Image text removal using inpainting with Generative Adversarial Networks

Department of Computer Science and Engineering

University of Ioannina

Ioannis Georvasilis

Supervisor: Christophoros Nikou

Co-supervisor: Giorgos Sfikas

June 2021

Abstract

In this work, we study the problem of text removal using inpainting techniques with Generative Adversarial Networks. It is a very challenging problem due to the fact that can be divided into 2 separate, that of text localization and that of inpainting the localized texts. We try several approaches to the problem, starting with the well-studied supervised training to study their results and improvements. We then try to take advantage of great inpainting success of conditional adversarial techniques using a cGAN. Finally, we use additive text localization networks to contribute to more realistically inpainting by facilitating the text localization task. Despite the fact that the additive networks significantly improved the inpainting results, we investigate whether the network's depth shrinkage by integrating attention gates (AG) will offer the same impressive results. Although, elimination of explicit external localisation modules does not offer more realistic inpainting, still contribute to localization task and the results are worth studying.

Περιληψη

Η παρούσα εργασία πραγματεύεται το πρόβλημα της αφαίρεσης κειμένου με τεχνικές ενδοχρωματισμού, κάνοντας χρήση των Παραγωγικώνν Αντιπαραθετικών Δικτύων. Το πρόβλημα παρουσιάζει πολλές προκλήσεις καθώς μπορεί να διαχωριστεί σε 2 επιμέρους προβλήματα, αυτό της ανίχνευσης του κειμένου και εκείνο του ενδοχρωματισμού των ανιχνευμένων κειμένων. Δοκιμάζουμε διάφορες προσεγγίσεις για το πρόβλημα, ξεκινώντας με την ευρέως μελετημένη επιβλεπόμενη διακασία μάθησης, ώστε να μελετήσουμε τα αποτελέσματά τους και τις βελτιώσεις που μπορούν να εφαρμοστούν. Στη συνέχεια προσπαθούμε να επωφεληθούμε απο τη μεγάλη επιτυχία της υπο συνθήκης αντιπαραθετικής διαδικασίας μάθησης χρησιμοποιώντας ένα Υπο-συνθήκη Παραγωγικό Αντιπαραθετικό Δίκτο, με τα αποτελέσματα να επιβεβαιώνουν τον ισχυρισμό. Τέλος, προκειμένου να απαλλαχθούμε απο το πρόβλημα του κειμενικού εντοπισμού, χρησιμοποιούμε πρόσθετα δίκτυα εντοπισμού κειμένου, ώστε να συμβάλουμε σε πιο ρεαλιστικό ενδοχρωματισμό. Παρά το γεγονός ότι τα πρόσθετα δίκτυα βελτίωσαν σημαντικά τα αποτελέσματα, διερευνούμε κατά πόσο η συρρίκνωση του βάθους του δικτύου ενσωματώνοντας πύλες προσοχής θα προσφέρει τα ίδια εντυπωσιακά αποτελέσματα. Παρόλο που η ενσωμάτωση αυτή δεν προσφέρει πιο ρεαλιστικο ενδοχρωματισμό, συμβάλλει σημαντικά στο διαδικασία εντοπισμού των κειμενικών περιοχών και τα αποτελέσματα σίγουρα αξίζουν περεταίρω μελέτη.

Acknowledgements

First of all I would like to thank my supervisor, professor Christophoros Nikou and my co-supervisor, research associate and visiting lecturer Giorgos Sfikas for their guidance and their continued motivation throughout the course of this work. Their suggestions, insights and answers to my questions have been invaluable for this work and are immensely appreciated. I have highly benefited from this co-operation and I am grateful for working under his direction. I would like also to thanks professor Aristidis Likas, for his course on Computational Intelligence thus by attending this course motivated me to work with neural networks and to undertake this project. Thanks are also extended to the academic faculty of the Computer Science and Engineering Department of the University of Ioannina, for equipping me with significant knowledge and contributing towards shaping my scientific way of thinking during the years of my undergraduate studies. At this point, I would also like to thank my dear friend and fellow student Andreas Sideras for all these years of collaboration and support. Last but not least, I would like to express my sincerest gratitude to my family who wholeheartedly supported me in this endeavor.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Image in-painting is the process of creating alternative structures and textures in the missing areas of an image in order to restore the full visual effects.

It is an essential component of many image editing operations, such as image target removal, image restoration, and image denoising. The final image in-painting result is determined by quality of filling damaged region and the primary challenge of image in-painting is to create the feasible structure with realistic texture.

Sometimes an image may contain text embedded on to it, which should be removed for aesthetic and other reasons. In current dissertation we will deal with text removal using inpainting, by experimenting on deep learning techniques which have achieved impressive results in recent years.

This thesis is divided into four chapters. In Chapter 2, we discuss a few pioneering architectures as well as a brief overview of the key building components of convolutional neural networks. In Chapter 3, we look into generative adversarial networks, beginning with the definition and progressing to more complicated ones that were used in this work. In chapter 4, we formulate the problem of text removal using inpainting and we outline our work based on networks and outlines from chapters 2 and 3. Finally, we present our results and compare them with those of recent work on this subject.

Chapter 2

Convolutional Networks

Convolutional Neural Network (CNN) is a type of deep learning model for processing data that has a grid pattern (i.e images), which is inspired by the organization of animal visual cortex and designed to automatically and adaptively learn spatial hierarchies of features, from low- to high-level patterns [2].

Convolutional Neural Network (CNN) is a mathematical construct that is typically composed of three types of layers - building blocks: convolution, pooling, and fully connected layers. The first two perform feature extraction, whereas the third, a fully connected layer, maps the extracted features into final output (i.e classification). A convolution layer plays a key role in CNN, which is composed of a stack of mathematical operations, such as convolution, a specialized type of linear operation. In digital images, pixel values are stored in a two-dimensional grid and a small grid of parameters called kernel, an optimizable feature extractor, is applied at each image position, which makes CNNs highly efficient for image processing, since a feature may occur anywhere in the image. As one layer feeds its output into the next layer, extracted features can hierarchically and progressively become more complex. The process of optimizing parameters such as kernels is called training, which is performed so as to minimize the difference between outputs and ground truth labels through an optimization algorithm called backpropagation and gradient descent, among others.

2.1 Convolutional Layer

A convolution layer is a fundamental component of the CNN architecture that performs feature extraction, which typically consists of a combination of linear and nonlinear operations, i.e., convolution operation and activation function.

Convolution is a specialized type of linear operation used for feature extraction, where a small array of numbers, called a kernel, is applied across the input, which is an array of numbers, called a **tensor**. An element-wise product between each element of the kernel and the input tensor is calculated at each location of the tensor and summed to obtain the output value in the corresponding position of the output tensor, called a **feature map** or **activation map**. This procedure is repeated applying multiple kernels to form an arbitrary number of feature maps, which represent different characteristics of the input tensors; different kernels can, thus, be considered as different feature extractors.

The input to a CNN is a tensor with the following shape: **(number of inputs) x (input height) x (input width) x (input channels)**. After passing through a convolutional layer, the image is abstracted to a feature map, with the following dimensions: **(number of inputs) x (feature map height) x (feature map width) x (feature map channels)**. In general, a convolutional layer within a CNN includes the following attributes:

- **Convolutional filters/kernels** defined by a width and height .
- **The number of input and output channels** (hyper-parameters). The number of input channels in one layer must equal the number of output channels (also known as depth) in that layer.
- **Padding** refers to the use of zeros around the input volume. It enables the application of the filter to the image's edges. It also allows for the spatial size of the output volumes to be controlled.
- **Stride**: The number of units or pixels moved by the kernel on each convolution is referred to as the stride. A longer stride results in a lower output volume.

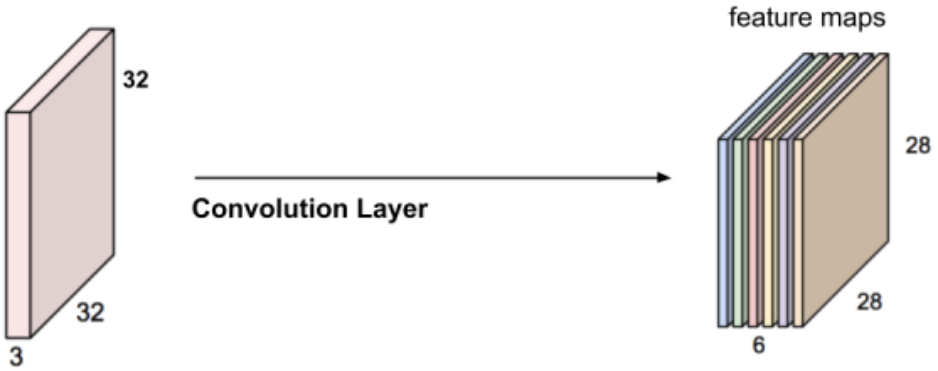

Figure 2.1: Applying 6 5 × 5 filters leads to an output volume of size 28 × 28 × 6.

Convolutional layers convolve the input and pass the output to the following layer. This is analogous to a neuron's response to a specific stimulus in the visual cortex. Each convolutional neuron only processes information for its own receptive field. Although fully connected feed-forward neural networks can be used to learn features and classify data, this architecture is problematic for bigger inputs such as high dimension images. Due to the vast input size, where each pixel is an important input characteristic, it would require a relatively high number of neurons, even in a shallow architecture.

The **transposed convolutional layer** unlike the convolutional layer, is upsampling in nature. Transposed convolutions (or deconvolution operations) are usually used in networks that must reconstruct an image. The word transpose means to cause two or more things to switch places with each other, and in the

context of convolutional neural networks, this causes the input and the output dimensions to switch. When downsampling and upsampling techniques are applied to transposed convolutional layers, their effects are reversed. The reason for this is for a network to be able to use convolutional layers to compress the image, then transposed convolutional layers with the exact same downsampling and upsampling techniques to reconstruct the image.

2.2 Pooling Layer

Along with standard convolutional layers, convolutional networks may add local and/or global pooling layers. Its purpose is to progressively reduce the spatial size of the representation. By combining the outputs of neuron clusters at one layer into a single neuron in the following layer, pooling layers reduce the dimensionality of data. This function aggregates feature map activations in a local neighborhood by replacing them with a summary of their data. Two main pooling operations are used in convolutional networks:

- **max pooling**, where the maximum value is computed from small local neighborhoods of feature maps that defined
- **average pooling**, where the average value of the local neighborhood is computed.

There are two main advantages. First, it gradually minimizes the size of the feature map, as well as the number of parameters and computation in the network, hence controlling over-fitting. Furthermore, it contributes to the representation being insensitive to tiny translations in the input. With a little translation, the max pooling layer will identify the same features on both the original and translated input.

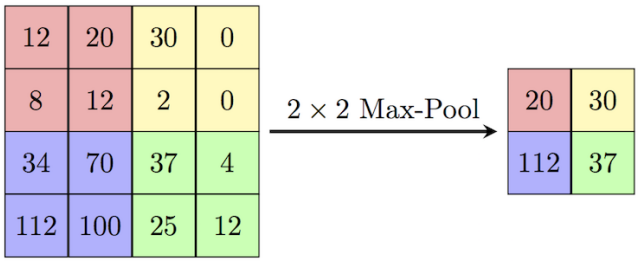

Figure 2.2: Max pooling operation.

2.3 Non-Linearity Layer

The outputs of a linear operation such as convolution are then passed through a nonlinear activation function.

Although smooth nonlinear functions, such as sigmoid or hyperbolic tangent (tanh) function, were used previously because they are mathematical representations of a biological neuron behavior, the most common nonlinear activation functions used presently are the rectified linear unit (ReLU) and (leaky ReLU).

$$\mathbf{Sigm}(x) = \frac{1}{1 + e^{-x}} \tag{2.1}$$

$$\mathbf{Tanh}(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{2.2}$$

$$\mathbf{Relu}(x) = \max(0, x) \tag{2.3}$$

$$\mathbf{LeakyRelu}(x) = \max(s \cdot x, x) \tag{2.4}$$

where s denotes the *slope* (common slope between 0.1 and 0.001).

Two additional major benefits of ReLUs over sigmoids, are sparsity and a reduced likelihood of vanishing gradient. The constant gradient of ReLUs results in faster learning. The ReLU is linear for $a > 0$ and zero for $a < 0$. It results to produce neurons with sparse activities, which means that neurons are often exactly zero. This occurs because of the ReLU's domain: activations below zero will produce a zero, that is half its domain.

2.4 Batch Normalization

Training Deep Neural Networks is complicated by the fact that the distribution of each layer's inputs changes during training, as the parameters of the previous layers change. This slows down the training by requiring lower learning rates and careful parameter initialization, and makes it notoriously hard to train models with saturating nonlinearities.

Batch Normalization is a supervised learning technique that converts intermediate layer outputs of a neural network into a standard format. This effectively 'resets' the distribution of the previous layer's output so that it can be handled more efficiently by the subsequent layer.

For a layer of the network with d-dimensional input, $x = (x^{(1)}, x^{(2)}, ..., x^{(d)})$, each dimension of its input is then normalized (i.e. re-centered and re-scaled) separately,

$$\hat{x_i}^{(k)} = \frac{x_i^{(k)} - \mu_B^{(k)}}{\sqrt{\sigma_B^{(k)^2} + \epsilon}} \tag{2.5}$$

where $\mu_B^{(k)}$ and $\sigma_B^{(k)^2}$ are the per-dimension mean and variance, respectively.

ϵ is added in the denominator for numerical stability and is an arbitrarily small constant. The resulting normalized activation $x_i^{(k)}$ have zero mean and unit variance, if ϵ is not taken into account.

This method results in faster learning rates since normalization assures that no activation value is too high or too low, and it also allows each layer to learn independently of the others. Normalizing inputs lowers the "dropout" rate, or the amount of data lost between processing layers. As a result, the network's accuracy improves dramatically.

2.5 Dropout

Deep neural networks trained on limited datasets have a tendency to overfit the training data.

This causes the model to learn the statistical noise in the training data, resulting in poor performance when the model is tested on new data, such as a test dataset. Overfitting increases generalization error.

Drop Nodes at Random: Dropout is a regularization method that approximates concurrent training of a large number of neural networks with various designs. During training, some layer outputs are rejected or "dropped out" at random. This has the effect of making the layer appear and be regarded as if it had a different number of nodes and connectedness to the preceding layer.

Dropout is implemented per-layer in a neural network. It can be used with most types of layers, such as dense fully connected layers, convolutional layers, and recurrent layers such as the long short-term memory network layer.

Dropout may be implemented on any or all hidden layers in the network as well as the visible or input layer. It is not used on the output layer.

2.6 Training procedure

Training a convolutional neural network is substantially the same as training any other feedforward neural network, and it employs the backpropagation technique.

The network is first constructed with random values in all of its weights and biases. The training examples are a collection of tuples of images and targets. Each training image is processed by the entire network, and the last layer produces a vector that is ideally identical to the target.

2.6.1 Loss Function

In machine learning (ML), the main purpose rely on minimizing or maximizing a function called "objective function" $J(\theta)$. The group of functions that are minimized are called "loss functions". Loss function is used as measurement of how good a prediction model does in terms of being able to predict the expected output.

A loss function is an error metric, a way to calculate the inaccuracy of predictions. The aim of deep learning models is to minimize this loss function value, and the process of minimizing the loss function value is called optimization.

2.6.2 Back Propagation

Backpropagation is an algorithm which helps neural networks to learn their parameters, mostly from errors in predictions.

Backpropagation computes the gradient of the loss function with respect to the network weights for a single input–output example more efficiently than a naive direct computation of the gradient with respect to each weight individually while training a neural network. Algorithm is computing the gradient of the loss function with respect to each weight using the chain rule, one layer at a time,

iterating backward from the last layer to prevent unnecessary calculations of intermediate terms in the chain rule (dynamic programming).

Learning rate is the step size of each iteration in the gradient descent or other optimazation algorithms. If the learning rate is too small, convergence will take a long time to happen, but if the learning rate is too large, there might be no convergence at all.

2.6.3 Gradient Descent and its variants

Gradient Descent is an optimization procedure that adjusts the neural network's internal weights in order to minimize the loss function value. After each iteration, the gradient descent method aims to reduce the loss function value by adjusting weights until additional adjustments yield little or no change in the loss function value, a process known as convergence.

For the complete training dataset, **Batch Gradient Descent** computes the gradient of the cost function to the parameters θ. Batch gradient descent can be highly slow and intractable for datasets that don't fit in memory because is needed to calculate the gradients for the entire dataset to execute just one update. Batch gradient descent also prevents us from updating our model online. Furthermore, for big datasets, Batch gradient descent performs redundant calculations since it recomputes gradients for similar cases before each parameter update.

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta) \tag{2.6}$$

Stochastic Gradient Descent (SGD) in contrast, which is a variant of gradient descent, performs a parameter update for each training example and label.

SGD does away with this redundancy by performing one update at a time. It is therefore usually much faster and can also be used to learn online. SGD performs frequent updates with a high variance that cause the objective function to fluctuate heavily.

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)}) \tag{2.7}$$

Momentum: SGD has difficulty navigating ravines, which are widespread around local optima and are regions where the surface curves considerably more steeply in one dimension than another. In these scenarios, SGD oscillates across the slopes of the ravine while only making hesitant progress along the bottom towards the local optimum. Momentum is a method that helps accelerate SGD in the relevant direction and dampens oscillations. It does this by adding a fraction γ of the update vector of the past time step to the current update vector:

$$v_t = \gamma \cdot v_{t-1} + \eta \cdot \nabla_\theta J(\theta) \tag{2.8}$$

$$\theta = \theta - v_t \tag{2.9}$$

Adaptive Moment Estimation (Adam) is another method that computes adaptive learning rates for each parameter.

Adam also keeps an exponentially decaying average of past gradients similar to momentum. Whereas momentum can be seen as a ball running down a slope, Adam behaves like a heavy ball with friction, which thus prefers flat minima in the error surface. We compute the decaying averages of past and past squared gradients m_t, u_t respectively as follows:

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t$$

$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2$$

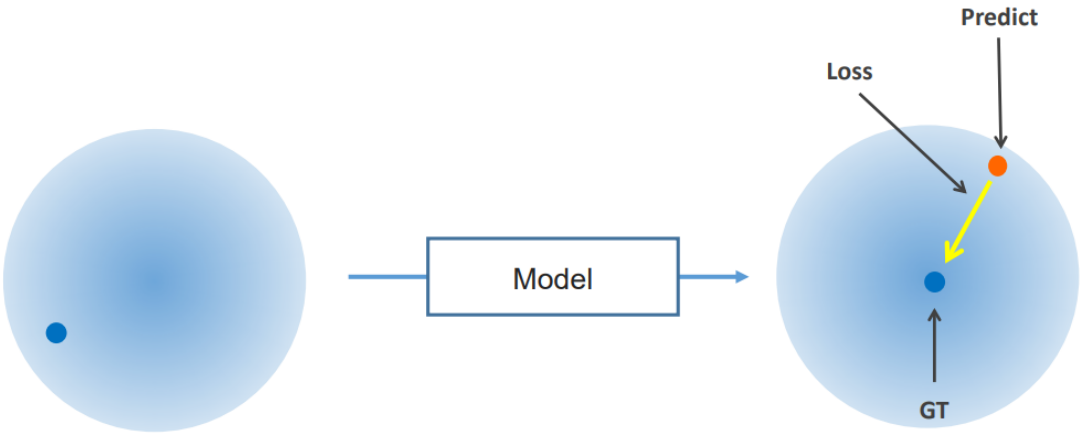

Figure 2.3: To find the local minimum of a function using gradient descent, we must take steps proportional to the negative of the gradient (move away from the gradient) of the function at the current point.

2.6.4 Noise2noise Training

Traditional deep learning image denoising relies on the availability of pairs of corrupted-clean images (xi, yi). A neural network that trained to predict y_i from x_i, may be used to denoise new noisy images measurement since the new data come from the same distribution. Because of that, we expect denoising similar to that observed during training. In other words, we train the model saying 'when you see a x_i, you have to show y_i'. In the context of learning to remove image corruptions, the simplest form of which is just noise, independent from pixel per pixel, the learning process entails training a model by showing many pairs of noisy images and the corresponding clean images and try to learn the mapping that when given a noisy image outputs the corresponding.

But what if clean targets (y_i) are difficult to obtain or even worst do not they exist? Authors [18] shows that if noise is zero mean and pixel per pixel independent, arises the property that the underlying clean image is the expectation of infinitely many noisy realisations. It basically just means that the color of every pixel in every noisy realization of this image is correct on average. So they proposed to train the deep network with the classic training process, feeding with noisy targets but replacing clean targets (that they do not exist) with noisy targets, that is different noisy realizations. After infinite steps, by showing lots and lots of noisy pairs, network will output the expectation of distribution of all possible noisy targets, that correspond to clean image.

Training procedure: When we train in the standard manner, when we have a clean target for our training example, the model outputs a value and essentially since the SGD takes place, loss gives us a gradient with respect to the parameters. Gradient is 'responsible' for tweeting the network parameters to move 'closer' to clean target.

On the other hand, if the loss computations arise from targets that are correct on average, then in terns out that the average of all of these gradients from all of training examples give us a false average gradient. But on average, because noisy targets drawn in from such a distribution that their expectation is, their gradient will actually point towards the right direction.

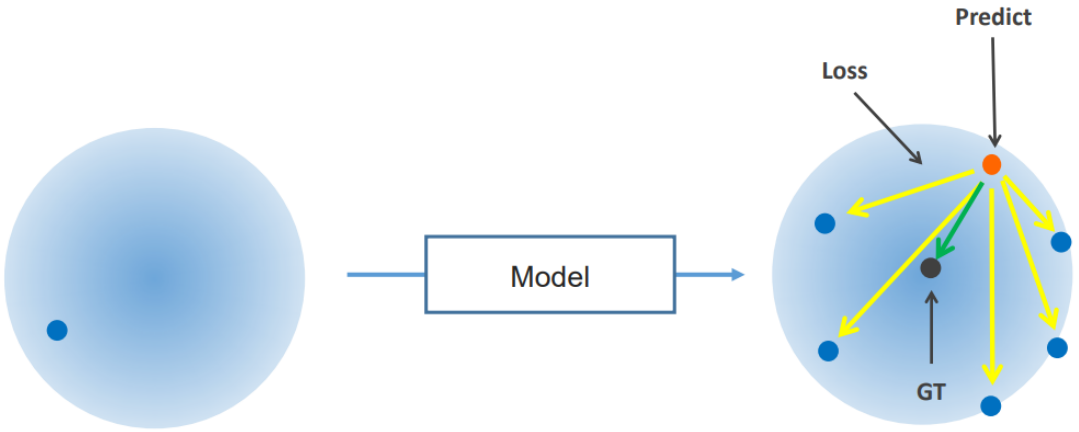

Figure 2.4: The expectation of distribution of all possible noisy targets correspond to clean image.

2.7 Deep CNN architectures

Autoencoders, a type of artificial neural network, have been used successfully in image processing, particularly in image reconstruction. The goal of image reconstruction is to create a new set of images that are comparable to the original input images. If given a series of noisy or incomplete images, this aids in producing noise-free or complete images.

2.7.1 Convolutional Autoencoder

Convolutional Autoencoders are a type of Convolutional Neural Network used for unsupervised learning of convolution filters. They are commonly used in image reconstruction to decrease reconstruction mistakes by learning the best filters. They can be trained in this task and then applied to any input to extract features.

An autoencoder is composed of two major components: an **encoder** that converts the input into code and a **decoder** that converts the code into a reconstruction of the input.

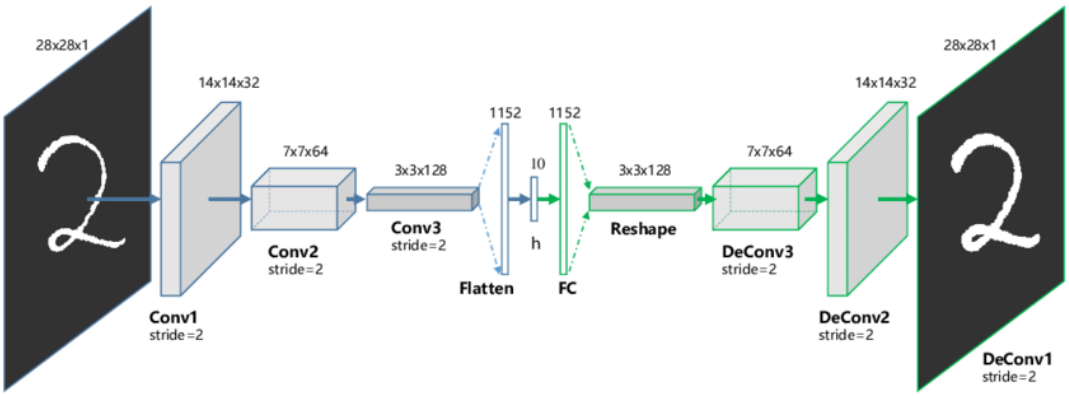

Figure 2.5: The block diagram of a Convolutional Autoencoder.

Convolutional autoencoders, forwards the input through consecutive convolutional blocks where at the encoder's part a convolutional block contains a convolutional, a non-linear, a batch normalization and a dropout layer. In a similar way but aiming to decode the forwarded input, what differentiate a deconvolutional block is that replaces convolutioan layer with transposed convolutional layer. What separates encoder and decoder is the flatten layers. They follow a typical MLP Architecture, since the feature maps are 'flattened' to an 1-dimensional vectors.

2.7.2 U-Net

The U-net architecture introduced and developed by Ronneberger et al. (2015) [25] was one of the first convolutional networks designed specifically for biomedical image analysis. This network aimed to address two domain-specific issues in medical image segmentation. The first issue is a scarcity of large datasets in this domain. The goal of this architecture is to produce competitive segmentation results with a small amount of training data. Traditional feed-forward

convolutional neural networks with fully connected layers at the end have a large number of parameters to learn, hence require large datasets. Encoder-decoder architectures like U-net have proven to be more effective even with small datasets, because the fully-connected layer is replaced on the decoder side with a series of up convolutions, which still has learnable parameters but far fewer than a fully-connected layer. The second issue that the U-net architecture addresses is accurately capturing context and localizing lesions at various scales and resolutions.

Architecture Details: The U-Net architecture consists of two portions, as shown in Figure . The left side is a contracting path, in which successive convolutions are used to increase resolutions and learn features. The right side is an expanding path of a series of upsampling operations. Low resolution features from the contracting path are combined with upsampled outputs from the expanding path. This is done via skip connections and is significant in helping gain spatial context that may have been lost while going through successive convolutions on the contracting path. The upsampling path utilizes a large number of feature channels, which enables the effective propagation of context information to lower resolution layers on the contracting path, allowing for more precise localization. This motivated the authors to make the expanding path symmetric to the contracting path, forming a U-shaped architecture.

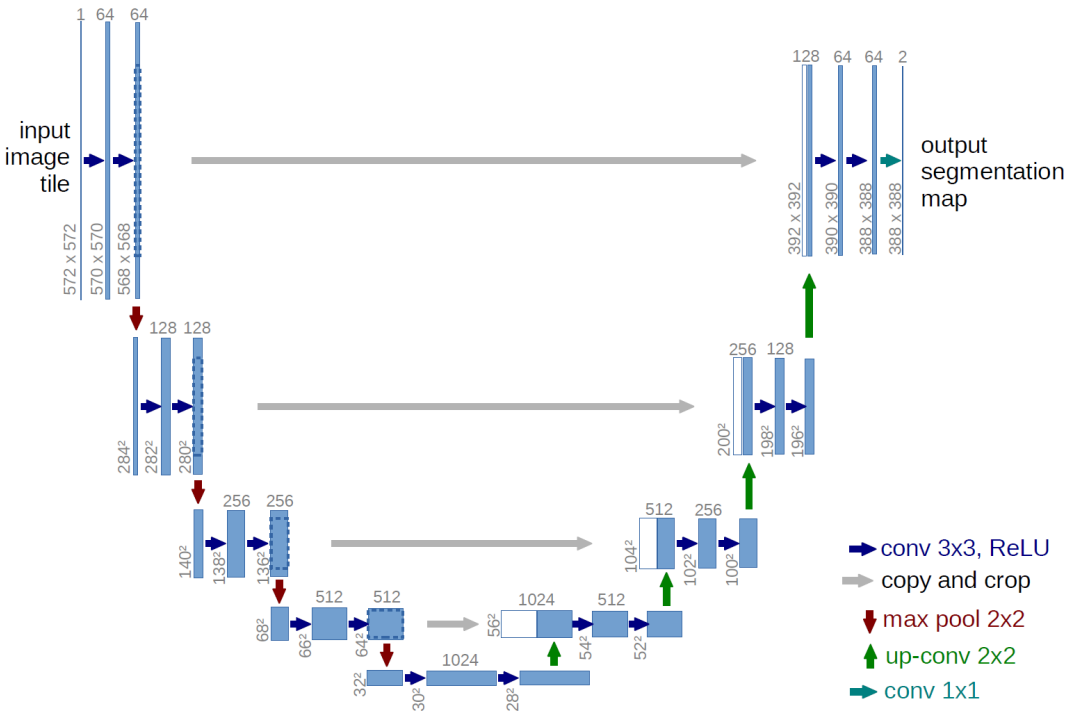

Figure 2.6: U-Net Architecture.

2.7.3 ResNet

The well-known vanishing gradient is one of the challenges that ResNets handle. This is due to the fact that when the network is deep enough, the gradients from which the loss function is derived simply drop to zero after numerous chain rule applications. As a result, the weights never update their values, and hence no learning occurs. Gradients in ResNets can flow straight via the skip connections from later layers to initial filters.

ResNet [11] tackles the degradation issue by introducing the residual network, which is built up of blocks known as residual blocks. Instead of attempting to learn an underlying mapping from x to $H(x)$, the authors claim that learning the difference between them, known as the residual, denoted as $F(x) = H(x) - x$, will be more successful. The residual may then be added to the input $H(x) = F(x) + x$ to compute the original mapping $H(x)$. Each ResNet block is made up of a sequence of layers and a shortcut link that connects the block's input to its output. The gradient signal in ResNets could travel back directly to early layers via the shortcut connections, alleviating the deep neural networks from the vanishing gradients suffer. Because these connections contribute no new parameters or computational complexity, it is possible to design networks with hundreds or even thousands of layers and yet achieve good performance.

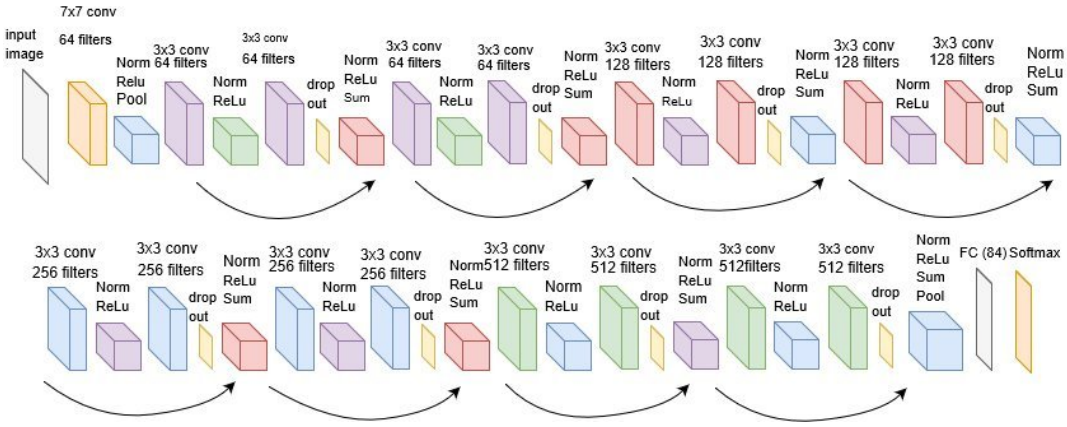

Figure 2.7: Residual network architecture as proposed by [1].

2.8 Attention Mechanisms

In the context of image segmentation, **attention** is a method of highlighting just the necessary activations during training. This eliminates computational resources that would otherwise be spent on irrelevant for the task activations, giving the network more generalisation power. In essence, the network may pay "attention" to certain areas of the image. Which part of features is more important than others depends on the context and is learned through training data by gradient descent.

There are two kinds of attention: hard and soft.

Hard attention works by highlighting relevant portions of an image through cropping or iterative region proposal. Because hard attention can only select one part of an image at a time, it is non-differentiable and requires reinforcement learning to train. Because it is non-differentiable, it means that the network may either pay "attention" to a specific region in an image or not, with no in-between. As a result, ordinary backpropagation is impossible, and Monte Carlo sampling [8] is required to calculate the accuracy at various stages of backpropagation.

Soft attention works by assigning different parts of the image with distinct weights. Those of high significance are given a higher weight, whereas areas of low relevance are given a lower weight. As the model is trained, greater emphasis is placed on regions with higher weights. These weights, unlike hard attention, can be given to several parts in the image. It remains a differentiable process, can be trained with standard backpropagation and the weighting is also trained such that the model gets better at deciding which parts to pay attention to.

They are used in several machine learning models, including natural language processing and computer vision. Furthermore, transformer networks make significant use of attention mechanisms in order to gain high expressive power and also can help computer vision systems based on Convolutional Neural Networks (CNNs).

Figure 2.8: Visualisation of soft attention weights.

2.8.1 Attention U-net

In normal CNN designs, the feature-map grid is gradually downsampled to capture a suitably broad receptive field and hence meaningful contextual information. In this way, features on the coarse spatial grid level model location and relationship between tissues at global scale.

Although, capturing locations of text instances with high form diversity remains tough. In order to improve the text inpainting, some frameworks [27, 29] rely on already established object localisation models to simplify the task into

separate localisation and subsequent inpainting steps. Authors [21], demonstrate that the same objective can be achieved by integrating **Attention Gates** (Soft Attention) in a standard CNN model. This does not necessitate the training of several models or the addition of a huge number of extra model parameters. Unlike the localisation strategy used in multi-stage CNNs, AGs suppress feature responses in irrelevant background areas without the need to trim a Region Of Interest between networks.

The spatial information produced during upsampling in the expanding path is inaccurate. As it mentioned before, in order to address this issue, the U-Net employs skip connections that mix spatial information from the downsampling path with spatial information from the upsampling path. However, because feature representation is weak in the initial layers, this results in many redundant low-level feature extractions.

Implementing soft attention at the skip connections will actively suppress activations in irrelevant regions, minimizing the quantity of redundant features brought over. The attention gates introduced by Oktay [26] uses additive soft attention. The attention gate takes in two inputs, vectors x and g, while g is taken from the next lowest layer of the network and also has smaller dimensions and better feature representation, given that it comes from deeper into the network.

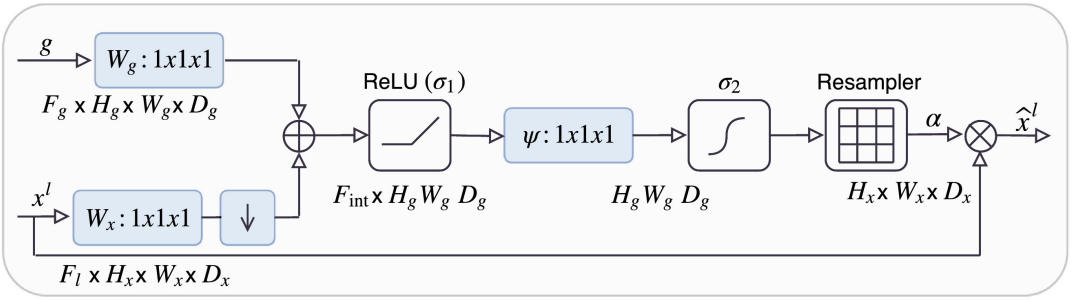

Figure 2.9: The schematics of the additive attention gates.

In the figure 2.9, vector x would have dimensions of (N, N, N) (filters x height x width) and vector g would be $(\frac{N}{2}, \frac{N}{2}, \frac{N}{2})$. Vector x undergoes a strided convolution, resulting in dimensions of $(N, \frac{N}{2}, \frac{N}{2})$, while vector g undergoes a 1x1 convolution, resulting in dimensions of $(N, \frac{N}{2}, \frac{N}{2})$. Element-wise summation process that follows, results in aligned weights becoming larger while unaligned weights become relatively smaller. The dimensions are reduced to $(1, \frac{N}{2}, \frac{N}{2})$ after passing through a ReLU activation layer and a 1x1 convolution. This vector is sent via a sigmoid layer, which scales it between $[0, 1]$, yielding the attention coefficients (weights), with coefficients closer to 1 indicating more significant features.

The attention coefficients are upsampled to the original dimensions (N, N) of the x vector using trilinear interpolation. The attention coefficients are multiplied element-wise to the original x vector, scaling the vector according to relevance. This is then passed along in the skip connection as normal.

During training progresses, the network learns to focus on the desired location. Because the attention gate is differentiable, it may be trained during

backpropagation, which means the attention coefficients get better at highlighting relevant regions.

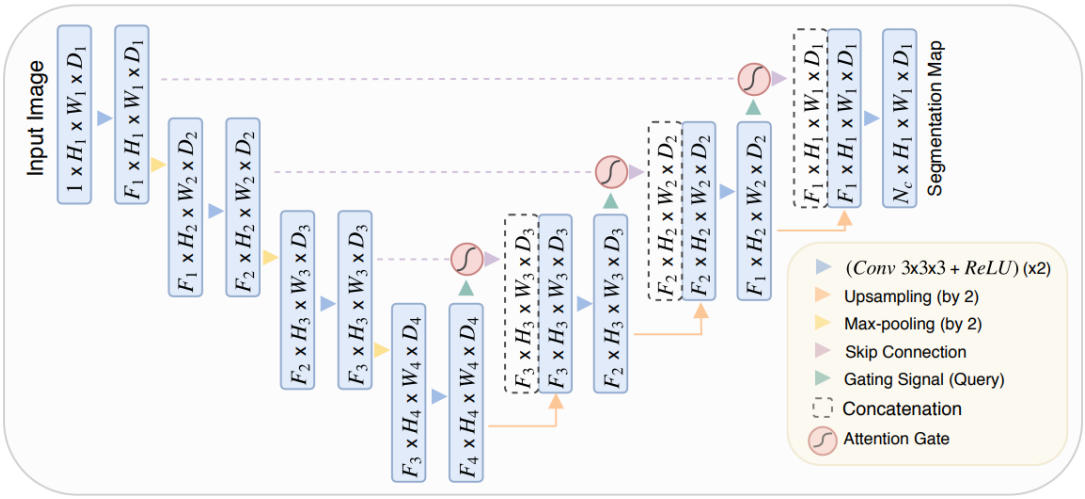

Figure 2.10: Attention U-net architecture as proposed by authors

Chapter 3

Generative Adversarial Networks

Generative Adversarial Networks (GANs), are a generative model that employs deep learning networks such as convolutional neural networks. It is a class of machine learning frameworks designed by Ian Goodfellow and his colleagues in 2014 [9].

Generative modeling is an unsupervised machine learning task that requires automatically learning the regularities or patterns of data such that the model may be used to produce or output new instances that could have been drawn from the original dataset.

GANs are an ingenious way of training a generative model by framing the problem as a supervised learning problem with two sub-models:

- generator model, which we train to generate new data,
- discriminator model, which attempts to classify examples as either real (from the domain) or fake (generated).

The two models are trained in an **adversarial zero-sum game** until the discriminator model is fooled around half of the time, indicating that the generator model is creating realistic data.

GANs are an exciting and rapidly changing field that fulfills the promise of generative models by generating realistic data across a wide range of problem domains, most notably in image-to-image translation tasks such as converting photos of summer to winter or day to night, and in generating photorealistic objects that even humans cannot tell are fake.

3.1 Generative models

A typical machine learning problem requires using a model to make a prediction (Discriminative modeling). In order to train a model, is required the use of a training dataset, where data contain input variables (x) and output class labels (y). As it mentioned in 2.6 model is trained by feeding it with data, having it predict outputs, and then correcting the model to make the predicted outputs more similar to the expected outputs. This model learning procedure is generally referred to as **supervised learning**.

Another learning procedure is one in which the data is only contain the input variables (x) and the problem has no output variables (y). The descriptive or

unsupervised learning, is the second major form of machine learning. We are just given inputs in this case, and the purpose is to uncover "interesting patterns" in the data. This is a lot less well-defined problem because we do not know exactly what kinds of patterns to look for and there isn't a clear error metric to utilize. A generative algorithm simulates how data was generated in order to classify a signal. To the question "Which category (supervised or unsupervised) most likely to generate this signal, depending on my generating assumptions?" the answer is that a supervised algorithm is unconcerned in how the data was generated; it simply classifies a given signal [3].

Some popular types of generative models are:

- Gaussian mixture model
- Hidden Markov model
- Bayesian network (e.g. Naive bayes, Autoregressive model)
- Boltzmann machine (e.g. Restricted Boltzmann machine, Deep belief network)
- Variational autoencoder
- Generative adversarial network

3.2 Training

As it mentioned before GANs, are a deep-learning-based generative model. GANs, in general, are a model architecture for training a generative model, and deep learning models are most commonly used in this architecture.

When the models are both multilayer perceptrons, the adversarial modeling framework is easiest to apply. To learn the generator's distribution p_g over data x, we define a prior on input noise variables $p_z(z)$, then represent a mapping to data space as $G(z;\theta_g)$, where G is a differentiable function represented by a multilayer perceptron with parameters θ_g. We also define a second multilayer perceptron $D(x;\theta_d)$ that outputs a single scalar. $D(x)$ represents the probability that x came from the data rather than p_g. We train D to maximize the probability of assigning the correct label to both training examples and samples from G.

In other words, D and G play the following two-player minimax game with value function $L(G, D)$:

$$\min_G \max_D L(D, G) = E_{x\sim p_{data}(x)}[\log D(x)] + E_{z\sim p_z(z)}[\log(1 - D(G(z)))] \quad (3.1)$$

In practice, $L(D, G)$ may not provide sufficient gradient for G to learn well. Early in learning, when G is poor, D can reject samples with high confidence because they are clearly different from the training data. In this case, $\log(1-D(G(z)))$ saturates. Rather than training G to minimize $\log(1-D(G(z)))$ we can train G to maximize $\log(D(G(z)))$. This objective function results in

the same fixed point of the dynamics of G and D but provides much stronger gradients early in learning.

GANs often suffer from a "mode collapse" where they fail to generalize properly, missing entire modes from the input data. For example, a GAN trained on the MNIST dataset containing many samples of each digit, might nevertheless timidly omit a subset of the digits from its output. Some researchers perceive the root problem to be a weak discriminative network that fails to notice the pattern of omission, while others assign blame to a bad choice of objective function.

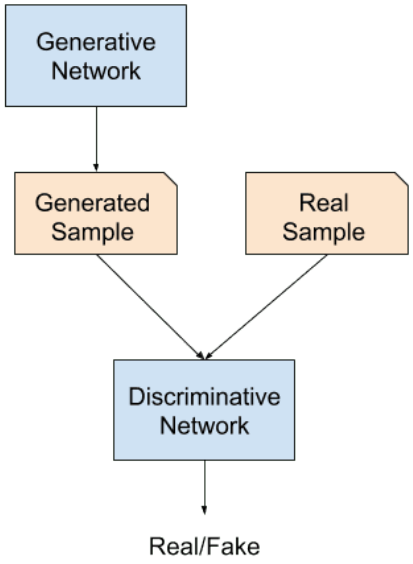

Figure 3.1: GANs training overview.

3.2.1 Deep Convolutional GAN

Attempts to scale up GANs using CNNs to model images in the past have failed. Although, after extensive model exploration, identified a family of architectures that resulted in stable training across a range of datasets and allowed for training higher resolution and deeper generative models. [24]

There are some major modifications such as:

- Spatial pooling functions (such as maxpooling) replaced with strided convolutions, allowing the network to learn its own spatial downsampling. By using this approach in generator, allowing it to learn its own spatial upsampling, and discriminator.

- Elimination of fully connected layers on top of convolutional features. The first layer of the GAN, which takes a uniform noise distribution z as input, could be called fully connected as it is just a matrix multiplication, but the result is reshaped into a 4-dimensional tensor and used as the start of the convolution stack.

- Usa of Batch Normalization which stabilizes learning by normalizing the input to each unit to have zero mean and unit variance. This helps deal

with training problems that arise due to poor initialization and helps gradient flow in deeper models.

- The Tanh function is used in the output layer, and the ReLu activation function is used in other layers. We observe that using a restricted activation function allows the model to learn faster to fill and cover the color space of the training distribution. For the discriminator, the leaky rectified activation function works well, especially in higher resolution models.

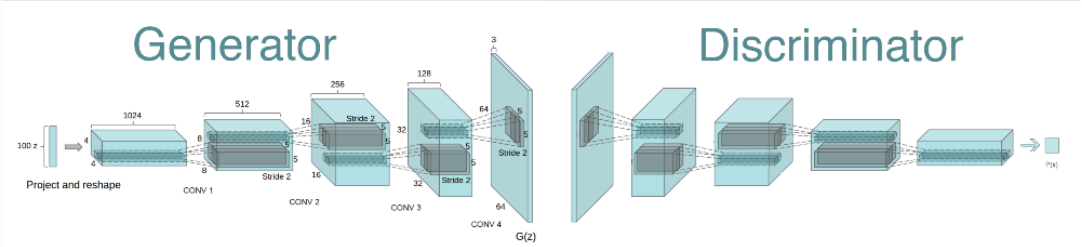

Figure 3.2: Deep Convolutional GAN.

3.2.2 Markovian discriminator (PatchGAN)

The L2 and L1 losses are widely known for producing fuzzy results in image generation problems . Although these losses fail to encourage high-frequency crispness, in many cases they nonetheless accurately capture the low frequencies. Authors [16] . in order to model high-frequencies, thought that could be sufficient to restrict the attention to the structure in local image patches.

Receptive field is the main idea behind the PatchGAN design. Sometimes it is called effective receptive field. One could assume that it is the relationship between one output activation of network to an area on the input image. A PatchGAN with the size NN, means that the output of the network maps to a NN square of the input image. In effect, a NN PatchGAN will classify NN patches of the input image as real or fake.

The receptive field is not the size of the output of the discriminator network. It is a model specification in terms of one pixel from the output activation map to the input picture. The model's output might be a single value or a square activation map of values predicting whether each patch of the input picture is real or false.

The receptive field has traditionally been defined as the size of the activation map of a single convolutional layer in relation to the layer's input, the size of the filter, and the size of the stride. This concept is generalized by the effective receptive field, which computes the receptive field for the output of a stack of convolutional layers in relation to the raw picture input.

The PatchGAN configuration is denoted as **C64-C128-C256-C512**, where **C** denotes a block of Convolution-BatchNorm-LeakyReLU layers and the number denotes the number of filters. In the first layer, batch normalization is not employed. The kernel size is set at 4x4, and all except the last two layers of the model utilize a stride of 2x2.

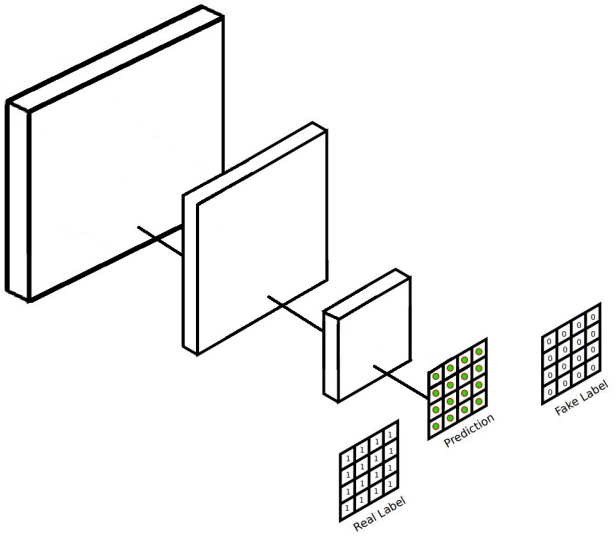

Figure 3.3: Markovian discriminator.

3.3 Generative adversarial networks variations

- **Progressive GANs**: Progressive Growing GAN is a GAN training method enhancement that enables for the sustainable training of generator models capable of producing huge, high-quality pictures. It entails beginning with a very small image and gradually adding layers to raise the output size of the generator model and the input size of the discriminator model until the required image size is reached.

- **Conditional GANs**: Generative adversarial nets can be extended to a conditional model if both the generator and discriminator are conditioned on some extra information y [20]. y could be any kind of auxiliary information, such as images, class labels or data from other modalities. We can perform the conditioning by feeding y into the both the discriminator and generator as additional input layer.
  In the generator the prior input noise $p_z(z)$, and y are combined in joint hidden representation, and the adversarial training framework allows for considerable flexibility in how this hidden representation is composed. In the discriminator x and y are presented as inputs and to a discriminative function (embodied again by a MLP in this case).

  The objective function of a two-player minmax game would be:

$$\min_G \max_D V(D,G) = E_{x\sim p_{data}(x)}[\log D(x|y)] + E_{z\sim p_z(z)}[\log(1 - D(G(z|y)))] \tag{3.2}$$

- **Image-to-Image Translation**: The objective of image-to-image translation is to learn the mapping between an input image and an output picture. It may be used for a variety of purposes, including collection style transfer, item transfiguration, season transfer, and photo enhancement. In the same way that automated language translation is defined, automatic

image-to-image translation is defined as the job of converting one potential representation of a scene into another given enough training data.

- **Inpainting**: Image inpainting is the technique of reconstructing missing sections of an image in such a way that onlookers cannot detect that these areas have been restored. This method is frequently used to eliminate undesirable items from images or to repair damaged areas of vintage photographs.

- **CycleGAN**: A large collection of paired examples is generally required to train a model for image-to-image translation. These datasets, such as images of artworks by long-dead artists, can be difficult and expensive to create, if not impossible in some situations. CycleGAN is a technique for automatically training image-to-image translation models without paired samples. The models are trained unsupervised using a set of images from the source and target domains that are not connected in any way. CycleGAN has been demonstrated on a range of applications including season translation, object transfiguration, style transfer, and generating photos from paintings.

- **Text-to-Image Synthesis**: The goal of text-to-image synthesis is to create visuals from natural language descriptions. A generated image is expected to be photo and semantically accurate. An image, in particular, should include adequate visual elements that are semantically aligned with the text description.

- **Super-resolution** : Super-resolution imaging (SR) refers to a group of approaches that increase (raise) an imaging system's resolution. The diffraction limit of systems is overcome in optical SR, whereas the resolution of digital image sensors is improved in geometrical SR. A low resolution image is frequently used as an input, and the same image is upscaled to a higher quality as the output. Where the details are essentially unknown, the details in the high resolution output are filled in.

Chapter 4

Image text removal using inpainting

4.1 Introduction to the problem

Inpainting is the process of filling-in missing or damaged image information. Its applications include the removal of scratches in a photograph, repairing damaged areas in ancient paintings, recovering lost blocks caused by wireless image transmission, image zooming and superresolution ,removing undesired objects from an image, or even perceptual filtering. Because areas missing or damaged in an image cannot be easily defined objectively, the user must select the area to be filled in, the inpainting region, in the inpainting problem. However, there are some occasions where this can be done. One such example, are the text characters printed in an image. Detecting and recognising these characters is significant in bi-modal internet data search (picture and text), and deleting them is important in the context of removing indirect adverts and for aesthetic reasons.

We address the issue of automatic inpainting-based image restoration following text detection and removal from photos. That rises the question that given an image with text characters, how to detect the exact position of the letters in the text, remove them, then fill in the spaces with inpainting methods.

4.1.1 Related work

For decades, image in-painting technology has been proposed. The image inpainting algorithms can be divided into three categories:

- Propagation-based algorithms
- Exemplar based algorithms
- Learning-based algorithms.

The first [12] is proposed to deal with small object removal, such as noise, which is accomplished by expanding the information in the existing region to the damaged region, but it may fail to restore the image with large damaged regions.

Exemplar based image in-painting methods [5] is realized by searching the similar image blocks in the information area and copying them to the damaged area. The method searches the image for the best matching patch for the target region, and once found, the pixel values are copied from the source region and pasted into the target region. The patch matching criteria, patch size, and filling order all have a significant impact on the results. However, it is usually difficult to restore the complex structure and rich details because no semantic information of image is utilized.

With the development of deep learning technology, especially the convolutional neural networks (CNNs) and generative adversarial networks (GANs), the Learning-based algorithms have been widely used to deal with more complex situations. In recent years, many works [23, 15, 7] treated the image in-painting as a conditional generation problem by learning the mapping function between the input corrupted image and ground truth one. The key point, like with the conventional Fully Connected Layer, is that all feature locations from the previous layer contribute to each feature site from the current layer. As a result, the network may learn the relationship between all of the feature locations and get a stronger semantic knowledge of the entire image.

4.2 Experiments details

4.2.1 Datasets

Before referring to the networks we used, the training details, the experiments' results and the conclusions that emerge from them, we should have a detailed discussion of the datasets used. More specifically, in order to succeed in the wild text in-painting, it must be understood that the dataset must contain a wide range of landscapes, portraits, shapes, textures, aesthetics, etc. In this way we will be able to succeed acceptable network generalization and therefore greater efficiency.

Figure 4.1: KonIQ-10K dataset: Random strings as text appended in random positions within each image.

For text in-painting with Generative Adversarial Networks we used the **KonIQ-10K** data set, consisting of **10,073 images**. This is the first in-the-wild database aiming for ecological relevance, with respect to the authenticity of distortions, the diversity of content, and quality-related indicators [13].

However, we know that the variety and variation of text instances encountered in everyday problems is quite large and even chaotic. Perhaps it would be naive and futile to try to produce such text instances (figure 4.1) to describe the actual distribution under which they are drawn.

Considering the above, we tried to work on the **SynthText in the Wild Dataset** which consists of **800 thousand** images with approximately **8 million synthetic text instances**. This is a synthetically generated dataset, in which text instances are spotted in a wide range of natural scene images, while taking into account the scene layout (figure 4.2). The reef we encountered is that despite the wide range of text instances, clean images do not exist. So the **noise2target** training technique could not be used.

How could we access the clean images? But of course training with the **noise2noise** technique. Indeed, by training under this training procedure, using a U-Net network and as an objective function the L1 norm, we managed to obtain the clean images of the dataset. In this way, ensuring greater diversity and variation of the text instances that are placed in the images and that the generator is called to locate and inpaint.

Datasets divided both into 2 subsets: training and test set. Training set contains the 80% of them, while 20% for validation. Also random selected from web a set of 32 images for **in-the-wild evaluation**, but the non-existence of ground truth does not allow us to evaluate the results with metrics, only visually.

Figure 4.2: SynthText dataset: consist of textually corrupted images.

Having mentioned almost all the background for CNNs, GANs and their variants, and also presented the datasets that used, we can now present the experiments in which we worked and studied, which of them yielded results, which did not and which need more study and research.

4.2.2 Text removal without clean data

In this experiment, the goal is to increase the diversity of text instances (shape, color, size, font, text content, rotation) but also to attach the text instances to the areas of the images in such a way that they correspond to reality. The SynthText dataset contains a large variety of such images but unfortunately the clear images are not provided. For this reason we try to train a network with U-Net architecture, and noise2noise training processes in order to recover clear images.

More specifically for each input image x with text instance ϵ_i (where $x = y + \epsilon_i$) of the dataset, we set as desired output the same image x but with a different text instance ϵ_j, where $j \neq i$. In addition, choosing $\mathcal{L}_{L1}$ as cost function means that the network's goal will be to find the median pixel color that corresponds to text removal goal. Note that the average ($\mathcal{L}_{L2}$) as cost function can not be used since the goal corresponds to find the average color of the text. Only damaged images appeared during the training and so we rejected it as an experiment. The results obtained after about 450 seasons are the following:

Figure 4.3: SynthText dataset: clean images retrieval. Noise2Noise training ($\mathcal{L}_{L1}$) yielded fruits and thus we are given the opportunity to train with clear targets.

Note that we did simultaneously training for all the images we had in our possession. In other words, we transfer the goal of network training from finding the median pixel color of an image and producing the clean image, to finding the median pixel color for each observation we have. Text and subtraction abstraction quality decreases slightly but convergence is achieved much faster (almost in half). The removal and inpainting quality of the text is slightly reduced but the convergence is achieved much faster (almost the half epochs). Network also tested to images that did not included in dataset. It seems (figure 4.4) that while the text is more or less correctly located (although there is room for improvement), there is a problem mainly in 'how' and with 'what' the network will inpaint it.

Figure 4.4: Noise2Noise simultaneous training ($\mathcal{L}_{L1}$) tested in images which we did not include in SynthText dataset.

4.2.3 Supervised training approach

The first and simplest idea is to train a U-Net or a ResNet (Resnet18 offered implemented by *torch* library), networks designed for image reconstruction, under a supervised manner using the L_1 or L_2 norm as an objective function. The inputs to the respective network will be images with noise-text attached to them and the target will be the clean images, i.e. images without text. After a reasonable number of seasons, the network will theoretically begin to detect and recognize areas of the image that contain text, then replacing the pixels' values in those areas with such values that minimizes error.

U-Net is all the experiments backbone. U-Net as described further in previous section, is a skip-connected fully convolutional neural (FCN) network. U-Net receives an input of size (256 x 256 x 3), that 3 corresponds to red, green and blue channels of an image. All images are resized to 256 square.

We define U-Net as follows:

- The **encoder** consists of five convolutional blocks that consists of a convolutional layer, batch normalization and ReLU activation, except the first that consists of 2 consecutive covolutional blocks. Kernel sizes set to 3, stride step to 1 as padding. For the downsampling operation, maxpool used on top of encoder's convolutional blocks.

- The **decoder** consists of five deconvolutional blocks with kernel size 3, stride step 1 and padding 1. It is also defined on top, an extra convolutional layer with kernel 3 and stride step 1 followd by a LeakyRelu, in order to map the last convolution block's channels to actual RGB channels. The output of each convolution layer is sent to a subsequent convolutional layer, which stacks the output of the corresponding convolutional layer in the encoder. For the downsampling operation, transposed covolusion used on top of decoder's convolutional blocks with stride 2, padding 1 and output padding 1.

The U-Net architecture can be summarized as **(C64-C64-C64-C64-C64-C128-C128-C128-C128-C128)** where C denotes Conv block. It needs to be mentioned, that U-Net architecture ended up being the above, after repeated experiments and evaluations.

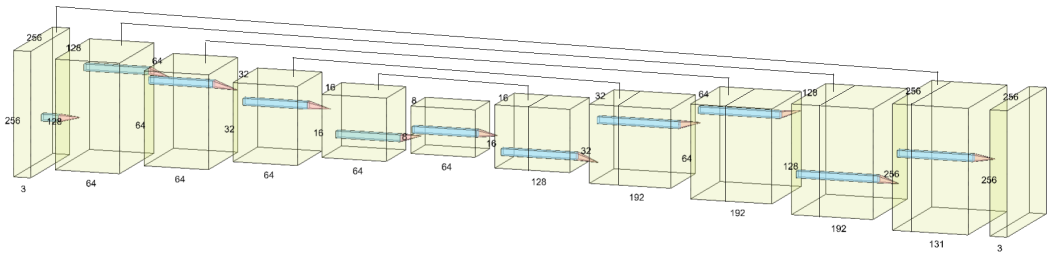

Figure 4.5: Our generator U-Net like architecture. Each convolution block containts 2d-convolutional layers, batch normalization layer and ReLU activation (except the last that used LeakyReLU with slope 0.1.) For downsampling used maxpool and for upsampling transposed convolution with stride 2.

4.2.4 Conditional adversarial training approach

Inspired by the work of [16], where with great success they enhance realism and accuracy in the generation and translation of images, we also introduced **adversarial loss**. This is due to the fact that we want the replacement of the pixels during the translation to be done in a realistic way in terms of the content of the image, which essentially means that one can not recognize text in images.

This means that we will need a **discriminative** architecture that will be trained for this very purpose, in parallel with the generator, in a minmax game. If we paraphrase the claim a little, we could say that we are introducing $L1$ loss in the adversarial training of the generator, hoping that the results will be more crispy, with more accuracy and improved textures. In other words, more realistic inpainting.

Expressing the objective function 3.2 as:

$$\mathcal{L}_{cGAN}(G, D) = E_{x,y}[\log D(x, y)] + E_{x,z}[\log(1 - D(x, G(x, z)))] \tag{4.1}$$

training the generator in order to minimize it and the disc to maximize it:

$$G^* = \arg\min_G \max_D \mathcal{L}_{cGAN}(G, D) \tag{4.2}$$

and finally introducing $\mathcal{L}_1$ loss:

$$\mathcal{L}_{L1}(G) = E_{x,y,z}[\|y - G(x, z)\|_1] \tag{4.3}$$

the objective function with which we did the training of the generator is the following:

$$G^* = \arg\min_G \max_D \mathcal{L}_{cGAN}(G, D) + \lambda \mathcal{L}_{L1}(G) \tag{4.4}$$

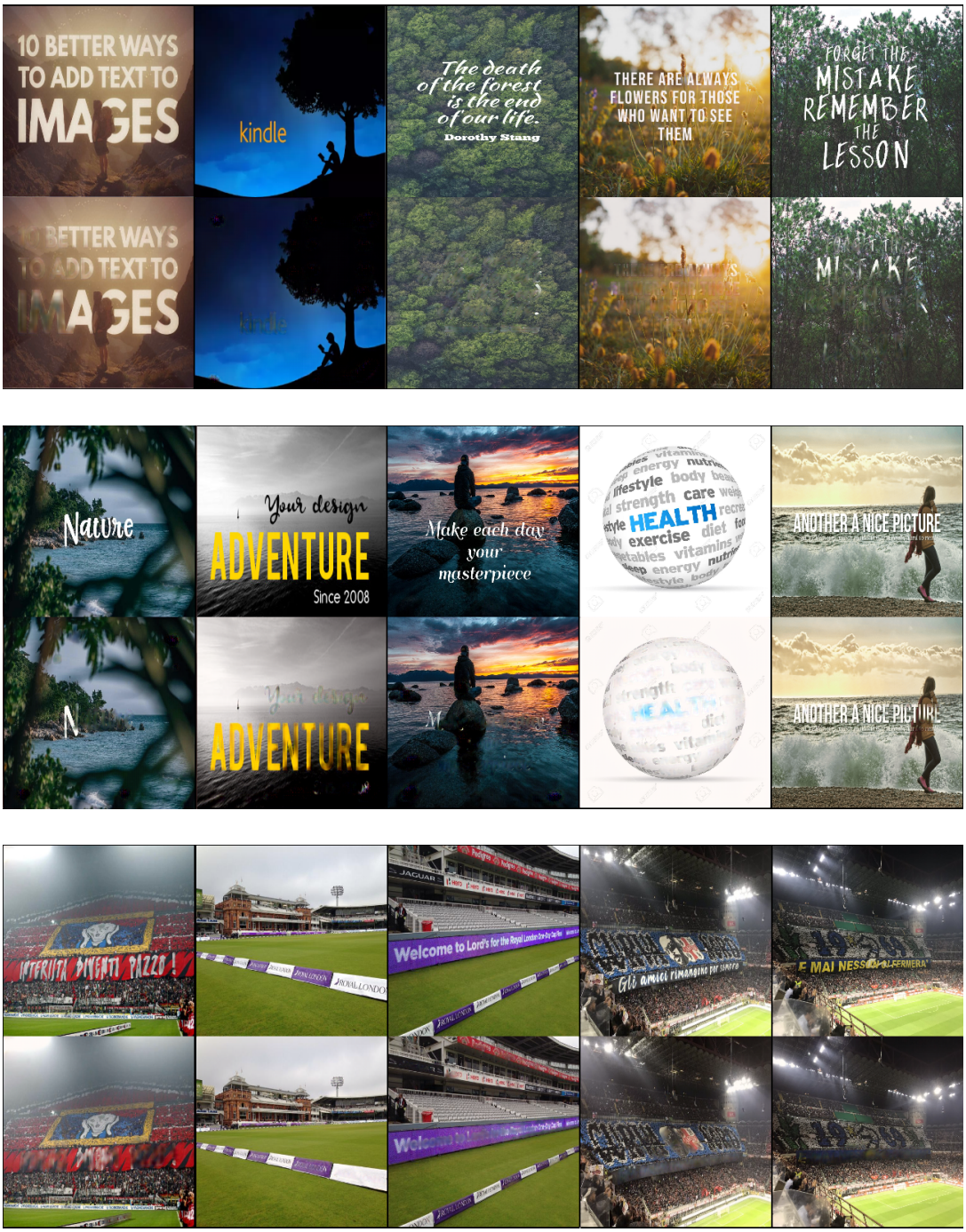

Figure 4.6: $\mathcal{L}_{L1}$ in-the-wild evaluation: **Supervised training**.

where λ expressing the contribution factor of $\mathcal{L}_{L1}$.

So we trained the above mentioned architectures under this view, in parallel with a discriminator that follows the patchgan architecture **(C64-C128-C256-C512)**.

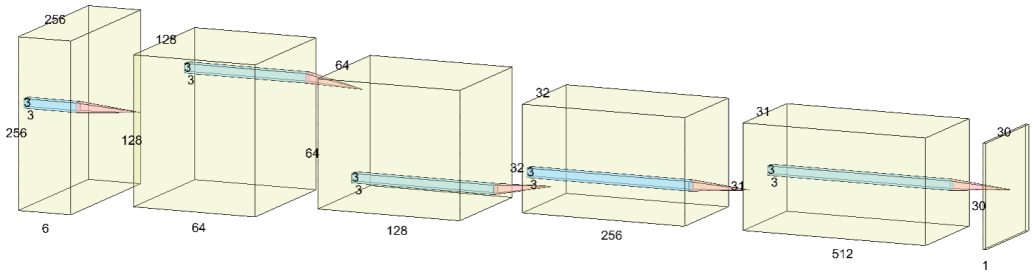

Figure 4.7: Our **discriminator (PatchGAN)** architecture. Each convolution block containts 2d-convolutional layers, with stride 2, padding 1 and kernel size 4. Batch normalization layer and LeakyReLU activation with slope 0.2 following the convolutional layer.

4.2.5 Attention mechanisms integration

By integrating attention mechanisms, we wanted to give the network the ability to focus its attention on the inpainting of the text instances, relieving it of the arduous task of recognizing and locating the text in the image.

Initially ,we thought that the integration of a 4th channel (mask) as input to the image - condition will reinforce this view. Inspired by [27], we extend generator architecture by using a pre-trained network [4] which when given an image with text as input, that outputs the areas where there is text. They proposed a scene text detection method to effectively detect text area by exploring each character and affinity between characters. To compensate for the lack of individual character level annotations, their proposed approach makes use of both the given character-level annotations for synthetic images and the estimated character-level ground-truths for real-world photos collected by the learnt intermediate model. The network is trained with the newly proposed affinity representation in order to assess affinity between characters.

By concatenating the output of the text detector to the input of generator (U-Net), we will study whether the attention mechanisms will help the network learn faster and whether the relief from the text localization problem will make the inpainting more realistic. Finally, it should be noted that the 4th channel is given also as an input to discriminator, in order for the adversarial game to be fair.

In the last experiment, we try to disengage from the use of additional networks (text detector) that make the network as a whole large. So we try, by incorporating Attention Gates in generator to achieve simultaneous training for text detection and in-painting. More specifically, the U-Net architecture on the encoder's part remains the same. There is no differentiation. In the decoder's part is where it take place the integration of Attention Gates. Each Attention Block contains convolutions with stride 1, kernel size 1 and zero padding. In order to perform the attention, after each up-sampling process in the decoder part and thus producing this signal g, it will be given as input to Attention Block with the signal x_l which is the output of the corresponding lth block in the encoder's part. The attention coefficients will be concatenated with the up-sampled signal g and finally will eventually be the input to the following

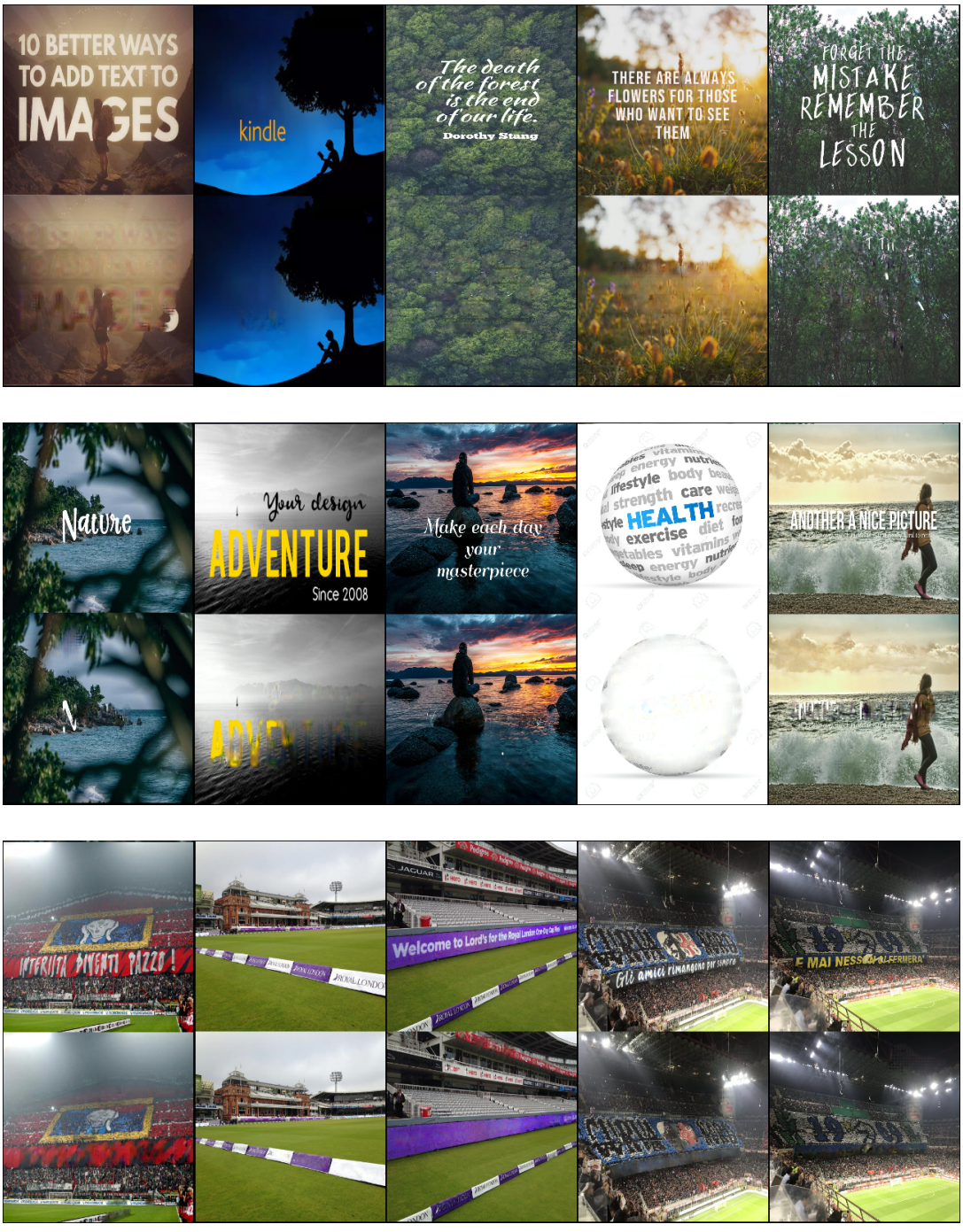

Figure 4.8: $\mathcal{L}_{L1+Adv}$ in-the-wild evaluation: **Conditional adversarial training**.

convolutional block. That process continues until the last convoltuional block.

4.2.6 Implementation and training details

The implementation is based on the *PyTorch* [22], an open-source machine learning library for Python. It is an optimized tensor library primarily used for Deep Learning applications using GPUs and CPUs and is one of the widely used Machine learning libraries, others being *TensorFlow* [19] and *Keras* [6]. It is

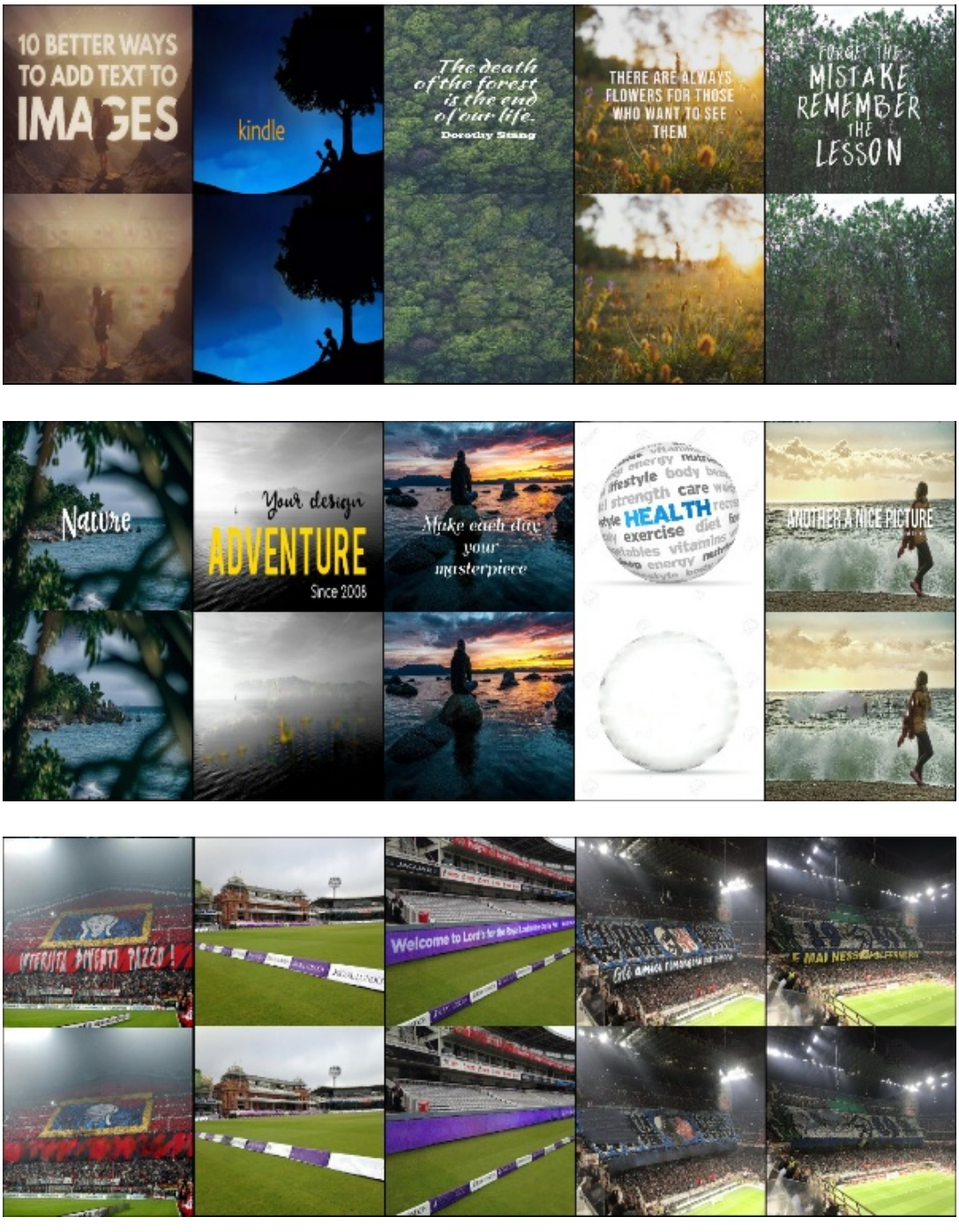

Figure 4.9: In-the-wild evaluation of text detector integration: Integrating **text detection mechanisms** (attention mechanism), leads to fast convergence and more realistically results.

based on the Torch library, largely created by Facebook's AI Research division for applications such as computer vision and natural language processing. It's advantages lie in two high-level features: i) Tensor computation (similar to *NumPy* [10]) with powerful acceleration via graphics processing units (GPU), ii) Deep neural networks based on an automated differentiation mechanism based on type.

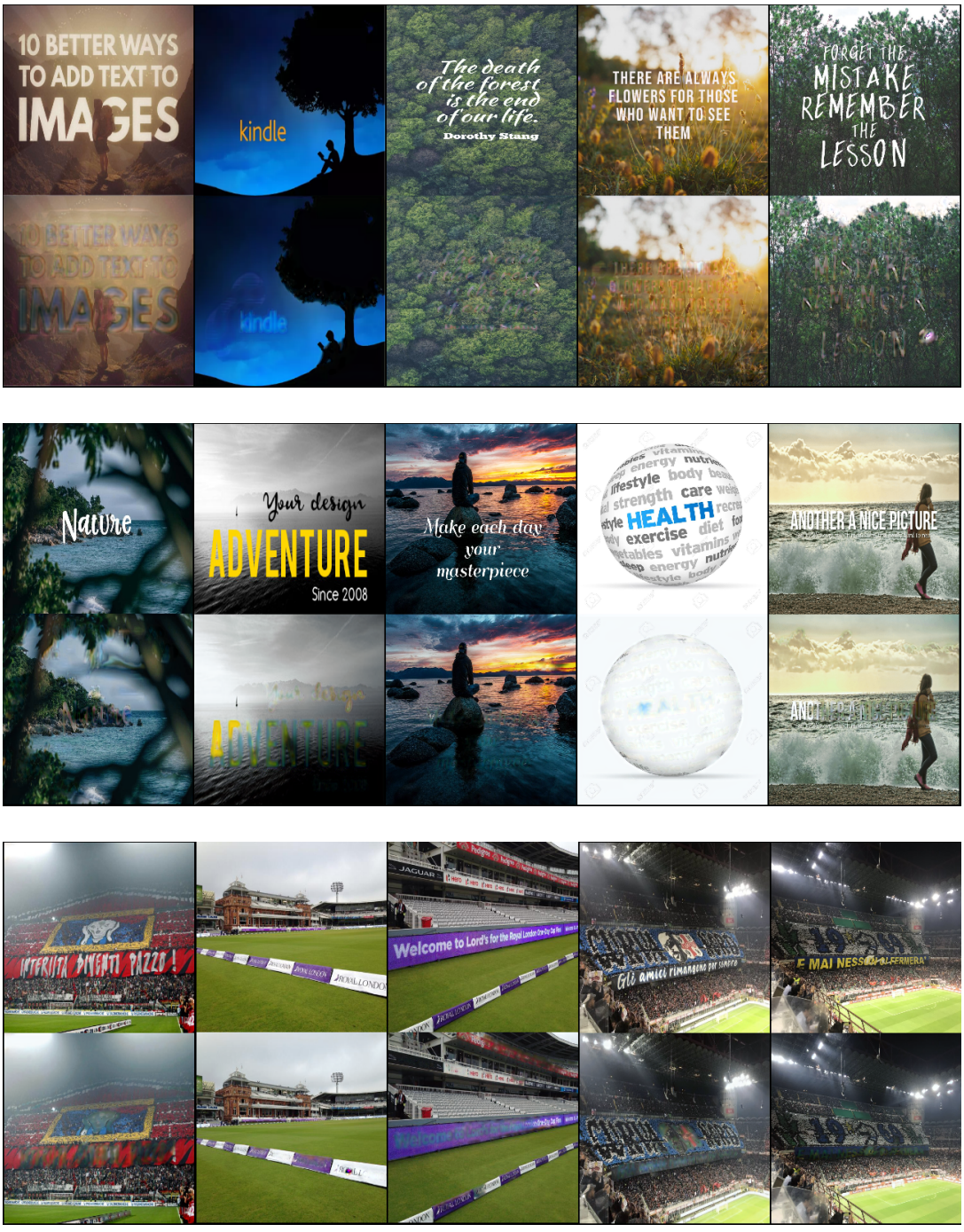

Figure 4.10: Conditional adversarial training ($\mathcal{L}_{L1+Adv}$ + attention gates integration): in-the-wild evaluation: Integrating **attention gates**, results to faster convergence and more accurate text localization compared to experiments that did not contain attention mechanisms. Although they seem to offer a focus of attention not as precise as with the integration of mask by text detector.

Specifically, for the data manipulation, preprocessing and network feeding, modules *nn.Dataset* and *nn.DataLoader* imported, while for data transformations the module *transforms* from *torchvision* library. Furthermore, libraries *Python Image Library (PIL)* [28], *Matplotlib (matplotlib)* [14], *opencv (cv2)* [17]

Figure 4.11: Attention Gates visualisation.

used to generate text and attach into images. In other words, the training data generated from the mentioned above libraries. The content and characters of text was random generated and python's modules *string* and *random* offered significant help in this.

All experiments that described in previous section (4.2) trained in GPU where used the Adam Optimizer with a learning rate $2 \cdot 10^{-4}$. Coefficients used for computing running averages of gradient and its square set to default (0.9, 0.999) and term added to the denominator to improve numerical stability set default 10^{-8}. Batch size set to 4

For SynthText dataset clean images retrieval (4.2.2) took almost 500 epochs for training, where $\mathcal{L}_{L1}$ defined as objective function and was completed when there were no further improvement of text removal task. The supervised training took place in 50 epochs, conditional adversarial training in 100 epochs while with the attention mechanisms integration the epochs reduced by half. ResNet18 and $\mathcal{L}_{L2}$ never employed as they did never offer improvement.

4.2.7 Evaluation metrics

We used two types of evaluation metrics that are widely used for image quality assessment: Peak Signal to Noise Ratio (**PSNR**) and Structural index Similarity (**SSIM**).

PSNR is used earlier than SSIM, is easy and has been widely used in various digital image measurements. Peak signal-to-noise ratio (PSNR) is an engineering term that refers to the ratio of a signal's maximum achievable power to the power of corrupting noise that influences the fidelity of its representation. PSNR is commonly stated as a logarithmic quantity using the decibel scale because many signals have a very wide dynamic range. It is often used to quantify reconstruction quality for lossy-compressed pictures and videos.

The structural similarity index measure (SSIM) is a newer measurement tool that is designed based on three factors i.e. luminance, contrast, and structure to better suit the workings of the human visual system. SSIM is a perception-based model that incorporates image deterioration as a perceived change in structural information while also includes crucial perceptual phenomena such as luminance and contrast masking components. The concept of structural information refers

to the idea that pixels have substantial interdependencies, especially when they are spatially adjacent. These dependencies contain crucial information about the structure of the items in the visual scene. Luminance masking is a phenomenon in which picture distortions become less obvious in bright areas, whereas contrast masking is a phenomenon in which distortions become less visible in areas where there is high activity or "texture" in the image.

$$PSNR = 10\log_{10}(\frac{MAX_I^2}{MSE}) \tag{4.5}$$

with:

- $MSE = \frac{1}{mn}\sum_{i=1}^{m-1}\sum_{j=1}^{n-1}[I(i,j) - K(i,j)]^2$
- MAX_I^2 is the maximum possible pixel value of the image
- I the actual image and
- K its noisy approximation

$$SSIM = \frac{(2\mu_x\mu_y + c_1)(2\sigma_{xy} + c_2)}{(\mu_x^2 + \mu_y^2 + c_1)(\sigma_x^2 + \sigma_y^2 + c_2)} \tag{4.6}$$

with:

- μ_x the average of x
- μ_y the average of y
- σ_x^2 the variance of x
- σ_y^2 the variance of y
- $\sigma_{x,y}^2$ the covariance of x, y
- $c_1 = (k_1 L)^2$, $c_2 = (k_2 L)^2$ two variables for division stabilization
- L the dynamic range of pixel-values

4.2.8 Experiments results

In this section we will present the experiments results for evaluating networks. As mentioned above, no ground truth provided for in-the-wild images that random collected from web, so we can not evaluate the performance, only visually. For this reason, we evaluated the methods in test set of SynthText dataset.

Initially it becomes clear that the increase of the diversity of the text instances after the retrieval of the clean images of SynthText dataset, leads to better results. All networks generalize better and can perceive and recognize a wide range of texts in images.

Supervised training, by pushing the network to learn by minimizing the Euclidean distance and according to the experimental results that emerged, it seems that the inpainting in the text areas is quite general and vague. In other words, with a single glance at the generator outputs, one can easily perceive in

	psnr	ssim
U-Net ($\mathcal{L}_{L1}$)	20.08	0.67
Attention U-Net ($\mathcal{L}_{L1}$)	21.28	0.71
U-Net ($\mathcal{L}_{L1+adv}$)	28.90	**0.78**
U-Net + mask ($\mathcal{L}_{L1+adv}$)	**29.11**	0.77
Attention U-Net ($\mathcal{L}_{L1+adv}$)	26.16	0.72

(a) SynthText dataset

	psnr	ssim
U-Net ($\mathcal{L}_{L1}$)	19.1	0.65
Attention U-Net ($\mathcal{L}_{L1}$)	20.11	0.70
U-Net ($\mathcal{L}_{L1+adv}$)	26.68	**0.74**
U-Net + mask ($\mathcal{L}_{L1+adv}$)	**27.09**	0.73
Attention U-Net ($\mathcal{L}_{L1+adv}$)	24.98	0.69

(b) KonIQ-10K dataset

Table 4.1: Methods evaluation after training on different datasets, a) SynthText dataset, b) SynthText dataset. The diversity of text instances significantly improves generalization ability and therefore more effective text removal.

which areas there was text in the image. This is due to the fact that Euclidean distance is optimized by averaging all feasible outputs, resulting in blurring [30]. Also, the task is hard to divide into 2 different tasks, text localization and inpainting.

Higher psnr is obtained by text localization network integration. What we noticed is that generator from very early, inpaint the text embedded on image without leaving any area of the image that contains text intact. This explained by the fact that generator from the very first back propagations, learns that its own target is to inpaint feasible and realistically the areas that mentioned by the input (mask). From the other hand, discriminator seems to encounter greater difficulties to separate between real and fake images. As a result, after some epochs it does not propagate useful gradient information to generator, thus generator does not learns as efficiently as assumed the inpainting task. Although, results (figure 4.9 and above table) reveals the most realistically inpainting.

Conditional adversarial approach leads to **higher ssim**. With this training approach for inpainting of textually corrupted images, we achieve better structural reconstruction. This is due to the fact that discriminator plays an important role in how generator learns the inpainting task. Despite the fact that the strangely structural textual areas remain intact, generator will give a greater focus for how to fool discriminator, leading to the results presented.

Finally, integration of attention gates results did not demenostrate our claim. Having said that enerator focuses his attention on text areas with the assistance of attention gates, which during training arise and are generated, he also pays attention to areas where there is no text. It's something that makes it difficult for generator to produce realistic results. Attention gates, while operating as text detectors in many cases, end up learning correlations between areas that are irrelevant to the text. In this way, if generator learns early on that he simply has to inpaint the text areas that highlited by attention coefficients, it means

that he will inpaint any area indicated. There are results and assumptions that definitely need more research and experimentation.

Chapter 5

Conclusion

In this work, we studied the problem of text removal using inpainting techniques with Generative Adversarial Networks. It is an open problem and there are several difficulties both in the localization of the text areas and in the way of their inpainting. The learning-based inpainting technique requires the training dataset to follow an in-the-wild perspective, as the diversity of fonts, the ways of attaching them in the images and the opacity level range is chaotic. So we experimented with learning based inpainting models by training them in a set of data that we attached our own text and in another one in which they were attached with in the wild technique. We also overcame the non-existence of clean images (images without text), by training with noise2noise training approach according to which we can recover textually corrupted images by having only those in our possession. Apart from the fact that we managed to retrieve them effectively, it seems that with the simultaneous training technique that we tried, the network manages, even in a not so efficient way, to remove text from images that did not exist in the entire training. The results show that in the wild text addition technique helps the learning process as a whole and leads to greater generalization.

Initially, we trained a U-Net with a supervised training approach and noticed that the results are quite good, for this simple approach. In addition to the blur in the results, the problem lies in the fact that it can not detect all kinds of texts with the result that the goal is not achieved to the extent we would like. The recent rapid development of GANs and the adversarial - unsupervised learning approach, pushed us to use it, assuming that on the one hand the inpainting will become more realistic but on the other hand that the discriminative architecture will help in the task of text localization. The results refute this claim, confirming once again the great success of the GANs.

Convolutional architectures' feature-map grid is gradually downsampling in order to capture the semantic information derived from the context. In this way, it remains difficult to locate text that show great schematic variation, which leads to failure to inpaint by generator. Under these assumptions we considered that the integration of attention mechanisms will help to more accurate localization of smaller or more diverse texts. We have integrated additional localization networks whose prediction of texts' position will be given as a 4-th additional channel at the input of the generator. Indeed, the inpainting now takes place in every text area, since the outputs of the external pre-training networks rep-

resent the attention of generator. Finally, in order to achieve comparable levels of attention but reducing the network's depth, we integrated Attention Gates (AGs) to generator network. We considered the AGs to offer the network more semantic and context information not only in terms of inpainting but also in terms of text location. From the experiments results obtained the model did not show good generalization, as no significant improvement in the results was observed after the introduction of these AGs. The experiments of integration of attention mechanisms with external localization networks, are considered successful since they managed to catch the best score after the evaluation presented, while with the integration of AG in U-Net architecture and supervised training we increased the performance even a little.

Bibliography

[1] Mujadded Al Rabbani Alif, Sabbir Ahmed, and Muhammad Abul Hasan. "Isolated Bangla handwritten character recognition with convolutional neural network". In: *2017 20th International Conference of Computer and Information Technology (ICCIT)*. 2017, pp. 1–6. DOI: 10.1109/ICCITECHN.2017.8281823.

[2] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. "Understanding of a convolutional neural network". In: *2017 International Conference on Engineering and Technology (ICET)*. 2017, pp. 1–6. DOI: 10.1109/ICEngTechnol.2017.8308186.

[3] Ng Andrew and Jordan Michael. "On Discriminative vs. Generative Classifiers: A Comparison of Logistic Regression and Naive Bayes". In: (2001). URL: http://ai.stanford.edu/~ang/papers/nips01-discriminativegenerative.pdf.

[4] Youngmin Baek et al. *Character Region Awareness for Text Detection*. 2019. arXiv: 1904.01941 [cs.CV].

[5] Pierre Buyssens et al. "Exemplar-Based Inpainting: Technical Review and New Heuristics for Better Geometric Reconstructions". In: *IEEE Transactions on Image Processing* 24.6 (2015), pp. 1809–1824. DOI: 10.1109/TIP.2015.2411437.

[6] Francois Chollet et al. *Keras*. 2015. URL: https://github.com/fchollet/keras.

[7] Ugur Demir and Gozde Unal. *Patch-Based Image Inpainting with Generative Adversarial Networks*. 2018. arXiv: 1803.07422 [cs.CV].

[8] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. Cambridge, MA, USA: MIT Press, 2016, pp. 590–599.

[9] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML].

[10] Charles R. Harris et al. "Array programming with NumPy". In: *Nature* 585.7825 (Sept. 2020), pp. 357–362. DOI: 10.1038/s41586-020-2649-2. URL: https://doi.org/10.1038/s41586-020-2649-2.

[11] Kaiming He et al. *Deep Residual Learning for Image Recognition*. 2015. arXiv: 1512.03385 [cs.CV].

[12] Somayeh Hesabi and Nezam Mahdavi-Amiri. "A modified patch propagation-based image inpainting using patch sparsity". In: *The 16th CSI International Symposium on Artificial Intelligence and Signal Processing (AISP 2012)*. 2012, pp. 043–048. DOI: 10.1109/AISP.2012.6313715.

[13] V. Hosu et al. "KonIQ-10k: An Ecologically Valid Database for Deep Learning of Blind Image Quality Assessment". In: *IEEE Transactions on Image Processing* 29 (2020), pp. 4041–4056.

[14] J. D. Hunter. "Matplotlib: A 2D graphics environment". In: *Computing in Science & Engineering* 9.3 (2007), pp. 90–95. DOI: 10.1109/MCSE.2007.55.

[15] Satoshi Iizuka, Edgar Simo-Serra, and Hiroshi Ishikawa. "Globally and Locally Consistent Image Completion". In: *ACM Transactions on Graphics (Proc. of SIGGRAPH 2017)* 36.4 (2017), 107:1–107:14.

[16] Phillip Isola et al. *Image-to-Image Translation with Conditional Adversarial Networks*. 2018. arXiv: 1611.07004 [cs.CV].

[17] Itseez. *Open Source Computer Vision Library*. https://github.com/itseez/opencv. 2015.

[18] Jaakko Lehtinen et al. *Noise2Noise: Learning Image Restoration without Clean Data*. 2018. arXiv: 1803.04189 [cs.CV].

[19] Martın Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[20] Mehdi Mirza and Simon Osindero. *Conditional Generative Adversarial Nets*. 2014. arXiv: 1411.1784 [cs.LG].

[21] Ozan Oktay et al. *Attention U-Net: Learning Where to Look for the Pancreas*. 2018. arXiv: 1804.03999 [cs.CV].

[22] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach et al. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[23] Deepak Pathak et al. *Context Encoders: Feature Learning by Inpainting*. 2016. arXiv: 1604.07379 [cs.CV].

[24] Alec Radford, Luke Metz, and Soumith Chintala. *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*. 2016. arXiv: 1511.06434 [cs.LG].

[25] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597 [cs.CV].

[26] Jo Schlemper et al. *Attention-Gated Networks for Improving Ultrasound Scan Plane Detection*. 2018. arXiv: 1804.05338 [cs.CV].

[27] Osman Tursun et al. "MTRNet: A Generic Scene Text Eraser". In: *2019 International Conference on Document Analysis and Recognition (ICDAR)* (Sept. 2019). DOI: 10.1109/icdar.2019.00016. URL: http://dx.doi.org/10.1109/ICDAR.2019.00016.

[28] P Umesh. "Image Processing in Python". In: *CSI Communications* 23 (2012).

[29] Jan Zdenek and Hideki Nakayama. "Erasing Scene Text with Weak Supervision". In: *2020 IEEE Winter Conference on Applications of Computer Vision (WACV)*. 2020, pp. 2227–2235. DOI: 10.1109/WACV45572.2020.9093544.

[30] Richard Zhang, Phillip Isola, and Alexei A. Efros. *Colorful Image Colorization*. 2016. arXiv: 1603.08511 [cs.CV].