BACHELOR'S THESIS

Degree in Mathematics

Title: Generative Adversarial Networks for Anomaly Detection in Images

Author: Guillem Batiste Ros

Advisor: Veronica Vilaplana

Department: Image Processing Group (UPC)

Academic year: 2017-2018

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat de Matemàtiques i Estadística

Universitat Politècnica de Catalunya
Facultat de Matemàtiques i Estadística

Degree in Mathematics

Bachelor's Degree Thesis

Generative Adversarial Networks for Anomaly Detection in Images

Guillem Batiste Ros

Supervised by Veronica Vilaplana

January, 2018

I am very grateful of having the opportunity to work with Professor Veronica Vilaplana and I would thank her for guiding me and sharing knowledge through all this months of work and for providing thoughtful advices in our meetings. I would also like to thank all Imatge Group research students for sharing their knowledge in the weekly study group sessions, and specially Marc Combalia and Irina Sanchez for providing help with technical details of the implementation.

Also, I would like to thank Professor Narciso Roman Roy for helping with all the procedural problems encountered.

Finally, I would also like to mention my family, specially *Eulàlia* and also *Irene*, for your patience and support during this journey.

Abstract

Anomaly detection is used to identify abnormal observations that don't follow a normal pattern. In this work, we use the power of Generative Adversarial Networks in sampling from image distributions to perform anomaly detection with images and to identify local anomalous segments within this images. Also, we explore potential application of this method to support pathological analysis of biological tissues.

Keywords

Anomaly Detection, Generative Adversarial Networks, Deep Learning, Medical Imaging

Contents

1. Introduction

1.1 Motivations

Context

At present, Artificial Intelligence (AI) and Machine Learning are rapid moving fields with high potential impact on society. Despite being founded around 1960, AI didn't started to stand out until recent years. Most argue that milestones in the field come by hand with concurrent advances in computational power and production of massive amounts of data.

Nowadays, AI it's not only found in research community but also in the industry and at last at our daily lives. Big corporations use Machine Learning to recommend products based on interests, to detect fraud, better translate texts, forecasting and even to analyze genome sequences among many other applications. Deep Learning is a sub-field of Machine Learning that is revolutionizing Computer Vision, and it's currently applied among others to object detection, image segmentation and recently to develop first autonomous cars systems.

Generative Adversarial Networks are a particular type of model used in Deep Learning. Concretely, they are *generative models*, mostly used to generate samples of images distributions. Despite being introduced in 2014, it's promising results on image generation made this models a highly active research topic that is heavily increasing everyday.

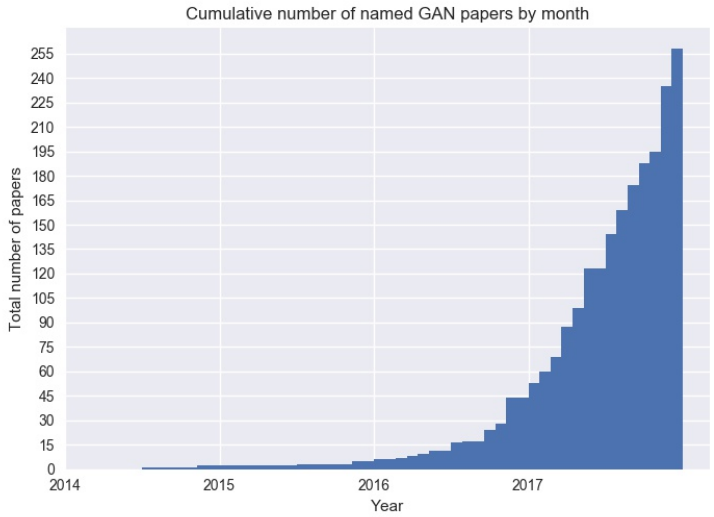

Figure 1: All the named GAN variants papers cumulatively since 2014. Credit: Bruno Gavranović.

Personal motivations

Personally, as a two degree Mathematics and Medicine student, I'm interested in potential applications of Mathematics to Medicine. The possibility to combine knowledge of so different areas of science sounds more than challenging. With recent advances, AI seems to be promising in many ways when it comes to Medicine. More concretely, we found that Deep Learning and GANs can be applied to Medical Imaging, an essential part of modern Medicine practice.

Usefulness

In Medical Imaging, Deep Learning is being applied to analyze different types of scans as Simple Rx, RMN and TC. Also, analysis of Histopathological Images is done in order to, for example, detect metastasis in breast cancer biopsies. Particularly, generative models are being used to synthesize new images, to achieve super-resolution, complete missing parts of images, to 'translate' one RMN sequence to another without the need of performing another scan,etc. As said, our chosen technique is generative modeling with GANs. We aim to apply GANs for a classical Computer Vision problem: Anomaly Detection in images. With that, taking inspiration from work done by Raymond A. Yeh et al. (see [14]) and Thomas Schlegl et al. (see [13]), our intention is to explore potential use of automatic Anomaly Detection methods in Histopathological Images. Ig good enough, this methods could support slide analysis work-flows providing regions of interest to focus on.

1.2 Work Overview

This work started as a research for potential applications of Deep Learning to Medical Imaging. In order to do this, we spent a great deal of time doing general research among a broad set of techniques to finally focus on some of them for the application part of this work.

This thesis follows the same order and starts with an explanation of all theoretical concepts in a brief and compact way. For the reading of this work, only mathematical background is expected. Starting from Machine Learning, the concepts are ordered to make the reader able to build the knowledge necessary to understand how the techniques explored are used in the application. Besides that, Machine Learning is a broad field so references will be provided to give the reader the opportunity to go deeper into the concepts exposed.

In the second part, a simple anomaly detection application prototype is explained and discussed. As long as the final application was an ambitious one, we decided to implement a simpler application first and test it on smaller data.

Finally, in the third part of this work we aim to apply the same techniques to a more difficult set of images such as histopathological images and explore their potential use for anomaly detection.

1.3 Scope of this memoir

This work is done in Computer Vision research group *Imatge* from UPC, so it involves mostly applied mathematics. Deep Learning was practically a new field to me, so I have spent great time learning nearly from scratch. In this memoir, several concepts studied are omitted because they are not essential to understand the final application, but are valuable knowledge I take from this work.

Additionally, ideas explored are code implemented, process that represents more than half of the work. In this process I learned and used different technologies: programming language *Python* (*numpy*, *scipy* libraries in particular) and *Tensorflow*, a libray from Google used for Machine Learning.
While those are not described in this work, it's learning process is worth hours of work but also knowledge now I already have. The final implementation can be seen in the shared copy of the code (see A.3).

2. Preliminaries

2.1 Machine Learning

Machine Learning is a field of Artificial Intelligence and Computer Science involving algorithms and systems that learn to perform tasks from data without human help. With the rise of computational power and huge structured data sets in the recent years, Machine Learning has become one of the most active research areas.

There are three types of machine learning:

- **Supervised Learning**: the model gets pairs of inputs and outputs, and then it learns to predict proper outputs to new inputs. Typical problems that use supervised learning are classification (the output is a discrete class) and regression (the output is the value of a continuous variable). One of the most simple examples of supervised learning is Linear Regression.

- **Unsupervised Learning**: only inputs are given to the model, and then it learns useful features to describe the hidden structure of the data. One typical example of unsupervised learning is clustering.

- **Reinforcement Learning**: an agent takes actions in an environment and receives feedback in form of rewards and punishments in order to learn a concrete behavior.

In this work we will focus on the first two types.

The *learning* has three components:

- **Representation** The model family must be represented in a formal language. This choice will determine the set of models that can be learned.

- **Evaluation** We will choose a metric or *score function* to evaluate performance of different learned models in the task. One example can be accuracy in prediction tasks.

- **Optimization** We will search among the possible models within the family by optimizing a *loss function* (see section 2.2).

There are many choices in this three components of the learning, and those will highly determine performance (see section 1.5 of [6]).

2.2 Optimization

All machine learning problems involve optimization of one or more functions. For most models, the goal will be to find a set of parameters θ that minimizes a loss function $\mathcal{L}$, which is a measure of the quality of the model for a particular set of parameters. One simple example of loss function is Mean Squared Error used in Linear Regression. Simple models such as linear regression define nice convex loss functions, but more complex ones such as those of Neural Networks aren't. For now, there are several optimization strategies that achieve good optimization in different contexts, but optimization remains an active filed of research.

Some optimization strategies involve computing gradients of $\mathcal{L}$ with respect to the parameters. Gradients provide directions in which we make small steps in order to update the parameters. The size of this steps is determined by the *learning rate*, which is an hyperparameter. In the case of Neural Networks, derivatives are computed with backpropagation algortihm (see section 3.2). Also, there are many ways that we can use this gradients to update our parameters. We will explain the ones used in this work.

Stochastic Gradient Descent
SGD is the most basic strategy used in optimization performed updating parameters in the direction in which the loss function decreases more.
Denoting the gradient by $\nabla_\theta \mathcal{L}$, s the learning rate and θ the parameters, the update is performed in the following way:

$$\theta = \theta - s \cdot \nabla_\theta \mathcal{L}$$

Momentum
Momentum is a method designed to accelerate learning, as long as SDG can be sometimes slow. This update strategy is motivated from a physical perspective of the problem where a ball is rolling down a landscape. Initializing the parameters with random numbers is equivalent to setting a particle with zero initial velocity at some location. In a physical system, this particle would have *momentum*, that acts both as a smoother and an accelerator. The local dynamics of this system gives momentum update almost always better and faster convergence rates than SGD (see section 8.3 of [6]).
In this strategy, we introduce velocity ν initialized as 0, and a coefficient ρ that is inspired in the coefficient of friction of the physical system. ρ is an hyperparameter than when cross-validated is usually set values that range from 0.5 to 0.99. The update is performed in he following way:

$$\nu = \rho \cdot \nu - s \cdot \nabla_\theta \mathcal{L}$$
$$\theta = \theta + \nu$$

Nesterov Momentum
This strategy is a variation of Momentum and has similar properties. Instead of computing the gradient in the current position θ, we will compute it in the position $\theta + \nu$ (where is going to end

up the current step). The update is performed in he following way:

$$\begin{aligned}\hat{\nu} &= \nu \\ \nu &= \rho \cdot \nu - s \cdot \nabla_\theta \mathcal{L} \\ \theta &= -\rho \cdot \hat{\nu} + (1+\rho) \cdot \nu\end{aligned}$$

Nesterov is the chosen strategy for this work as long as it gives the best performance.

There exist other parameter update methods (see chapter 8 of [6]). Per-parameter adaptive learning rate methods such as AdaGrad and RMSprop are specially interesting but won't be covered in this work.

2.3 Hyperparameters and data split

Often representation of the models and optimization strategies depend on parameters that are external to model parameters, such as learning rate (size of the step in optimization, see section 2.2). Those are called *hyperparameters*. Their values are determined with evaluation in the validation, set which is a part of the dataset (see data split below).

Typical workflow in the processes of training machine learning algorithms involve a split of the data in three parts:

- **Training set** The data used in the learning process.
- **Test set** Usually we are interested in how well our model generalizes on data that it has not seen before. Therefore, we use our score metric to evaluate these performance measures using the *test set*, that is separate from the data used for training.
- **Validation set** Part of the data used to determine or *tune* values of the hyperparameters.

3. Deep Learning

3.1 Neural Newtorks

Neural Networks (NN) are the canonical deep learning models. They are called *networks* because they are represented typically by composing many functions with different relationships.

NN can be seen as a directed acyclic graph describing how functions are composed together. Typically this functions are applied to different *units* or *neurons*, and are chained one after another. Each function in the chaining of composition determines a *layer* (which contains different neurons). The *depth* of the model is given by the number of layers. The first layer is called *input layer*, the final layer is called *output layer* and the intermediate layers are called *hidden layers*.

One common architecture for a NN layer is a *fully-connected* or *dense* layer, in which every neuron is connected to every neuron of previous layer.

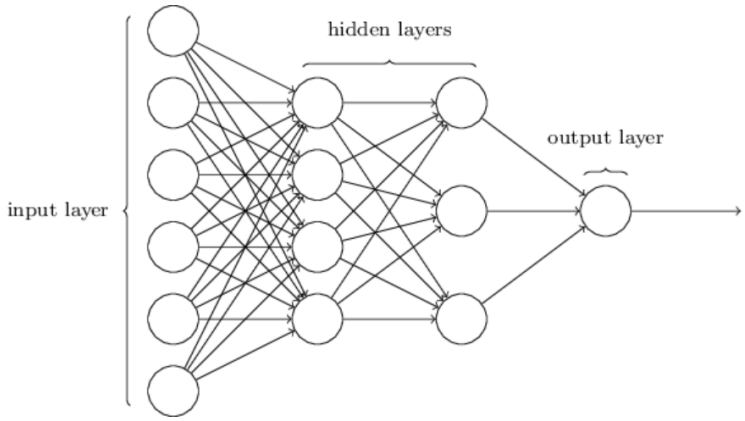

Figure 2: Overview example of a neural network architecture with fully connected layers.

Neurons

Neurons are basic units of a neural network. They are inspired on the biological model of a neuron: they receive inputs from other neurons and compute it's own output value. Their outputs are computed as:

$$a = f\left(\sum_i w_i x_i + b\right)$$

That is a linear combination of it's inputs x_i, an added term of bias b and then a function f or *activation* function (typically non-linear).

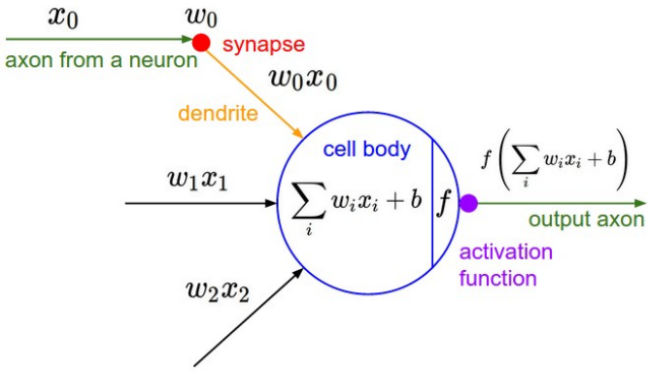

Figure 3: Mathematical model of a neuron. Input signals come from other neurons and interact multiplied by the weights w_i with the neuron. A bias term b is added. The firing response of the neuron is determined by activation function f.

There are many choices in activation functions with particular properties. The most common are:

- **Sigmoid** While very used initially, sigmoid non-linearity has fallen in disuse for it's problems in gradient saturation and non zero centered outputs (see [11]).

$$\sigma(x) = \frac{1}{1 + e^x}$$

- **ReLU** Rectified Linear Unit non-linearity was introduced by Krizhevsky et al. in [9] and it was found to greatly accelerate SGD compared with sigmoid and other activation functions, but also has problems of some neurons never activating. However, this problem can be partly addressed with proper setting of the learning rate (see [9]).

$$f(x) = max(0, x)$$

- **Leaky ReLU** This non-linearity was designed to address activation problems of ReLUs, and it's done adding a small constant α in order to have a small negative slope for $x < 0$.

$$f(x) = 1(x < 0)(\alpha x) + 1(x >= 0)(x)$$

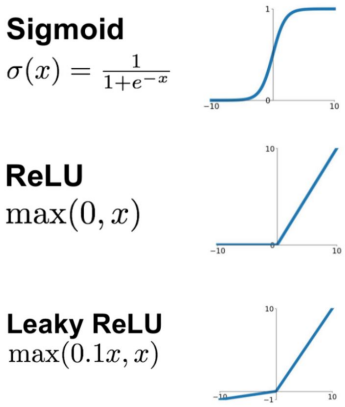

Figure 4: Common non-linear activation functions. In this work ReLUs and Leaky ReLUs will be used.

3.2 Backpropagation

In the core of the Neural Networks learning process there is the *backpropagation* algorithm. NNs are trained with gradient based optimization strategies (see section 2.2). Backpropagation provides a way to compute derivatives with respect to all network parameters in order to perform optimization.

Notation

- Let be w^l_{jk} the weight for the connection for the k^{th} neuron in the $(l-1)^{th}$ layer to the j^{th} neuron in the l^{th} layer.
- Let be b^l_j the bias of the j^{th} neuron in the l^{th} layer.
- Let be a^l_j the activation of the j^{th} neuron in the l^{th} layer.

Note the relation

$$a^l_j = f\left(\sum_k w^l_{jk} a^{l-1}_k + b^l_j\right)$$

which can be written in vectorized from

$$a^l = f\left(w^l a^{l-1} + b^l\right)$$

where w^l is the matrix of weights connecting to the l^{th} layer, and b^l and a^l are the vectors of biases and activations for each unit in the l^{th} layer.

Given the loss function $\mathcal{L}$, the goal of backpropagation is to compute the derivatives of each parameter with respect to $\mathcal{L}$. This is $\partial\mathcal{L}/\partial w$ and $\partial\mathcal{L}/\partial b$ for every weight and bias in the network.
We start defining two intermediate quantities:

- The *weighted input* z^l defined as $z^l = w^l a^{l-1} + b^l$ to layer l.
- The *error* δ_j^l in the j^{th} neuron and the l^{th} layer, defined as $\delta_j^l = \frac{\partial \mathcal{L}}{\partial z_j^l}$

The four equations of backpropagation are:

$$\delta^L = \Sigma'(z^L) \cdot \nabla_a \mathcal{L} \tag{1}$$

$$\delta^l = \Sigma'(z^l) \cdot (w^{l+1})^T \cdot \delta^{l+1} \tag{2}$$

$$\frac{\partial \mathcal{L}}{\partial b_j^l} = \delta_j^l \tag{3}$$

$$\frac{\partial \mathcal{L}}{\partial w_{jk}^l} = a_k^{l-1} \cdot \delta_j^l \tag{4}$$

Where $\Sigma'(z^l)$ is the square matrix whose diagonal entries are the values of $f'(z_j^l)$, and whose off-diagonal entries are zero.

Note that equation 1 gives an expression for computing the error vector of the output layer L. The equation 2 gives an expression for computing error vectors in a layer in terms of the error vector of the next layer. This way, the error recursively *backpropagates* from last to first layer. Finally equations 3 and 4 give expressions for computing the derivatives we want in terms of the errors. A proof of the equations can be found at [1].

3.3 Convolutional Neural Networks

Convolutional Neural Networks (CNN) are a type of Neural Networks designed particularly for images. This particular type of networks assume that inputs are always grid rectangular (as images) and take advantage of their spatial structure. This advantage makes convolutions networks train faster than conventional networks and achieve much better performance in most learning tasks involving images.

The main ideas they take advantage of are:

- **Local receptive fields** In order to compute features to classify images is enough to look on a part instead of the whole image.

- **Shared weights and biases** The weights and biases used to detect the same feature in a part of the image can be used too in different parts of the image.

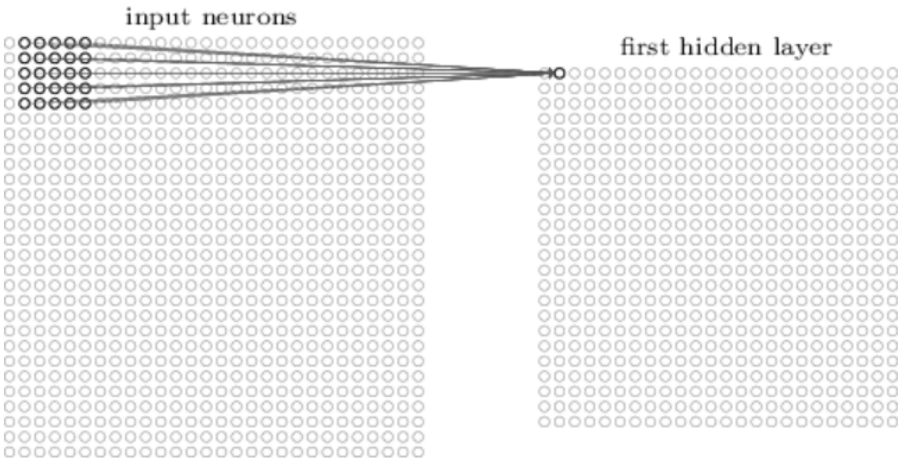

Figure 5: Local receptive filed of size 5x5 of the second neuron in first hidden layer.

Convolutions

In CNNs we have *filters* that are convolved around the image and provide a *feature map*, which is part of the next layer. Filters are matrices of certain size that contain the weights. The output in the first hidden layer of the j^{th}, k^{th} neuron for a filter of size *axb* will be computed by element-wise multiplication of the filter, adding the bias and computing activation function, that is:

$$f\left(b + \sum_{l=0}^{a-1}\sum_{m=0}^{b-1} w_{l,m} a_{j+l,k+m}\right)$$

This operation is called a **convolution**. With this operation, units organized into the same feature map share weights while different units cover different positions in the image. For each layer, we can have a certain number of filters (weight matrices), and for each filter we have a feature map.

It's important to note that the sharing weights and biases for same filter in all regions of the image greatly reduces the amount of parameters per layer compared with fully connected layers, which will result in a faster training of convolutional neural networks and the possibility of building deeper networks with same computational power constraints.

Pooling

Another strategy used in CNNs is *pooling* layers. This type of layers are usually used after convolutional layers. Pooling layers simplify the amount of information provided by convolutional layers

summarizing small regions of feature maps into condensed feature maps. This can be achieved by computing the max value in that region or averaging them.

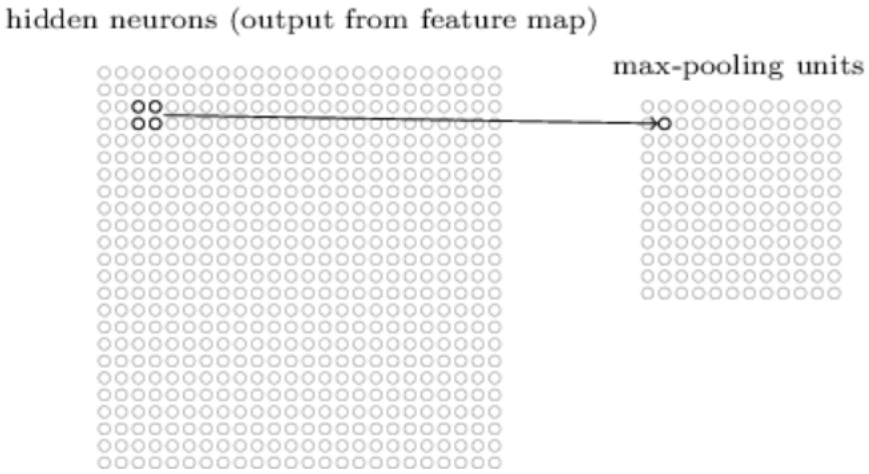

Figure 6: Max-pooling a region of size 2x2 of a hidden layer.

With max pooling the number of parameters is also greatly reduced while most of information found in convolutional layer is kept.

Final architectures of CNNs are composed of different combinations of types and number of layers, number of filters, size of filters, etc. One example can be seen in figure 7. Also, there are a lot of other strategies used and other types of layers that won't be explained in this work (see chapter 9 of [6]).

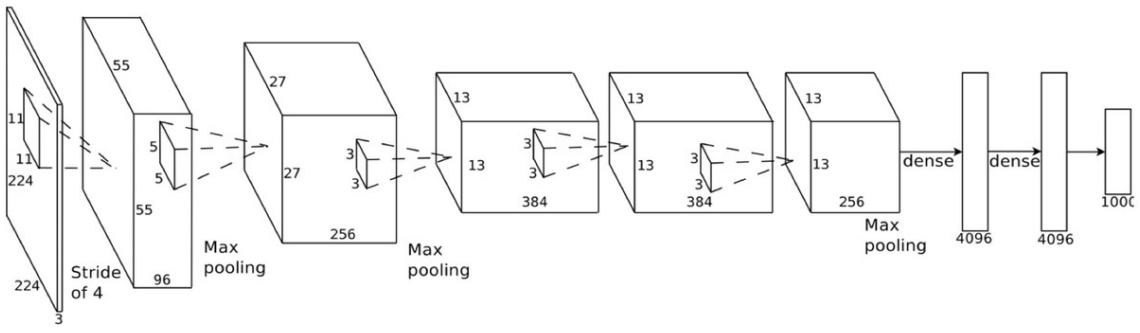

Figure 7: Example of full architecture of a Convolutional Neural Nework

4. Generative Adversarial Networks

4.1 Generative Models

Generative models learn to generate new samples from a distribution. They take an input data set that follows a theoretical probability distribution p_{data} and learn an estimation distribution p_{model} in an unsupervised way.

One of the approaches of the learning with generative models is **maximum likelihood estimation**. In this case, learning a probability distribution means learning a density function p_{model}. The core idea is to define a model that provides an estimate of a probability distribution, parameterized by parameters θ. Defining a parametric family of densities $(P_\theta)_{\theta \in \mathbb{R}^d}$ and with a training data set $\{x^{(i)}\}_{i=1}^m$, the problem to solve is:

$$\theta^* = \max_{\theta \in \mathbb{R}^d} \frac{1}{m} \sum_{i=1}^{m} \log P_\theta(x^{(i)})$$

The result is an explicit density function P_θ^* for p_{model}. However, we need P_θ^* to exist, to live within the parametric family $(P_\theta)_{\theta \in \mathbb{R}^d}$ and to be computationally tractable. This means only model families that respect this constraints can be used.
For this reasons, under model family misspecification and finite data (which is what we mostly find in real world problems) this approach fails to capture a good approximation of p_{data} (see [5]).

Other approaches such as **Variational Autoencoders (VAE)** address tractability and existence problems. Rather than estimating P_θ^*, choose a prior simple $p(z)$ distribution (Gaussian, Uniform,...) from which we can easily sample, and then try to learn a transformation (typically with a neural network) from prior simple to p_{data}:

$$G_\theta : \mathcal{Z} \to \mathcal{X}$$

where $z \in \mathcal{Z}$ and $\mathcal{X}$ the p_{model} manifold. This way, despite we don't have an explicit density function, with that transformation we can generate samples that follow p_{model}. By optimizing on θ, we can learn a G_θ^* that samples from an approximation close to p_{data}. This approach allows to easily generate samples more than knowing the exact numerical value of the density.
VAEs also address intractability issues defining a lower bond $\mathcal{L}$ of the likelihood. For many models it is possible to define a tractable lower bond even the log likelihood is not (see [5]). However, optimizing a lower bond instead of true likelihood often leads to learn weak approximations for p_{data}.

The approach used in this work is **Generative Adversarial Networks (GANs)**, introduced by Goodfellow et al. in [4], explained in detail in section 4.2. GANs use the same sampling strategy as VAEs, but don't rely it's optimization in a lower bond of likelihood nor involve any explicit density functions. Instead, they optimize sampling distribution trying to find Nash Equilibrium between two players in a minimax game. The quality of the samples obtained with GANs is state of the art (see an example in figure 8), but the training is delicate and unstable (see section 4.5).

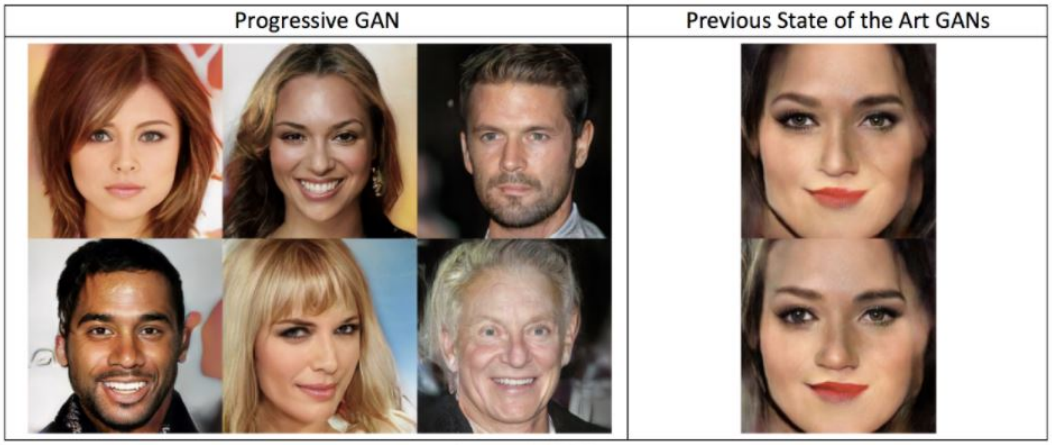

Figure 8: Samples generated from a GAN trained on CelebA-HQ dataset compared with previous results on same dataset. Image taken from *Progressive Growing of GANs for Improved Quality, Stability, and Variation* published in 2017 (see [8]).

4.2 Introduction to the GAN framework

Generative Adversarial Networks are generative models composed of two parts. One is the discriminator D, which takes samples as an input and classifies them as belonging to a data distribution p_{data} or not. The other is the generator G, which generates samples from noise as input. The goal of the generator is to fool the discriminator, and the goal of the discriminator is to correctly classify the inputs as being real (belonging to p_{data}) or fake (generated by G).

Intuitively, we can think about a two player game in which the discriminator learns to accurately distinguish between real and generated images, so the generator learns to produce more and more realistic images that are close to the real ones in order to fool the discriminator.

4.3 Formal definition

A **structured probabilistic model** is a way of describing a probability distribution,using a graph to describe which random variables in the probability distribution interact with each other directly.

Definition 4.1. A Generative Adversarial Networks is a structured probabilistic model containing latent variables $z \in \mathcal{Z}$ and observed variables $x \in \mathcal{X}$ related by a generator and a discriminator.

Definition 4.2. A generator G is a function $G_{\theta^{(G)}} : \mathcal{Z} \to \mathcal{X}$ with parameters $\theta^{(G)}$, differentiable with respect to the input and parameters.

Definition 4.3. A discriminator D is a function $D_{\theta^{(D)}} : \mathcal{X} \to \{0, 1\}$ with parameters $\theta^{(D)}$, differentiable with respect to the input and parameters.

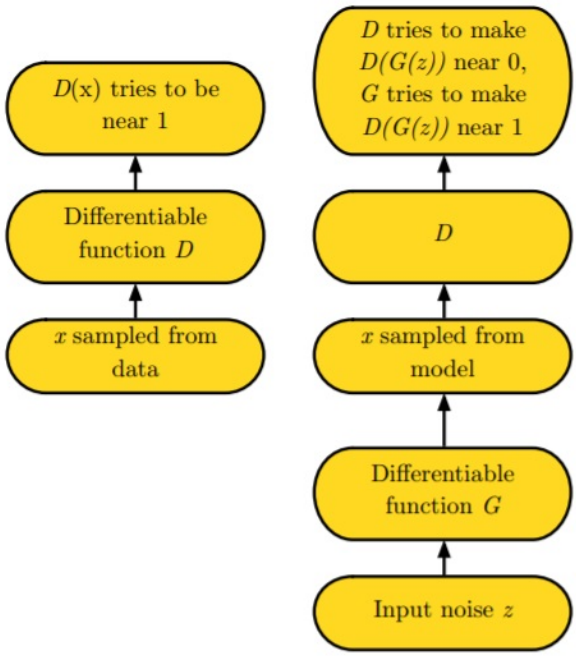

Figure 9: An overview of the architecture of the GAN framework. The game is composed of two parts: Left part shows scenario where random samples from training set are passed to the discriminator. Right part shows the scenario where generated samples are passed to the discriminator. The training process involves steps in each scenario (see section 4.4). Image extracted from [6].

4.4 The training process

The training process is done following a two player minimax game with value function:

$$\min_{\theta^{(G)}} \max_{\theta^{(D)}} V(D, G) = -\frac{1}{2}\mathbb{E}_{x\sim p_{data}}[\log D(x)] - \frac{1}{2}\mathbb{E}_{z\sim p(z)}[\log(1 - D(G(z)))]$$

In the case where G and D are represented by a neural networks, we can compute the derivatives of G and D with respect to $\theta^{(G)}$ and $\theta^{(D)}$ using *backpropagation* (see 3.2). The full training algorithm for GANs is composed of optimization steps in both the generator and discriminator:

for number of training iterations **do**
  for k steps **do**
    • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
    • Sample minibatch of m examples $\{\boldsymbol{x}^{(1)}, \ldots, \boldsymbol{x}^{(m)}\}$ from data generating distribution $p_{\text{data}}(\boldsymbol{x})$.
    • Update the discriminator by ascending its stochastic gradient:

$$\nabla_{\theta_d} \frac{1}{m} \sum_{i=1}^{m} \left[\log D\left(\boldsymbol{x}^{(i)}\right) + \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right) \right].$$

  end for
  • Sample minibatch of m noise samples $\{\boldsymbol{z}^{(1)}, \ldots, \boldsymbol{z}^{(m)}\}$ from noise prior $p_g(\boldsymbol{z})$.
  • Update the generator by descending its stochastic gradient:

$$\nabla_{\theta_g} \frac{1}{m} \sum_{i=1}^{m} \log\left(1 - D\left(G\left(\boldsymbol{z}^{(i)}\right)\right)\right).$$

end for

Figure 10: Training algortithm of GANs.

Note that k is an hyperparameter that represents the number of times we train the discriminator before we train the generator. The value of this hyperparameter is currently in discussion (see [4]), but mostly $k = 1$ is used for providing good results and for being the least expensive option in computational terms. Also, note that the expectations are computed as approximations in batches of size m.

Convergence of the algorithm Convergence of the Algorithm is proven in [4] giving G and D enough capacity (no limits in representability). However, GANs can represent a limited family of p_{model} distributions, so p_{model} converges to an approximation of p_{data}.

4.5 Instability of training

As discussed in *Towards Principled Methods for Training Generative Adversarial Networks* [2], GAN training can be instable, which leads to bad results or even non convergence of the algorithm. While will not be discussed in this work, it's important to take those in mind when training a GAN. One solution can be to use known stable architectures such as DCGAN [12], which we will use. New architectures with better theoretical bases for stability are being proposed and developed. In particular, in this work we will try a variant of the original GAN, WGAN as proposed in [3] that tries to address stability issues.

5. GANs for Anomaly Detection in images

5.1 Introduction

Given a data set that follows a distribution, anomaly detection methods determine which instances of it are unusual.

In this work, we use the power of GANs in sampling from image distributions to implement an unsupervised anomaly detection method. The goal of the method is to identify anomalous images and moreover, **to detect local anomaly segments** within this images. The data set used in this section is MNIST, a collection of handwritten digit images that is widely used in computer vision. The dataset contains 55000 training images, 5000 validation images and 10000 testing images. The images are in grey scale and have dimensions of 28x28 meaning that we work in a 784-dimensional space.

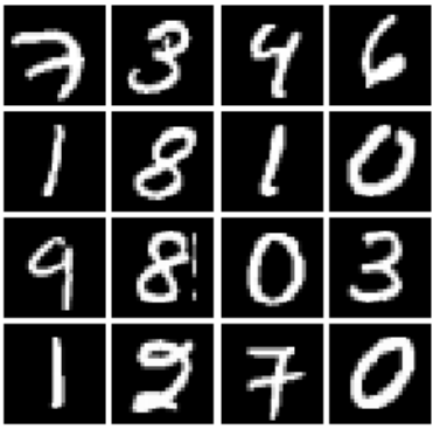

Figure 11: Some examples of MNIST images.

To perform anomaly detection, we will take MNIST images and put a mask in them in order to generate anomalous images.

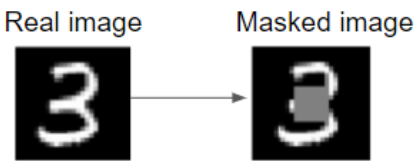

Figure 12: Example of masked 'anomalous' image.

5.2 Problem statement

We are given a set of N 'normal' images (without anomalies) and we train a GAN to learn the manifold $\mathcal{X}$ where 'normal' images live. By learning that image distribution, we are encoding an approximation of 'normal' variability. Details can be seen in section 5.3.

On the other hand, GANs provide the mapping $z \mapsto x$, but don't provide the inverse mapping $x \mapsto z$. For anomaly detection, we would like to find correspondent z to a given image x.
One approach could be with a search for a good candidate z in latent space $\mathcal{Z}$. Linear interpolation in latent space shows that points close to each other seem to generate visually similar images.

Figure 13: Generated images resulting from linear interpolation between points in latent space.

This shows that latent space has smooth transitions and justifies this approach. In this work, we implement a method to make a search in latent space: given a query image x, we try to find the best point z in $\mathcal{Z}$ such that, being G the generator of a GAN trained on a data set, $G(z)$ is the most similar to x.
If x follows the distribution of the training data, that is $x \in \mathcal{X}$, then $G(z)$ and x will be almost equal. If x doesn't follow the distribution, as in the case of an anomalous image, the method won't be able to find a good z to match whole image, so $G(z)$ and x will be different in anomaly regions. Details of this part can be seen in section 5.4 and 5.5.

Finally, for testing we use images with anomalous segments never seen during training. With our method we aim to identify potential anomalous regions within those images. Details of this part can be seen in section 5.6.

5.3 Learning to generate MNIST Digits

As described in section 4, we trained a GAN on MNIST to be able to generate samples that follow an approximation of MNIST dataset distribution.

In this case, we implemented different types of GANs: GAN as presented in the original paper [4], WGAN with gradient penalty proposed in paper [7] and DCGAN, as proposed in [12] . While

evaluation of the quality of generated images hard since there isn't a quantitative metric, results provided by DCGAN are by far better looking than the others.

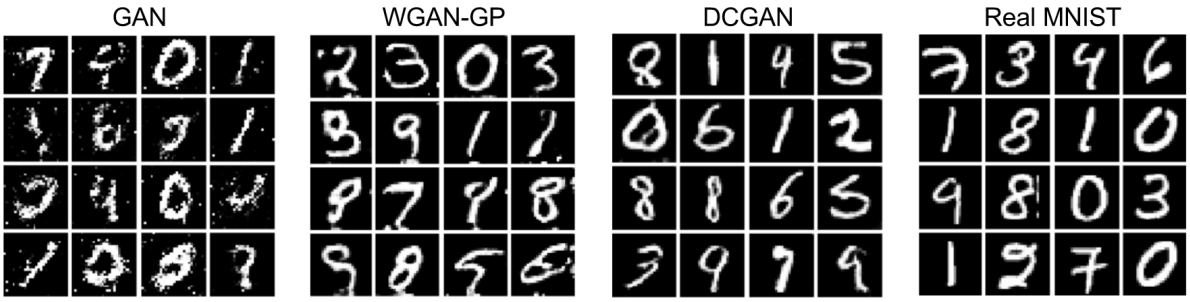

Figure 14: Generated samples after 5000 iterations of training.

Also, DCGAN converged much faster than others. In the view of the results, we have done the final implementation with DCGAN. For concrete details of the networks used as generator and discriminator and the whole architecture, see table 1 in the appendix.

5.4 Search in latent space

Once the GAN is trained, we fix the generator and discriminator parameters.

To find a z such that $G(z)$ is most similar to input image x, we start randomly sampling z_0 from p_z distribution, and compute $G(z_0)$ image. Then we use a loss $\mathcal{L}$ (see section 5.5) that defines the degree of similarity between input image and current image $G(z_0)$. Finally, we compute gradients (using backpropagation as described in section 3.2) to update z_0 so we get a new position z_1 in latent space. In an iterative process, we update position in latent space for k steps in order to find the best candidate z^*.

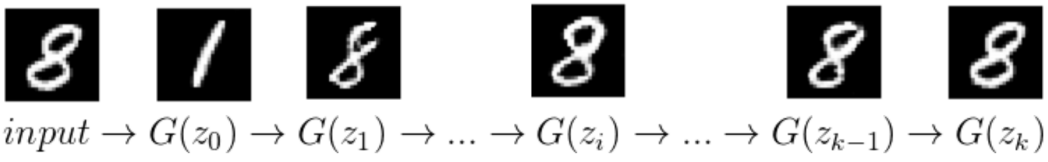

Figure 15: Example of search in latent space in k steps, starting from a random position z_0. Note that in this case input image doesn't contain any anomalous segment, so that the search can find z such that G(z) is almost equal than input image.

The formal implementation of this search is the following algorithm:

Algorithm 1 Search in Latent Space

input: image x, generator G_θ, discriminator D_θ, hyperparameters

initialize z_0, v
for $i = 1, ..., k$ **do**
    $v_i \leftarrow \rho \cdot v_{i-1} - s \cdot \nabla_z \mathcal{L}(G(z_{i-1}), x)$
    $z_i \leftarrow z_i - \rho \cdot v_{i-1} + (1 + \rho) \cdot v_i$
end for
return $z, \mathcal{L}(G(z_{i-1}), x)$

There are some points we want to note with respect to the algorithm:

- For the position update we have choosen Nesterov update strategy (see section 2.2) since it worked better than the others.
- We perform the search with multiple z points, meaning that we find multiple position candidates and average losses among them. This is more robust because of individual searches can get stuck in bad local optima. Concretely, we use a batch size of 128 position candidates.
- k (nº of iterations), ρ (friction coefficient of Nesterov Momentum) and s (learning rate) are hyperparameters in addition to the loss hyperparameters (see section 5.5).

Search results examples

As previously explained, performing a search with a 'normal' input is expected to return almost an equal image. However, we must take in account that input images come from the test set, and have never been seen during training. On the other hand, image inputs that contain anomalous segments are expected to return results that match the input image except in anomalous regions.

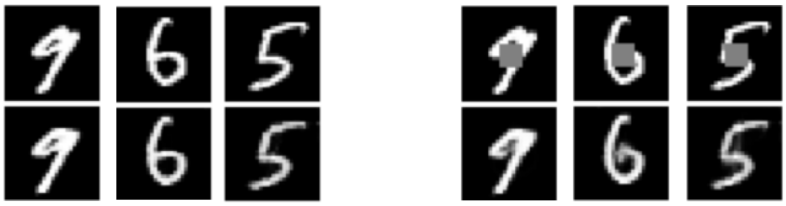

Figure 16: Top row show input images and bottom row show search results for 'normal' input images (left) and for anomalous input images (right). Note that for anomalous images, results are close to it's corespondent unmasked digits, that is exactly what we want to detect anomalies (see 5.6).

It is obvious that search results depend on which type of mask we are using, it's size and pixel values, and the type of digit used. Also, on the loss and the choices of search hyperparameters. Evaluation of the results of the search has been done in section 5.7. In order to get better results, we tried different losses (see final choices in 5.5) and we tuned the hyperparameters (see section 5.7). Final values used as hyperparameters can be found at the apendix section A.2.

5.5 Loss functions

In order to define a 'similarity' measure for searching in latent space, we define the loss in two parts, the context loss $\mathcal{L}_c$ and the perceptual loss $\mathcal{L}_p$.

Context Loss. We define a L^2 norm that measures pixel-wise differences between the images.

$$\mathcal{L}_c(z_i) = ||x - G(z_i)||^2$$

Perceptual Loss. In order to keep generated images visually consistent within the distribution, we define a loss that penalizes anomalous segments in generated images. This way, despite an anomalous input image, we keep our latent space search within the learned distribution. In order to achieve this, we penalize big differences in high level features of the image using the last feature layer f of the discriminator.

$$\mathcal{L}_p(z_i) = ||f(x) - f(G(z_i))||^2$$

The final loss $\mathcal{L}$ is a combination of this perceptual and contextual losses, with α as a hyperparameter (see final value in A.2):

$$\mathcal{L}(z_i) = \alpha \cdot \mathcal{L}_c(z_i) + (1 - \alpha) \cdot \mathcal{L}_p(z_i)$$

Visual interpretation of Perceptual Loss

To get an intuition of what the term $\mathcal{L}_p$ is doing, we present an small experiment designed to show how $\mathcal{L}_p$ for an anomalous image is related to the results of the search. In order to do this experiment, we have chosen a difficult search: the number seven with an anomaly in it's horizontal line. It's important to note that only a portion of sevens in MNIST images have an horizontal line, which makes the search end in 'sevens' without horizontal lines. In addition, we have performed an incomplete search (less iterations). This choices have been made in order obtain some bad position candidates among the 128 returned.

If we compute $\mathcal{L}_p$ scores between input x and all 128 candidates and sort them from best to worst score, we obtain the results shown in the following figures.

Figure 17: From left to right, top three scores of perceptual loss (lower score is better).

We can observe that lower scores of $\mathcal{L}_p$ are given to those candidates that share features with the input. This results also show that, despite having an anomaly in one part of the image, features that are not affected by the anomaly help the search finding good candidates.

Figure 18: From left to right, bottom three scores of perceptual loss (lower score is better).

This results show that $\mathcal{L}_p$ seems to be giving high values to candidates that don't share features with the input, and moreover, to anomalous images that don't follow the digit distribution.

5.6 Anomaly Detection

With search candidates z^*, we compute the map of differences $|x - G(z*)|$, that we will call **anomaly density map**, in order to find segments with high density regions of differences. If input image x has corrupted segments that don't follow the learned distribution, the differences map will show those as anomalous regions. To do that, we compute differences for all 128 candidates returned from search, and then average them.

Figure 19: Anomalous images (first column) and it's mean anomaly density map (second column) for six first images of MNIST test set. White indicates big difference and black indicates no difference.

As can be seen, detection of anomalies isn't obviously perfect. The fact that $\mathcal{L}_c$ is an average of the differences between input image and generated image (see 5.5), makes the search try to match also the anomaly region which have pixel value of 0.5. As long as $\mathcal{L}_c$ averages over all image, the scale of the digit will be slightly smaller than input one in order to lower $\mathcal{L}_c$ despite having differences in all border pixels. For this reason, we can see residual differences located in the borders of the digit.

Postprocessing

In order to address some of the issues described, we perform a postprocess of the anomaly density map in order to eliminate small border regions and fill holes in high density regions.
In the same spirit that CNNs (see 3.3), we use a convolution with a filter of ones and size 3x3 over the image and then a threshold t. This way, final pixel values are weighted to the values of it's neighbours. This process penalizes regions with low density than end up with values under t while regions with high density end up with values above t, resulting in low density regions to disappear and high density regions to fill holes inside. See second part of figure 20 for an example of this process.

Prediction of anomaly region

The prediction is done averaging among all postprocessed density maps. For evaluation porpuses, the prediction is finally binarized with a threshold of 0.5 in order to be compared with ground truth.

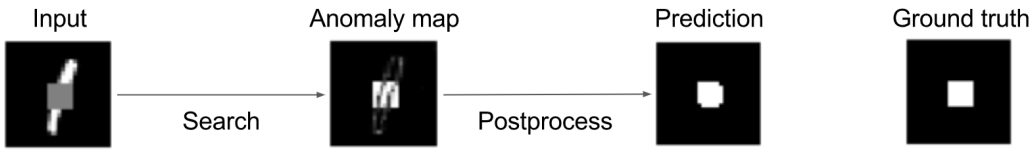

Figure 20: An overview of the full process from input image to final prediction of anomaly region.

5.7 A metric for evaluating predictions

We have chosen Jaccard index, also known as *intersection over union* (IOU) to evaluate our results. IOU is computed as:

$$IOU(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

being A and B the predicted region and ground truth in our case. A property of IOU is that $0 \leq IOU(A, B) \leq 1$. Since it's a bounded index, we can use it to compare different predictions.

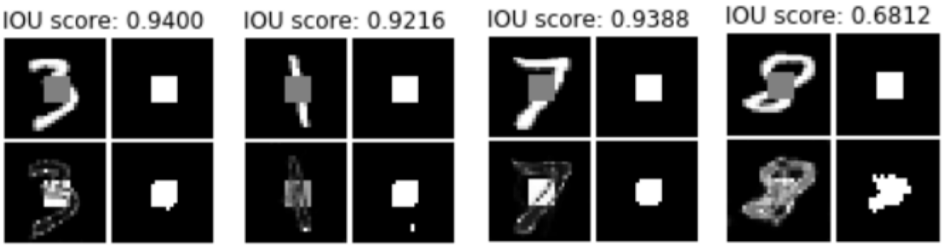

Figure 21: Examples of IOU scores. First row: Input image and ground truth. Second row: Anomaly map and prediction.

Global score

As a global metric of our method, we average IOU scores among random sample of 1000 test images. The same metric will be used in order to do hyperparameter tunning.

Generalization

In terms of performance on test images, we achieved a global score of 0.87, which suggests that proposed method is good in detection of local anomalies in MNIST images. Further evaluation must be a comparison with other anomaly detection methods and with different types of masks, which can be a continuation of this work.

6. Detecting anomalies in Histopathological Images

Examination of histological preparations is a labor intensive process done by pathologists. The result of this examination, as metastasis observation in breast cancer biopsies, determine diagnostics, treatment decisions and disease prognostics among other things. Slides are large and can contain a lot of different tissue cells, but pathological regions can be relatively small.

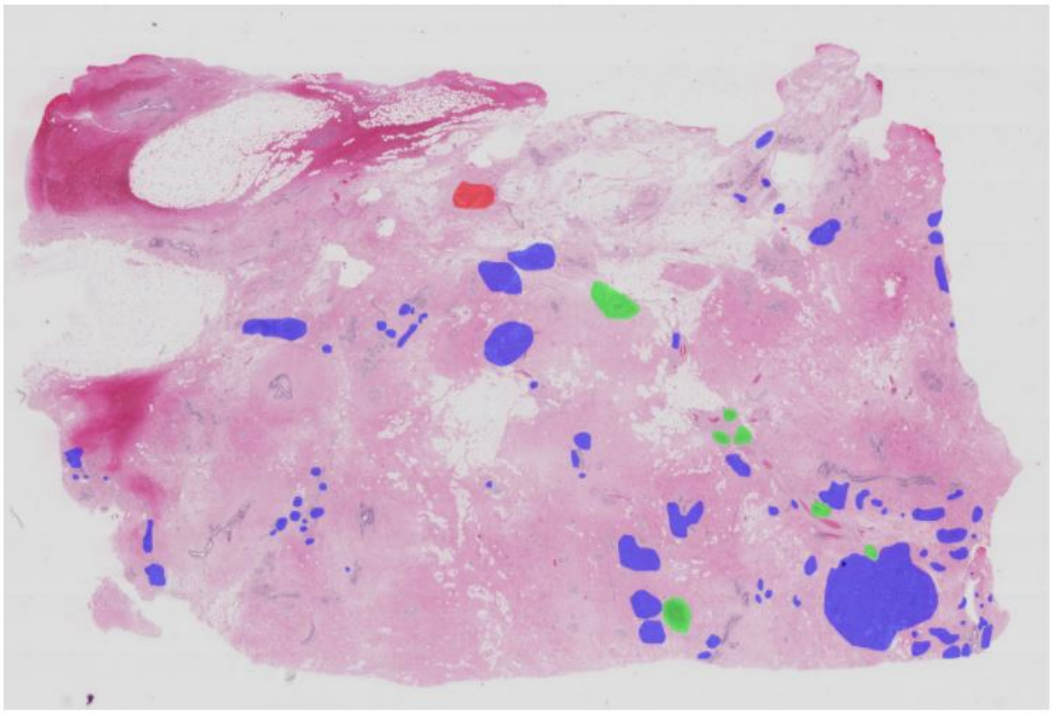

Figure 22: Example of whole slide image with annotations of pathological regions from ICIAR segmentation competition.

Slide examination can be seen as an anomaly detection process where the pathologist knows the distribution of the healthy tissues and detect anomalies that can be found in them. Actually, pathologists work is much more complex than that, since they can make a diagnostic out of the examination, but a good anomaly detection method in histopathological images could be useful providing region proposals as potentially anomalous. Then, one could analyze directly this regions instead of the examining the whole slide. It's obvious that such a method can be difficult to develop, especially with current lack of data in this particular field and limited resources. For that, our application aims only to explore the potential use of concrete techniques in this particular task.

6.1 The dataset

ICIAR Grand Challenge provides individual microscopy images for classification as well as whole slide images for segmentation. Whole slide images have additional technical difficulties because of being very large and occupy near 8GB of RAM per slide when fully opened. For this reasons, we decided to use individual microscopy images which are easier to work with.

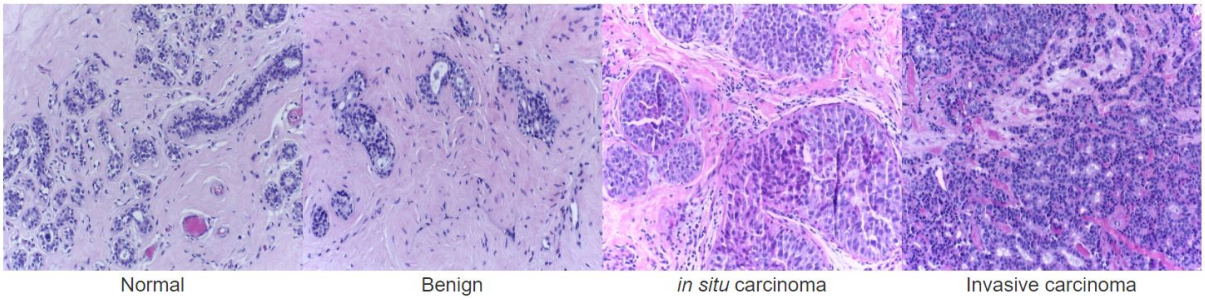

Figure 23: Example image for each class of ICIAR classification competition.

For each class, the dataset contains 100 microscopy images. We used *Normal* class images (which we will refer from now as healthy) to train our GAN and *Invasive carcionma* images as anomalous images.

6.2 Preprocessing of images

In order to have enough data, we cutted images in patches of size 64x64, which resulted in a training set of 73000 images and 1000 test images.

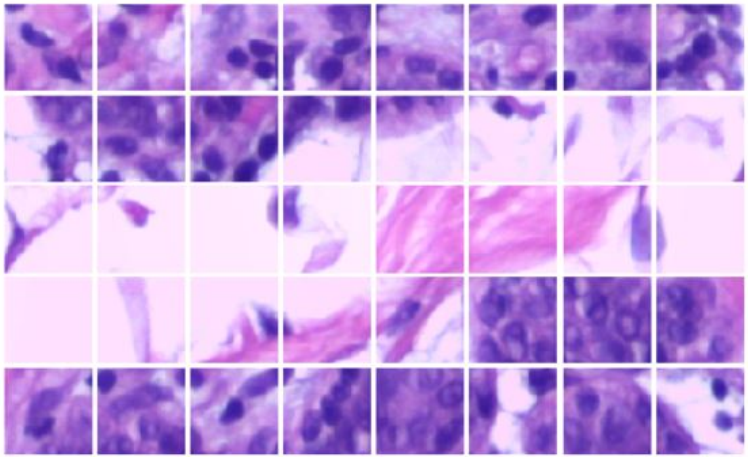

Figure 24: Example patches for healthy images.

6.3 The complexity of the task

Compared to previous section, there are a some differences that greatly increase the difficulty of performing anomaly detection in histopathological images compared to MNIST digitis:

- **Bigger, RGB images**. With an patch size of $64x64x3$ we are working on a 12288-dimensional space instead of the 784-dimensional space of MNIST images.

- **Variable complex images**. In comparison with white digits above a black background, histological images have a plenty of combinations of colors, shapes and textures. Thus, they end up with much higher variability than MNIST digits.

- **Anomalies**. Unlike masks used in previous section, 'anomalies' in histopathological images are not just structural and constant. For what can we say, pathological regions seem to contain high density of cellular nucleus that also have different sizes and forms than healthy ones. For this reason, it isn't obvious which zoom scale is optimal for anomaly detection. Since computations are expensive with bigger images, we have chosen a minimum reasonable size of 64x64 in which there are around 0 to 5 cell nucleus.

- **GAN training**. As stated in section 4, GAN training is unstable an delicate. While GAN training on the MNIST dataset is simple and well studied, their strategies often fail in other datasets. Additionally, faster training can lead to better results due to many iteraitons of he process, while longer and costly trainings have resource constraints. For comparison, training with MNIST lasted about 5 minutes while in this part, the training lasted more than 10h.

6.4 Enconding healthy tissue variability with GANs

We have the dataset of 73000 patches to train a GAN in order to generate sample healthy patches. Model used was a DCGAN [12] adapted to new image sizes. After 100 epochs of training (in one epoch all training examples have seen once), the results are the following:

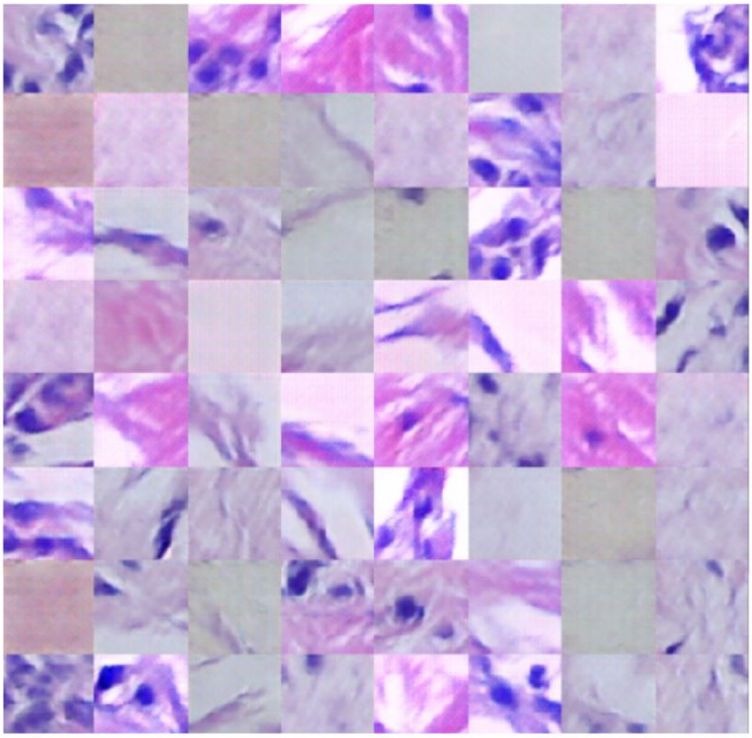

Figure 25: Generated samples after 100 epochs of training. Note that there are some 'greyish', 'empty' patches, which are those that not contain cells and are often found in whole images (see images in figure 23).

Our trained GAN is able to generate realistic images, what suggests that it can learn a good approximation of healthy tissue variability. On the other hand, we think that better results can be achieved with more data and patches of different sizes. Also, some preprocessing of the images like color normalization and other processes can be done. This will be part of the continuation of this work.

Better evaluation of this results can be done with latent space interpolations, which will be also part of the continuation of this work.

6.5 Latent space search and anomaly detection

In this section we use same methods as with MNIST digits.

For anomaly detection, we perform a search in the latent space and compute a map of differences (anomaly density maps). For healthy patches, the search is expected to find good candidates resulting in anomaly density maps with low values. On the other hand, the search with pathological patches is expected to find bad results ending with higher anomaly density maps. Also, we try a search with composed patches, with one healthy part and one pathological part. We expect to see higher

differences in the pathological part compared with the healthy part.

Results for some examples are summarized in figure 26:

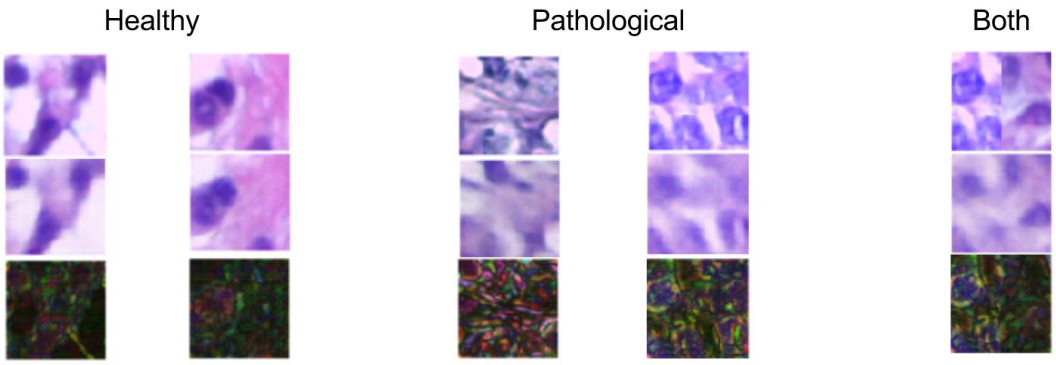

Figure 26: Anomaly detection in histopathological images. *First row*: input patches (healthy, pathological, and patch composed of pathological part on the left and a healthy part on the right. *Second row*: latent spaces search average results. *Third row*: averaged anomaly densities maps scaled by a factor of 3 for better visualization. The search is able to find good candidates for healthy patches resulting in low differences in the anomaly map. Search with pathological patches provide bad results ending with higher differences. For composed patch, note the predominance of differences in the left part of the anomaly map.

Because of resource and time limitations (see some of the difficulties in 6.3), anomaly detection has not been done in a whole set of patches, which is necessary for further evaluation of the results and will be part of the continuation of this work. Instead, a set of 30 examples has been examined and average results have been presented.

Results provided for healthy patches seem to confirm that GAN training was good enough to learn an approximation of health data manifold and search is performed correctly.

Also, results for pathological patches show good performance in detecting whole anomaly patches. Proper postporcesing of anomaly density maps and determination of an anomaly threshold t will be part of the continuation of this work.

The results for composed patches show that, with proper postprocessing, local anomaly segments can be detected within patches. The use of real patches with both healthy and pathological segments instead of composed ones will be part of continuation of this work.

In summary, we think that results provided in this section show potential in whole-slide local segment anomaly detection and justify further development of proposed methods.

7. Conclusions

7.1 Evaluation of the results

The goal of implementing an application that uses GANs for anomaly detection in image distributions has been achieved, with distribution encoding with GANs, latent space search and anomaly detection as sub parts of the initial goal.

Initial prototype for detecting anomalies in MNIST digits has provided very good results.

Detecting anomalies in histopathological images is an ambitious and complex task. In this part, presented application can be further developed to provide anomaly region proposals for whole slide images. Still, presented results in this part are promising and are something we are really proud of.

Finally, a large and diverse set of theoretical concepts and techniques have been analyzed and learned, which is valuable knowledge we keep from this work.

7.2 Future work

From a theoretical sight, an immediate continuation of this work is to try more robust and stable models for learning image distributions. Also, design better loss functions for latent space search could further improve results. Inverse mapping from images to latent space could be tried with different approximations, in particular we would like to try Conditional GANs [10].

On the other hand, we would like to further develop our anomaly detection method in histopathological images: changing patch size, trying normalization of colors in images, postprocessing of anomaly density maps, evaluation in whole set of patches and finding best and bigger data sources. Also, we would like to use the application to perform anomaly detection in whole slide images, and compare performance with other methods.

8. Bibliography

References

[1] M. A.Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015.

[2] M. Arjovsky and L. Bottou. "Towards Principled Methods for Training Generative Adversarial Networks". In: *ArXiv e-prints* (Jan. 2017). arXiv: 1701.04862 [stat.ML].

[3] M. Arjovsky, S. Chintala, and L. Bottou. "Wasserstein GAN". In: *ArXiv e-prints* (Jan. 2017). arXiv: 1701.07875 [stat.ML].

[4] I. J. Goodfellow et al. "Generative Adversarial Networks". In: *ArXiv e-prints* (June 2014). arXiv: 1406.2661 [stat.ML].

[5] Ian J. Goodfellow. "NIPS 2016 Tutorial: Generative Adversarial Networks". In: *CoRR* abs/1701.00160 (2017). arXiv: 1701.00160. URL: http://arxiv.org/abs/1701.00160.

[6] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[7] Ishaan Gulrajani et al. "Improved Training of Wasserstein GANs". In: *CoRR* abs/1704.00028 (2017). arXiv: 1704.00028. URL: http://arxiv.org/abs/1704.00028.

[8] Tero Karras et al. "Progressive Growing of GANs for Improved Quality, Stability, and Variation". In: *CoRR* abs/1710.10196 (2017). arXiv: 1710.10196. URL: http://arxiv.org/abs/1710.10196.

[9] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks". In: *Advances in Neural Information Processing Systems 25*. Ed. by F. Pereira et al. Curran Associates, Inc., 2012, pp. 1097–1105. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf.

[10] Mehdi Mirza and Simon Osindero. "Conditional Generative Adversarial Nets". In: *CoRR* abs/1411.1784 (2014). arXiv: 1411.1784. URL: http://arxiv.org/abs/1411.1784.

[11] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. "Understanding the exploding gradient problem". In: *CoRR* abs/1211.5063 (2012). arXiv: 1211.5063. URL: http://arxiv.org/abs/1211.5063.

[12] Alec Radford, Luke Metz, and Soumith Chintala. "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks". In: *CoRR* abs/1511.06434 (2015). arXiv: 1511.06434. URL: http://arxiv.org/abs/1511.06434.

[13] Thomas Schlegl et al. "Unsupervised Anomaly Detection with Generative Adversarial Networks to Guide Marker Discovery". In: *CoRR* abs/1703.05921 (2017). arXiv: 1703.05921. URL: http://arxiv.org/abs/1703.05921.

[14] Raymond A. Yeh et al. “Semantic Image Inpainting with Perceptual and Contextual Losses”. In: *CoRR* abs/1607.07539 (2016). arXiv: 1607.07539. URL: http://arxiv.org/abs/1607.07539.

A. Appendix

A.1 MNIST DCGAN

Discriminator D	**Generator** G
input batch of images: array of size [128,784]	input batch of noise: array of size [128,96]
Conv: 32 Filters, 5x5, Stride 1, Leaky ReLU	Fully Connected: size 1024, ReLU
Max Pool: 2x2, Stride 2	BatchNorm
Conv: 64 Filters, 5x5, Stride 1, Leaky ReLU	Fully Connected: size 7 x 7 x 128, ReLU
Max Pool: 2x2, Stride 2	BatchNorm, resize
Fully Connected: size 4 x 4 x 64, Leaky ReLU	BatchNorm
Fully Connected: size 1	Conv Transposed: 64 Filters, 4x4, Stride 2, ReLU
	Conv Transposed: 1 Filter,4x4, Stride 2, TanH

Table 1: Details of the architectures used in MNIST sample generation.

A.2 Hyperparameters

Hyperparameter	**Value**
k	2000
ρ	0.99
s	1e-2
α	0.5

Table 2: Hyperparameter values used for anomaly detection in MNIST (section 5).

Hyperparameter	**Value**
k	3000
ρ	0.95
s	1e-1
α	0.5

Table 3: Hyperparameter values used for anomaly detection in Histopathological images (section 6).

A.3 Code

Code can be found in the following repository:
https://github.com/guillembatiste/TFG/tree/master/