UNIVERSITAT DE BARCELONA

FUNDAMENTALS OF DATA SCIENCE MASTER'S THESIS

Generating Synthetic Intestine Images

Author:
Stefan IVANOV

Supervisor:
Santi SEGUÍ

A thesis submitted in partial fulfillment of the requirements for the degree of MSc in Fundamentals of Data Science

in the

Facultat de Matemàtiques i Informàtica

June 28, 2019

UNIVERSITAT DE BARCELONA

Abstract

Facultat de Matemàtiques i Informàtica

MSc

Generating Synthetic Intestine Images

by Stefan IVANOV

Capsule endoscopy is a non-invasive medical procedure used to record images of the gastrointestinal tract. While this method is a better alternative for patients, it presents a difficulty to doctors who need to go over as much as 50000 images. Scientists are developing machine learning algorithms that will automatically throw away images free of any anomalies. Like other medical applications, however, available data to train such models is sparse. Therefore, we attempt to create synthetic images that can be used as substitution. For the purpose we have used generative adversarial networks (GANs) as they have recently shown great promise for problems like this one. Training a classifier on both the real and synthetic data, we achieve an increase in the classification accuracy for a dataset of intestine images.

Acknowledgements

Many thanks to Pablo Laiz for always being close to answer questions about deep learning.
An honourable mention of Alvaro Parafita who is way better at GANs than me.
Much appreciation of Elia Ficapal for it would have been harder to work on this project without the coffees we shared.
And, most of all, I am very grateful to Santi Segui for not only being a resourceful and helpful professor, but also quite the cool supervisor.

Contents

Chapter 1

Introduction

1.1 Capsule Endoscopy

Capsule endoscopy is a modern medical procedure used to record images of the gastrointestinal tract to be analysed for medical diagnosis. Patients swallow a capsule containing a tiny camera and it travels trough their body, taking as much as 50000 (or more) images, transmitted wirelessly to a portable device nearby. The novelty of this approach is it allows doctors to examine parts of the tract unreachable by alternative endoscopy methods, and furthermore it is non-invasive. Common reasons for using capsule endoscopy include searching for polyps, uclers and tumors, as well as diagnosis of unexplained bleeding or other diseases. Caffrey et al., 2008 provide further details about capsule endoscopy. Examples of usage are presented by Zhuan Liao et al., 2010 (investigate detection, completion, and retention rates of capsule endoscopy in the small bowel) and Tong et al., 2012 (investigate iron deficiency anemia without evidence of gastrointestinal bleeding using capsule endoscopy).

The problem associated with this method is the infeasibility of a doctor manually reviewing so many photos taken inside the body. While diagnosis should certainly be done by a professional, automatically removing pictures that, with high certainty, don't have any anomalies on them helps the process significantly. Researchers have developed machine learning algorithms to recognise those photos, but they face some difficulties with the data to train them. For instance, data from different patients has high variability. Furthermore so if different capsule cameras are used. Most of all, as in other medical applications, data is simply not enough. Even worse, the positive and negative class are often extremely unbalanced. Consider the 50000 images the camera in the capsule takes - even if the patient had cancer, it would only show on a very few of those frames. There are different techniques to alleviate this problem, most frequently data augmentation. This means creating new images by applying simple operations like rotation, translation, scaling etc to the existing ones. While this enlarges the dataset artificially, it doesn't create much variability, as new images are essentially the old ones with some transformation. Recently, scientists' interest has moved onto generating synthetic image data.

1.2 Generating Synthetic Data

The main goal of this project is to successfully generate synthetic data, given a set of real data it should resemble. In this application, we have images of intestines produced from capsule endoscopy. The images come from different classes, depending on what they capture - blobs, bubbles, walls etc. We need to be able to produce images that look true and that capture the diversity of the classes.

Synthetic image generation is a field that has fascinated machine learning enthusiasts with its numerous applications ranging from rather useful, like image inpainting (repairing an image by filling up the missing part, see Yu et al., 2018) and text to image translation (Zhang et al., 2017), to some quite quirky, like cross-domain transfer (recreating a painting in the style of another artist, see Jones, 2017 and figure 1.1) and creating fictional celebrity faces (Hart, 2017).

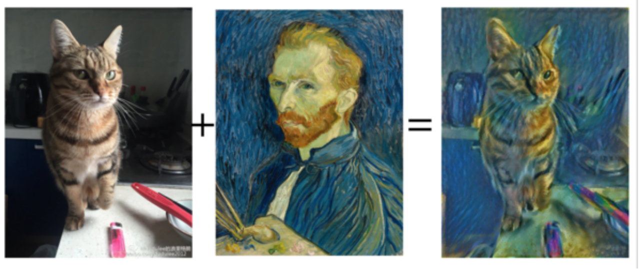

FIGURE 1.1: Example of generating synthetic imagery - cross-domain adaptation; in this case we transform a real photo to look like a painting in the style of Van Gogh; source: *Deep learning for hackers with MXnet (2): Neural art*

A traditional approach to synthetic image generation is variational autoencoders (VAE), where we encode the real data to a latent space, represented by a mean and variance. This allows us to sample this space for vectors, which when decoded produce new synthetic data. The original paper on the subject by Kingma and Welling, 2014 contains the details and figure 1.2 provides a schema of the main idea. While VAEs offer neat probabilistic reasoning and relatively easy implementation, recently another method has shown promise with a much higher visual fidelity and variety - generative adversarial networks (GANs). This is precisely why we have chosen to tackle the problem by using GANs. In this paper we analyse their performance for the task we have set.

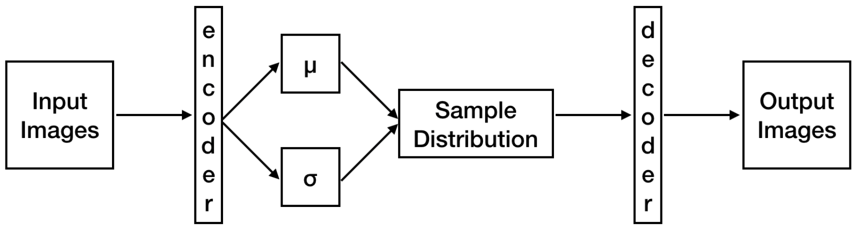

FIGURE 1.2: This diagram shows the main idea behind VAEs as explained here

1.3 Main Contributions

The main steps we followed while working on this project are as follows:

- Extensivelly researched the theory behind deep learning, convolutional neural networks, generative adversarial networks and their variants.
- Experimented substantially on the MNIST dataset to produce first GAN versions.
- Moved on to working with the intestine dataset, implemented the original GAN.
- GAN wouldn't train well, so optimizations like Wasserstein GAN were employed.
- Conditional GAN was implemented to impose label conditioning.
- The network performance was assessed and analyzed, we check if automatic image classification is improved with the use of the generated data.

1.4 Report Structure

This section serves to outline the general structure the report will follow.

First, in the next chapter the reader is introduced to the scientific advances we have based this work on. We begin by a brief explanation of convolutional neural networks and how they are used in GANs. We explore in detail the ideas behind GAN and how researchers improved the original model to the state-of-the-art.

Next, in chapter 3 we describe the work done by providing details on the implementation, discussing the problems encountered and how they were solved, as well as analysing the results.

Finally, the last chapter offers a further discussion about the limitations of the project, but also the possible improvements and extensions.

Chapter 2

Background

2.1 Convolutional Neural Networks

In the 1960s multiple scientists contributed to the derivation of backpropagation, notably Dreyfus, 1962, who based his work completely on the chain rule. In the 1970s the general method for automatic differentiation (AD) was published by Linnainmaa, 1976, which together with the rapid development of cheap, powerful GPU-based computing systems in the years to come, ultimately led to the birth of deep learning. As reviewed in Schmidhuber, 2015, deep neural networks revolutionized the field of machine learning and proved successful in many applications like classification, regression etc. Most importantly, they allowed breakthroughs in areas previously lagging like speech recognition (Hannun et al., 2014), natural language processing (Hochreiter and Schmidhuber, 1997) and, what will be of particular interest to us, computer vision.

Traditionally, computer vision methods for detection, segmentation and other tasks were based on hard manual preprocessing steps such as noise reduction, background subtraction or contrast enhancement, followed by error-prone feature extraction like Canny edge detector, localized interest points and so on. Applying deep learning to images largely removed the need for these obsolete methods and offered great results. Papers like Lee, 2016 explore the differences and tradeoffs between the two broad approaches.

Problem is, applying a deep neural network to an image is not straightforward as we need to preserve the structure. The idea of convolutional neural networks (CNNs) came to tackle this issue. Lecun et al., 1998 was the first to recognize that the ability of multi-layer networks trained with gradient descent to learn complex, high-dimensional, non-linear mappings from large collection of examples makes them obvious candidates for image recognition tasks and suggested the convolutional networks relying on three architectural ideas: local receptive fields, shared weights and spatial sub-sampling.

What this means in practice is a filter (kernel) is applied to a window of fixed size moving throughout the image. Each convolutional layer tries to learn the parameters for a set number of such filters. Essentially, every filter should be able to recognize a specific pattern, such as horizontal/vertical edges, colour schemes etc. In addition, pooling layers are usually used to reduce the dimensionality of the image and make the computation more efficient. Most often the max pooling approach is followed, meaning that from a small window of nearby pixels we only take the maximum value. Finally, after the convolutional layers, most architectures have some fully connected layers, as in the traditional multi-layer perceptron, to produce the final result. Figure 2.1 is an example of a CNN that uses all of the above concepts.

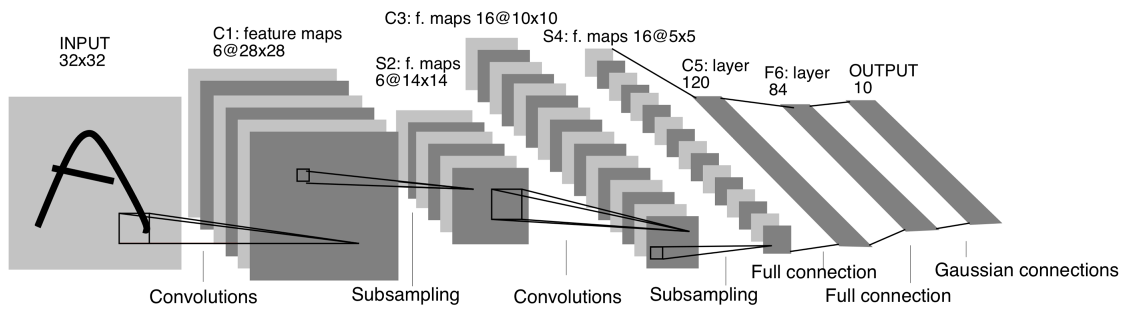

FIGURE 2.1: Lecun et al., 1998 present the architecture of the first CNN LeNet, which consists of a few convolutional and pooling layers, followed by fully connected layers. This network can be used to classify the MNIST digits.

CNNs revolutionized computer vision and their building blocks have been used in many architectures for different purposes. Krizhevsky, Sutskever, and Hinton, 2012 built AlexNet to win the ImageNet Large-Scale Visual Recognition Challenge in 2012, which is a classification task with more than 1000 classes. He et al., 2016 from Microsoft showed an improvement in CNNs by introducing the residual block in their ResNet.

Residual blocks are based on skip connections to a few layers forward. Backpropagating through those allows us to learn initial layers as fast as the final layers. Thus, a residual learning framework is shown to ease the training of networks that are substantially deeper, solving problems like vanishing gradient.

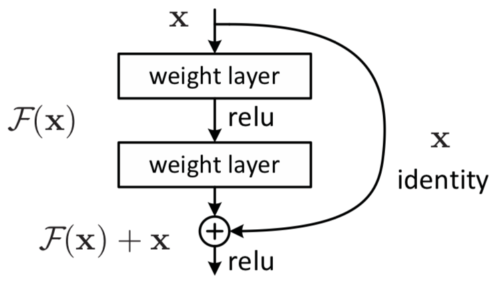

FIGURE 2.2: He et al., 2016 shows the structure of a residual block. A skip connection is introduced and instead of learning the output, we optimize with regards to the difference between output and input - the residual, hence the name.

In this work we have reused these ideas to implement CNNs successfully.

2.2 Generative Adversarial Networks (GANs)

GANs, originally suggested by Goodfellow et al., 2014, propose an adversarial process to estimate a generative model. Two networks are trained simultaneously: a generator that captures the data distribution, and a discriminator that estimates the probability that a sample came from the training data.

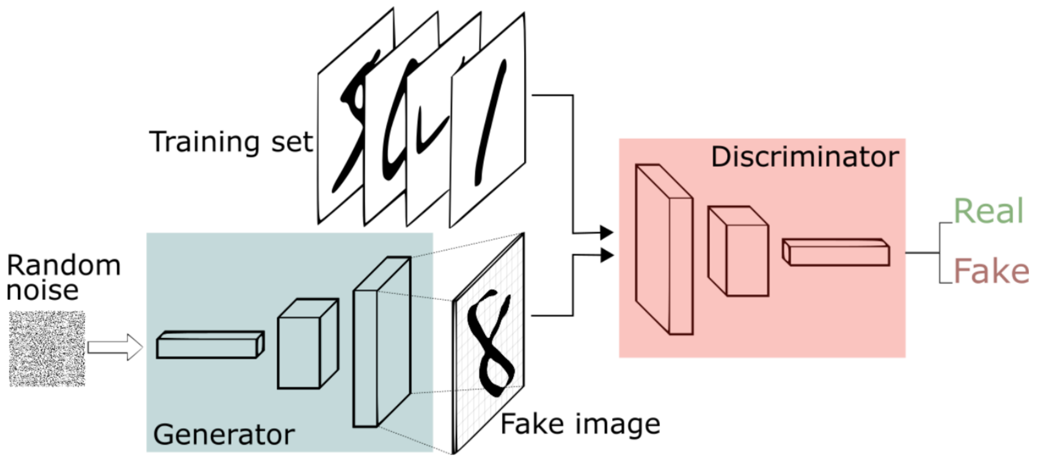

FIGURE 2.3: GAN general diagram; source: *An intuitive introduction to Generative Adversarial Networks (GANs)*

The training process goes as follows: the generator G creates fake images from random noise. The discriminator D learns to distinguish between the fake data and the real set. This in turn improves G as it continuously tries to fool D. This concept is illustrated on figure 2.3.

In other words, D and G play the following two-player minimax game with value function V(G,D), which we need to optimize:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[logD(x)] + \mathbb{E}_{Z\sim p_z(z)}[log(1-D(G(z)))] \tag{2.1}$$

The discriminator is a binomial classifier labelling images as real or fake, thus the architecture of a CNN is usually applied. The generator is similar in structure, but instead of downsampling, we upsample to produce an image from the noise.

Although GANs have shown great promise and have been used successfully in many image generations scenarios, they are particularly hard to train. Even intuitively, it is easier to recognize the artist that painted an image, rather than to paint it yourself. The difficulty lies mostly in GANs' adversarial nature. For instance, if the discriminator becomes too good at its job, the generator gradient vanishes and it learns nothing. This is called the diminishing gradient problem. People also often face mode collapse, that is the generator tricks the discriminator by learning a particular mode, but as real-life data distributions are multimodal, this results in a greatly limited variety of the produced samples. Furthermore, the model parameters easily oscillate, destabilize and sometimes this causes non-convergence, as noted by Hui, 2018. These are some of the problems we encountered while working on this project.

As the scientific community realized the prospects of GAN, many variations and improvements appeared. For example, CycleGAN as proposed by Zhu et al., 2017 is well suited for the domain adaptation example from figure 1.1. Stacked GAN (SGAN, see Huang et al., 2016) uses a hierarchy of top-down stacked GANs, "each learned to generate lower-level representations conditioned on higher-level representations", thus improving image quality (originally, images produced by GANs were often blurry or had checkers patterns). Similarly, Progressive GAN, suggested by Karras et al., 2017, tries to improve results for higher resolution images, but with another approach - by starting at a smaller resolution and continuously growing the network to "model increasingly fine details as training progresses". In the end of 2018, Karras, Laine, and Aila, 2018 from Nvidia built Style GAN on top of the progressive architecture to achieve the state-of-the-art in high resolution human faces generation. In this paper, we will focus on a few GAN improvements as described next.

2.2.1 DCGAN

Radford, Metz, and Chintala, 2015 published a paper making specific recommendations on the architecture of GANs, including:

- Use fully convolutional neural networks as proposed by Springenberg et al., 2014. This means replacing the pooling layers with strided convolutions. Stride is the distance between spatial locations where the convolution kernel is applied. Having a stride larger than 1 results in a dimensionality reduction.
- Furthermore, they remove the fully connected layers, leaving only convolutional layers.
- Batch normalization is recommended to stabilize the learning by normalizing the input to each unit to have zero mean and unit variance. This helps with some of the training problems we mentioned.
- The generator uses a ReLU activation, except for the last layer, which uses tanh. The discriminator uses the leaky ReLU.

This improved network was called Deep Convolutional GAN (DCGAN) and we have implemented it in this project.

2.2.2 Conditional GAN

GANs showed good ability to learn representations with rich variety, but initially we couldn't control the modes of data to be generated. Sometimes it is necessary to enable the generator to produce images of a particular class, as in our case. This can be achieved by simply inputting the class label as an additional parameter in the model. This modification by Mirza and Osindero, 2014 is called Conditional GAN (CGAN). Figure 2.4 illustrates how the generator and disriminator are changed.

This changes the loss function as follows, where now we introduce the label y:

$$\min_G \max_D V(D,G) = \mathbb{E}_{x \sim p_{data}(x)}[logD(x|y)] + \mathbb{E}_{Z \sim p_z(z)}[log(1 - D(G(z|y)))] \quad (2.2)$$

Alternative ways of conditioning the generator's output exist too. Chen et al., 2016 suggest InfoGAN, which can learn meaningful latent variables without any labels on the data. That is, part of the generator output is reserved for a salient variable c, which is sampled from a distribution, learned by the discriminator. As this is more complex and CGAN is more widely adopted, we have stuck with it for this project.

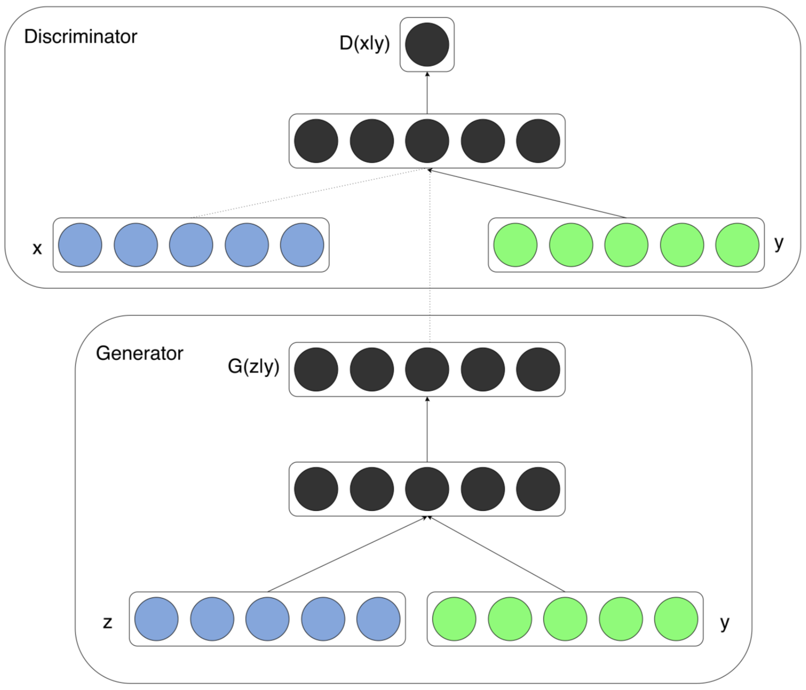

FIGURE 2.4: Conditional GAN architecture as defined by Mirza and Osindero, 2014. Note both generator and discriminator take an additional input y - the label.

2.2.3 Wasserstein GAN

We already discussed how GANs are infamously hard to train. Often the algorithm never converges, and researchers tried to come up with a new cost function that will solve this issue. Let's try to give some insight into the problem:

A generative model tries to learn a probability distribution q that will closely resemble a given distribution p. Distribution similarity is traditionally measured with KL- or JS-divergence, defined as follows:

$$D_{KL}(p||q) = \sum_{x \in \mathcal{X}} P(x) log \frac{P(x)}{Q(x)} \tag{2.3}$$

$$D_{JS}(p,q) = \frac{1}{2} D_{KL}(p||\frac{p+q}{2}) + \frac{1}{2} D_{KL}(q||\frac{p+q}{2}) \tag{2.4}$$

Although JS-divergence is preferred, because it is symmetric, both suffer the same problem, intuitively explained by Hui, 2018. In essence, when the distributions are too different, the gradient diminishes. And when the gradient is close to 0, the generator learns nothing from the gradient descent. In practice, an optimal discriminator produces good information for the generator to improve. But if the generator is not doing a good job yet, we fall into the described pitfall.

Therefore, Arjovsky, Chintala, and Bottou, 2017 introduce a new cost function, using the Earth-Mover (or Wasserstein) distance instead. Defined as the minimum cost of transporting mass in converting the data distribution q to the data distribution p, it benefits from a smooth gradient everywhere, thus escaping the aforementioned problem. Wasserstein GAN (WGAN), as it was named, learns no matter if the generator is performing or not, which is the important breakthrough. We no longer need to balance generator and discriminator carefully. The better the discriminator, the higher quality the gradients we use to train the generator.

$$D_W(p,q) = \inf_{\gamma \in \Pi(p,q)} \mathbb{E}_{(x,y)\sim\gamma}[||x-y||] \tag{2.5}$$

is the Wesserstein distance, where $\Pi(p,q)$ denotes the set of all joint distributions $\gamma(x,y)$ whose marginals are respectively p and q. Due to the infimum, this is further reduced to a more tractable form as follows:

$$D_W(p,q) = \sup_{||f||_{L\leq 1}} \mathbb{E}_{x\sim p}[f(x)] - \mathbb{E}_{x\sim q}[f(x)] \tag{2.6}$$

To optimize this new cost function, we train a neural network similarly to the original GAN, but now we remove the sigmoid activation and instead have the outputs as scalar scores rather than probabilities. This score is interpreted as a judgement of how real the input image seems. To reflect the new role of the discriminator, it is renamed to a critic.

The fact that the Earth-Mover distance is continuous and differentiable means that we can train the critic till optimality. Empirically, the authors of the paper show that this not only stabilizes training, but also successfully avoids mode collapse.

2.2.4 WGAN with Gradient Penalty

Notice in the formula (2.6), f needs to be a 1-Lipschitz function. Lipschitz continuity limits how fast a function can change: there exists a real number such that, for every pair of points on the graph of this function, the absolute value of the slope of the line connecting them is not greater than this real number. To enforce this constraint, the authors of the WGAN paper use weight clipping, meaning they clamp the weights to a fixed box (usually [-0.01,0.01]) after each gradient update. This, however, is not a reliable method. "If the clipping parameter is large, then it can take a long time for any weights to reach their limit, thereby making it harder to train the critic till optimality. If the clipping is small, this can easily lead to vanishing gradients when the number of layers is big, or batch normalization is not used."

Gulrajani et al., 2017 suggest an alternative that will improve training of WGANs. Since a differentiable function f is 1-Lipschitz if and only if it has gradients with norm at most 1 everywhere, we can penalize the model if the gradient norm moves away from its target value 1. Thus, we define a new cost function for WGAN with gradient penalty (WGAN-GP) as follows:

$$L = \mathbb{E}_{\tilde{x}\sim p}[f(\tilde{x})] - \mathbb{E}_{x\sim q}[f(x)] + \lambda \mathbb{E}_{\hat{x}\sim P_{\hat{x}}}[(||\nabla_{\hat{x}} f(\hat{x})||_2 - 1)^2], \tag{2.7}$$

where x represents the real data, $\tilde{x}$ is as produced by the generator, and $\hat{x}$ is sampled from x and $\tilde{x}$ with t uniform from 0 to 1:

$$\hat{x} = t\tilde{x} + (1-t)x$$

As one can notice, this is the same cost function as before, but with the gradient penalty added. λ is a coefficient that the authors set to 10. An important note here is that while previous GAN implementations usually did batch normalization in the critic, this should now be avoided since we penalize the norm of the critic's gradient with respect to each input independently, rather than with respect to the batch.

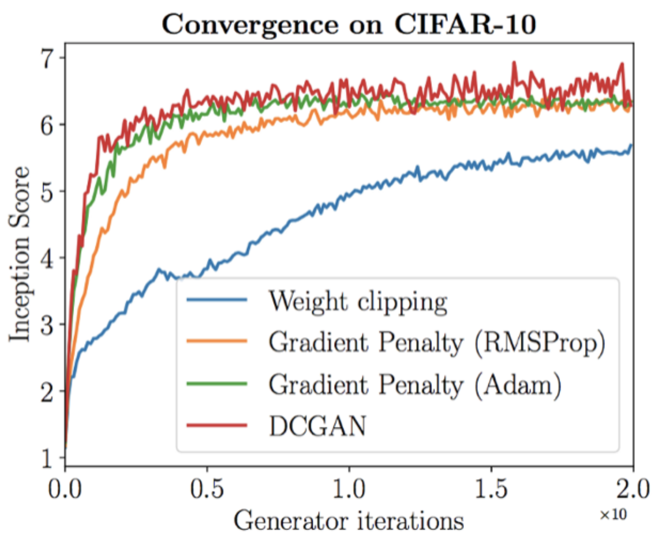

FIGURE 2.5: Gulrajani et al., 2017 present results on GAN convergence

On figure 2.5 we can see a comparison of the convergence of the original WGAN with weight clipping and WGAN-GP for a popular dataset CIFAR-10. Obviously, the result is much better. Furthermore, it is comparable with DCGAN, but more stable.

Chapter 3

Methods

3.1 Data Exploration

The data for this project comes from capsule endoscopy performed on 50 patients. We have footage of the small intestine in healthy subjects obtained using PillCam SB2. The images are divided into 7 classes depending on the particularities captured by the camera - bubbles, clear blob, dilated, turbid, undefined, wall and wrinkles. Figure 3.1 shows examples. Classification and labeling is performed by an expert, so we consider this the ground truth. The images are available in 256x256 resolution. Throughout the project we experimented with different sizes, starting with smaller images as it is easier and then trying to reproduce the results for bigger ones. We had 20000 total images per class, which in principle is more than enough to train a GAN. We used 1000-3000 per class in practice. More detailed information on the dataset can be found in the paper by Seguí et al., 2016, where they try to improve anomalies recognition on it by feature learning.

The nature of this dataset makes it rather challenging to work with. As one can observe on figure 3.1 some classes are quite similar and can be confused for one another. We aim to see if GANs can recreate the variety, but also learn the differences between the classes.

3.2 Implementations

All the implementations for this project are done in Keras. Along with TensorFlow and PyTorch, it is one of the widely used frameworks for deep learning development. We have chosen Keras for its powerful capabilities, but also for its simplicity and ease of use as it was developed as a high-level API with a focus on enabling fast experimentation.

Due to the challenge that the intestine dataset presents, we began work on this project by experimenting with an easier set first - the fashion MNIST. This is a set, included in Keras, that consists of images of clothing and accessories, divided into 10 classes. It provides an easy way to learn how to use and tweak GANs.

We started by implementing a version of the original GAN. The code can be found in *gan_fashion_mnist.ipynb* and is based on work by Rowel Atienza, as linked from the notebook. In general, all implementations follow this structure:

- We define a class to hold all the GAN methods.

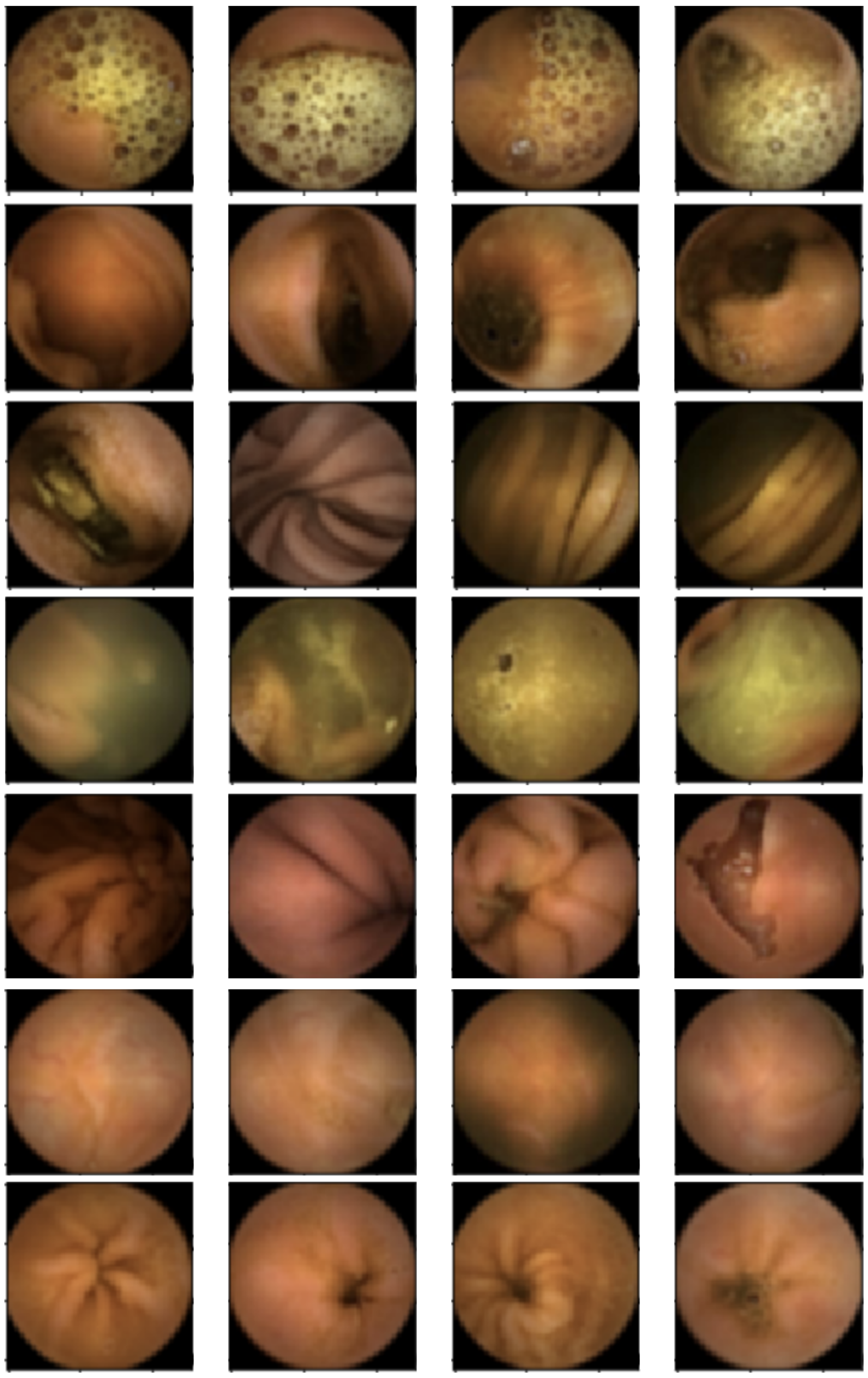

FIGURE 3.1: The set of real images. Each row contains four samples of one class, from top to bottom - bubbles, clear blob, dilated, turbid, undefined, wall and wrinkles

- We define the architecture of the generator and the discriminator and then combine them into the adverserial network. We compile the model with the optimizer and loss function we have chosen. In the beginning, those were respectively Adam and a simple binary crossentropy.
- We train as follows: We randomly select a batch of real images and generate a batch of fake images, then training the discriminator on all those to let it learn to distinguish. After this, we train the generator by inputting noise in the adversarial network and telling the discrminator the images produced by it are real. We repeat the process for the desired number of iterations.
- We usually include a method to plot the produced synthetic images.

For this simple dataset this showed satisfactory results, but we wanted to implement WGAN too. This improvement can be found in *wgan_fashion_mnist.ipynb*, following examples from the same project. You can notice the new Wasserstein loss, as well as the weight clipping as described in Chapter 2. Another thing that changes in the training process is we now train the discriminator 5 times in every iteration since in this case we want it to be as good as possible.

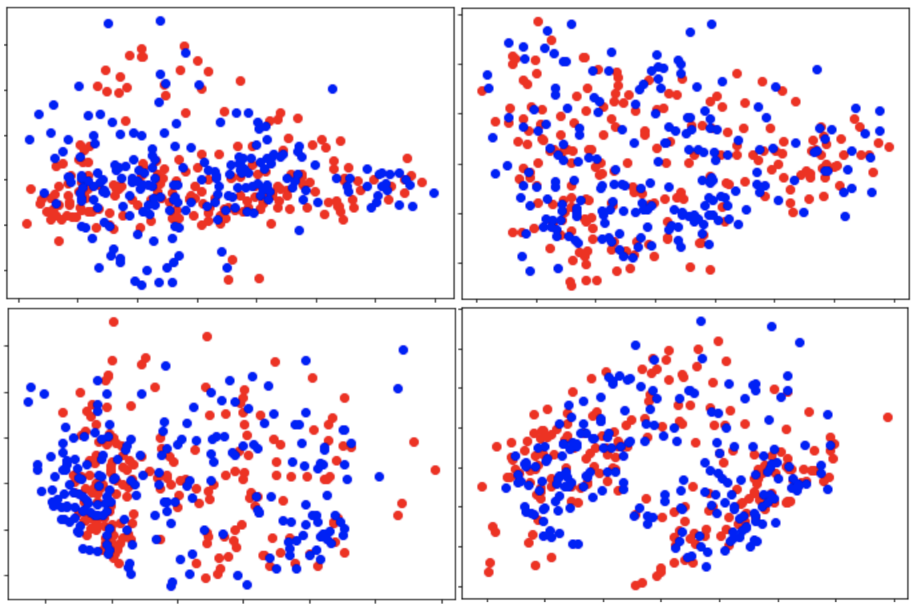

FIGURE 3.2: Principal component analysis on the images from 4 different classes. Red points represent real samples and blue points represent the generated ones. It seems the distribution is preserved within each class, which is the expected result. For instance, notice on the lower right graph there seems to be two big clusters in both red and blue.

Next, we needed to write the code for the cGAN to impose the label as a condition. The implementation is in *cgan_fashion_mnist.ipynb*. The resulting images looked quite good, but we did some detailed experimentation to make sure our GANs are working well before moving on to the real dataset. For example, we performed PCA on the data points to reduce them to 2D and plot them, expecting that real and fake samples from the same class should be close together. The plot of all classes gets a bit cluttered and one cannot assess it very well, so we decided to do separate plots for each class. We present some of them here on figure 3.2. It seems that the distribution of the points is preserved in the fake samples.

Furthermore, we tried to define a network that classifies the images into the 10 classes and check if training it on the extended set (with the synthetic images added) would improve accuracy. The final validation accuracy for real images only was 0.91, for fake images only - 0.82 and for combined - 0.91. It doesn't show an improvement yet, but it is still a good result. We decided not to lose more time on this set as a better score might depend on its specifities, whereas it is not our focus.

At this point we moved to the intestine dataset to try and reproduce the results. As the set is more challenging, this introduced certain difficulties. In the next sections, we focus on those and how we had to modify the GANs so they produce satisfying results.

3.3 Network Architecture

One of the challenges of GANs is choosing the right architecture for both generator and discriminator. The toyset we used has black and white images of size 28x28 pixels. Thus, a shallow convolutional neural network as in figure 3.3 was sufficient.

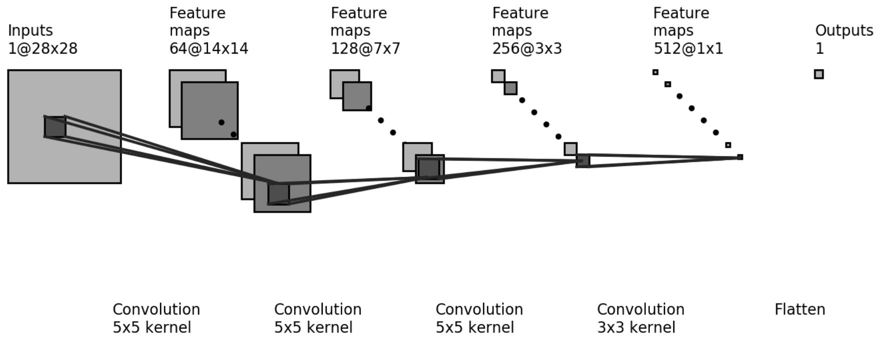

FIGURE 3.3: Architecture of the discriminator. The generator is symmetrical, but we start with the noise, reshaping it into an image and then upsampling instead of downsampling. The diagram is generated by adapting the code from https://github.com/gwding/draw_convnet

For the intestine dataset, where images are bigger and have 3 color channels, this network wouldn't learn the particularities. Results were blurry and unclear. We decided we need to implement DCGAN as defined in Chapter 2. However, as explained earlier, deeper networks sometimes enable additional problems like vanishing gradient. Thus, we decided to add skip connections, resulting in an architecture similar to the ResNet, visualized on figure 3.4.

In yellow you can see the input - a 64x64 image with 3 channels. We reduced the size of the original 256x256 images to 64x64 as smaller images are generally easier and we wanted to take a methodical approach starting with these. Each blue rectangle is a convolution with a kernel of size 5x5. We do a few of those one after the other and we call this a block. Before each block, we reduce the dimensionality by setting a stride of 2. At the end of each block we add a skip connection. Thus, the pink rectangles combine the results from the last convolution and the one in the beginning of the block. As in ResNet, we use an addition operator to do this (although

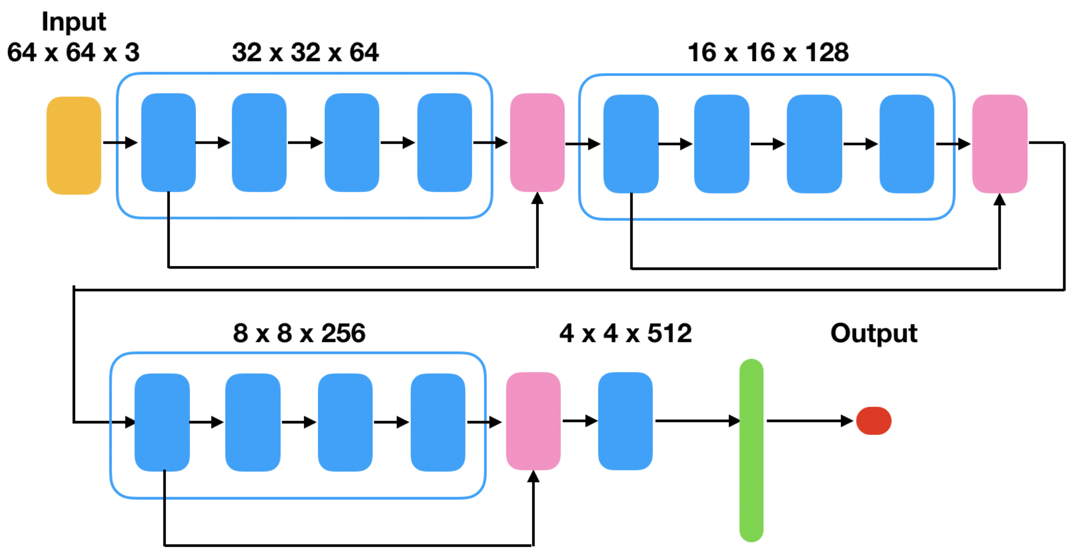

FIGURE 3.4: Diagram of the final architecture.

we experimented with other possibilities as well - multiplication, concatination etc). Finally, we do one last convolution to bring the tensor to a size of 4x4x512, which is reasonable to flatten (this is the green rectangle). And then we add a fully connected layer from it to the desired output (the red dot).

Once we moved on to bigger images, it was really easy to just add new blocks of convolutions - one more for a 128x128 input, and two more for a 256x256 input. This architecture has around 15 million parameters, which nowadays is perfectly normal. Similarly to before, the architecture for the generator is completely symmetrical.

We implemented this architecture within a WGAN. This meant changing the loss function, but also the activation function in the critic as explained in Chapter 2. Furthermore, instead of Adam, we used a RMSprop optimizer as it is empirically shown it works better with WGANs. The code is in *wgan.ipynb*. Results were, however, unsatisfactory as evident on figure 3.5. The next section regards our efforts to fix the GAN and the uncertainties we faced.

3.4 Improving Image Quality

To improve the image quality, we experimented with numerous tweaks briefly outlined here:

- The design of generator and discriminator:
  Before stopping at the architecture as described above we continuously tried to tweak the number of layers, the kernel size, the activation functions, the number of channels per convolution, using pooling as opposed to strides, changing the dropout or batch normalization etc

- The parameters of the GAN:
  This means the optimizers, the loss functions, the particular implementation (especially for Wasserstein, which seemed quite puzzling at first and took some time to get it right)

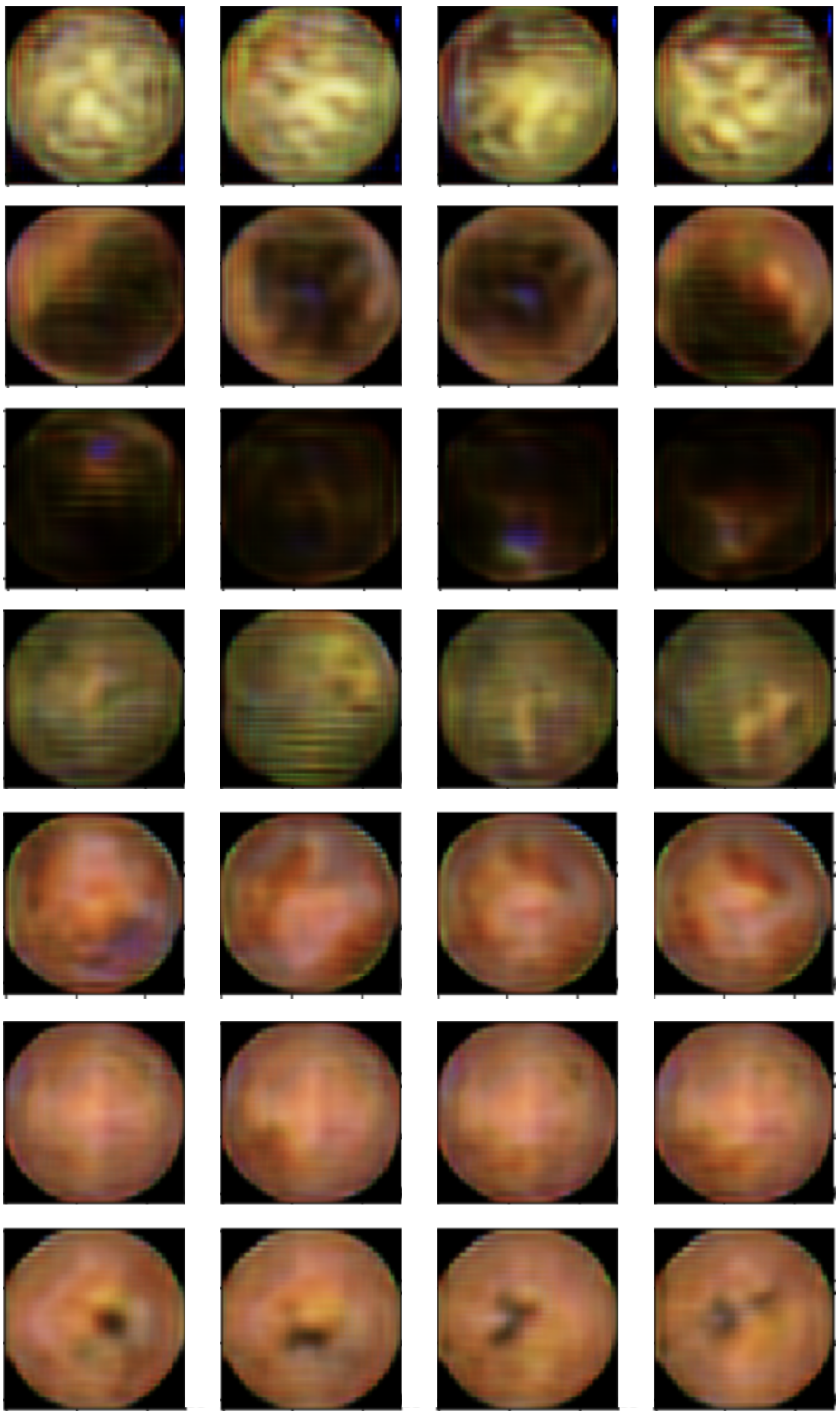

FIGURE 3.5: The first results. One can see the network is learning some of the characteristics of the different classes, but they are too blurry or have a checkboard pattern and other unwanted artefacts.

- The training procedure:
  There were a few considerations we had here. For one thing, people often train on a random batch each iteration. We experimented with the alternative of reshuffling the training data on each epoch and going through all of it to make sure we make full use of all the available data. This turned out irrelevant in the end.
  Then we had some uncertainties coming from the conditional nature of our GAN. For instance, when we train the critic on real and fake data, we were wondering whether the fake images should be assigned the same labels as the real ones. Also we couldn't figure out if we need to train the network on real images with wrong labels to make it understand the classes better. Currently we only train on real images with the corresponding labels and fake images with any labels. The problem is that the critic, in essence, needs to learn two things - distinguishing real from fake, but also distinguishing classes. More on this in the next section.
  Another important point is to freeze the generator when training the critic, and vice versa.

- Inputting the labels:
  Another source of confusion was the particular way that we input the labels into the conditional versions of the generator and discrminator. The original paper on cGAN doesn't specify the correct way and we found out alternatives exist. The problem comes from the fact that the two inputs have a very different format. This is easily solved for the generator, where we can simply concatenate the label to the noise vector. Even so, we would either first learn an embedding or use the one hot encoded label. Both options provide a good result. For the critic it is harder, because we have a fully convolutional network that takes an image as input, so we cannot simply concatenate another vector. The widely adopted approach seems to be to embed the label into a space as big as the flattened image and then multiply these two quantities. After this, we reshape back into an image shape and start convoluting. This is what we ended up using in the end, although we had doubts that maybe "stamping" the label over the image like this is causing the bad quality of the resulting synthetics. The main alternative we experimented with is performing all the convolutions and downsamplings until the tensor is small enough to be flattened. It's only then that we would input the label via concatenation and add a few fully connected layers to ensure the network learns it before the final output. However, the network never learnt the label successfully. Another thing we tried is to embed the label into 64x64 (for a 64x64 image) and then add it as a new channel in the input. Thus, instead of a 64x64x3 image, we would input a 64x64x4 image. This also proved unsuccessful. Finally, the most interesting suggestion is proposed by Tang et al., 2018 who do a one hot channel encoding. That is, each label gets its own 64x64 channel full of zeros. For the label that corresponds to the image, we fill in the channel with ones. Although, it works in their algorithm for artificially aging faces with GANs, we couldn't make use of their approach in the end.

- How long to train:
  Of course, we also experimented with the number of epochs. We gave the networks enough time to train - sometimes tens of thousands of epochs for hours or even days for the final trials. To check if it makes sense to train more, we kept track of the loss for both discrminator and generator, and visualized

it using TensorBoard. You can see on figure 3.6 it converges close to 0 for both networks. This should mean the network has learnt well. Even so, the results were still questionable.

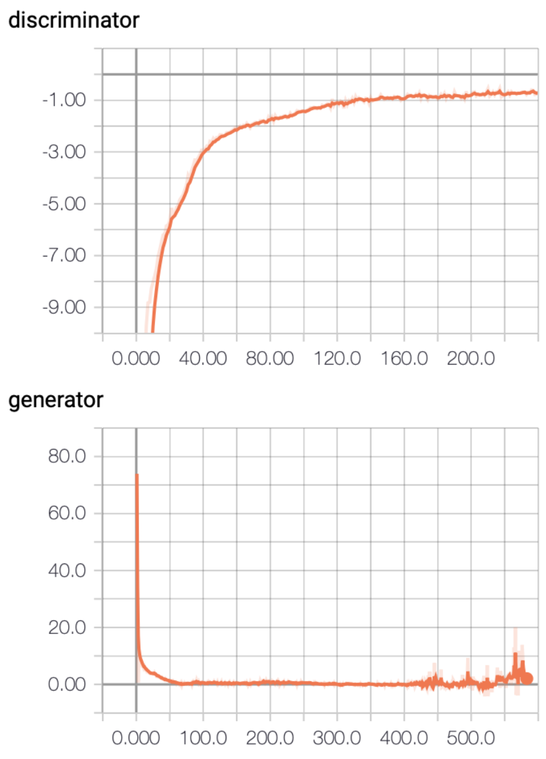

FIGURE 3.6: The Wasserstein loss for the critic and the generator. It is low, close to 0, so we assume the network is learning well. We notice some unstability in the end.

The situation finally improved when we changed to WGAN-GP. This meant replacing the weight clipping with the gradient penalty. Also we made sure to remove batch normalization in the critic as is recommended. As explained in Chapter 2 this network performs better than the original WGAN. However, it introduced a new problem initially - the GAN would produce good images, but wouldn't recognize the classes. We inspect that in the next section.

3.5 Classification Loss

At this point, we also trained a 1-class GAN and it was working perfect. It's undoubtedly easier to learn generating one class, than a few of them. As mentioned previously, the problem is that in a multi-class environment the critic needs to learn two things - distinguishing real from fake images, but also distinguishing classes. We tried tweaking the training procedure to accommodate for this. Finally, we decided to change the loss function by adding a penalty for misclassification, inspired by the age module in the paper about artificially aging faces by Tang et al., 2018. Actually, this technique has been applied numerous times and it proves successful. The new loss function is as follows:

$$L = \mathbb{E}_{\tilde{x}\sim p}[f(\tilde{x})] - \mathbb{E}_{x\sim q}[f(x)] + \lambda\mathbb{E}_{\hat{x}\sim P_{\hat{x}}}[(||\nabla_{\hat{x}} f(\hat{x})||_2 - 1)^2] - \sum_{c=1}^{7} y_{\tilde{x},c} log(p_{\tilde{x},c})$$

This is exactly the same as before, except for the last additive, which is simply the categorical cross entropy loss for classification problems. To calculate it, we build a new neural network that will classify images into one of our 7 possible classes. For the purpose we can use the same architecture or even a much simpler one as this is an easier task. We just need to change to a softmax activation function in the output layer, so that we get the probability of each class. Training this network on the same data that we use for the GAN for about 50 epochs, we get an accuracy of 0.95, which is very good. Then when we evaluate the loss metrics of the GAN, for each generated image we further run it through the classifier and it gives us what we consider to be the true label. If it doesn't match the one we fed to the generator, then we penalize the model.

With this addition, the WGAN-GP was working very well. Code can be found in *wgan-gp.ipynb*. One can observe the additional loss function added to the generator model. The resulting final synthetic images are shown on figure 3.7.

3.6 Assessment

To a human eye, the produced synthetic images already looked good, albeit not perfect. Sometimes they would be a bit blurry, specifically for the 64x64 case. This is why we added the extra blocks to run the algorithm on 128x128 and 256x256, which expectedly improved resolution. We ran a few tests to make sure the network is working correctly. The final model that we assessed was run for two days on a GPU.

For one thing, we manually gave the generator random noise and a label to see if it produces an image from the class we expect. Then we analysed the output of the critic for this synthetic image and all possible labels. As should be, the score for the correct label is the highest. It's interesting to see for which classes the scores are similar as this means the network is uncertain about those and might confuse them. For example, undefined and wrinkles are often similar.

Apart from that, we wanted to find an actual measure for the GAN's performance. Inception Score (IS) is often used for this purpose. Proposed for the first time by Salimans et al., 2016, it tries to judge the dataset of produced synthetic images on two criteria: if they look like something sensible and if they have variety. This is done by running the images through a classifier, combining the label probability distribution over the whole set and calculating the divergence from the real distribution, using KL-divergence as defined in equation (2.3). Usually GANs are scored on the Inception classifier, hence the name of the measure, but it is very different from our problem (produced by Google, it classifies images into 1000 different classes). Thus, we cannot use the measure directly. We tried tuning it to use our own classifier that we came up with in the previous section, but the score didn't make sense compared to other ISs we found in literature, so we are not sure if it is directly comparable. IS is considered imperfect due to its nature of linking the score to the classifier. Frechet Inception Distance (FID), proposed by Heusel et al., 2017 offers an improvement, but still uses the Inception network.

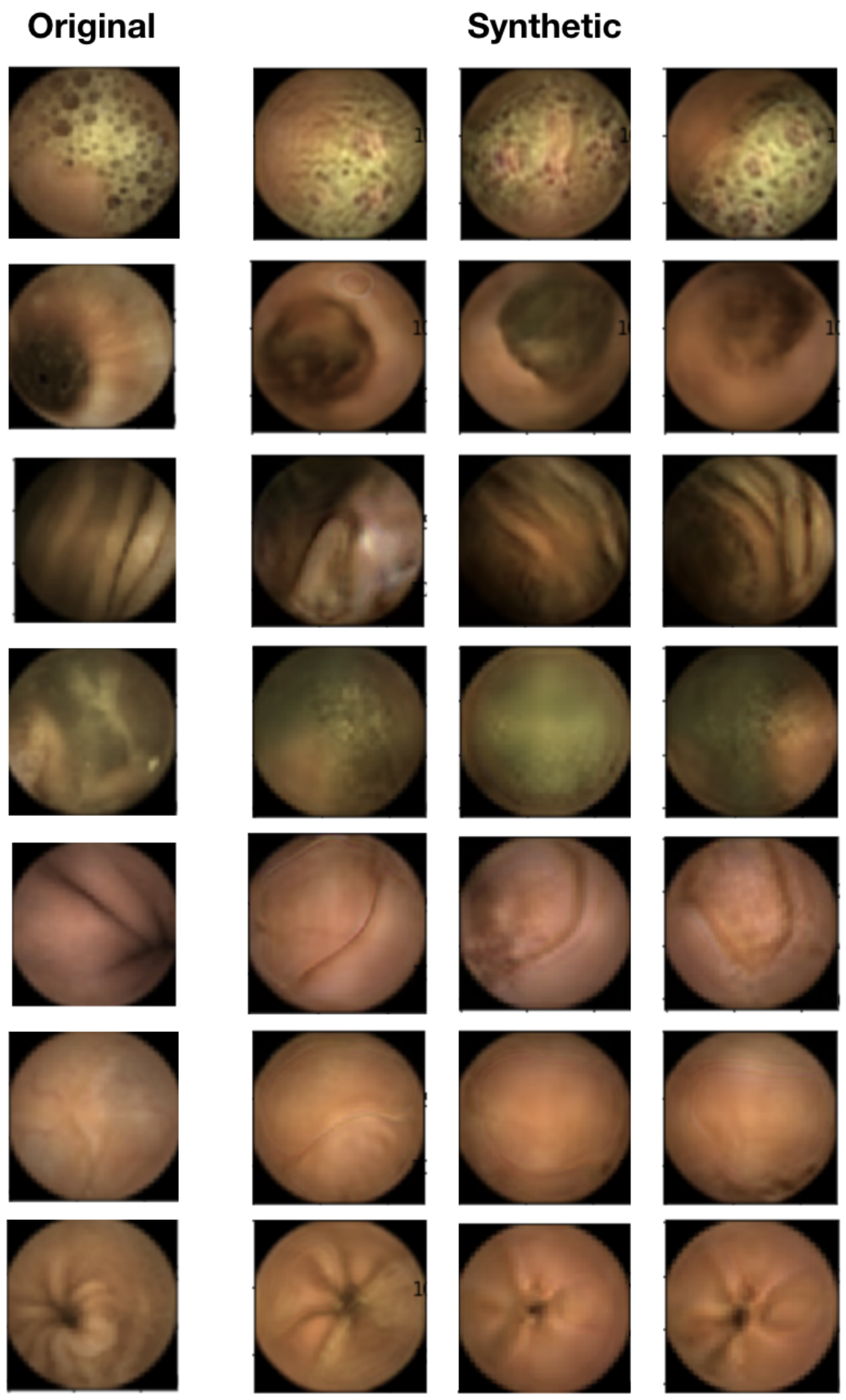

FIGURE 3.7: The final results. The leftmost image serves to remind of the real class. The rest of the images are synthetically generated.

Either way, we decided to do something that is essentially very close to the idea of IS. Going back to the beginning, the idea of this project was to create synthetic data that would aid classification algorithms with insufficient training data. Thus, we wanted to check if the new images would improve the classification accuracy. That's why we performed the following test:

- Divide the real images into a train and test set.
- Train a first classifier on the train set. Evaluate on the test set and on the synthetic data.
- Train a second classifier on the synthetic images. Evaluate on the test set and the train set.
- Train a third classifier on both the train set and the synthetic images. Evaluate on the test set.

We have used the same architecture as in the previous section, but we trained for 25 epochs. Results are provided in table 3.1.

accuracy	synthetic data	train set	test set
1st classifier	**0.91**	*0.89*	**0.75**
2nd classifier	*0.99*	**0.66**	**0.63**
3rd classifier	*0.98*	*0.98*	**0.78**

TABLE 3.1: Evaluation results, where the classifiers are as described in the text. The values in italic are in training time. The values in bold are reported on unseen data.

We can observe some overfitting of the classifiers as the test accuracy is lower than the train accuracy, although we have designed the network with dropout specifically to avoid this. Either way, this will not be of concern for the moment. We can also see that when training on the real data, the classifier is performing perfectly on the synthetic one. We could hope for an improvement the other way around, but the lower scores are probably because of some artefacts that sometimes appear on synthetic images. Finally, in the last row we can see that when we trained on all the available data, the accuracy in the test set increased slightly. Therefore, we can conclude the GAN is producing good images and the synthetic data can indeed be used for the goal we pursued.

The code for these tests, including the attempts to score with IS, can be found in *eval.ipynb*

Chapter 4

Conclusion

In the last chapter we would like to summarize what we have achieved and discuss what we believe can be done next.

4.1 Future Work

As stated in the beginning, the main purpose of generating the synthetic images is to use them in training algorithms for automatically recognizing anomalies when performing capsule endoscopy on a patient. Those could be bleedings, ulcers, polyps. Recognizing polyps is particularly challenging, even for humans, because of the different shapes they could have. The initial goal of the project was to yield an algorithm that will produce real-looking images with polyps on them, but due to the difficulty of training the GANs we only developed the network for the easier set of the intestine images. Indeed, in medical applications the positive class usually suffers from scarcity, so ideally we would like to turn our focus to generating the polyps. This section serves to offer insights on how we can extend the project to solve this problem.

In particular, we are interested in reproducing or improving the results of Shin, Qadir, and Balasingham, 2018. They try to take real images and put realistic looking polyps on top of them. To do so, instead of inputting random noise in the generator, they input an edge filtering of the original and a random binary polyp mask. Then the network should learn to generate a synthetic image, keeping the edges, but adding a polyp on top. This is done to make sure that polyp and environment are harmonious in the synthetic images, while maintaining the overall content of the colonoscopy images. This is an interesting case of a cGAN, where instead on a label we condition on the edges and mask. The concept is shown on figure 4.1. We consider this a possible and logical continuation of the work on this project.

4.2 Summary

The invention of wireless capsule endoscopy has had a tremendous impact on medicine, revolutionizing the management of dozens of conditions and diseases. The biggest challenge to its adoption, however, has been the limits of the human eye and human attention. Artificial intelligence and deep learning algorithms that can learn from raw and unprocessed imagery are considered the solution. Michael F. Byrne and Fergal Donnellan, 2019 claim that a human can discern 3 features for polyp

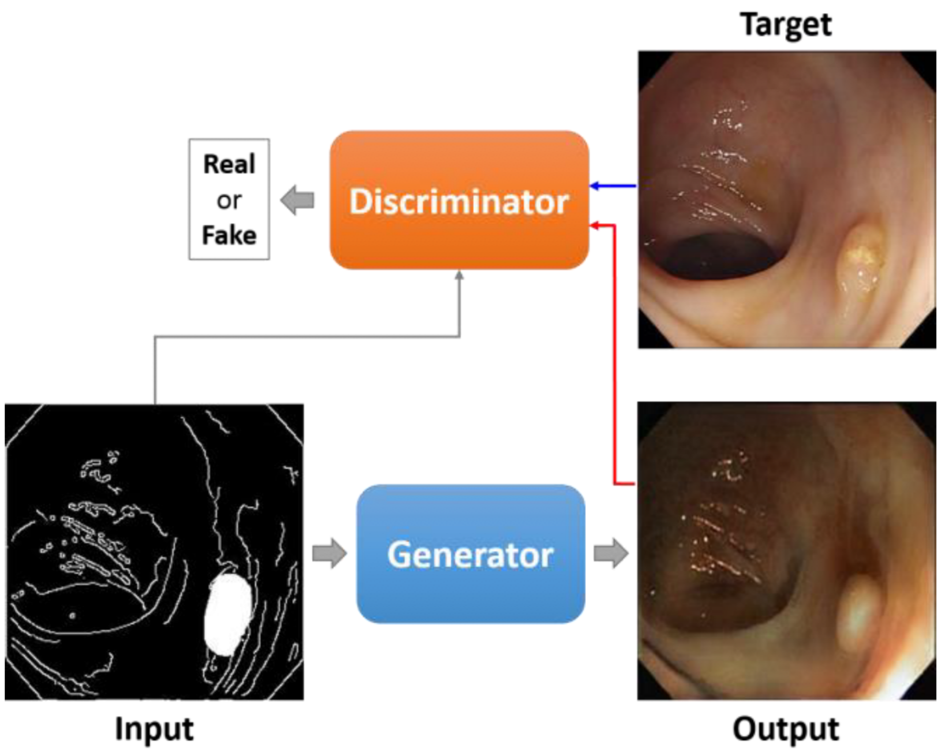

FIGURE 4.1: The polyp generation GAN proposed by Shin, Qadir, and Balasingham, 2018. Note that a different capsule camera has been used, therefore the images look different to the ones in this project.

differentiation, whereas their algorithm for polyp detection was seeing over 1000 discriminating features.

As is usually the case, the quantity and quality of available data is key to a well performing algorithm. Different strategies have been utilized to tackle insufficient training data. We are concerned with artificially enlarging the training set by adding new synthetically generated images. In this project we have used a GAN to achieve this.

GAN employs a minimax game between two adversarial agents - discrminator and generator. While the discriminator tries to distringuish real from fake images, the generator gets better at producing the latter. We have implemented the original GAN, as well as necessary improvements like the conditional variant where we impose a condition on the generator and the Wasserstein GAN, including gradient penalty, where we use a new loss function to stabilize training.

With this we have achieved our goal of improving classification accuracy for a dataset of various intestine images, ranging from showing clear blobs to bubbles to wrinkles etc. In particular, the accuracy on a test set has increased from 0.75 to 0.78 when training on both the real and synthetic data. Results show promise to be applied to the harder task of recognizing polyps, the next step for the development of this project.

While we have tried to make the GAN as good as possible by continuously testing different configurations for its parameters and hyperparameters, we recognize its imperfections. GANs are a subject of numerous ongoing studies - researches try to apply new loss functions, change the architectures and introduce other novelties.

We are certain that more experiments could be run to further improve the quality of the output.

Bibliography

Arjovsky, Martin, Soumith Chintala, and Léon Bottou (2017). “Wasserstein GAN”. In: URL: https://arxiv.org/abs/1701.07875.

Caffrey, Colm Mc et al. (2008). “Swallowable-Capsule Technology”. In: *IEEE Pervasive Computing* 7.1, pp. 23–29. URL: https://ieeexplore.ieee.org/document/4431853/authors#authors.

Chen, Xi et al. (2016). “InfoGAN: Interpretable Representation Learning by Information Maximizing Generative Adversarial Nets”. In: URL: https://arxiv.org/abs/1606.03657.

Deep learning for hackers with MXnet (2): Neural art. URL: https://no2147483647.wordpress.com/2015/12/21/deep-learning-for-hackers-with-mxnet-2/ (visited on 05/03/2019).

Dreyfus, Stuart (1962). “The numerical solution of variational problems”. In: *Journal of Mathematical Analysis and Applications* 5.1, pp. 30–45. URL: https://www.sciencedirect.com/science/article/pii/0022247X62900045?via%3Dihub.

Goodfellow, Ian J. et al. (2014). “Generative Adversarial Networks”. In: URL: https://arxiv.org/abs/1406.2661.

Gulrajani, Ishaan et al. (2017). “Improved Training of Wasserstein GANs”. In: URL: https://arxiv.org/abs/1704.00028.

Hannun, Awni et al. (2014). “Deep Speech: Scaling up end-to-end speech recognition”. In: URL: https://arxiv.org/abs/1412.5567.

Hart, Matthew (2017). “Nvidia’s A.I. Generates Perfect Headshots of Fake Celebrities”. In: *Nerdist.com*. URL: https://nerdist.com/article/nvidia-ai-headshots-fake-celebrities/ (visited on 04/29/2019).

He, Kaiming et al. (2016). “Deep Residual Learning for Image Recognition”. In: *2016 IEEE Conference on Computer Vision and Pattern Recognition*. URL: https://ieeexplore.ieee.org/document/7780459.

Heusel, Martin et al. (2017). “GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium”. In: URL: https://arxiv.org/abs/1706.08500.

Hochreiter, Sepp and Jürgen Schmidhuber (1997). “Long Short-term Memory”. In: *Neural Computation* 9.8, pp. 1735–1780. URL: https://arxiv.org/abs/1404.7828.

Huang, Xun et al. (2016). “Stacked Generative Adversarial Networks”. In: URL: https://arxiv.org/abs/1612.04357.

Hui, Jonathan (2018). “GAN - Why it is so hard to train Generative Adversarial Networks!” In: URL: https://medium.com/@jonathan_hui/gan-why-it-is-so-hard-to-train-generative-advisory-networks-819a86b3750b.

Jones, Kenny (2017). “GANGogh: Creating Art with GANs”. In: *towardsdatascience.com*. URL: https://towardsdatascience.com/gangogh-creating-art-with-gans-8d087d8f74a1 (visited on 04/30/2019).

Karras, Tero, Samuli Laine, and Timo Aila (2018). “A Style-Based Generator Architecture for Generative Adversarial Networks”. In: URL: https://arxiv.org/abs/1812.04948.

Karras, Tero et al. (2017). "Progressive Growing of GANs for Improved Quality, Stability, and Variation". In: URL: https://arxiv.org/abs/1710.10196.

Kingma, Diederik P and Max Welling (2014). "Auto-Encoding Variational Bayes". In: URL: https://arxiv.org/abs/1312.6114.

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton (2012). "ImageNet classification with deep convolutional neural networks". In: *NIPS'12 Proceedings of the 25th International Conference on Neural Information Processing Systems* 1, pp. 1097–1105. URL: https://dl.acm.org/citation.cfm?id=2999257.

Lecun, Y. et al. (1998). "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11, pp. 2278–2324. URL: https://ieeexplore.ieee.org/document/726791.

Lee, Andy (2016). "Comparing Deep Neural Networks and Traditional Vision Algorithms in Mobile Robotics". In: URL: https://www.cs.swarthmore.edu/~meeden/cs81/f15/papers/Andy.pdf.

Linnainmaa, Seppo (1976). "Taylor expansion of the accumulated rounding error". In: *BIT Numerical Mathematics* 16.2, pp. 146–160. URL: https://link.springer.com/article/10.1007%2FBF01931367.

Michael F. Byrne, MD and MD Fergal Donnellan (2019). "Artificial intelligence and capsule endoscopy: Is the truly"smart"capsule nearly here?" In: *Gastrointestinal endoscopy* 89.1, pp. 195–197. URL: https://www.giejournal.org/article/S0016-5107(18)32937-7/fulltext.

Mirza, Mehdi and Simon Osindero (2014). "Conditional Generative Adversarial Nets". In: URL: https://arxiv.org/abs/1411.1784.

Radford, Alec, Luke Metz, and Soumith Chintala (2015). "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks". In: URL: https://arxiv.org/abs/1511.06434.

Salimans, Tim et al. (2016). "Improved techniques for training GANs". In: URL: https://arxiv.org/pdf/1606.03498.pdf.

Schmidhuber, Juergen (2015). "Deep Learning in Neural Networks: An Overview". In: *Neural Networks* 61, pp. 85–117. URL: https://arxiv.org/abs/1404.7828.

Seguí, Santi et al. (2016). "Generic Feature Learning for Wireless Capsule Endoscopy Analysis". In: URL: https://arxiv.org/pdf/1607.07604.

Shin, Younghak, Hemin Ali Qadir, and Ilangko Balasingham (2018). "Abnormal Colon Polyp Image Synthesis Using Conditional Adversarial Networks for Improved Detection Performance". In: URL: https://ieeexplore.ieee.org/abstract/document/8478237.

Silva, Thalles. *An intuitive introduction to Generative Adversarial Networks (GANs)*. URL: https://medium.freecodecamp.org/an-intuitive-introduction-to-generative-adversarial-networks-gans-7a2264a81394 (visited on 05/17/2019).

Springenberg, Jost Tobias et al. (2014). "Striving for Simplicity: The All Convolutional Net". In: URL: https://arxiv.org/abs/1412.6806.

Tang, Xu et al. (2018). "Face Aging with Identity-Preserved Conditional Generative Adversarial Networks". In: URL: https://www.researchgate.net/publication/329743796_Face_Aging_with_Identity-Preserved_Conditional_Generative_Adversarial_Networks.

Tong, Jessica et al. (2012). "Diagnostic yield of capsule endoscopy in the setting of iron deficiency anemia without evidence of gastrointestinal bleeding". In: *Canadian Journal of Gastroenterology* 26.10, pp. 687–690. URL: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3472906/.

Yu, Jiahui et al. (2018). "Generative Image Inpainting with Contextual Attention". In: URL: https://arxiv.org/abs/1801.07892.

Zhang, Han et al. (2017). "StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks". In: URL: https://arxiv.org/abs/1612.03242.

Zhu, Jun-Yan et al. (2017). "Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks". In: URL: https://arxiv.org/abs/1703.10593.

Zhuan Liao, MD et al. (2010). "Indications and detection, completion, and retention rates of small-bowel capsule endoscopy: a systematic review". In: *Gastrointestinal Endoscopy* 71.2, pp. 280–286. URL: https://www.giejournal.org/article/S0016-5107(09)02540-1/references.