original image, thus maintaining its structural, textural and semantic integrity. Inpainting techniques have numerous practical applications, such as restoring old or damaged photographs, removing unwanted objects or artefacts and filling in occlusions in panoramic images. The origins of image inpainting can be traced back to the domain of art conservation, wherein skilled artists and restorers manually retouch damaged paintings or photographs. In the digital era, researchers have developed various algorithmic approaches to automate the process of image inpainting, broadly categorized into two groups: non-learning-based methods and learning-based methods.

Non-learning-based methods, also known as traditional or deterministic methods, rely on mathematical models and image priors to estimate the missing information. These techniques include diffusion-based methods and patch-based methods. **Diffusion-based methods** propagate information from the surrounding area into the missing region, primarily preserving the smoothness and structure of the image. In contrast, **patch-based methods** search for similar regions in the known image area and use them to synthesize the missing parts. Although these methods can achieve satisfactory results in certain scenarios, they may struggle with complex textures, fine details, or large missing regions.

With the advent of deep learning, **learning-based methods** have emerged as a powerful alternative for image inpainting. These methods leverage the representation learning capabilities of neural networks to model complex patterns and relationships within images. Techniques such as Convolutional Neural Networks (CNNs), Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs) have shown remarkable success in various inpainting tasks[40]–[43]. CNNs, for instance, can be trained to predict missing pixels based on the surrounding context, while GANs and VAEs can generate coherent and plausible image patches by learning the underlying distribution of the image data.

In order to develop a more comprehensive understanding of the fundamental **technical principles** underlying image inpainting, it is essential to delve into several critical aspects

that encompass the field's primary methodologies, techniques and approaches:

- **Local and global consistency** – A critical aspect is to maintain both local and global consistency in the reconstructed image. Local consistency refers to the coherence of textures, colours and patterns in the immediate vicinity of the missing region, while global consistency relates to the preservation of the image's overall structure and semantics. Achieving this balance is a challenging task, as it requires considering both the immediate surroundings and the broader context of the image.

- **Non-local self-similarity** – Natural images often exhibit a high degree of self-similarity and repetitive patterns. Non-local self-similarity refers to the observation that similar patches can be found in disparate regions of the image. Exploiting this property can facilitate the search for suitable candidates to fill the missing regions, particularly when using patch-based inpainting methods.

- **Ill-posed nature** – Image inpainting is inherently an ill-posed problem, as there are often multiple plausible solutions for a given missing region. The absence of a unique solution complicates the task, requiring the development of sophisticated algorithms capable of generating visually coherent and semantically consistent results.

- **Handling diverse image characteristics** – Images can possess a wide variety of characteristics, including different structures, textures and semantics. Developing inpainting algorithms that can effectively handle such diversity is challenging, as it requires accounting for the specific properties of each image and adapting the inpainting process accordingly.

In the following discussion, we will shed light on some of the prominent **challenges** that arise in the context of image inpainting, highlighting the complexities and obstacles that must be addressed to achieve effective and visually coherent results:

- **Large missing regions** – Inpainting algorithms may struggle with situations where the missing region is extensive, as the available information from the known region might be insufficient to accurately reconstruct the occluded area.

- **Insufficient self-similarity** – Some images may not exhibit sufficient self-similarity or repetitive patterns, making it challenging for patch-based methods to find suitable candidates for filling the missing regions.

- **Complex textures and structures** – Inpainting algorithms can face difficulties when dealing with complex textures and structures, as accurately reconstructing these features requires a high level of fidelity and coherence.

- **Subjectivity in visual perception** – The notion of visual similarity is often subjective, which complicates the task of evaluating the success of inpainting algorithms. Aiming to mimic human visual perception, inpainting methods must account for the nuances

of human perception and generate results that are not only visually coherent but also semantically consistent.

Although substantial advancements have been made in recent years, image inpainting has always been a demanding endeavour owing to its ill-posed nature and the heterogeneous properties of images, including aspects such as texture, structure and semantics. Acknowledging the inherent challenges of image inpainting, three notable non-learning-based algorithms that have made significant contributions to the domain will be concisely presented.

Navier-Stokes

The **Navier-Stokes**[44] algorithm, also known as fluid-flow-based inpainting, is an image inpainting technique introduced in 2001. It is inspired by the principles of fluid dynamics and aims to reconstruct missing or corrupted regions in an image by propagating structural and textural information from the surrounding known areas. The algorithm operates by calculating the isophote (i.e. curves on an image that connect points of equal intensity or brightness) and Laplacian fields (i.e. spatial distribution of the second derivatives of an image, capturing its local curvature and edges) for the known regions of the image, which provide information about the structure and texture. It then uses the Navier-Stokes equations, which describe the motion of fluid substances, to model the propagation of these fields from the known region into the missing region.

The fundamental approach is heuristic in nature. Initially, the process traverses along the edges from known to unknown regions, as edges are intended to be continuous. It maintains the isophotes while aligning gradient vectors at the boundary of the inpainting region. Afterwards, colour is introduced to minimize the variance within the given area. The inpainting process is iterative, with each iteration refining the reconstructed region to minimize discrepancies between the known and missing areas. The algorithm prioritizes the completion of image structure and edge information, which is achieved through the use of an edge-preserving smoothing technique, namely anisotropic diffusion.

When employing the Navier-Stokes inpainting algorithm, several challenges may arise, affecting its performance and output quality. One notable issue is the algorithm's sensitivity to noise, which can result in the amplification of noise artefacts during the inpainting process. Furthermore, the method may struggle to handle complex textures and structures, often leading to less satisfactory reconstructions in such cases. Another challenge is the computational efficiency of the Navier-Stokes algorithm, as its iterative nature and reliance on solving partial differential equations can render it less suitable for large-scale or real-time inpainting applications.

Telea

The **Telea**[45] algorithm, developed by Alexandru Telea in 2004, is an image inpainting technique based on the fast-marching method. It aims to reconstruct missing or corrupted regions in an image by propagating information from the surrounding known areas. The algorithm prioritizes the completion of image structure and edge information. This method is founded upon the Fast Marching Method (i.e. numerical algorithm for solving the Eikonal equation, nonlinear partial differential equation describing the propagation of wavefronts, used for efficiently computing shortest paths or wavefront propagation).

Given a region within an image to be inpainted, the algorithm commences at the region's boundary and progresses inwards, gradually filling the boundary first. It considers a small neighbourhood surrounding the pixel to be inpainted. This pixel is then substituted with a normalized weighted sum of all known pixels in the neighbourhood. The selection of weights is crucial, with the higher weight assigned to pixels closer to the point, proximate to the boundary's normal, and located on the boundary contours. Upon inpainting a pixel, the algorithm proceeds to the next nearest pixel using the Fast Marching Method. FMM ensures that pixels close to the known ones are inpainted first, emulating a manual heuristic operation.

When using the Telea inpainting algorithm, certain challenges may arise that impact its performance and the quality of the generated results. One key limitation stems from its diffusion-based nature, which can lead to over-smoothing and the loss of intricate details in the inpainted regions. Moreover, the algorithm's local connectivity assumptions may hinder its ability to accurately propagate coherent structures and textures over larger distances or in cases where the missing region spans a significant portion of the image. Additionally, the selection of parameters, such as the neighbourhood's radius, can substantially influence the outcome, necessitating careful tuning to achieve optimal results.

PatchMatch

The **PatchMatch**[46] algorithm, proposed in 2009, is an image inpainting technique that employs patch-based methods to reconstruct missing or corrupted regions in an image. The algorithm efficiently searches for approximate nearest-neighbour matches between patches in the image, enabling the synthesis of visually coherent and plausible results.

The algorithm operates iteratively, commencing with the random initialization of an offset field, followed by a random search to identify better-matching patches. Subsequently, the best matching patches are propagated along a scanline order, enabling the efficient transfer of information across the image. Lastly, after a number of an empirically determined number of interactions, the best matching patches are blended to inpaint the missing region, producing a visually consistent reconstruction. The PatchMatch algorithm has been pivotal in the development of patch-based inpainting techniques, as it offers an efficient means of searching and propagating information while generating visually coherent outcomes.

In the context of employing the PatchMatch inpainting algorithm, various challenges may emerge, affecting its efficacy and the quality of the resulting inpainted images. One primary concern is the algorithm's computational complexity, which may limit its applicability in large-scale or real-time applications. Additionally, the tuning of parameters, such as patch size and search radius, can significantly impact the output quality, requiring careful adjustment to achieve desired results. Furthermore, the PatchMatch algorithm may encounter difficulties in handling artefacts and discontinuities, leading to the generation of visually inconsistent or implausible reconstructions.

Chapter 2
State of the Art

Image inpainting, a key research area in the field of image processing, focuses on the reconstruction of missing or deteriorated parts of images while preserving visual coherence and contextual consistency. Over the years, numerous methods have been proposed to address this complex task, evolving from traditional patch-based techniques to more sophisticated deep-learning approaches. In the context of this chapter, we will explore the most prominent and recent developments in image inpainting, with a particular emphasis on techniques employing Convolutional Neural Networks and Generative Adversarial Networks. These **state-of-the-art** methods have significantly advanced the field, demonstrating their ability to generate plausible and visually appealing content while overcoming limitations faced by earlier methods. The outcomes derived from the application of state-of-the-art methods are demonstrably represented in figures 2.2 and 2.3, wherein the efficacy of these techniques in image inpainting is visually substantiated.

In the pursuit of advancing image inpainting techniques, various **innovative concepts** have emerged, each contributing unique strengths to the reconstruction process. These approaches encompass a diverse range of strategies, addressing challenges such as edge preservation, texture consistency and computational efficiency:

- **Partial convolutions**[47] leverage a specialized convolution operation that allows for the integration of masked and unmasked regions within an image. By utilizing a binary mask in tandem with the input image, partial convolutions adapt the convolution operation to account for the presence of missing or corrupted pixels. This is achieved by normalizing the convolution output with the sum of the mask values within the convolution window, effectively excluding missing pixels from the calculation.

- **Gated convolutions**[40] introduce a gating mechanism to control the flow of information within convolutional neural networks. This mechanism operates by computing a gating signal, typically using a sigmoid activation function, which is then element-wise multiplied with the output of a standard convolution layer. The gating signal modulates the degree to which each feature map influences the final output, effectively learning to weigh the importance of different regions within the image.

- **Fourier convolutions**[48] employ the Fourier domain to perform convolutions more efficiently. By leveraging the Convolution Theorem, which states that the convolution of

two functions in the spatial domain is equivalent to the element-wise multiplication of their Fourier transforms, Fourier convolutions facilitate faster computation. Implementing Fourier convolutions involves changing the input image and filters into the Fourier domain, performing element-wise multiplication and subsequently applying the inverse Fourier transform to obtain the final result.

- Using a **coarse inpainting network followed** by a **refinement network**[40],[43],[49] represents a two-stage approach to image inpainting, which aims to balance the efficiency and effectiveness of the reconstruction process. In the first stage, the coarse inpainting network generates a preliminary, low-resolution reconstruction of the missing regions, focusing on capturing the overall structure and global context. In the second stage, the refinement network takes the output from the coarse network and enhances it by adding finer details, textures and colours, producing a high-resolution and visually coherent final result. This hierarchical approach enables the model to progressively refine the inpainted regions, yielding more accurate and visually pleasing reconstructions. One such architecture can be explored in Figure 2.1

- **Local and global refinement**[49] aims to address the challenge of reconstructing missing regions by integrating both local and global context information. This method typically employs a multi-scale or multi-branch architecture, wherein one branch focuses on capturing fine-grained local features, such as textures and edges, while another branch is dedicated to extracting global structures and contextual relationships. The outputs of these branches are combined and refined to produce a final inpainted result that preserves both local details and global coherence.

- **Contextual attention mechanisms**[43],[50] enable models to selectively focus on relevant regions in the input image by dynamically weighing their contributions during reconstruction. These mechanisms operate by learning spatial correlations between the missing region and its surrounding context, thereby facilitating the transfer of appropriate textures and structures from the context to the missing area.

- **EdgeConnect**[51] focuses on preserving edge information in the image reconstruction process. This method employs a two-stage process, consisting of an edge generator and an image completion network. The edge generator is responsible for predicting the structure of the missing regions by extending the edges found in the surrounding context. Subsequently, the image completion network utilizes the generated edge map, along with the original degraded image, to fill in the missing regions with appropriate colours and textures.

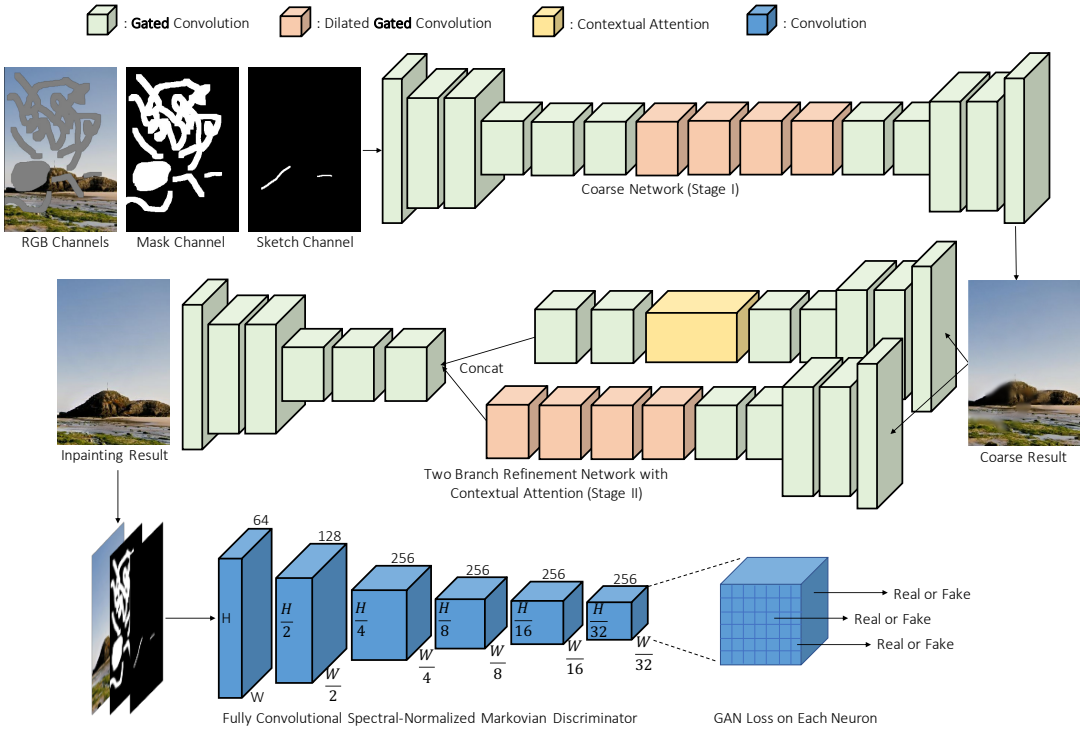


Figure 2.1: Coarse inpainting network followed by a refinement network architecture[40]

Figure 2.2: Inpainting example using the Free-Form architecture[40]

Figure 2.3: Inpainting example using the Fourier Convolutions architecture[48]

Chapter 3
Preliminary Work

3.1 Datasets Description

In this segment of the discussion, an in-depth exploration of the two datasets, integral to the development of the project, will be undertaken. The forthcoming discourse will clarify the specifics, characteristics and distinctive aspects of each dataset, delivering a comprehensive understanding of their role within the scope of the project.

3.1.1 CIFAR-10

The **CIFAR-10**[1] (Canadian Institute for Advanced Research) dataset is a widely used dataset for machine learning and computer vision research. It contains a total of 60,000 images divided into 10 distinct classes: aeroplanes, cars, birds, cats, deer, dogs, frogs, horses, ships and trucks, as depicted in Figure 3.1. Each class contains 6,000 images and the dataset is further divided into **50,000 training images** and **10,000 testing images**. The images in the CIFAR-10 dataset are colour and relatively small, each with a dimension of **32×32** pixels[52]. The dataset, a ubiquitous tool for machine learning and computer vision initiatives, presents unique characteristics that can be both advantageous and disadvantageous in the context of inpainting degraded colour images:

- **Variety of classes** – The CIFAR-10 dataset's broad variety of categories provides a balanced and diverse training platform, enabling a model to learn intricate features and distinctions across different classes.
- **Established benchmark** – Owing to its extensive use in academic research, it serves as an excellent benchmark for evaluating the effectiveness of different algorithms. Numerous state-of-the-art models have been tested on this dataset and their performances have been recorded, facilitating a comparative study.
- **Size and complexity** – Despite the relatively small size of the images, they are complex enough to pose a substantial challenge, thus testing the robustness of machine learning models.

[1] cs.toronto.edu/~kriz/cifar.html

- **Low resolution** – The primary limitation of the CIFAR-10 dataset is the low resolution of the images. While this reduces computational requirements, it could potentially hamper the ability to inpaint high-resolution images as the model may not learn sufficient high-frequency details.

- **Limited domain** – Despite the variety of classes, the CIFAR-10 dataset is relatively limited in its representation of the real world. The classes are constrained and there's an absence of images depicting complex scenes, variations in lighting or images captured from varying perspectives. Hence, a model might not perform well on more diverse and complex images.

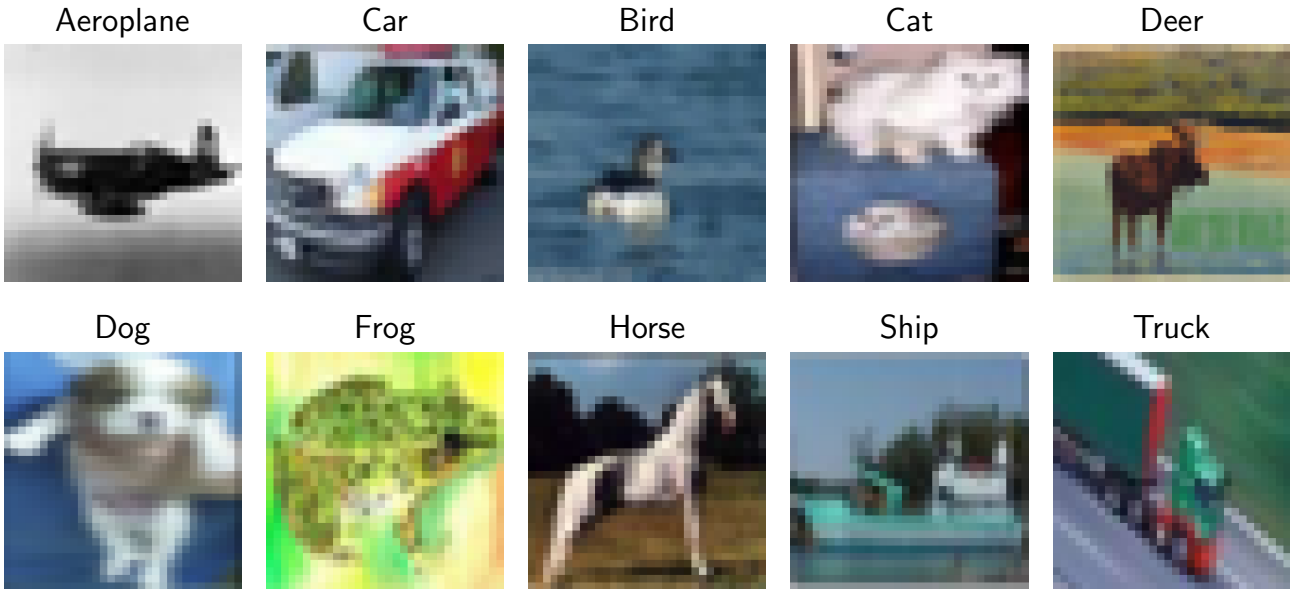


Figure 3.1: Image samples from the CIFAR-10 dataset

In the search for addressing the challenge of image inpainting, a notable constraint is encountered when leveraging the standard CIFAR-10 dataset: the diminutive image size of 32×32 pixels. The network architectures that are going to be presented in the subsequent chapter involve deep convolutional networks, which are characterized by a progressive reduction of spatial dimensions as we move deeper into the network. This inherent feature of such architectures is incompatible with the small size of standard CIFAR-10 images, leading to difficulties in the meaningful extraction of higher-level features due to rapid downscaling. To circumvent this limitation and ensure a sustainable depth of learning, we opt for an upscaled version of the CIFAR-10 dataset, presenting images at a resolution of **64×64** pixels[2], as illustrated in Figure 3.2. This approach retains the diverse content of the original dataset while accommodating the requirements of our selected deep-learning models. Consequently, the larger images facilitate the capture of more detailed features and enable a more robust learning process, improving the potential performance of the inpainting task at hand.

[2] `kaggle.com/datasets/joaopauloschuler/cifar10-64x64-resized-via-cai-super-resolution`

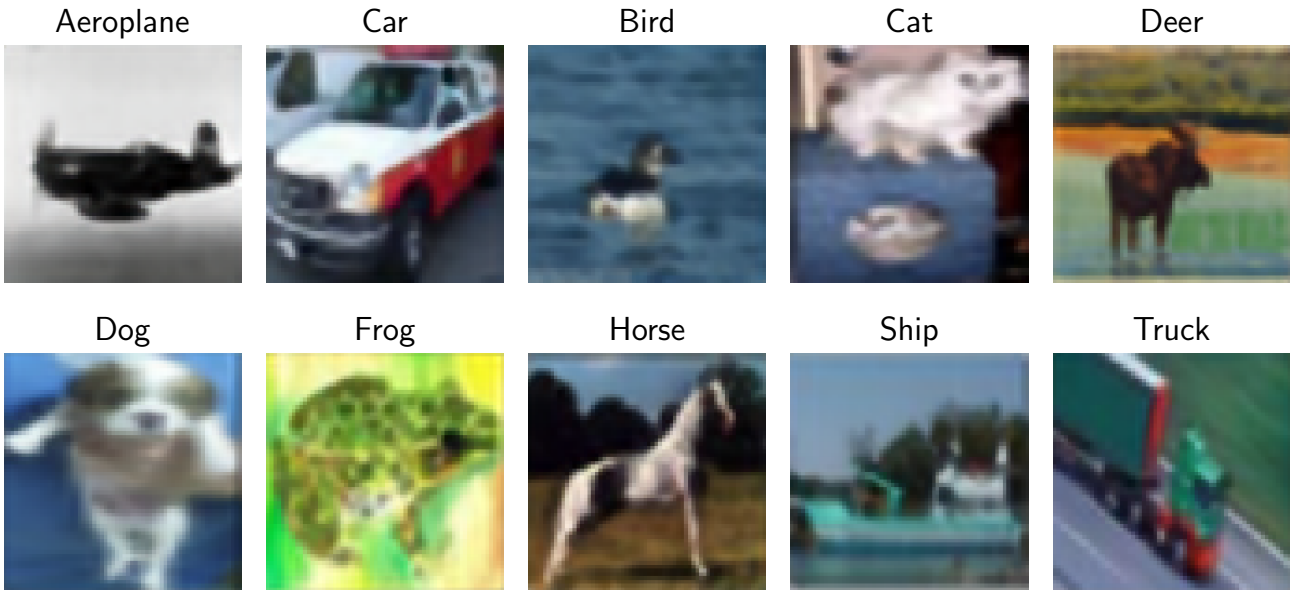


Figure 3.2: Upscaled of the same image samples from the CIFAR-10 dataset

The selection and usage of the CIFAR-10 dataset and its upscaled version in the incipient phases of this project proved to be an instrumental decision. With a principal goal of developing and refining the image inpainting architectures, the CIFAR-10 dataset offered a convenient yet diverse array of images for initial testing and training. The modest size of the dataset, coupled with the relatively higher resolution of the upscaled version, facilitated an environment favourable to rapid experimentation and iterative improvement of the models. This combination enabled the project to strike a crucial balance: extracting meaningful, high-level features critical for the inpainting task while ensuring that model training and testing could be executed within reasonable time frames, a process that would have been substantially more challenging with larger and more complex datasets.

3.1.2 Common Objects in Context

The **Common Objects in Context**[3] (COCO) database is a large-scale, richly annotated dataset designed to spur advancements in object detection, segmentation, person keypoints detection, stuff segmentation and caption generation. The dataset comprises 330,000 colour images of varying dimensions, encompassing 1.5 million object instances across 90 categories[53], as represented in Figure 3.3. In the context of degraded colour image inpainting, the COCO dataset brings its own strengths and potential weaknesses:

- **High diversity and complexity of scenes** – COCO contains images with complex everyday scenes, often with multiple objects per image, providing a diverse and challenging learning environment. But this complexity is beneficial for developing a more robust inpainting model that can handle real-world scenarios.

[3] `cocodataset.org`

- **Large scale** – The sheer number of images and object instances is an advantage for training deep learning models, which often benefit from larger datasets.
- **High resolution** – Images in the COCO dataset are of higher resolution compared to CIFAR-10. This provides more detailed visual information, which is crucial for a task such as image inpainting.
- **Processing power and preparation time** – The size and complexity of the COCO dataset require more computational resources to process, which slows down model training. Also, given the dataset's richness and diversity, preprocessing and setting up the data for this task is more time-consuming compared to the simpler CIFAR-10 dataset.

Figure 3.3: Image samples from the COCO dataset

Following the initial development and refinement of the models using the CIFAR-10 dataset, the transition was made to the Common Objects in Context dataset for the crucial phases of final model training, testing and evaluation. The choice of the dataset served as a strategic operation designed to elevate the inpainting system's performance in real-world scenarios. COCO's complex, high-resolution images and various object instances provided a rich learning environment that far exceeded the simplified scenarios presented by CIFAR-10. While more computationally demanding, the extensive array of real-world scenes in the dataset ultimately afforded a broader and deeper understanding of varied contexts and challenges in image inpainting.

To conclude, the datasets used in this project were the upscaled version of CIFAR-10 and COCO. The CIFAR-10 was used in its original form, maintaining the initial division of training and testing sets. In contrast, the COCO dataset was further partitioned, being divided into a **training set of 80,000 images**, a **testing set of 20,032 images** and a **validation set of 20,032 images**, while the rest up to 330,000 were discarded due to hardware constraints. The specific number of 20,032 for the test and validation sets was intentionally chosen to be compatible with the batch size employed in the model training process.

3.2 Image Pre-processing

Data pre-processing is an indispensable step in the machine learning pipeline. This stage aims to convert raw, unprocessed data into a structured format that is suitable for machine learning algorithms. As revealed in Figure 3.4, the steps constitute loading the image dataset, performing random cropping, generating degraded images through the application of masks, normalizing the images and caching the processed dataset.

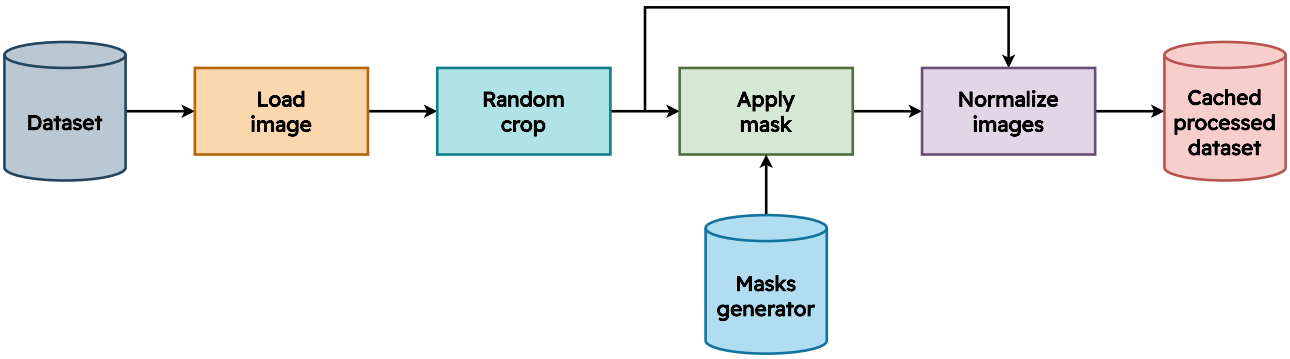


Figure 3.4: Image pre-processing pipeline with masks generator

Loading of the image dataset

The initial step in the pre-processing pipeline involves preparing and loading the image dataset. This process begins with sourcing the images from the internet. Once downloaded, these images are partitioned into distinct subsets for training, testing and validation purposes. This division is performed only once to ensure consistency across experimental iterations. Thereafter, these datasets are imported into the system by reading them from the disk and loading them into the Random Access Memory (RAM). This loading step is crucial as it enables efficient and speedy access to the images during the next processing operations.

Random cropping of images

Upon the successful loading of the images, the next step is the application of random cropping, a process specifically used when handling the COCO dataset. This decision is driven by two primary factors. Firstly, due to hardware limitations, the full processing of high-resolution images, which can contain millions of pixels, becomes computationally demanding. By focusing on a random subset of these pixels, the dimensionality of the dataset is effectively reduced, thereby significantly enhancing computational efficiency. Secondly, the application of random cropping is instrumental in obtaining images of uniform size (chosen **128×128**), a prerequisite for the employed network architectures.

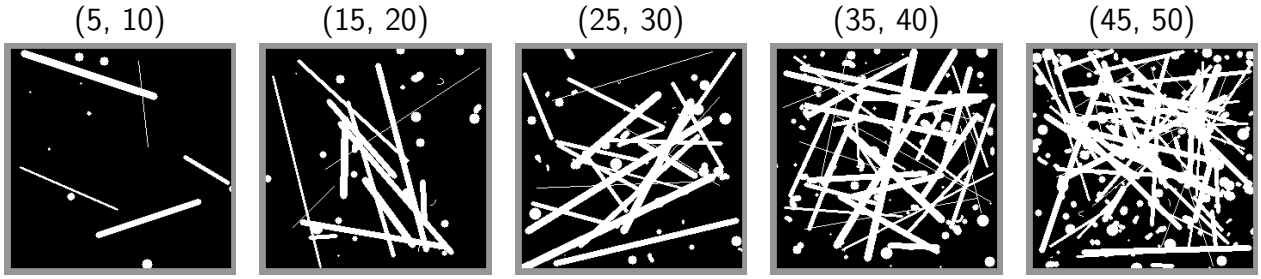
Moreover, as highlighted in Figure 3.5, random cropping serves as a form of **data augmentation**. This technique, by increasing the diversity of the training data, aids in

Figure 3.5: Random cropped regions from images

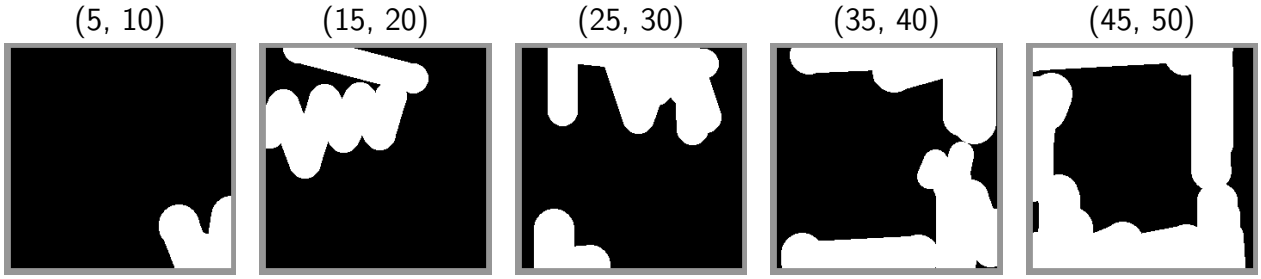
mitigating the risk of overfitting, thereby enhancing the generalization capability of the model. Thus, the process of random cropping, while primarily employed due to hardware constraints and the necessity of uniform image sizes, also contributes significantly to the robustness of the model.

Application of masks

The subsequent step in the pipeline involves the generation and application of masks. Within the scope of image processing, a mask is a binary image that delineates which pixels of an image should be subjected to processing. For this particular task, masks are employed to demarcate the degraded regions within the cropped areas of the images that necessitate inpainting.



(a) Masks generated using random shapes (lines, circles and distorted ellipses)



(b) Mask generated using the modified freeform algorithm (brush-like strokes)

Figure 3.6: Comparison of generated masks

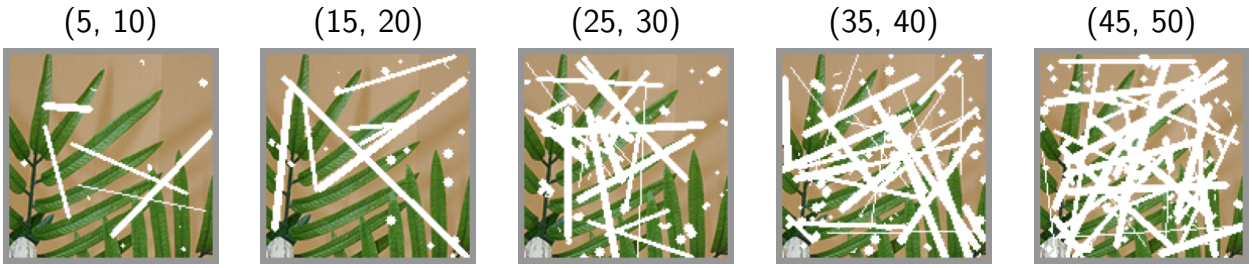
As illustrated in Figure 3.6a, the generation of these masks was initially accomplished using a simple algorithm that produced random shapes, such as lines, circles and distorted

ellipses. This approach provided a basic method for creating diverse and randomly degraded regions within the images. However, to enhance the realism and complexity of the degradation, the mask generation process was later updated to incorporate an algorithm derived from [40]. This updated algorithm generates brush-like strokes, as presented in Figure 3.6b, thereby creating masks that simulate more natural and realistic degradation patterns.

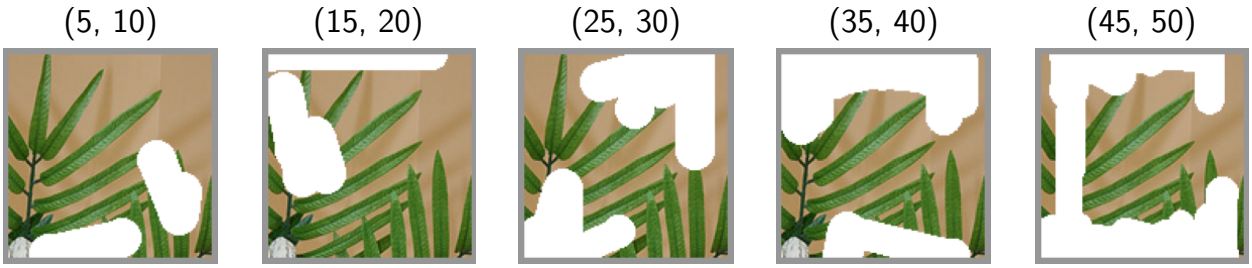
A key aspect of the mask generation process is the adherence to what is termed a **degradation ratio interval** (DRI), as seen in the titles of the two previously mentioned figures. This is a predefined interval that specifies, in percentages, the area of the image that should be covered by the degradation. The degradation ratio interval thus provides a control mechanism for the extent of degradation introduced into the images, ensuring a balanced and controlled distribution of degradation across the dataset.

Regardless of whether they are produced by the initial or the updated algorithm, these masks serve a pivotal role in the pre-processing pipeline. They are applied to the cropped regions of the images, with the '1' values in the mask signifying the pixels in the image that should be regarded as degraded, as illustrated in Figure 3.7. This process of applying the masks effectively transforms the original images into degraded versions, where degraded regions have the maximum attainable pixel value (255 for an 8-bit image).

Note: The masks and masked images depicted in the preceding figures are framed with a slight grey border, a design choice implemented to enhance their distinguishability and facilitate easier visual interpretation for the reader.



(a) Images affected by masks generated using random shapes



(b) Images affected by masks generated using the modified freeform algorithm

Figure 3.7: Example of a cropped image masked with different degradation ratio intervals

Normalization of images

Following the masking process, the original image (referred to as the ground truth) and the masked image (the one designated for inpainting) are assembled into a **pair**. Following this pairing, each image undergoes an essential pre-processing step known as normalization, a fundamental procedure that involves scaling the initial values of the data to a specific range. This process is instrumental in reducing computational complexity and facilitating faster convergence of the learning algorithm.

In the absence of normalization, the range of pixel values could be considerably large (e.g. $[0, 255]$ for an 8-bit image), which could induce numerical instability during the learning process. By scaling the values to a smaller range, such issues can be effectively mitigated. Therefore, both the ground truth and the masked image in each pair are normalized in the $[0, 1]$ range, ensuring a stable and efficient learning process.

Caching of the processed dataset

The final stage in the pre-processing pipeline involves caching the processed image pairs. Caching is a technique that temporarily stores the processed data in a storage area, specifically the system's RAM in this case, to facilitate quicker access during subsequent processes. This step is particularly beneficial in enhancing the efficiency of the deep learning tasks by reducing the time taken to access the processed data. After the completion of the first epoch, the processed image pairs are cached into the system's RAM. This strategy ensures faster access times during the training process, thereby significantly improving the overall computational efficiency.

An additional reason for the implementation of caching in this context is the time-consuming nature of the mask generation process. Due to a limitation in the `TensorFlow` library, this process cannot be parallelized, resulting in an additional time overhead for mask generation. By caching the processed image pairs after the first epoch, the need to regenerate the masks for each epoch is eliminated, thereby saving considerable processing time. While the cache could also be stored on disk for larger size, the choice of RAM as the caching medium in this context offers the advantage of speed, which is crucial for the iterative nature of the training process and particularly beneficial given the time-intensive nature of the mask generation process.

Chapter 4
Proposed Solutions

Recognizing the necessity for a robust and adaptive approach, we explore the implementation of two distinct deep learning architectures, U-Net and ResNet, which have demonstrated remarkable efficacy in image processing tasks. Subsequent sections provide a comprehensive examination of these architectures, detailing their design, implementation and specific applicability to our task at hand. Finally, a separate section offers a thorough review of the conducted experiments and the ensuing results, thus quantifying the effectiveness of the proposed solutions in addressing the challenge of image inpainting.

4.1 U-Net

The U-Net model, a distinguished and influential architecture within the domain of deep learning, has garnered considerable recognition for its exceptional performance in the realm of image segmentation tasks. As depicted in Figure 4.1, the architecture's defining characteristic is its symmetric, expansive pathway, a design that facilitates precise localisation[54]. This attribute is particularly advantageous in tasks that necessitate exact boundary demarcation, such as semantic segmentation. The model's unique design, coupled with its proven effectiveness, positions it as an optimal choice for such complex image-processing tasks. The ensuing discussion will delve into the details of the U-Net model, elucidating its architectural components and operational mechanisms.

The U-Net model is composed of several **blocks**, each performing a specific function. These blocks can be broadly categorised into the **encoder** (i.e. *downsampling* or *contraction* path), the **decoder** (i.e. *upsampling* or *expansion* path) and the **bottleneck** (i.e. *skip/shortcut connections*).

Encoder

The primary function of the encoder (also referred to as the downsampling or contraction path) is to systematically reduce the spatial dimensions of the input image while simultaneously increasing the depth of the feature maps. This process facilitates the extraction of high-level semantic features from the input data. The encoder achieves this by applying a series of transformations that progressively abstract the information, thereby enabling the

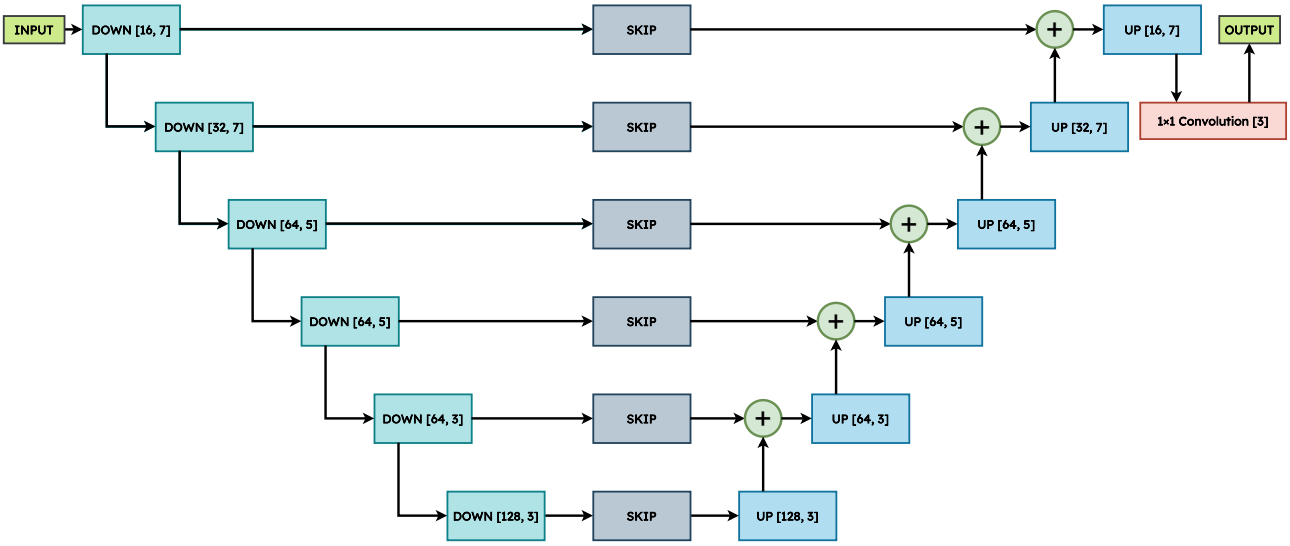


Figure 4.1: General overview of the U-Net architecture

model to discern complex patterns and relationships within the data.

The encoder constitutes the initial half of the U-Net model, being composed of a series of convolutional blocks, as outlined in Figure 4.2. Each block within the downsampling path commences with a `K×K` convolution operation, where `K` denotes the kernel size and `C` represents the number of channels or filters. This operation serves to extract features from the input image. Following the convolution operation, a `2×2` max pooling operation is executed. This operation effectively reduces the spatial dimensions of the input, thereby downsampling the image. The max pooling operation aids in expanding the field of view of the convolutions and mitigating the computational complexity of the model. Following the max pooling operation, the batch normalisation step follows. This technique standardizes the outputs of the max pooling operation, enhancing the stability and performance of the model. Post batch normalisation, a Leaky ReLU activation function with a negative slope of `0.2` is performed. This function introduces non-linearity into the model, empowering it to learn more intricate representations. Following the first Leaky ReLU, another `K×K` convolution operation with `C` channels is performed, succeeded by batch normalisation and another Leaky ReLU activation function. The block concludes with a dropout operation at a rate of `5%`.

Decoder

The primary role of the decoder (alternatively known as the upsampling or expansion path) is to restore the spatial dimensions of the abstracted feature representation received from the encoder, effectively reconstructing the image. It accomplishes this by progressively increasing the resolution of the feature maps while simultaneously decreasing their depth. This process enables the model to generate a detailed and high-resolution output from the high-level features extracted by the encoder. In the specific context of image inpainting, the

decoder's role is to use these high-level features to accurately fill in the missing or corrupted parts of the image, thereby producing a complete and coherent output

As reflected in Figure 4.2, the decoder forms the latter half of the U-Net model, being comprised of a series of up-convolutional blocks. Each block within the upsampling path initiates with a batch normalisation operation. This technique normalises the inputs, enhancing the stability and performance of the model. Following batch normalisation, a `K×K` convolution operation with `C` channels is performed, serving to extract features from the input. Succeeding the convolution operation, another batch normalisation is applied, followed by a Leaky ReLU activation function with a negative slope of `0.2`. The block then incorporates a dropout operation at a rate of `5%`. Following the dropout, a `1×1` convolution operation with `C` channels is performed, succeeded by batch normalisation and another Leaky ReLU activation function. The block concludes with another dropout operation at a rate of `5%`, followed by a `2×2` transposed convolution operation with `C` channels. This operation effectively increases the spatial dimensions of the input, thereby upsampling the image.

Bottleneck

Situated between the encoder and decoder in the U-Net model, the bottleneck path, also called skip/shortcut connections, plays a crucial part. Its primary role is to extract the most abstract and high-level features from the input data, acting as a bridge between the encoder and decoder paths. This layer is of foremost importance as it encapsulates the most compressed representation of the input data, which is then expanded by the decoder to generate the final output. These connections, both long and short, are instrumental in addressing the vanishing gradient problem. Long skip connections specifically allow for the transfer of features from the encoder to the decoder, aiding in the recovery of spatial information that may have been lost during the downsampling process. Short skip connections, on the other hand, contribute to the stabilisation of gradient updates.

The bottleneck block is composed of a distinct sequence of operations, setting it apart from the other blocks in the model, as suggested in Figure 4.2. Each bottleneck block initiates with a `1×1` convolution operation with `4` filters. This operation is designed to extract high-level features from the input. Following the convolution operation, batch normalisation is applied. Ensuing batch normalisation, a Leaky ReLU activation function with a negative slope of `0.2` is used. The block then incorporates a dropout operation at a rate of `5%`. Contrary to the other blocks, the bottleneck does not include a max pooling or transposed convolution operation. As illustrated in Figure 4.1, the U-Net model employs a concatenation operation to integrate the high-level feature representations extracted by the bottleneck blocks with the spatially detailed information reconstructed by the upsampling blocks.

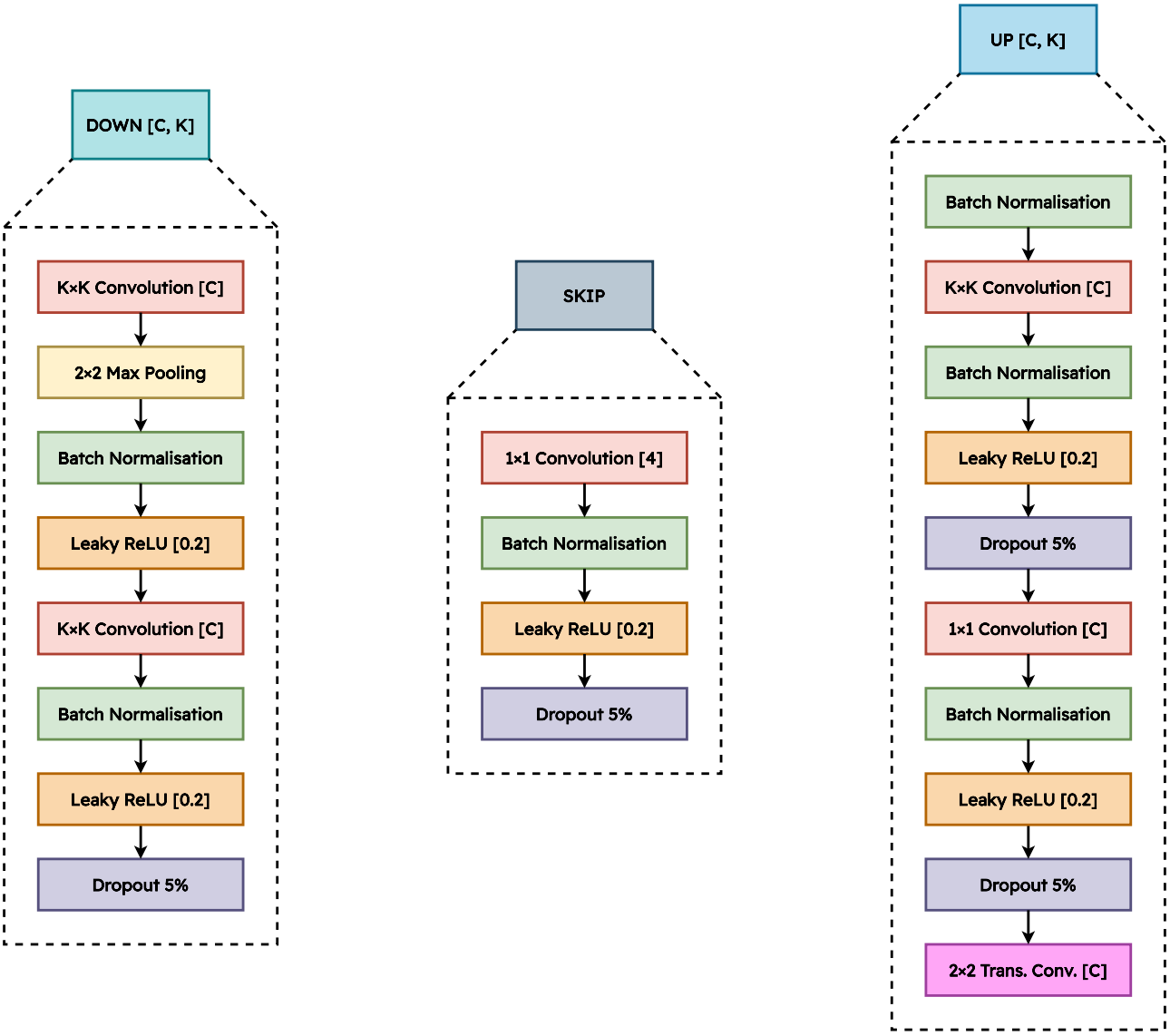


Figure 4.2: Detailed examination of U-Net's incorporating blocks

Ouput

As indicated in Figure 4.1, the terminal layer of the UNet model is a 1×1 convolution with 3 channels. This layer is instrumental in mapping the deep feature representation, established by the network, to the inpainted RGB image. The output of this layer is not a conventional probability map, but rather a reconstructed RGB image where each pixel's value is determined by the network's output.

4.2 ResNet

The **Residual Network** (ResNet), a cornerstone in the realm of deep learning architectures, is the focal point of this section, as depicted in Figure 4.3. ResNet is predicated on the innovative concept of residual learning. This paradigm-shifting model is characterised

by its depth, comprising a multitude of interconnected layers, each integral to the model's ability to interpret and learn intricate patterns embedded within the data. The architecture of ResNet is ingeniously designed to circumvent the vanishing gradients problem, a common pitfall associated with increasing network depth. This is achieved through the introduction of shortcut or skip connections, which facilitate the propagation of gradients throughout the network. These connections, also known as **residuals**, enable the model to learn an identity function, ensuring the network's performance does not deteriorate with increasing depth. As such, ResNet has emerged as a robust and efficient framework in the field of computer vision, setting a new standard for tasks such as image classification and recognition.

As highlighted in Figure 4.3, the architecture commences with a `3×3` convolutional layer with `32` filters. This layer is responsible for extracting low-level features from the input data. Following this layer, a batch normalisation process is applied, which standardises the inputs to the next layer, improving the stability and speed of learning. Succeeding batch normalisation, a `3×3` max pooling operation is performed. This operation reduces the spatial dimensions of the input, effectively downsampling the image and increasing the field of view of the convolutions, which aids in reducing the computational complexity of the model. Following the max pooling operation, the architecture incorporates a series of residual blocks. After the sequence of residual blocks, a Leaky ReLU activation function with a negative slope of `0.2` is used, introducing non-linearity into the model. Following the Leaky ReLU, a `5×5` convolution operation with `64` filters is performed, followed by batch normalisation and another Leaky ReLU activation function. This sequence of operations allows the model to extract even more complex features from the input data. The architecture concludes with a `4×4` transposed convolution operation with `3` channels. This operation effectively increases the spatial dimensions of the input, thereby upsampling the image and enabling the reconstruction of the original input image from the abstracted feature representation.

Each residual block commences with a Leaky ReLU activation function with a negative slope of `0.2`. Following this, the architecture incorporates a residual node. This node is composed of a series of operations, starting with a `3×3` convolution operation with `32` channels. This operation serves to extract features from the input data. The convolution operation is followed by batch normalisation. Subsequent to batch normalisation, a dropout operation at a rate of `10%` is performed as a form of regularisation that helps prevent overfitting. After the dropout operation, another Leaky ReLU activation function with a negative slope of `0.2` is used, followed by another `3×3` convolution operation with `32` filters and batch normalisation. This sequence of operations allows the model to extract even more complex features from the input data. The output of the Leaky ReLU at the beginning of the block and the output of the residual node are then summed up. This operation allows the gradient to flow directly through the network, mitigating the vanishing gradient problem and enabling the training of extremely deep networks.

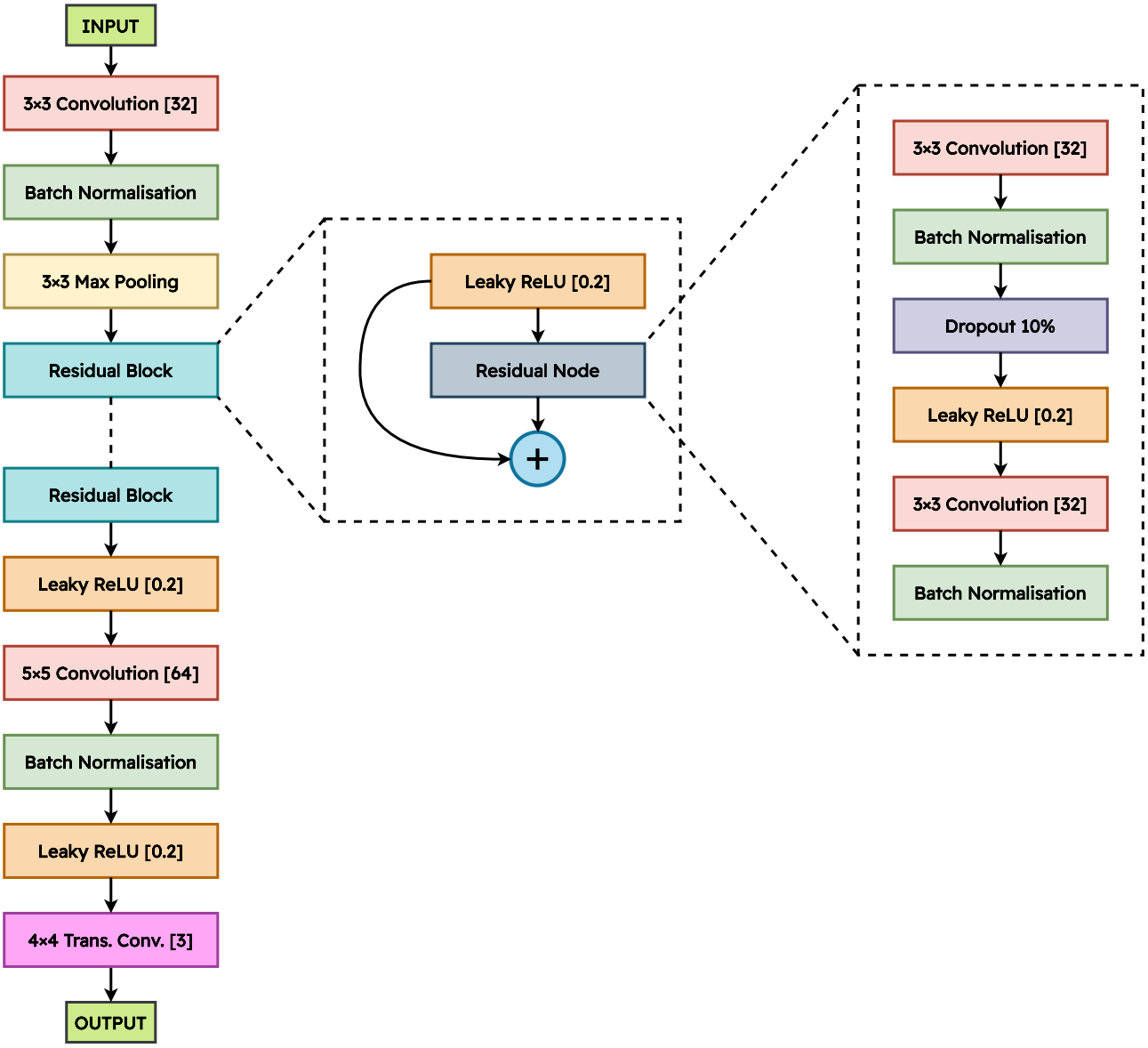


Figure 4.3: In-depth look at the ResNet architecture and its encompassing layers

4.3 Experiments and Results

In this segment, detailed discussions on the experiments conducted, the results derived and additional aspects of the implemented solutions are to be presented. The chosen evaluative metrics, namely the Peak Signal-to-Noise Ratio (PSNR), Structural Similarity Index Measure (SSIM) and Learned Perceptual Image Patch Similarity (LPIPS), are to be employed for the quantitative assessment of the solutions' performance.

In the incipient phase of this project, the mask generation algorithm of choice was the random shapes one, as described in section 3.2. This initial stage established baseline values for the chosen metrics, applying them to the masked images, as illustrated in Table 4.1. Furthermore, traditional inpainting methodologies, discussed previously in section 1.4, were applied to the upscaled CIFAR-10 dataset. This approach allowed for the computation of the

DRI	PSNR [dB]	SSIM [%]	LPIPS [%]
(5, 10)	17.99	61.56	60.88
(15, 20)	13.44	31.34	44.29
(25, 30)	11.24	19.95	34.26
(35, 40)	9.77	14.87	26.36
(45, 50)	8.69	12.09	20.21

Table 4.1: Baseline values on the 64×64 upscaled CIFAR-10 dataset (random shapes mask generation)

same metrics, facilitating a comprehensive evaluation of the inpainting outcomes yielded by these classic methods, as denoted in Table 4.2.

	Telea			Navier-Stokes			PatchMatch		
DRI	PSNR [dB]	SSIM [%]	LPIPS [%]	PSNR [dB]	SSIM [%]	LPIPS [%]	PSNR [dB]	SSIM [%]	LPIPS [%]
(5, 10)	38.96	98.88	99.11	42.57	99.59	99.77	39.35	98.99	99.01
(15, 20)	33.74	96.52	96.09	36.73	98.57	98.93	34.50	97.04	95.51
(25, 30)	31.01	93.73	91.96	33.45	97.07	97.33	32.11	94.95	91.25
(35, 40)	28.95	90.43	87.30	30.89	94.91	94.76	30.40	92.62	87.09
(45, 50)	27.20	86.53	82.67	28.68	91.79	91.10	28.86	89.75	83.03

Table 4.2: Results for classic methods on the 64×64 upscaled CIFAR-10 dataset (random shapes mask generation)

In order to validate the effectiveness of the U-Net architecture and affirm its implementation correctness, the inference phase was executed on an identical test dataset to the one used by the traditional inpainting methods. This allowed for a direct and equitable comparison of the U-Net architecture's results with those obtained by traditional methods. The outcomes of this comparative analysis, as catalogued in Table 4.3, provide valuable insights into the performance of the proposed deep learning solution vis-à-vis that of the traditional inpainting methodologies.

	MAE			MSE			SSIM-MAE		
DRI	PSNR [dB]	SSIM [%]	LPIPS [%]	PSNR [dB]	SSIM [%]	LPIPS [%]	PSNR [dB]	SSIM [%]	LPIPS [%]
(5, 10)	33.77	96.40	97.75	32.75	95.57	97.07	30.71	96.64	96.74
(15, 20)	31.70	95.18	97.40	31.82	95.11	97.11	29.22	94.70	96.56
(25, 30)	31.23	94.29	96.53	31.36	93.28	95.57	30.14	94.32	95.23
(35, 40)	29.83	92.50	95.61	30.55	91.86	95.09	28.46	93.57	94.92
(45, 50)	28.84	91.27	94.26	29.45	89.67	92.86	26.99	91.91	88.49

Table 4.3: Results for U-Net on the 64×64 upscaled CIFAR-10 dataset (random shapes mask generation)

During the network's training phase, three distinct loss functions were employed: Mean Squared Error (MSE), Mean Absolute Error (MAE) and a composite function designated as SSIM-MAE. The SSIM-MAE loss function represents a combination of MAE and the Structural Similarity Index Measure (SSIM) loss. The composition of the SSIM-MAE is determined by a weighting constant, α (alpha), which falls within the interval $(0, 1)$. The

role of α is to balance the contributions of MAE and SSIM when computing the composite loss. Through empirical analysis, an optimal alpha value of 0.75 was determined. It should be noted that this loss function can be viewed as a simplified version of the one delineated in [55], wherein the Multi-Scale SSIM is employed instead of the simple SSIM. The specific mathematical formulations that characterise each of these loss functions are to be detailed in the next equations:

$$\mathcal{L}_{\text{MSE}} = \frac{1}{T_b} \sum_{k=1}^{T_b} \text{MSE}\left(\mathbf{y}_k, \hat{\mathbf{y}}_k\right) , \tag{4.1}$$

$$\mathcal{L}_{\text{MAE}} = \frac{1}{T_b} \sum_{k=1}^{T_b} \text{MAE}\left(\mathbf{y}_k, \hat{\mathbf{y}}_k\right) , \tag{4.2}$$

$$\mathcal{L}_{\text{SSIM-MAE}} = \frac{1}{T_b} \sum_{k=1}^{T_b} \left[\alpha \cdot \text{MAE}\left(\mathbf{y}_k, \hat{\mathbf{y}}_k\right) + (1-\alpha) \cdot \left(1 - \text{SSIM}\left(\mathbf{y}_k, \hat{\mathbf{y}}_k\right)\right)\right] , \tag{4.3}$$

where T_b denotes the number of samples in the current batch, $\mathbf{y}$ represents the ground-truth image value and $\hat{\mathbf{y}}$ signifies the predicted value.

The optimisation algorithm of choice was AMSGrad, the refined variant of the Adam optimiser, as detailed in section 1.2, with a designated learning rate of 0.01. In an effort to refine the training process, the feature involving callbacks during training available in `TensorFlow` was leveraged. Specifically, three distinct callbacks were employed to boost the network's performance and training robustness:

- **Reduce Learning Rate on Plateau** (`ReduceLROnPlateau`) – This callback dynamically adjusts the learning rate by a fixed constant, lowering it up to a predetermined minimum value if the validation loss fails to show improvement over a specified number of epochs.

- **Early Stopping** (`EarlyStopping`) – This mechanism halts training when the validation loss ceases to improve for a fixed number of epochs. This number is invariably larger than the one specified for the previous callback. This strategy, coupled with a very large number of training epochs, prevents the wasteful continuation of training when no further improvement is discernible.

- **Model Checkpoint** (`ModelCheckpoint`) – This callback saves the trained model to the disk if the validation loss registers an improvement in the current epoch. At the end of the training phase, it restores the model's weights to the best-saved ones, thereby ensuring that the optimal model configuration is retained.

Throughout the training process, the evaluation was confined to two metrics: PSNR and SSIM. This selection was necessitated by the fact that the use of LPIPS, while insightful, is computationally intensive and its application would have slowed down the training process. A visual representation of the loss and evaluation metrics' evolution, corresponding to each of the three types of loss functions employed, is presented in Figure 4.4.

DRI	PSNR [dB]	SSIM [%]	LPIPS [%]
(5, 10)	17.31	81.37	65.05
(15, 20)	12.90	57.43	51.98
(25, 30)	10.75	40.80	46.50
(35, 40)	9.31	29.62	42.74
(45, 50)	8.27	22.10	39.11

Table 4.4: Baseline values on the 128×128 random-cropped COCO dataset (random shapes mask generation)

The succeeding phase involves the training, evaluation and testing of the U-Net model using larger 128×128 patches from the COCO dataset. Mirroring the previous approach, a baseline was established on the masked images, as catalogued in Table 4.4 and the performance metrics were computed employing traditional inpainting methods, as documented in Table 4.5. Following this, the U-Net model, trained with the same methodology, was deployed for the inpainting task and the corresponding test metrics were computed, with the results collated in Table 4.6.

	Telea			Navier-Stokes			PatchMatch		
DRI	PSNR [dB]	SSIM [%]	LPIPS [%]	PSNR [dB]	SSIM [%]	LPIPS [%]	PSNR [dB]	SSIM [%]	LPIPS [%]
(5, 10)	36.40	97.77	98.41	36.67	97.95	98.53	36.44	97.60	98.33
(15, 20)	31.46	93.77	95.26	31.51	94.21	95.56	31.38	93.18	95.16
(25, 30)	28.87	89.48	91.48	28.93	90.16	91.91	28.96	88.30	91.77
(35, 40)	27.06	84.84	87.01	27.07	85.70	87.54	27.19	82.90	88.07
(45, 50)	25.54	79.79	81.95	25.49	80.78	82.46	25.78	77.04	84.13

Table 4.5: Results for classic methods on the 128×128 random-cropped COCO dataset (random shapes mask generation)

	MAE			MSE			SSIM-MAE		
DRI	PSNR [dB]	SSIM [%]	LPIPS [%]	PSNR [dB]	SSIM [%]	LPIPS [%]	PSNR [dB]	SSIM [%]	LPIPS [%]
(5, 10)	27.11	81.55	87.34	26.45	80.05	86.87	24.49	83.16	87.15
(15, 20)	26.06	78.07	85.93	25.55	77.44	83.29	23.77	82.13	86.91
(25, 30)	25.08	73.60	80.78	25.35	73.57	77.35	23.50	77.02	79.24
(35, 40)	24.77	70.13	78.34	23.76	67.72	71.93	22.79	73.47	71.93
(45, 50)	24.26	66.44	73.22	23.43	65.06	69.24	22.66	69.94	73.44

Table 4.6: Results for U-Net on the 128×128 random-cropped COCO dataset (random shapes mask generation)

Following the initial approach, we now turn our attention to another hybrid loss function known as LPIPS-MAE, which incorporates the LPIPS from [7]. Similar to SSIM-MAE, LPIPS-MAE incorporates a parameter α (alpha) to balance the contribution of MAE and LPIPS components when calculating the hybrid loss. The impetus for introducing this loss function lies in LPIPS' superior capacity to extract perceptual information from images compared to SSIM, potentially leading to more visually appealing inpainting results. Moreover, SSIM presents a limitation when applied to colour images, a shortcoming not shared by LPIPS. Additionally, the α parameter, integral to the computation of the LPIPS-MAE loss

function, was empirically determined. Following experimentation, an optimal alpha value of 0.75 was ascertained, providing the most effective balance between the MAE and LPIPS components of the loss function. The mathematical formulation of the LPIPS-MAE loss function is as follows:

$$\mathcal{L}_{\text{LPIPS-MAE}} = \frac{1}{T_b} \sum_{k=1}^{T_b} \left[\alpha \cdot \text{MAE}\left(\mathbf{y}_k, \hat{\mathbf{y}}_k\right) + (1-\alpha) \cdot \text{LPIPS}\left(\mathbf{y}_k, \hat{\mathbf{y}}_k\right) \right] . \tag{4.4}$$

The choice of coupling MAE with these two loss functions, as opposed to MSE, might initially seem counterintuitive. However, empirical evidence gathered during the training phase, as depicted in Figure 4.5, suggested that MSE demonstrated a more unstable and oscillating behaviour compared to MAE. Thus, the selection of MAE was made to account for this observed behaviour. Furthermore, it was considered that for larger patches such as those from the COCO dataset, the SSIM and LPIPS components of the loss functions may exhibit instability. Hence, the MAE component provides a more reliable and stable counterbalance.

While the initial random shapes mask generation algorithm proved instrumental in validating the model's performance and correctness, providing a fundamental benchmark, it fell short in generating masks of practical use. Consequently, the subsequent phase of development necessitated a transition towards a more sophisticated masking technique. The modified freeform algorithm, as highlighted in section 3.2, was adopted for its ability to generate masks that mimic brush-like strokes, thereby providing a more realistic and practical approach to the task at hand.

In the initial stages of the experimental approach, the U-Net model was constructed employing traditional convolution operations. To explore potential improvements, an experiment was conceived where gated convolutions were integrated into the model, as elucidated in section 1.3. Unfortunately, the attempt to enhance the model with gated convolutions led to unsatisfactory outcomes. Specifically, the loss function exhibited high levels of oscillation during the training phase, implying a lack of stability and consistent convergence. Owing to these erratic and suboptimal results, this trajectory in model development was abandoned.

The unsuccessful implementation of gated convolutions within the U-Net model did not deter further exploration. In a successive experiment aimed at enabling the use of gated convolutions, a different network architecture, reminiscent of a component of the pipeline detailed in [40], was put into practice. This architecture, namely ResNet, featuring 13 residual blocks, is further delineated in section 4.2.

To ensure an effective comparison between the original U-Net and the newly implemented architecture, which, for the purposes of this discourse, will henceforth be referred to as GatedResNet, training was performed on the gated model using each of the four loss

functions previously described. However, to simplify the comparison, this training was limited to a specific degradation ratio interval of $(25, 30)$. This narrower focus allowed a more nuanced evaluation, as represented in Table 4.7, thus paving the way for the determination of the optimal architecture and, correspondingly, the most suitable loss function.

Network	PSNR [dB]	SSIM [%]	LPIPS [%]
U-Net (MSE)	22.30	68.97	76.81
U-Net (MAE)	21.54	63.60	71.54
U-Net (SSIM-MAE)	21.49	72.27	73.04
U-Net (LPIPS-MAE)	21.67	64.62	70.22
GatedResNet (MSE)	23.82	72.88	78.29
GatedResNet (MAE)	23.03	71.21	73.81
GatedResNet (SSIM-MAE)	21.37	67.11	68.62
GatedResNet (LPIPS-MAE)	23.74	76.50	80.62

Table 4.7: Test metrics for (25, 30) degradation ratio interval (freeform mask generation)

Following this particular comparison, the model yielding the highest metrics, as presented in Table 4.7, was subjected to training across the remaining degradation ratio intervals. Thereupon, this model was affirmed as the final choice for the image inpainting task. The inpainting process for a masked image proceeds as follows:

1. The image along with the mask are segmented into 128×128 patches. This process includes padding where necessary to accommodate the prescribed patch size.

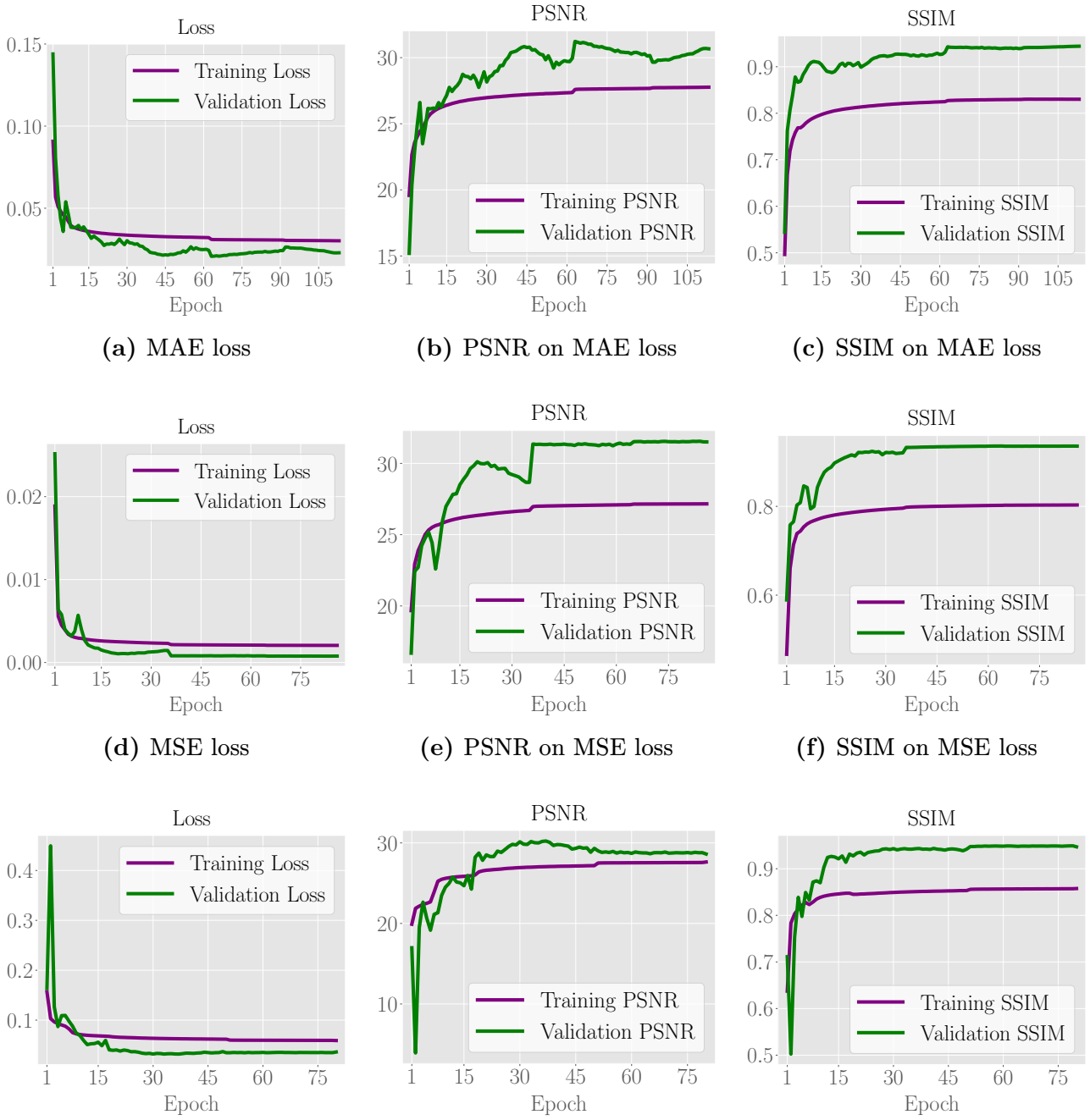
2. Each patch undergoes an evaluation process. If the mask of the current patch contains values of '1', indicating the presence of a masked area, the corresponding image patch is subject to inpainting. The appropriate model for this step is determined based on the percentage of values of '1' present in the mask, denoting the degradation ratio.

3. The final step involves reconstructing the full image from the inpainted patches. If padding was employed during the segmentation process, it is duly eliminated during image reconstruction to restore the image to its original dimensions.

As depicted in Figure 4.6, a three-part illustration is presented: the original image, the corresponding mask that denotes the region to be inpainted and the resultant image after the application of the mask, which represents the preliminary image intended for the inpainting procedure. In order to facilitate an efficient comparative analysis between the GatedResNet architecture trained with the LPIPS-MAE loss and the three classical inpainting algorithms discussed in section 1.4, we performed the inpainting process on the previously masked image using these four different methods. The resulting images from these distinct approaches are illustrated in Figure 4.7. To further refine the comparison and shed light on the intricacies of each method's inpainting process, specific patches have been selected and extracted from

the inpainting regions of each image. These chosen patches serve as a magnified lens into the procedural details, offering a more granulated view of the variances between each approach. By this means, the aim is to provide a robust and comprehensive exploration of the process and the performance of the inpainting algorithms.

In light of the prior discussion, it would be instructive to refer to Appendix A for details regarding the implemented solution and to find out the course of action on how to access it. Furthermore, a comprehensive collection of supplemental results, specifically images, is presented for examination in Appendix B.



(a) MAE loss **(b)** PSNR on MAE loss **(c)** SSIM on MAE loss

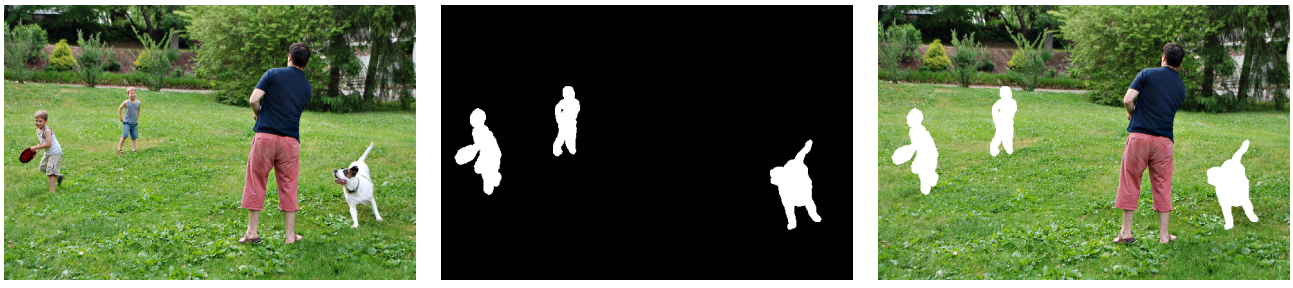
(d) MSE loss **(e)** PSNR on MSE loss **(f)** SSIM on MSE loss

(g) SSIM loss **(h)** PSNR on SSIM loss **(i)** SSIM on SSIM loss

Figure 4.4: Comparison of training metric of U-Net on upscaled CIFAR-10 (random shapes mask generation)

(a) MSE loss **(b)** MAE loss

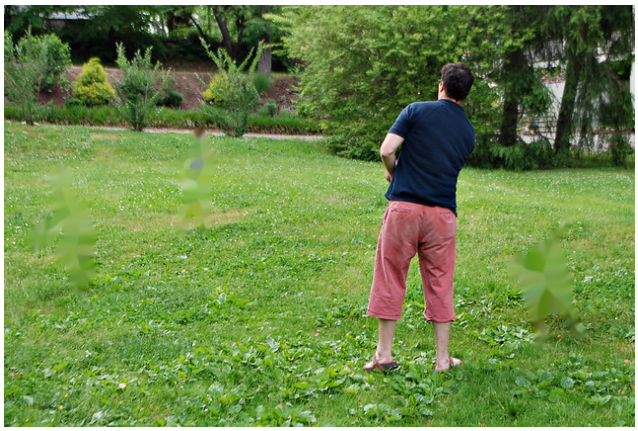
Figure 4.5: Training loss of GatedResNet on COCO using MSE and MAE loss (freeform mask generation)



(a) Original image **(b)** Inpainting mask **(c)** Masked image

(d) Original image - Patches **(e)** Inpainting mask - Patches **(f)** Masked image - Patches

Figure 4.6: The masking process for an image



(a) Navier-Stokes

(b) Telea

(c) Navier-Stokes - Patches

(d) Telea - Patches

(e) PatchMatch

(f) GatedResNet

(g) PatchMatch - Patches

(h) GatedResNet - Patches

Figure 4.7: Comparison of inpainting methods

Conclusions

Final Remarks

Image inpainting, as affirmed through this thesis, remains one of the challenging tasks within the domain of deep learning. Its complexity is underlined by the necessity for intricate pattern recognition, a high-level understanding of image structure and the subsequent generation of plausible and visually consistent content. This task's resource-intensive nature further accentuates its complexity, demanding extensive computational power and high-quality datasets to successfully develop and train models.

In this examination, we navigated through these challenges by implementing two distinct neural network architectures: U-Net and ResNet. The choice of these architectures was inspired by their successful precedence in image-related tasks. Our exploration into these architectures was complemented by an analysis of their performance, yielding satisfactory results that contribute to our understanding of applying deep learning techniques to image inpainting.

However, it is crucial to highlight that the boundary of this exploration was in part dictated by the available hardware. The performance and potential of deep learning models are closely tied to the computational power at their disposal. Therefore, the limitations of accessible hardware manifested as restrictions on the achievable complexity of the models and the speed of their training and testing processes.

Personal Contributions

In the course of completing this thesis, a multitude of personal contributions has been made, showcasing an engagement with the topic and the practical aspects of machine learning, image processing and deep learning techniques:

- Development and implementation of an efficient pre-processing pipeline, specifically designed to handle raw images extracted from two datasets, namely upscaled CIFAR-10 and COCO. This pipeline was created using various Python libraries such as `Numpy`, `Pillow`, `OpenCV`, `TensorFlow` and the native `multiprocessing` package. It ensured that the datasets were aptly and swiftly prepared for the subsequent stages of the project.

- Research into existing image inpainting methodologies. This task entailed investigating

relevant literature and technical studies to draw valuable insights and techniques, which were then incorporated into the core methodology of this project.

- Designing and implementing the proposed deep learning architectures, U-Net and ResNet. The chosen architectures were motivated by their success in related applications, as indicated in the literature. These networks were brought to life using `TensorFlow`.
- Careful orchestration of the training process for the proposed architectures. This entailed a cycle of choosing suitable loss functions, monitoring model performance across training epochs and an iterative process of hyperparameter tuning. Furthermore, the trained models' performance was assessed on the pre-processed datasets.

Future Work

Future research directions offer several exciting possibilities. Firstly, exploring state-of-the-art network architectures, such as multi-stage ones that incorporate Generative Adversarial Networks, could refine the current model. While U-Net and ResNet have shown their worth, cutting-edge architectures are going to enhance the performance of the inpainting system. Secondly, adopting advanced loss functions might improve model accuracy. Despite the effectiveness of the current loss functions, adaptive ones could offer more precise reconstructions, making a significant impact on the system's performance. The use of a more diverse database, such as ImageNet, could also provide additional robustness to the system. Though the current datasets were useful, ImageNet's greater diversity could enhance the system's adaptability to various image inpainting scenarios. Lastly, using devices with higher computational capabilities could accelerate training times and allow for the exploration of more complex models, possibly leading to better results. As the field evolves, more potent resources will play an increasingly significant role. In summary, these future directions can push the envelope in the field of degraded colour images inpainting, taking it closer to the cutting edge.

Bibliography

[1] C. Vertan and M. Ciuc, *Tehnici fundamentale de prelucrarea și analiza imaginilor*, Romanian. Bucharest, Romania: Matrix Rom, 2007, pages 8–9, ISBN: 978-973-755-207-5.

[2] R. Gonzalez, R. Woods, and S. Eddins, *Digital Image Processing Using MATLAB*, 3rd edition. Gatesmark Publishing, 2020, pages 16–17, ISBN: 978-098-208-541-7.

[3] Z. Wang, A. Bovik, H. Sheikh, and E. Simoncelli, "Image quality assessment: from error visibility to structural similarity", *IEEE Transactions on Image Processing*, volume 13, number 4, pages 600–612, 2004. DOI: 10.1109/TIP.2003.819861.

[4] B. Li, X. Peng, Z. Wang, J. Xu, and D. Feng, "AOD-Net: All-in-One Dehazing Network", in *IEEE International Conference on Computer Vision (ICCV)*, 2017, pages 4780–4788. DOI: 10.1109/ICCV.2017.511.

[5] C. Ledig, L. Theis, F. Huszár, *et al.*, "Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network", in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pages 105–114. arXiv: 1609.04802 [cs.CV].

[6] J. Nilsson and T. Akenine-Möller, "Understanding SSIM", 2020. arXiv: 2006.13846 [eess.IV].

[7] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang, "The Unreasonable Effectiveness of Deep Features as a Perceptual Metric", in *Computer Vision and Pattern Recognition Conference (CVPR)*, 2018. arXiv: 1801.03924 [cs.CV].

[8] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition", in *3rd International Conference on Learning Representations, ICLR, San Diego, CA, USA, May 7-9, Conference Track Proceedings*, 2015. arXiv: 1409.1556 [cs.CV].

[9] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks", in *Advances in Neural Information Processing Systems*, volume 25, Curran Associates, Inc., 2012. DOI: https://doi.org/10.1145/3065386.

[10] *Brain Basics: The Life and Death of a Neuron*. [Online]. Available: https://www.ninds.nih.gov/health-information/public-education/brain-basics/brain-basics-life-and-death-neuron.

[11] I. Neutelings, *Neural networks*, May 2022. [Online]. Available: https://tikz.net/neural_networks/.

[12] Y. LeCun, B. Boser, J. S. Denker, *et al.*, "Backpropagation Applied to Handwritten Zip Code Recognition", *Neural Computation*, volume 1, number 4, pages 541–551, 1989. DOI: 10.1162/neco.1989.1.4.541.

[13] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition", *Proceedings of the IEEE*, volume 86, number 11, pages 2278–2324, 1998. DOI: 10.1109/5.726791.

[14] J. C. Duchi, E. Hazan, and Y. Singer, "Adaptive Subgradient Methods for Online Learning and Stochastic Optimization", in *COLT 2010 - The 23rd Conference on Learning Theory, Haifa, Israel, June 27-29, 2010*, A. T. Kalai and M. Mohri, Eds., Omnipress, 2010, pages 257–269. [Online]. Available: https://www.jmlr.org/papers/volume12/duchi11a/duchi11a.pdf.

[15] D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization", in *3rd International Conference on Learning Representations, ICLR, San Diego, CA, USA, May 7-9, Conference Track Proceedings*, 2015. arXiv: 1412.6980 [cs.LG].

[16] S. J. Reddi, S. Kale, and S. Kumar, "On the Convergence of Adam and Beyond", in *6th International Conference on Learning Representations, ICLR, Vancouver, BC, Canada, April 30 - May 3, Conference Track Proceedings*, 2018. arXiv: 1904.09237 [cs.LG].

[17] M. D. Zeiler, "ADADELTA: An Adaptive Learning Rate Method", *CoRR*, 2012. arXiv: 1212.5701 [cs.LG].

[18] T. Dozat, "Incorporating Nesterov Momentum into Adam", 2016. [Online]. Available: https://cs229.stanford.edu/proj2015/054_report.pdf.

[19] I. Loshchilov and F. Hutter, "Decoupled Weight Decay Regularization", in *7th International Conference on Learning Representations, ICLR, New Orleans, LA, USA, May 6-9*, OpenReview.net, 2019. arXiv: 1711.05101 [cs.LG].

[20] G. Hinton, N. Srivastava, and K. Swersky, *Course: Neural Networks for Machine Learning - Lecture 6*, 2014. [Online]. Available: https://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf.

[21] X. Chen, C. Liang, D. Huang, *et al.*, "Symbolic Discovery of Optimization Algorithms", 2023. arXiv: 2302.06675 [cs.LG].

[22] N. M. Nawi, M. R. Ransing, and R. S. Ransing, "An Improved Learning Algorithm Based on The Broyden-Fletcher-Goldfarb-Shanno (BFGS) Method For Back Propagation Neural Networks", in *Sixth International Conference on Intelligent Systems Design and Applications*, volume 1, 2006, pages 152–157. DOI: 10.1109/ISDA.2006.95.

[23] D. J. Montana, L. Davis, *et al.*, "Training feedforward neural networks using genetic algorithms", in *IJCAI*, volume 89, 1989, pages 762–767. [Online]. Available: https://www.ijcai.org/Proceedings/89-1/Papers/122.pdf.

[24] K. Fukushima, "Cognitron: A self-organizing multilayered neural network", *Biological Cybernetics*, volume 20, number 3, pages 121–136, Sep. 1975, ISSN: 1432-0770. DOI: 10.1007/BF00342633.

[25] A. L. Maas, A. Y. Hannun, and A. Ng, "Rectifier nonlinearities improve neural network acoustic models", in *International Conference on Machine Learning*. [Online]. Available: https://ai.stanford.edu/~amaas/papers/relu_hybrid_icml2013_final.pdf.

[26] L. Lu, Y. Shin, Y. Su, and G. E. Karniadakis, "Dying ReLU and initialization: Theory and numerical examples", *Communications in Computational Physics*, volume 28, number 5, pages 1671–1706, Jun. 2020. arXiv: 1903.06733 [stat.ML].

[27] D. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)", in *4th International Conference on Learning Representations, ICLR, San Juan, Puerto Rico, May 2-4, Conference Track Proceedings*, 2016. arXiv: 1511.07289 [cs.LG].

[28] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter, "Self-Normalizing Neural Networks", *CoRR*, 2017. arXiv: 1706.02515 [cs.LG].

[29] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A Simple Way to Prevent Neural Networks from Overfitting", *Journal of Machine Learning Research*, volume 15, number 56, pages 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html.

[30] C. Shorten and T. M. Khoshgoftaar, "A survey on Image Data Augmentation for Deep Learning", *Journal of Big Data*, volume 6, number 1, page 60, Jul. 2019, ISSN: 2196-1115. DOI: 10.1186/s40537-019-0197-0.

[31] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift", *CoRR*, 2015. arXiv: 1502.03167 [cs.LG].

[32] S. Santurkar, D. Tsipras, A. Ilyas, and A. Madry, "How Does Batch Normalization Help Optimization?", 2019. arXiv: 1805.11604 [stat.ML].

[33] G. Yang, J. Pennington, V. Rao, J. Sohl-Dickstein, and S. S. Schoenholz, "A Mean Field Theory of Batch Normalization", 2019. arXiv: 1902.08129 [cs.NE].

[34] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks", in *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, series Proceedings of Machine Learning Research, volume 9, May 2010, pages 249–256. [Online]. Available: https://proceedings.mlr.press/v9/glorot10a.html.

[35] K. He, X. Zhang, S. Ren, and J. Sun, "Delving Deep into Rectifiers: Surpassing Human-Level Performance on ImageNet Classification", 2015. arXiv: 1502.01852 [cs.CV].

[36] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition", in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pages 770–778. arXiv: 1512.03385 [cs.CV].

[37] Y. Tai, J. Yang, and X. Liu, "Image Super-Resolution via Deep Recursive Residual Network", in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pages 2790–2798. DOI: 10.1109/CVPR.2017.298.

[38] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning", 2018. arXiv: 1603.07285 [stat.ML].

[39] E. Shelhamer, J. Long, and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation", *CoRR*, 2016. arXiv: 1605.06211 [cs.CV].

[40] J. Yu, Z. Lin, J. Yang, X. Shen, X. Lu, and T. Huang, "Free-Form Image Inpainting With Gated Convolution", in *IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pages 4470–4479. arXiv: 1806.03589 [cs.CV].

[41] D. Pathak, P. Krähenbühl, J. Donahue, T. Darrell, and A. A. Efros, "Context Encoders: Feature Learning by Inpainting", in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pages 2536–2544. arXiv: 1604.07379 [cs.CV].

[42] R. A. Yeh, C. Chen, T. Y. Lim, A. G. Schwing, M. Hasegawa-Johnson, and M. N. Do, "Semantic Image Inpainting with Deep Generative Models", in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pages 6882–6890. arXiv: 1607.07539 [cs.CV].

[43] J. Yu, Z. Lin, J. Yang, X. Shen, X. Lu, and T. S. Huang, "Generative Image Inpainting with Contextual Attention", in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pages 5505–5514. arXiv: 1801.07892 [cs.CV].

[44] M. Bertalmio, A. Bertozzi, and G. Sapiro, "Navier-stokes, fluid dynamics, and image and video inpainting", in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, 2001. DOI: 10.1109/CVPR.2001.990497.

[45] A. Telea, "An Image Inpainting Technique Based on the Fast Marching Method", *Journal of Graphics Tools*, volume 9, number 1, pages 23–34, 2004. DOI: 10.1080/10867651.2004.10487596.

[46] C. Barnes, E. Shechtman, A. Finkelstein, and D. B. Goldman, "PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing", *ACM Transactions on Graphics (Proc. SIGGRAPH)*, volume 28, number 3, August 2009. [Online]. Available: https://gfx.cs.princeton.edu/pubs/Barnes_2009_PAR.

[47] G. Liu, F. A. Reda, K. J. Shih, T.-C. Wang, A. Tao, and B. Catanzaro, *Image Inpainting for Irregular Holes Using Partial Convolutions*, 2018. arXiv: 1804.07723 [cs.CV].

[48] R. Suvorov, E. Logacheva, A. Mashikhin, *et al.*, "Resolution-robust Large Mask Inpainting with Fourier Convolutions", in *IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, 2022, pages 3172–3182. arXiv: 2109.07161 [cs.CV].

[49] W. Quan, R. Zhang, Y. Zhang, Z. Li, J. Wang, and D.-M. Yan, "Image Inpainting With Local and Global Refinement", *IEEE Transactions on Image Processing*, volume 31, pages 2405–2420, 2022. DOI: 10.1109/TIP.2022.3152624.

[50] J. Liu, M. Gong, Z. Tang, A. K. Qin, H. Li, and F. Jiang, "Deep Image Inpainting With Enhanced Normalization and Contextual Attention", *IEEE Transactions on Circuits and Systems for Video Technology*, volume 32, number 10, pages 6599–6614, 2022. DOI: 10.1109/TCSVT.2022.3175171.

[51] K. Nazeri, E. Ng, T. Joseph, F. Z. Qureshi, and M. Ebrahimi, "EdgeConnect: Generative Image Inpainting with Adversarial Edge Learning", 2019. arXiv: 1901.00212 [cs.CV].

[52] A. Krizhevsky, "Learning Multiple Layers of Features from Tiny Images", Tech. Rep., 2009.

[53] T. Lin, M. Maire, S. J. Belongie, *et al.*, "Microsoft COCO: Common Objects in Context", *CoRR*, 2014. arXiv: 1405.0312 [cs.CV].

[54] O. Ronneberger, P. Fischer, and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation", in *Medical Image Computing and Computer-Assisted Intervention – MICCAI*, Springer International Publishing, 2015, pages 234–241, ISBN: 978-3-319-24574-4. arXiv: 1505.04597 [cs.CV].

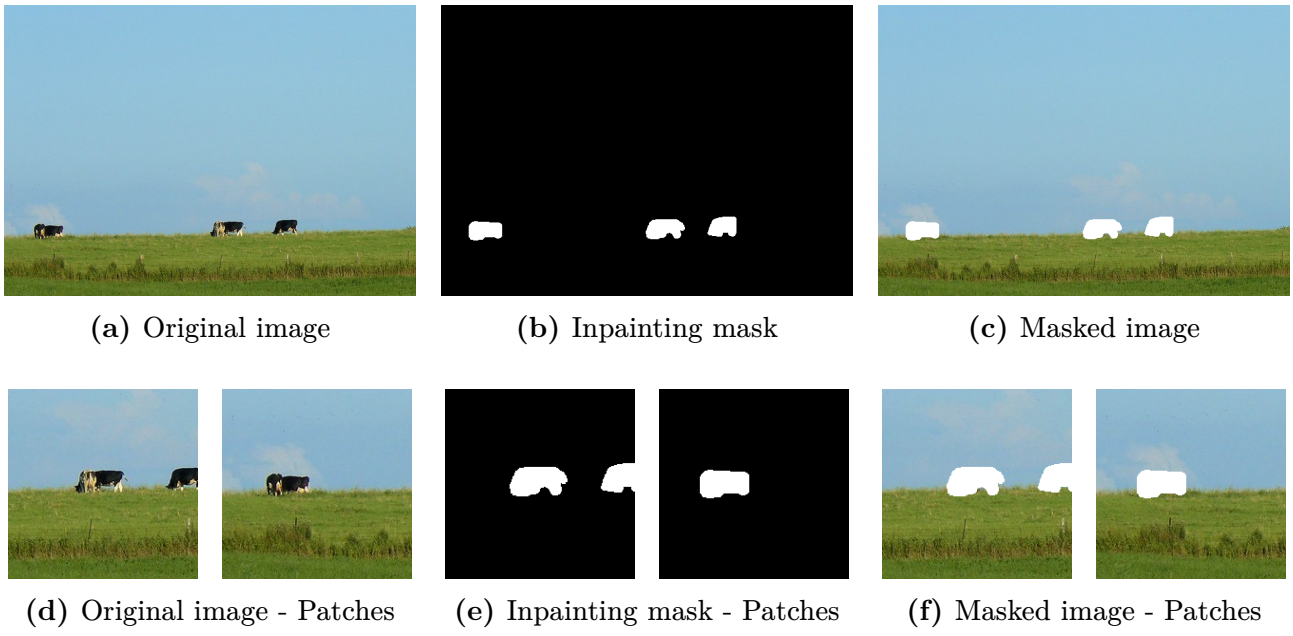
[55] H. Zhao, O. Gallo, I. Frosio, and J. Kautz, "Loss Functions for Image Restoration with Neural Networks", *IEEE Transactions on Computational Imaging*, volume 3, number 1, pages 47–57, 2017. arXiv: 1511.08861 [cs.CV].

Appendix A
Source code

Further details of the project implementation, including the complete source code in Python, relevant data files and supplementary resources, can be found in the official GitHub repository for this project. To access this repository, please follow the URL below:

`github.com/cristiancristea00/Image-Inpainting`

Appendix B
Additional results



(a) Original image **(b)** Inpainting mask **(c)** Masked image

(d) Original image - Patches **(e)** Inpainting mask - Patches **(f)** Masked image - Patches

Figure B1: Masking for an image with inpainting applied at a transition patch

(a) Original image **(b)** Inpainting mask **(c)** Masked image

(d) Original image - Patches **(e)** Inpainting mask - Patches **(f)** Masked image - Patches

Figure B2: Masking for an image with inpainting applied on a non-uniform local patch

(a) Navier-Stokes

(b) Telea

(c) Navier-Stokes - Patches

(d) Telea - Patches

(e) PatchMatch

(f) GatedResNet

(g) PatchMatch - Patches

(h) GatedResNet - Patches

Figure B3: Comparison for an image with inpainting applied at a transition patch

(a) Navier-Stokes

(b) Telea

(c) Navier-Stokes - Patches

(d) Telea - Patches

(e) PatchMatch

(f) GatedResNet

(g) PatchMatch - Patches

(h) GatedResNet - Patches

Figure B4: Comparison for an image with inpainting applied on a non-uniform local patch