POLITECNICO DI TORINO

Master degree course in Data Science and Engineering

Master Degree Thesis

Multimodal-source image generation with deep learning

Supervisors
Prof. Paolo Garza
Dr. Ruben Cartuyvels (KU Leuven)
Dr. Erfan Ghaderey (KU Leuven)

Candidate
Fabrizio Lande
matricola s276334

Academic year 2020-2021

Summary

The focus of this work is to present a new way of synthesizing images starting from a descriptive input text and a reference image by using a model extensively based on deep learning and generative adversarial networks.

Acknowledgements

This project would not have been possible without the support of my thesis supervisors Dr. Ruben Cartuyvels and Dr. Erfan Ghaderey, two PhD students at KU Leuven University that guided me through the development process until the end of my staying abroad as part of my Erasmus EU project. Many thanks to my Politecnico supervisor, Paolo Garza, who read my numerous revisions and helped produce a mature text.
Thanks to Politecnico di Torino for giving me the opportunity of spending this last year abroad in Belgium, partially providing me with the financial means to complete this project. Also thanks to KU Leuven University, that regardless the harsh times we went through, has offered me support and the technical tools to actually see my code running by means of their Flemish super computer (VSC). And finally, thanks to my brother and parents, my girlfriend and numerous friends who endured this long process with me, always offering support and love.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Computer automated image generation is a fascinating topic though relatively new. Only in recent years the scientific community has managed to see its ideas implemented in actual algorithms and models capable of returning good quality images with realistic subjects depicted inside. Even though this field is vast and only partially discovered, a few deep learning techniques have already proven to be good candidates for facing such task: generative adversarial networks are now days responsible for some of the most realistic pictures ever created by a computer, finding applications in a variety of scenarios. It has been attempted to upscale preexisting images for textures quality improvement in games [40] but also astronomical images [41], generate novel art [36], but also malicious applications like Deepfake [33], a deep learning model capable of producing fake and possibly incriminating photographs and videos.
In this work however, I will present how I generated realistic images of two hundred species of birds using as input some plain English text, describing the features of the subject, and some reference images containing small portions of real bird parts, that the model has to use in conjunction with the textual description to generate that particular parts in terms of shapes, textures and colors. Everything will be based on the Attention GAN model, which I have used as backbone and that ultimately I attempted to upgrade with the introduction of the additional input defined by the reference image. For this reason this work and the model that I ended up with, RaGAN, has to be considered a multi-source image generator that uses deep learning. This idea can have interesting applications in the field of arts and artists support by means of helping them with initial sketches to build upon in order to create novel animals and creatures that satisfy a written description and now an

additional reference image in order to include a specific element in the final image. The result can then be taken on its own or used as inspiration for drawing something more refined. Another field of application can be identikit enhancement. Given a written deposition of the suspect and a picture of a specific characteristic of their face, like their eyes or something that a camera has captured, it could be possible to generate the sketch with more realism because it would also contain the portion of his or her face depicted in the reference image. These claims can be considered quite advanced for what the model is actually able to achieve at this stage, but I do believe that the direction this work is aiming at will eventually turn into a mature model capable of delivering either or both applications.
In the following chapters I will first go through some related works, papers and architectures developed in past years, that I studied during my preliminary approach with the whole topic. I tired to follow a chronological order that carries the reader from the invention of GAN architectures all the way to Attention GAN and other similar models and finishing with other techniques that use different inputs for synthesizing images.
In the third chapter I will formalize my model as a natural extension of the Attention GAN architecture, showing issues and adopted implementation solutions I found in my journey.
In the fourth chapter I will dive deeper into what concerns the training phase, the supercomputer I had to use and all the low level details related to this crucial step of the model. Finally in the last chapter I will show my conclusions explaining pros and cons of the proposed implementation and discussing some possible further work, what it could have been done with more resources and time and where to improve in order to go further.
The vast majority of this work has been carried on during my stay abroad at KU Leuven University in Belgium, as part of my Erasmus project. At the beginning, this thesis as a whole was focused on a completely different task based on Lewis et al. work [21]. However, after a couple of months of individual study, my initial supervisor and I figured out that what we were trying to achieve was mathematically impossible. Therefore we moved towards a slightly different iteration of the initial idea, which however once again ended as soon as we reached a theoretical dead end, that made further work impossible. Nevertheless, I carried on and as soon as we discussed and planned the last version of my work, that ultimately became what I present here, I started working and studying for it and I kept on doing so until I completed everything I planned to do, or when there were no more available resources to run any other experiment.

Chapter 2

Related work

2.1 Introduction

Text to image generation and in general image generation is undoubtedly a very interesting task where deep learning can really shine and show true potential. In fact, such a complicated assignment cannot be solved using traditional methods that we might take from computer vision for example or from linguistics, if we narrow the job down to input text only. For once, these other techniques lack in generalization which is key if we really want to face the problem of generating something new, pictures of new animal species for instance.
Throughout my preliminary research I have discovered a certain polarization towards the use of the GAN architecture of solving this kind of assignment but in a later section I will show how also other techniques do exist and have been used. Though, a lot of work has been published regarding the topic of image generation throughout the (few) years since the invention of the GAN architecture. In particular, I will go through all the works that I have come in contact with in the first months of familiarization with the subject presenting them in a chronological order, hoping to carry the reader from the origin of the architecture all the way to more recent applications and implementations.
What I present here is by no means the completed research scenario of all possible published papers and works that the scientific community has generated in later years. While I am reading these lines new studies have been for sure released and many more will be presented in the future. For these reasons what I will show here has to be considered as a good small portion of what has been done in the context of image generation and what I came in

contact with during my initial gathering of information related to this topic before jumping into my own implementation.

2.2 The origin of GAN

Throughout my studies and personal research I have discovered that a big leap forward in the topic of image generation has been given by the generative adversarial network architecture (GAN) since its public release in 2014 [10].
This very interesting approach in generating new, synthetic instances of data that can pass for real data, has since then played a big role in image generation, video generation and voice generation. The base concept is the following: the model is actually composed of two models, a generator G and a discriminator D. They are trained in a competing (adversarial) fashion meaning that each has to prevail in doing its task over the other. The generator has to generate a "credible" output and the discriminator has to discriminate fake outputs from real data. In the context of image generation, the generator takes as input what can be considered as an initial canvas some random noise and tries to modify it with the help of convolutional layers into an image that has to be as close as possible to real data or, to put it more formally, to the target distribution. On the other hand, the discriminator has to be able to decide, given a real image and one generated by the model, which is which. Whenever the discriminator fails at this task (it is fooled by the generator) an associated loss increases (discriminator loss). If the discriminator instead successfully classifies the fake as fake the generator loss will increase, punishing its poor result.

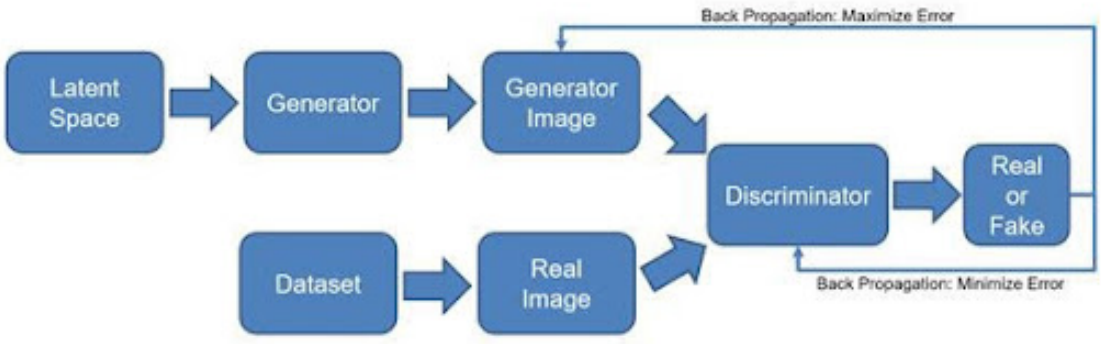

Figure 2.1. Basic GAN architecture. Source: public domain.

Mathematically speaking, following the description presented in the paper, D and G play a two-player min-max game with value function V(D, G):

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

The classic back propagation through chain rule is then used to update the two models. A big advantage of this architecture is that the generator does not directly see real data and the only "contact" it has with them is through back propagation. This means that components from the input are not copied by the generator. However, this advantage is also a big disadvantage of the structure. The two models must be synchronized during training, avoiding in particular that the generator is updated more often than the discriminator or else it would cause the Helvetica scenario or mode collapse: G will put too much probability density in a small area of the data space This in practice can be translated as: if the generator learns that one image (or small group of images) is good enough to fool the discriminator (that has not been updated enough) it will try to use that image (or small group of images) more often if not always, therefore forcing the discriminator to use less features to define if an image is fake or not worsening its ability to "understand" what it has to classify.
Due to this intrinsic back and forth behaviour, the training period is typically long for such models and requires many passes through the training set before the generator is good enough to produce high quality images. Theory [8] suggests that this juggling should end once we reach a state of convergence, defined as the state where the discriminator has a 50% accuracy. I must add to this definition that in the context of image generation the quality of the output images is itself a good way to see if the model is mature or not. I will show in later chapters how I decided to stop training my model, following the number of epochs and the training technique described in the Attention GAN paper but also the visual results.
Taking a more practical approach to this theoretical introduction, I can say that from what I have seen, studied and used, a generator is nothing more than a series of upsample blocks that accept an initial random-based input vector and enlarge it to the desired output dimensions injecting meaning to the new dimensions with the help of convolutional blocks that span the intermediate features and extract new ones. On the other hand, discriminators are quite the opposite because they receive as input these higher dimensional vectors with rich and meaningful features and have to reduce them

in steps in order to obtain an output vector that is then passed to a fully connected classifier. We can indeed see the discriminators like classic CNN networks used for classifications as the ones commonly presented in courses like Alexnet [17]. So the real key feature of GAN based models is the way they get trained, and not so much the building components, that are quite similar to previous architectures seen in other deep learning scenarios.

2.3 Text to image generation, a first example

In subsequent works that started with the GAN architecture, not only the generator and discriminator internal structure has been improved but also certain aspects of the model itself, like the generator input, have been revised. In the work of Reed et al. [30] in 2016 we can see for example the introduction of a textual input concatenated to the canonical random noise. This work is actually a big change in direction from previous attempts that generated images from a textual description but used variational auto-encoders [16, 18, 11]. Not only the resulting quality was lower but the whole architecture was more complicated than the adopted GAN.
The main goal of this work is to draw specific objects, described through input text, in specific positions and/or regions of the image by giving bounding boxes or punctual positions as side inputs[1].

For the sake of simplicity I will briefly present the structure of the model that uses bounding boxes but the one that uses point coordinates is quite similar. As it is visually presented in Figure 2.3, a first feature matrix is generated using the input text that is embedded into feature vectors and then expanded to match the normalized bounding box size. This creates a matrix where only the area covered by the bounding box receives values coming from the embedded text while the rest stays at zero. This masking matrix is brought back to a smaller feature space by passing through convolutional and pooling layers and then it is concatenated with the input vector noise. This new feature vector undergoes two different paths where a series of deconvolution layers upsample the vector enough times to make it reach the wanted final dimensions. The only difference is that the local features matrix is additionally masked with a new matrix that sets to zero all the regions outside the input bounding box. The resulting matrices are

[1]Two models have been developed, one using bounding boxes and the other two dimensional points.

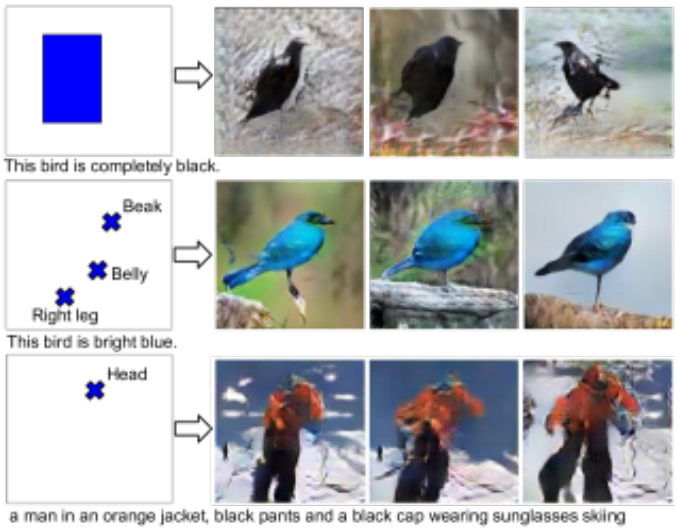

Figure 2.2. GAWWN qualitative input and output

then concatenated and a final deconvolutional block is used to upsample the vector and return the final image. The discriminator operates in the opposite way and from the image it generates local and global feature matrices. The local one is enriched by the input text embedding that is first spatially replicated to match its dimensions and then masked using the same masking matrix presented before. A convolutional block reduces the dimensions of both local and global matrices making them become vectors that are added together and a final fully connected layer produces the binary output.

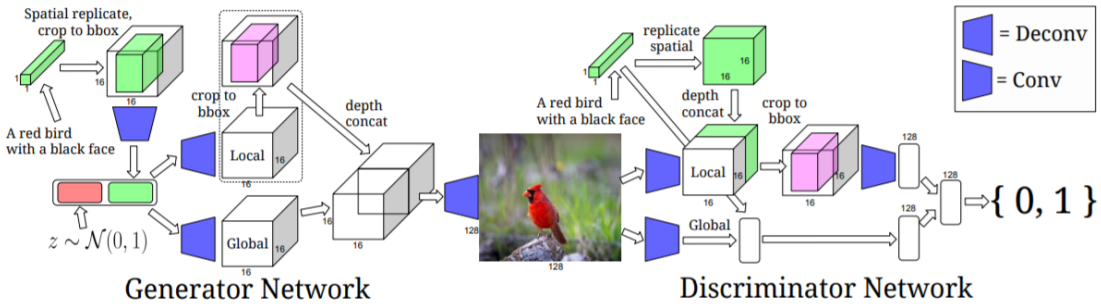

Figure 2.3. GAWWN architecture using bounding boxes. Source: [30]

From this work we can extract a valuable lesson that we will see common in other papers: input text, once embedded into feature vectors is simply

concatenated to the input random noise. This concept is key because I decided to do the same when introducing the additional input reference image features to the model.It also highlights the presence of punctual information contained in the CUB dataset, which is what they used here and what I have used to extract reference images with as I will show in chapter three.

2.4 Two fundamental models

In 2017 two new models based on GAN architecture improve even more the quality of the results, managing to obtain more crisp and detailed pictures starting from text. In this section I will briefly discuss them, namely StackGAN and AttnGAN, highlighting similarities and differences in implementation and philosophy.

Figure 2.4. Qualitative results of StackGAN (left) and AttnGAN (right). Source: [47, 45]

2.4.1 StackGAN

Zhang et al. [47], starting from the idea of stacking two generators and discriminators one after the other, create what's known in literature as StackGAN. The structure can be ideally extended to additional stages but this paper shows a two steps process. The first generator takes the input text in embedded format concatenated with random noise and generates a first,

low quality image. This process is done in almost the exact same way that I will present for AttnGAN in chapter three and consists of a series of upsampling blocks that take the input sentence feature vector and scale it up to match the dimensions of the first low quality output image. The second generator takes the features of output image extracted by a first discriminator and by downsampling, mainly using convolutional and pooling layers, it converts it to a matrix that contains all the semantic information that is then concatenated to another matrix obtained from downsampling the output image directly. This big matrix goes through a residual block that enriches its information content and a last upsampling block for generating the final result. The idea of splitting the process into two steps is important in order to make the generation task easier. The first step generator produces an idea of shapes and colors, with less consideration over ed and polished features. The second generator works over this first attempt, refining shapes and textures and also up-scaling the image to the final dimensions. Figure 2.5 shows this architecture more in detail.

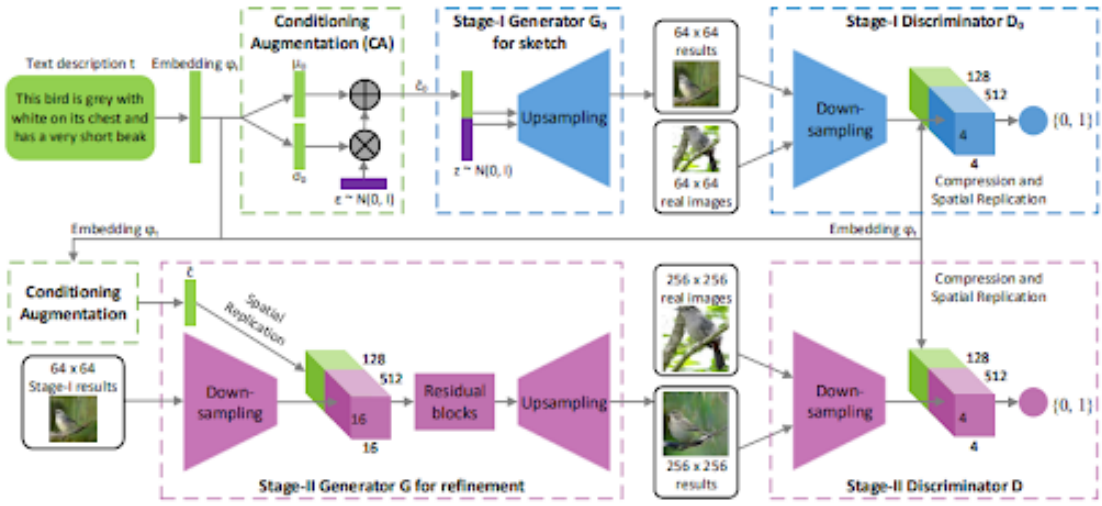

Figure 2.5. StackGAN architecture. Source: [47]

Qualitatively, the image results of this model are very much comparable to Attention GAN, another model that I will present later. However in terms of pure scores, they are a bit lower and for this reason and also because my thesis supervisors had more experience with the aforementioned Attention GAN I ultimately decided not to work on this architecture.

2.4.2 AttnGAN

Xu et al. [45] tackle the problem with another cut. Once again the input is a sentence describing how the subject of the image has to be[2]. The key concept of this work is a novel attention driven fine-grained image generation. The model learns to give the right attention at the right portion of the image for each input word. The concept of attention in the context of deep learning can be summed up as a weight vector that defines how important certain features are. Here, it is a two dimensional matrix that defines how important every pixel is according to each word. This model accepts input sentences of at most eighteen words, therefore it also generates, and we are able to visualize, up to eighteen different attention maps. Not only does this method allow for better quality images, more related to the input text, but in addition it offers an intrinsic way to study how the model works because we can take a look at these maps in order to check if the model "has learned" the meaning of those words or not and where in the picture each word is more used.
Conceptually this model is composed of three components: a multi-stage generator, a set of discriminators, thus resembling a GAN architecture, and a deep attentional multimodal similarity model (from now on DAMSM). The generator actually consists of three generators chained one after the other, each one producing a different quality output image. The paper suggests that potentially even more generators can be concatenated but for stability and use of resources they decided to stick with just three. Each output image is fed to a discriminator, therefore there are three of them. Each discriminator is trained separately and it checks for the quality of the image conditioned and unconditioned to the input sentence. This ensures that the generated output is not only credible but also related to the input sentence. Finally the last model piece, the DAMSM, compares features extracted from the highest quality image with those extracted at word level from the input image and generates an additional loss, that enforces even more semantic connection between the input and the output. This novel block is actually trained before the rest of the model, and it is done so that it learns a common semantic space where word features and image features have to be mapped to be compared.
As I will show in chapter three, my supervisors and I have decided to use this model and paper as a starting point to begin our research towards an ever

[2]The target distribution was either a few hundred races of birds contained in CUB dataset, or images from COCO dataset.

more advanced setup due to the brilliant concept brought by the DAMSM block and general architecture. For this reason I have decided not to go into too much detain in this section about lower level implementation stuff and how every component works, leaving the reader with hopefully a first impression of the model.

2.5 Image manipulation through text

IN 2020, a few years later AttnGAN and StackGAN publications, Li et al. [22] create a release their ManiGAN model. Here, instead of focusing on generating photo-realistic images they decided to concentrate their efforts in creating a model able to manipulate specific visual attributes of given images using natural descriptors. In practice, an input image and text are given. The image has to be modified according to the text. This means change in colors, texture and background.

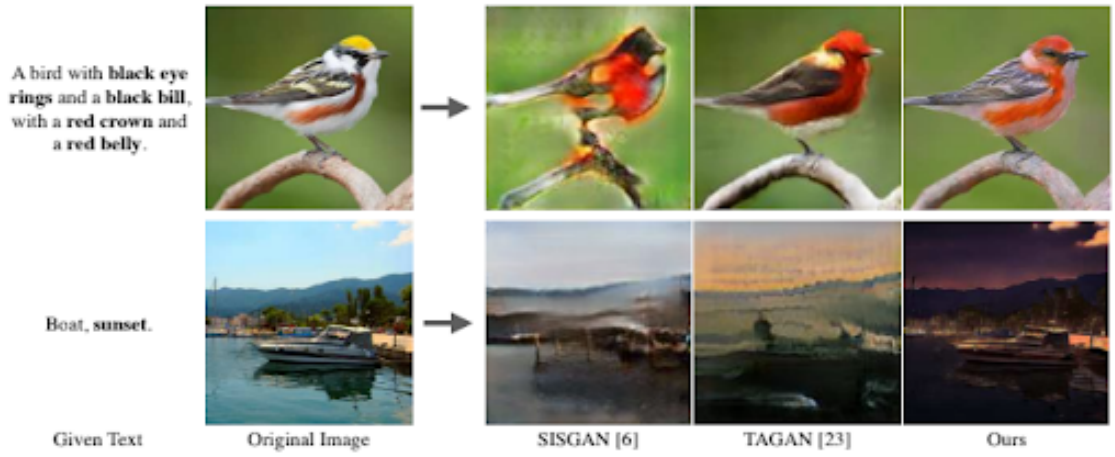

Figure 2.6. ManiGAN qualitatively result. Source: [22]

Even though it is not properly a text to image generation task it is interesting to see how they managed to modify some features of the image, which is what I attempted to do with an input reference image in my work. In addition, an interesting thing to notice about this work is the introduction of a new evaluation metric that defines the goodness of the result, that compares at pixel level the input and output image with respect to the similarity value between the final image and the input text through image and text encoders in a way that is similar to what DAMSM model does in AttnGAN. Indeed,

multiple times in the paper we can find references about Attention GAN, not only for the implementation of this correlation block inspired by DAMSM but also for the concept of a pretrained text encoder for embedding the input text.
The structure of the model itself is quite similar in many aspects to the one proposed by Xu et al. and ultimately similar to my own architecture. Figure 2.7 shows it in detail.

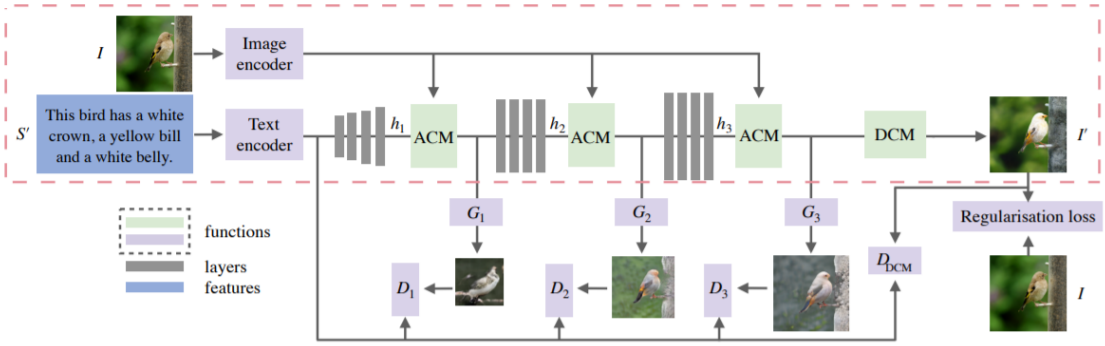

Figure 2.7. ManiGAN architecture. Source: [22]

As we can see, three generators are concatenated one after the other and each output is passed to a specific discriminator that also takes as input the embedded text, in order to check for correlation. The main difference is the input image that has to be modified, which is fed in multiple stages of the model.

2.6 Beyond GAN-based architectures

It must be said that text to image generation can also be done with different architectures than GAN based ones. In the paper Image generation from scene graphs [15], published in 2018, they show how it is possible to first generate a scene graph, a structure that encodes relations among words and objects, and then pass it to the model as input. This graph is used to predict bounding boxes and organize the objects in the scene (scene layout). This layout is then fed to a refinement network that starting from a noise canvas and using these bounding boxes as guidelines generates the final image. In practice the layout map is downsampled and concatenated to the noise vector first, and to the final feature matrix obtained by convoluting and upsampling the initial features. This work claims to be better than previous attempts

at generating complex scenes that usually end up to be messy and blurred. However, I did not consider this implementation too much since I preferred to work with input sentences instead of scene graphs, which require an extra step than actually just using plain text.

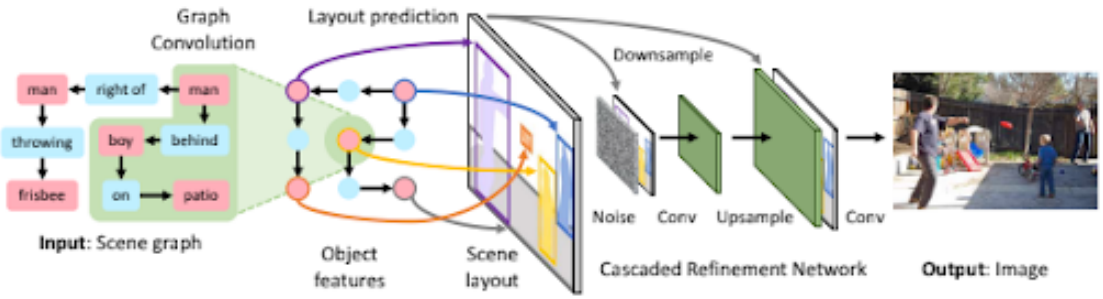

Figure 2.8. Scene-graph architecture, Source: [15]

2.7 Beyond text-to-image generation

As a conclusion to this chapter I would like to talk about an interesting work published in 2020 that has a somewhat opposite task to the one I have tried to solve in this thesis. Multimodal transformer for multimodal machine translation [46] aims to introduce information from different sources to improve natural language translation. A regular translation model takes as input a sentence in source language and returns the sentence in target language. This model instead takes a reference image in parallel with the input sentence. It learns a combined representation of information merging sentence and image features. This ensures that useless information inside the image is not considered since only what can be linked to the input sentence is actually extracted from the visual features and sent to the model. On the other hand text embedding is enriched by the visual embedding accounting for a more sophisticated description of the meaning. Unfortunately, the way this link between words and features of the reference image is created, is deeply based on the transformer architecture, that ultimately is the backbone of the entire translator system. It is therefore needless to say that I did not pursue this line for introducing an additional input to my model.
Another big portion of the image generation field is covered by what is known as *image generation from semantic layout*. A semantic layout, or segmentation map, is nothing more than an image containing shapes, usually of

different colors, each defining a different object in that specific portion of the canvas. These segmentation maps can be generated by hand, by coloring shapes using the predefined color scheme on a simple drawing program like MS Paint, or even starting from real images fed to convolutional models like U-Net [31] in order to extrapolate shapes.
Semantic maps can then be used in different ways. In SPADE [28] they use them to define the structure of the scene. Colors and textures are instead taken from a reference style image, a realistic image the model uses to extract the general style from.

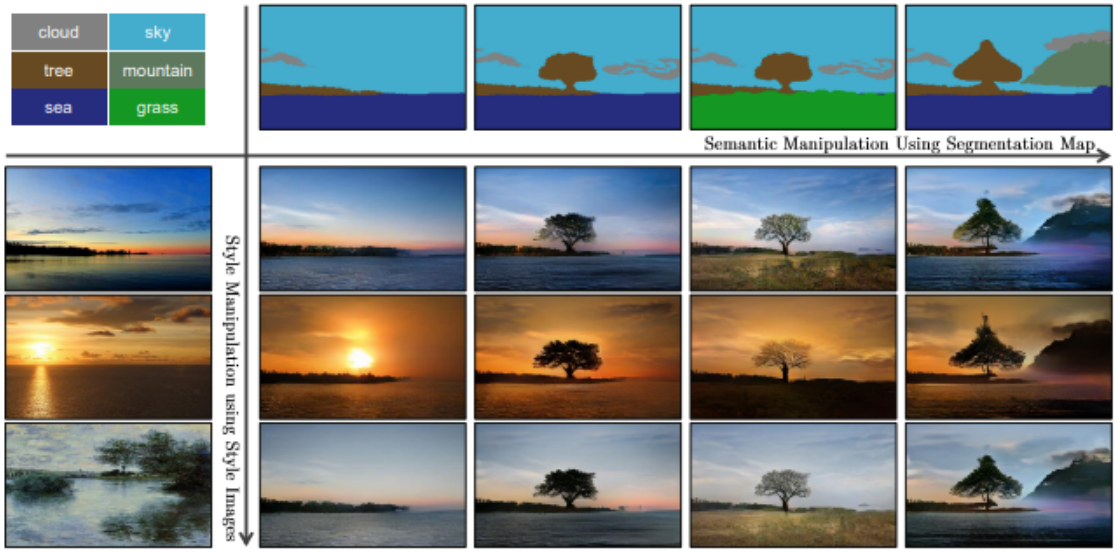

Figure 2.9. SPADE input (left and top) and output (center). Source: [28]

The generation of the final image is obtained thanks to multiple residual network blocks that, starting from random noise, modulate it all the way to the end receiving at intermediary steps the segmentation map. The style is instead passed through batch norm blocks.
On a later paper published in 2020 [24] Liu et al. argue that the semantic map has to be used to modulate the convolutional kernels of the generator of a GAN based architecture, so that the intermediate feature maps are more enhanced by the semantic features and ultimately should return a better quality image with particular improvement on the details. In practice the semantic map is passed to a weight prediction network that directly controls the convolutional blocks of the actual generator as showed in Figure 2.10.

Later on the same year, another work based on input semantic layout was published. Object-centric image generation from layout (OC-GAN) [35] tries

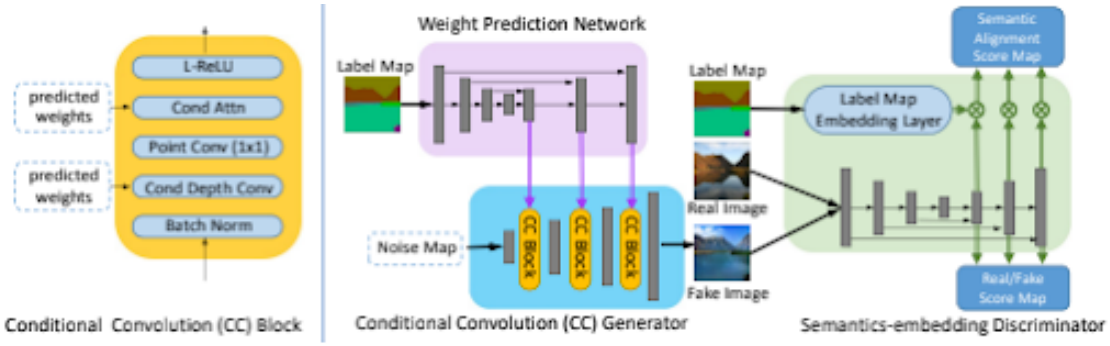

Figure 2.10. Convolutional layer kernels predicted using weight prediction network. Source: [24]

to solve the problem of generating multiple instances of the same object that appear in a close space. Previous image generation techniques have shown difficulties in reproducing non overlapping and crisp images of multiple items very close to one another. For this reason this work overcomes prior models like SPADE in generating complex scenes that now look more sharp in this particular setting.

Figure 2.11. OC-GAN output compared to previous models like SPADE. Source: [35]

To make this model work they make use of hollow bounding boxes (they

can be overlapped with others). With those they generate a scene graph which is used as input. The quality of the result is kept in check with a loss function that compares global and local features extracted from the input scene graph and output image.

2.8 Contribution

What my work tries to do is to extend the well founded GAN architecture, as it proves to be a solid infrastructure for image generation. In particular I try to expand the one developed in the AttnGAN paper, in order to support an additional input represented by a reference image that graphically expresses a random anatomical piece of a bird, whether it is a beak, a wing or anything else concerning the structure of the animal. Such an image will be used by the model in conjunction with the input text in order to reproduce the related portion of the output image in terms of texture and/or shape.
Not only does this idea require a prior knowledge of the presented works but it also requires an aware observation of the implemented techniques in order to find inspiration in making such extension of the model. The principal difficulties that the project has to face are how to insert this new input, how to make sure that the model actually uses it and how to do so in the context of attention so that we can also have a method to evaluate how the input image is exploited. Finally, since the task is arguably more difficult than text to image generation, another issue is to define a novel evaluation metric to express the effective goodness of the results.
All of this must be done with the idea in mind of not worsening the image quality: the final pictures have to be the visual representation of the input text and have to be as sharp as the ones returned by the original attention GAN model but they must also contain features that resemble the input reference picture.
I would like to point out that this idea, as far as my supervisors and I know, has never been officially tried, making this thesis work a novelty in this research field.

Chapter 3

Reference attention GAN

In this chapter I will discuss the architecture and the implementation of what I have called Reference attention GAN or RefAttnGAN for short. I will take a closer look at the backbone structure, that is, the Attention GAN architecture, and I will move from there to the model I created as a natural extension of it.

3.1 Attention GAN architecture

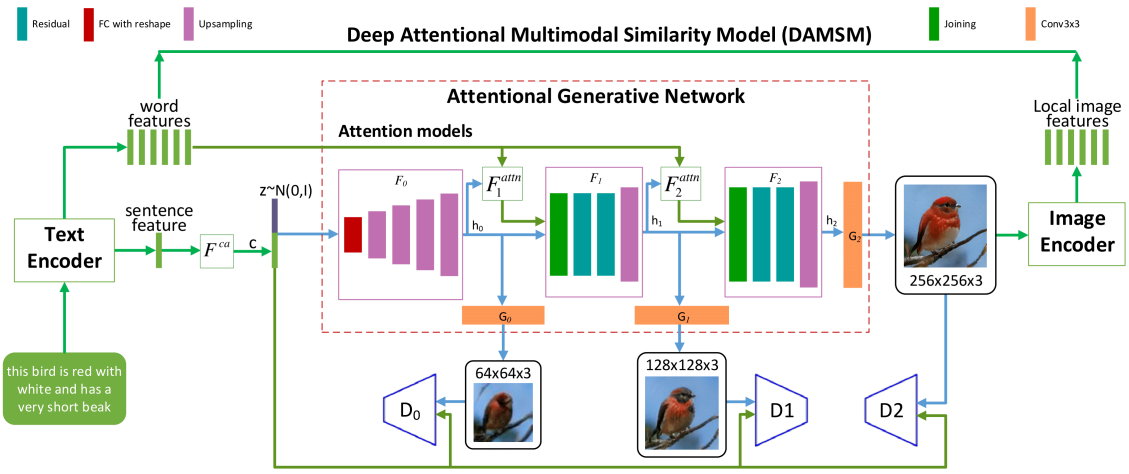

Figure 3.1. Attention GAN block architecture as presented in the paper. Source: [45]

Moving from left to right in Figure 3.1 I am going to discuss all the used components, briefly considering what they do internally and what they take

as input and return as output.

3.1.1 Text encoder

The input of the whole model is a plain English text descriptive of the picture we want to obtain. This work makes use of CUB dataset [42] and Coco dataset [23]. The first one contains pictures and descriptive sentences of each one (at most ten per picture) related to American birds of 200 species. Coco is instead a miscellaneous collection of "common objects in context" meaning that it contains pictures of everyday items, animals, means of transport and more. The latter is objectively more complicated to work with since its data distribution is wider and more sophisticated. For this reason in my work I limited the number of datasets used only to CUB. Therefore I will use examples from its data only from now on.
The input text is fed to a text encoder that embeds it into a sentence feature vector and a word feature matrix where each column is actually a vector containing the embedding of each word. This encoder is actually a long-short term memory recurrent neural network, or LSTM for short . Without going into too much detail, and using my personal knowledge on this type of neural networks obtained by following the course of "Natural language processing" [20] while I was attending lectures at KU Leuven, I will try to describe it. A regular RNN is a network often used for dealing with textual input, usually converted into numbers, and returns an embedding of each input word while simultaneously tracking its internal state, updating it each time a new input is received.

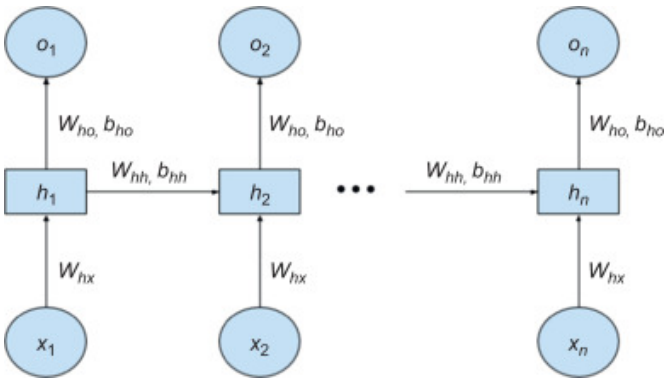

Figure 3.2. Recurrent neural network architecture concept. Source: public domain.

In practice, each word of a given text (an ordered sequence of words) is first transformed into numbers with the help of a look up table and becomes what in Figure 3.2 is X_i. Each encoded word is fed to the model one by one, respecting the original order. The network combines them with its state (at the beginning of each sentence embedding phase its internal state is reinitialized) returning an output embedding and an updated internal state. This state, which is nothing more than a weight matrix, should contain the semantic information given by all the words the model has seen before. However it has been proven that RNNs have an intrinsic problem related to this which is overcome by the introduction of an RNN variant called LSTM [27]. Even though the RNN model updates the very same weight matrix, its updates occur for each word the it receives as input. For this reason, back propagation is said to update the model through time, because it is like it traces back all the state updates. If the sentences are too long this might cause vanishing or exploding gradients. The example given by the above cited "TowardsDataScience" page gives a good example of this. Let's assume that the weight matrix of the model is just a scalar with value in one case 0.9 and in the other 1.1 and let's assume that the input sentence that has just passed is one hundred words long. If we use back-propagation multiplying these values one hundred times we would get a gradient that is either around zero (vanishing gradient) or very big (in the thousands order), exploding gradient. LSTM tries to solve this problem by actively "forgetting" certain words that are less relevant to be remembered in the internal state that won't be recorded. The internal structure contains some gates controlled by activation functions that decide whether or not to open or close them in order to let the word go to the state (being recorded), not go (being forgotten) or stay in a candidate state that will eventually become part of the state with the right weighting factor.
This encoder is used at training time after being pretrained in conjunction with the image encoder on the right hand side of Figure 3.1 as it is discussed in more detail in the section related to DAMSM.
The use of this pretrained LSTM text encoder is what makes AttnGAN step away from prior models: both local and global features of the sentence are used, meaning that information of the single word and of the whole sentence are considered and actually used in different ways.

3.1.2 Conditional augmentation block

The block labeled F_{ca} in Figure 3.1 receives the sentence feature vector and creates what is the actual input of the generator, the c_{code}, the conditioning vector. To put what this model actually does in a simple form I will cite what stated by my supervisor Dr. Ruben Cartuyvels: F_{ca} randomly samples the input vector from a normal distribution whose mean and co-variance matrix depend on the sentence level feature vector itself. This process has many advantages like adding randomness to the input data, adding concreteness to its content, which is just an arbitrary embedding of the input words that intrinsically contain arbitrary information, producing more image-text pairs as a single input sentence is sampled differently each time, so each epoch pass that uses the same sentences actually uses slightly different versions each time, therefore making the generator network more robust since it sees more examples.
The c_{code} is concatenated with a noise vector of fixed size, z_{code}, sampled from a normal distribution N(0, 1) just like it has been done in other models as presented in chapter two. This resulting vector (c_{code} + z_{code}) is what actually is fed to the generator network.

3.1.3 Generator network

Within this big network, that contains smaller concatenated blocks, all the magic happens that transforms the vector returned by F_{ca} into hidden states h_i later converted into RGB images. As I briefly discussed in chapter two, AttnGAN generator network is composed of a modular structure that can be repeated in multiple stages. This structure contains blocks labelled F_i, $F_{attn_{i+1}}$ and G_i and returns respectively an hidden state vector h_i eventually converted into RGB image, by G_i, at a certain resolution based on the stage i and an attention vector $Attn_i$. F_i block is used to upscale the input hidden state vector (either the c_{code} + z_{code} or the hidden state returned by a previous step) to a higher dimensional vector h_{i+1}. This is done in the case of F_0 with a first concatenation of the input vectors by means of a fully connected layer followed by a batch normalization. After this phase there is a series of up-scaling blocks: an upsample layer using “nearest” mode followed by a convolutional layer. The result is the hidden state vector h_i, containing latent information upscaled from the input. This vector is directly fed to the next F_i block. It is also passed to generator G_i, a convolutional layer followed by a tanh activation function that returns a three dimensional matrix, which

will become a real image (of size channels x height x width), but also to the $F_{attn_{i+1}}$ block. This attention block is actually the core component of the whole model as it introduces the attention mechanism to the second and third F_i block based on word features. In practice it takes the current hidden state and word feature matrix e and computes the following functions:

$$s_{j,i} = h_j^T e_i \tag{3.1}$$

$$\beta_{j,i} = \frac{exp(s_{j,i})}{\sum_{k=0}^{T-1} exp(s_{j,k})} \tag{3.2}$$

$$c_j = \sum_{i=0}^{T-1} \beta_{j,i} e_i \tag{3.3}$$

$s_{j,i}$ is the dot product between hidden state j and features of word i and defines the weight the model gives to region j when considering word i. These weights are normalized and become the $\beta_{j,i}$ coefficients used in the third equation for weighting each word feature vector i. We end up with C_j, a word-context vector that defines for each region j of the hidden state the cumulative attention associated with each word of the input sentence. This attention vector is then concatenated to the hidden state and the residual is computed and upsampled.

3.1.4 Discriminators

Real and fake images are embedded into vectors by the discriminators and later classified. In addition, input text global features are checked versus the generated images in order to account for the counterpart of what generators do.

For each generator and therefore for each output image resolution, a discriminator is set up and trained independently from the others in order to determine whether the generated images are realistic enough, classified as real even though fake or not. In practice, each discriminator is just a sequence of convolutional blocks followed by batch normalization layers and activation functions, in this case LeakyReLU that convert the input image into a feature vector. It is important to briefly discuss the used activation function. It is based on the more known rectifier linear unit (ReLU) but differs from it as it has a small slope for negative values instead of being flat at zero [44] and it is commonly used for training adversarial networks. It can be shown that this activation function solves intrinsic problems coming from

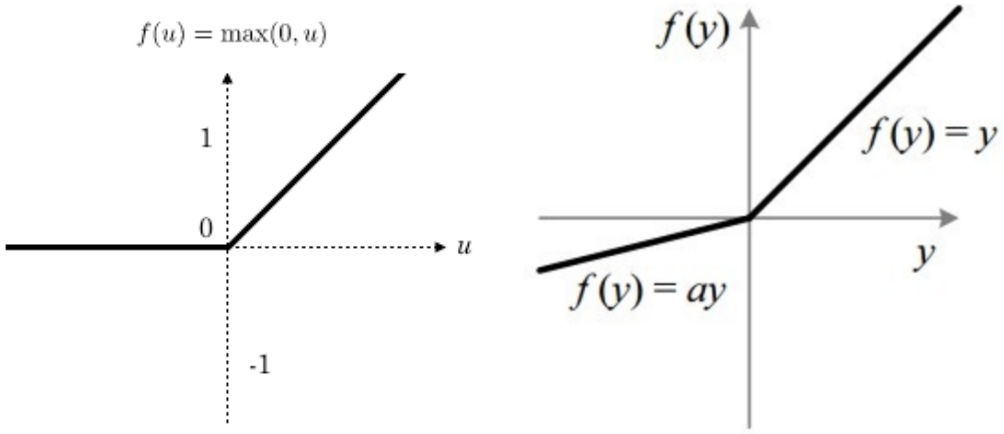

Figure 3.3. ReLU (left) vs LeakyReLU (right) slope comparison. Source: public domain.

ReLU and speeds up the training process [25], which is very important since the model consists of three discriminators.

3.1.5 DAMSM

DAMSM block represents what Attention GAN really brings to the table of image generation from input text. This portion of the model can, and has to be considered something separated from the rest but very important for the whole model to function properly. It's purpose is to define the matching score between the input text and the output image (only the highest quality image is considered). Word features coming out of the text encoder and region features extracted from the output image from an image encoder (an InceptionV3) are mapped into a common space, which makes it easier to semantically match them later. This forced mapping allows for semantic connections and proportional distancing in this common space of features coming from different sources. As I have studied in the course "Information retrieval and Search engines" [19] attended at KU Leuven, pouring information from multiple sources is a good option for later encode input queries (in this case the input text) and documents (in this case output images) and retrieve the most meaningful matches, in practice the closest in terms of euclidean distance.

This training of this bloc does not require the rest of the model at all

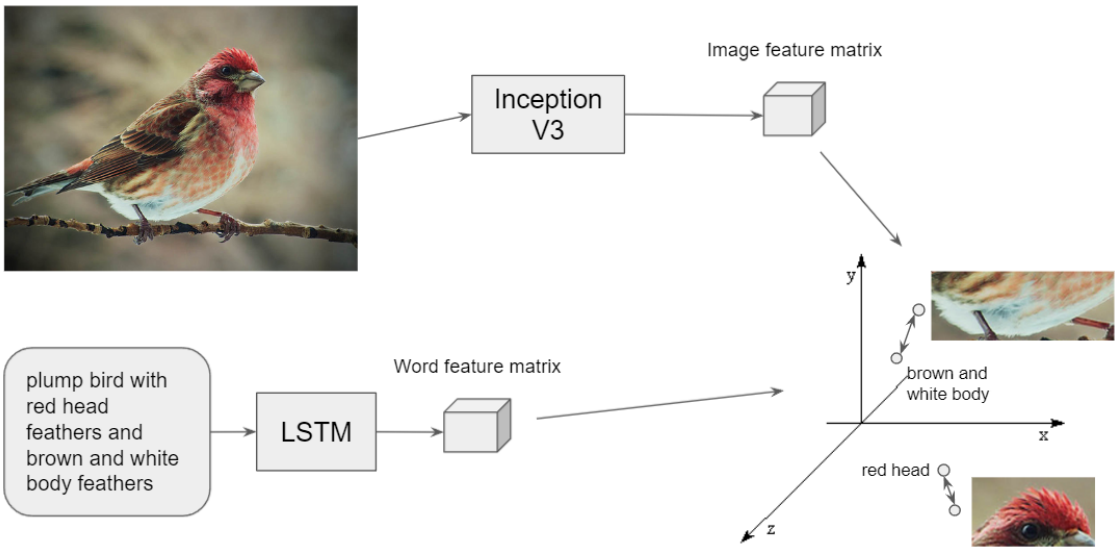

Figure 3.4. Multi-source feature mapping to common space.

since it just needs text-image pairs directly from the training data. Indeed, DAMSM is trained from scratch before the generator itself. Pretrained LSTM and InceptionV3 models are taken from the Pytorch library and fine tuned in order to minimize the so called word and sentence loss terms conditioned to the image features and vice-versa. Without going into too much detail, word, sentence, region and image features are extracted from both models and compared using dot product and later normalized. By taking vectors in pairs from these four feature types we end up with four loss terms that define cross modal losses:

$$L_{DAMSM} = L_1^w + L_2^w + L_1^s + L_2^s \tag{3.4}$$

3.1.6 Image encoder

The image encoder used is a pretrained InceptionV3 model [29] downloaded from the official Pytorch models repository [2] and fine tuned for the mapping of extracted features in this common space. This model is an improved version of more known convolutional neural networks like AlexNet [17] or ResNet [12] commonly presented in deep learning courses, and takes the image embedding to the next level. As the name implies, this is actually the third version of this architecture, though not the last, as Inception v4 does

exist. The idea behind this new convolutional network is to solve a few problems concerning older CNETs. Input images have a huge variation in the location of information and size, that is, where the main object is in the picture and how much surface it covers. Therefore, it's hard to define a correct kernel size for each convolutional block. It has to be big enough to extract global features but also still small enough to capture details. This is partially dealt with by stacking multiple convolutional kernels with different sizes but this causes very deep networks, prone to overfitting and with gradient related issues. In addition, stacking up kernels cannot be the solution as they are traditionally the source of computationally heavy operations which cause clunky and long training sessions. Inception models try to solve all these issues going "wider" instead of "deeper" by using multiple convolutional kernels at the same level. The output of each block is then concatenated and sent to the next layer. 1x1 blocks are also used before bigger kernels to collapse the input channels down to one in order to limit the number of operations needed.

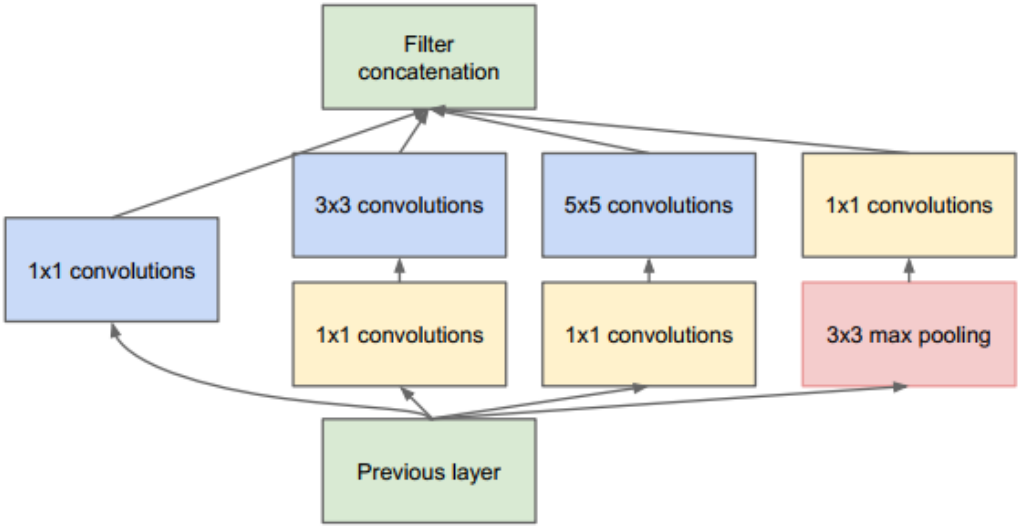

Figure 3.5. Inception basic block. Source: public domain.

Further versions present improvements by using a different factorization of bigger kernels into smaller ones to reduce computational costs but the concept remains the same. Inception V3 keeps all these ideas and changes the optimizer and the loss function for its training phase.

3.1.7 Loss function and training overview

As always, for training a deep learning model we need a target function to minimize, a loss function that should determine how badly the model is performing, therefore requiring it to go as low as possible during the training phase. The entire training process, whether it is for DAMSM or the generator itself, is computed on the train set of the chosen CUB dataset [42]. This collection of data contains 11788 images of 200 subcategories of birds. Roughly half are part of the training set and the rest is test set. Each image has in addition up to 15 part locations, coordinate positions of bird parts (very useful for my work), one bounding box containing the bird only and ten single-sentence descriptions. The official website states that it also contains other things but for this work purposes, only the ones listed above are what has been used.

Figure 3.6. CUB dataset birds-200-2011. Image taken from official CUB website.

Just like with any other generative adversarial network, the training process consists in alternating the training of the generators with the one of the discriminators. For the first group, the loss function that we try to minimize is the following:

$$G_{totLoss} = \sum_{i=0}^{m-1} L_{Gi} + \lambda L_{DAMSM}(+KL_{loss}) \tag{3.5}$$

The first term is the sum of all three losses of the three generators, each

consisting in the contribute of conditional and unconditional adversarial loss with respect to the input text $\bar{e}$:

$$L_{Gi} = -\frac{1}{2}E_{\hat{x}_i \sim p_{Gi}}[\log(D_i(\hat{x}_i))] - \frac{1}{2}E_{\hat{x}_i \sim p_{Gi}}[\log(D_i(\hat{x}_i, \bar{e}))] \tag{3.6}$$

Basically the first term computes the loss function depending on the respective discriminator D_i without considering the input sentence, while the latter term does consider the input sentence features. Each term has the same weight so the generator is pushed to produce a realistic image as much as it has to make it closely related to the input sentence.
The second term of 3.5 is the loss computed by the DAMSM block, that compares word features and region features of the highest quality image and returns a value that defines how correlated they are as briefly presented in Equation 3.4.
There is actually a third term to the total generator loss expressed in 3.5, that is not clearly specified in the paper but that I did find in the code and is defined as Kullback-Leibler divergence, or KL loss. In mathematical statistics, this divergence, D_{KL} (also called relative entropy), is a measure of how one probability distribution is different from a second, reference probability distribution [43]. It comes from the sampling of the sentence feature vector computed in the conditional augmentation block. As the definition states, it compares the input distribution with the sampled one. Unlike other losses, the correct behaviour for this one is slowly increasing since the sampled distribution has to be more and more different from the input one, though it does not have to explode or diverge.
Discriminators are trained separately from one another with a more complicated loss function that once again considers a conditioned and unconditioned term with respect to the input sentence, in order to check for image realism but also image relation with input text. Since discriminators are actually convolutional layers that output image features later converted into logits, each defining the probability of the image to be real (logit close to 1) or fake (logit close to 0), the loss function is actually a combination of multiple cross-entropy losses.

$$\begin{aligned} L_{Di} = &-\frac{1}{2}E_{\hat{x}_i \sim p_{data_i}}[\log(D_i(\hat{x}_i))] - \frac{1}{2}E_{\hat{x}_i \sim p_{Gi}}[\log(1 - D_i(\hat{x}_i))] + \\ &-\frac{1}{2}E_{\hat{x}_i \sim p_{data_i}}[\log(D_i(\hat{x}_i, \bar{e}))] - \frac{1}{2}E_{\hat{x}_i \sim p_{Gi}}[\log(1 - D_i(\hat{x}_i, \bar{e}))] \end{aligned}$$

3.2 Reference attention GAN architecture

Now that the reader has a clear understanding of the base AttentionGAN model, I will present what my integration is to this work and I will show how each new block and concept has been harmonically added into it. The final model structure is presented in figure 3.7, where the new blocks are presented in orange while the old ones are in blue, and will be used as a map for presenting it just like I did for AttnGAN.

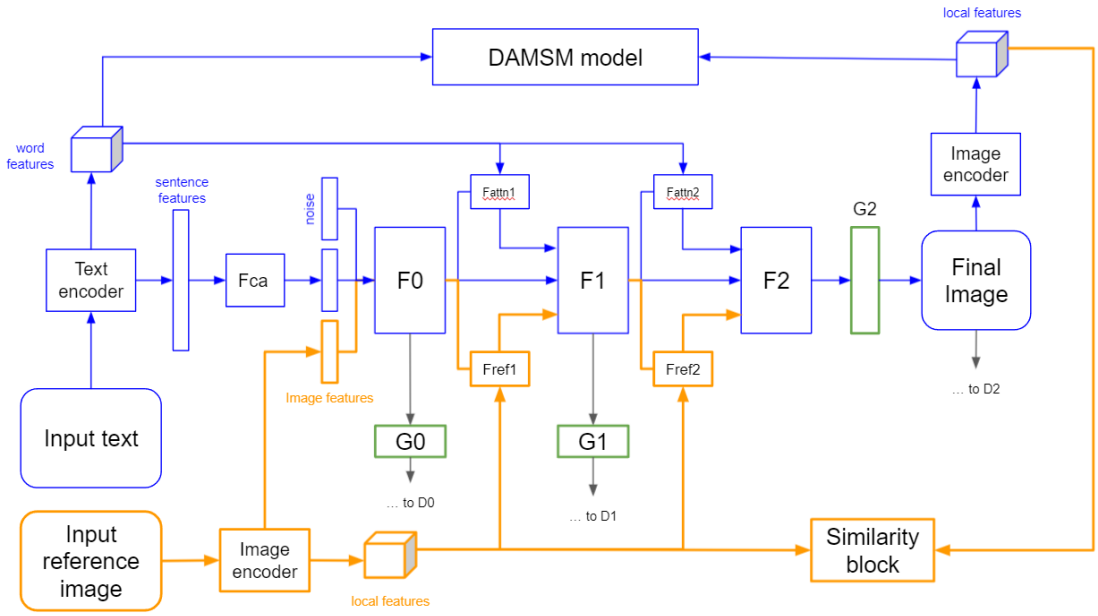

Figure 3.7. Reference attention GAN architecture

3.2.1 Reference images

The input reference image is a 50x50 pixel RGB picture cropped from real images of the train set using the part locations already present in the dataset as center position for each square crop. In case of locations that do not allow for a perfect square image, I added some black padding in order to ensure the same input size. Once all the possible reference images are created (I end up with roughly 100 thousands of them), the dataset script automatically assigns them to a pre existing text-image pair via random sampling. In practice, the dataset, instead of returning image and text, it returns the triplet (text, reference image, image) for the model to use.

Figure 3.8. Some generated reference images

As we can see in Figure 3.8, some of them are arguably hard to identify even for us humans. It is not always clear what portion of the bird they represent and where to use them each time, a problem that has some consequences in the results.
Following the arrow coming out of the reference image in Figure 3.7 we find an image encoder. It is a pretrained inceptionV3 (actually the very same used for the final image embedding) and it extracts features from the reference image. Global features, expressed in one single vector are then passed to the generator as input while local features, extracted from multiple regions of the image, are stored in a local feature matrix and used later.

3.2.2 Generator network

Just like it is done for concatenating sentence features and noise, I decided to insert in the same way the global reference image features. For this reason, the first fully connected layer of F_0 has been adequately modified in order to do just this, while keeping the rest of its internal layers the same. It must be said that this has been done for simplicity purposes. As I will discuss in chapter five, much more could have been done in this work if I had more time and resources, and one thing was certainly to engineer better the rest of the layers of this and other blocks so as to account for the increase of the input dimensions and complexity.
F_0, just like before, upsamples and convolutes the input and generates a hidden stat h_0. This state is passed to the next block and to $F_{attn}1$ as always but in addition it is also passed to $F_{ref}1$ block, a personal invention that covers the same role of its counterpart $F_{attn}1$ but with respect to the local features of the input reference image. In a very similar way to how F_{attn} block generates an attention map based on the word features, this block does the same using region features of the image. Therefore, this technique mimics what the original model does, supporting the concept of attention, which can

be exploited later for visualization purposes. Indeed, Equation 3.3 can be utilized in this context too simply by changing the vector $\bar{e}$ with $\bar{r}$, that is the reference image local feature matrix. With this, later blocks can have a perception of the input reference image that conditions them to produce a final output that should resemble some features extracted from it. In addition, the input is not simply forwarded at the beginning only. Instead, it walks with the hidden state enriching it, making it more "aware" of additional input features every time a F_{ref} block injects attention vectors based on it. Just like in the original model, this block and this process of attention is repeated between F1 and F2, which have been correctly modified in order to accept this additional attention vector.

3.2.3 Similarity block

The use of local features is not relinquished to the attention mechanism just presented. Actually its main purpose is to check for coherence in the final image. For this reason I also created what I have called "similarity block" which is the DAMSM counterpart. Before, with the DAMSM block we were comparing information from different sources but with the same content. Defining a bird with words or using a picture arguably gives the same information and for this reason the idea of mapping them to the same space made sense. In the case of a reference image, that has to express information for just a portion of the total information contained in the image, we cannot do this. This is why for example I have not retrained the DAMSM model so that the text encoder and the two image encoders mapped in the same space, because it was not the correct thing to do. Instead, I had to find a way to check for similarity between the reference and final image but in a posterior fashion. In practice what I do is to exploit the already extracted local features of the final image and compare them with those of the reference image. The problem is that the reference image has to be compared only to a portion of the final output, and in addition the right portion. To make myself more clear I try to give an example: if the input reference image represents a beak, we want to compare how similar it is to the portion of the final image where the beak of the bird is contained, and nothing more.

One option that I thought of consisted in pre training an object locator capable of identifying the exact position of the specific bird part defined in the reference image onto the final image, cropping out of it and creating a 50x50 patch, extract local features using the same inception model and comparing

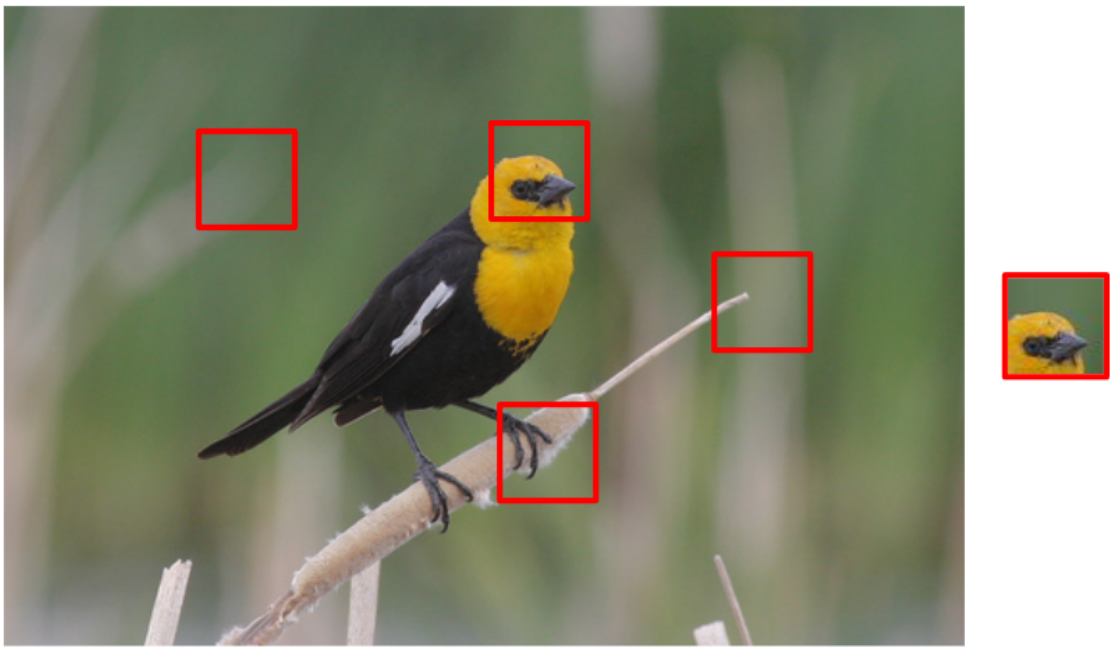

Figure 3.9. Finding most similar patch in final image

the feature vectors with cosine similarity for example. If in theory this idea works and makes the comparison meaningful, in practice it means to train another model, without the insurance of it to work properly. It is definitely something to try but as my supervisors suggested to me, this path was too hard and long to pursue and so I opted for something else. Another option was to embed the final image into a feature vector or even using the one that the inception model naturally returns and toss it to a series of fully connected layers that had to be trained to return the coordinates of all the 15 possible bird parts. This idea came to mind after reading a few works conducted by my professor supervisors Mrs. Moens at KU Leuven [7, 6]. The advantage of this idea is that in future, the model could have accepted multiple reference images and could have compared them to the final image because each possible bird part was being located into the final image. Once again, the solution required an external model to be trained, and the limited resources did not allow it. Another option to be discarded.
What I actually ended up doing was to create what I defined in Figure 3.7 as "similarity block" that in reality simply computes a similarity function and is not a trainable part of the model.

$$maxSim_i = max_j(cosSim(r_i, fi_j)) \tag{3.7}$$

$$sim = \frac{1}{L}\sum_{i=0}^{L-1} maxSim_i \tag{3.8}$$

$$Sim_{loss} = 1 - sim \tag{3.9}$$

The function takes as input r_i and fi_j, local features of the reference and final image respectively. They are compared in pairs and for each patch i of the reference image I save the highest similarity score, $maxSim_i$. I have L patches so I end up with L similarity scores defining for each region i of the reference image the most similar patch in the final image. They get collapsed into one value by taking the average, sim. This average similarity score is converted into a loss in Equation 3.9 by subtracting it from one. This loss is added to the generator loss so that now the they have to account for producing a realistic image that is representative of the input text and that has a region similar to the input reference image.

$$G_{totLoss} = \sum_{i=0}^{m-1} L_{Gi} + \lambda_1 L_{DAMSM}(+KL_{loss}) + \lambda_2 Sim_{loss} \tag{3.10}$$

λ_2 is a hyper-parameter I added in order to weight more or less this last loss term.
The concept behind this formulation is simple: we are comparing patches extracted from the reference and final image and we want to consider the most similar pair of patches. For this reason every time I take the maximum similarity value.
The advantage of this loss comes from the fact that it makes it easy to identify a similar region and hopefully a region that is actually associated with the relevant portion of the image expressed in the reference image itself. On the other hand we cannot control the precision of this match which is arguably quite low at the beginning, when the generated images are not realistic at all and therefore it's quite hard to find similarities even for us humans. In any case, this added loss term should enforce the concept of usage of the reference image in the final synthetic one.

3.2.4 Discriminators

It is important to notice that the global features of the input image $\bar{r}$ are also used by the discriminators for computing an additional term of conditioned loss that gets added to the previous ones:

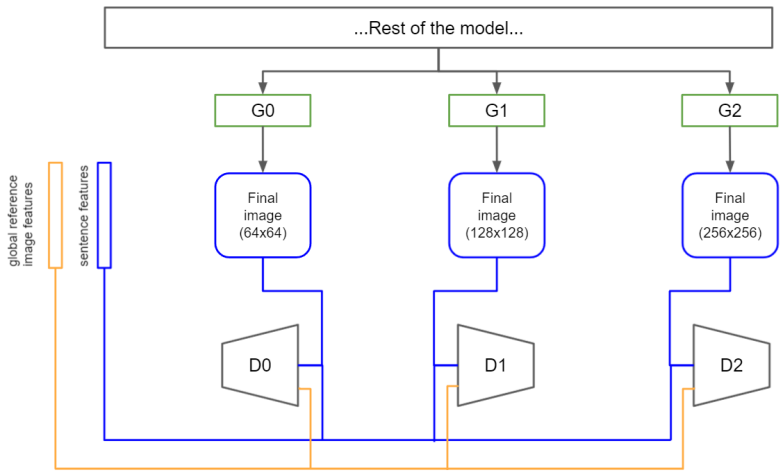

Figure 3.10. Reference attention GAN discriminators

$$L_{Di} = -\frac{1}{2}E_{\hat{x}_i \sim p_{data_i}}[\log(D_i(\hat{x}_i))] - \frac{1}{2}E_{\hat{x}_i \sim p_{Gi}}[\log(1 - D_i(\hat{x}_i))] +$$
$$-\frac{1}{2}E_{\hat{x}_i \sim p_{data_i}}[\log(D_i(\hat{x}_i, \bar{e}))] - \frac{1}{2}E_{\hat{x}_i \sim p_{Gi}}[\log(1 - D_i(\hat{x}_i, \bar{e}))] +$$
$$-\frac{1}{2}E_{\hat{x}_i \sim p_{data_i}}[\log(D_i(\hat{x}_i, \bar{r}))] - \frac{1}{2}E_{\hat{x}_i \sim p_{Gi}}[\log(1 - D_i(\hat{x}_i, \bar{r}))]$$

Following the ideas behind the original terms conditioned by the sentence features, I decided to also create terms that should depend on the reference image. In this way the discriminators now check for three things: realistic image, or better, an image in the target distribution, an image conditioned to the sentence, what is described in words has to be present in the picture and finally an image that contains features of the reference image.
I must point out that as I explain in chapter four, some experiments have been carried using the original discriminators configuration, that does not consider the loss terms conditioned to the reference image. For the sake of completeness I decided to present the architecture with this variation nonetheless.

Chapter 4

Experimental procedure

In this section I will show more in depth some technical implementation details, training setup, framework and conducted experiments on RaGAN. In addition I will show some qualitative results in order to better appreciate the output images the model is able to generate. In the final section I will discuss some quantitative metrics for evaluating and comparing the models even with respect to AttentionGAN.

4.1 A cumbersome model

Aside from the huge leap forward that GAN architectures have brought to the table of deep learning, a big technical disadvantage of models based on this structure is the fact that they are usually quite resource intensive, requiring a particular setup not easily obtainable for an amateur programmer or a student. Whenever we launch our training script we are asking in this case Python interpreter to load in memory a big generator, usually composed of multiple blocks, multiple discriminators and additional side components that support the entire structure like the various encoders presented in chapter three. In addition, to make computations faster and in some cases even feasible at all, the required device and memory is actually a GPU, as it proves to be better and faster than a CPU especially when dealing with big deep learning models and matrix calculations [38].
Since the very beginning, when I first downloaded the official Attention GAN code from the Git page specified in their paper [37], I faced the problem that my personal machine, that I thought was powerful enough for the task, was simply not able to even pass the loading stage of a pretrained version of the model, available to download on the same repository. My Nvidia

GTX 1660Ti with 6Gb of dedicated RAM kept on returning the same error: "CUDA out of memory". The original model requires 14 GB of memory to be loaded and some more to run depending on the batch size. Therefore running the model, or even worse, running the even bigger RaGAN locally, quickly became out of discussion. Fortunately though, my thesis supervisors at KU Leuven suggested me to request an account at VSC, Vlaams Supercomputer Centrum [3]. Thanks to my KU Leuven student account, I managed to get access at a Tier-2 cluster called Genius after creating and exchanging a pair of SSH keys. Once the account was created I was able to submit jobs through the command line and upload and download code via WinSCP [4].

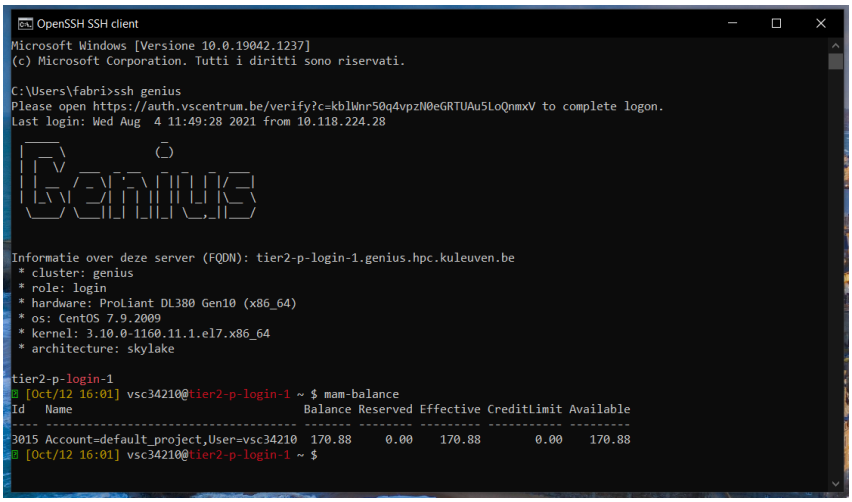

Figure 4.1. Accessing Tier-2 Genius cluster from command line.

With this setup, I was able to run very long training sessions without any issue. Unfortunately, as I will discuss more in depth in chapter five, the provided account came with a limited amount of credits[1], a VSC currency that gets consumed at each job submission proportional with the specified amount of time requested. This alone limited the number of training and planned experiments quite a lot.

[1]As student I only had access to 2000 credits. Each model training costed me around 500 to 600 credits.

4.2 Preliminary steps

Once the job runs on the cluster, the only way to inspect what is happening is through two files automatically generated on my remote account, containing standard output and standard error. If this manual inspection was ok at the beginning, it quickly became obvious that I needed something else to control the training session even without always being forced to connect to the supercomputer. In order to solve this issue, I decided to use Neptune.ai [1], an online tool for keeping track of various deep learning experiments with a simple interface. The only thing required is to add a few lines of codes that log onto my account whatever I want, from strings, to numerical data to images. I decided to send used configuration, losses and stream outputs in order to check for what parameter I was using, how the losses were behaving and at what epoch did the training arrive.
Even though the real runs of the code have been executed on the VSC supercomputer, this server based setup made the debugging phase clunky at best and definitely time consuming, without considering user credits depletion. For this reason I decided define some configuration files that, if passed as parameter to the main script, are used to overwrite the default values of specific model values such as embedding vector dimensions, batch size and so on. In this way whenever I was working on my computer I specified a configuration file that made the model smaller, reducing input and output vector dimensions for example, or reducing the number of epochs, so that I could run it locally. In this way I could always make sure that the code was at least reasonable before sending it to the supercomputer where it was run with the full dimensions. This has been particularly important at the very beginning: the code I have downloaded from the official Attention GAN Git repository is written in Python 2.7 in conjunction with a series of required libraries that function with that version of Python. At first I tried to use the same setup, therefore I created an Anaconda environment with the specified libraries and versions. However I quickly came to the conclusion that this arrangement could not work anymore. Due to new versions of required libraries being released, and old ones being discontinued and no longer supported, I was not able to use them or even install them at all in some cases. Therefore I decided to move to Python 3.7 and the latest supported versions of the needed libraries, Numpy, Pythorch, TorchVision and Pillow just to name the main ones.
Updating the code to the next Python version has not been that complicated, at most time consuming. From what I experienced, it has been a

matter of finding newer functions for those that are no longer supported, adequately adapting the parameters to the different function signatures and in some cases rewriting in a more modern and elegant fashion some code pieces. However, I would like to underline the fact that I simply revamped the code but I have not changed the logic behind with the idea to make AttentionGAN run.

4.3 Sanity check

Once the Attention GAN code was free of bugs or running errors my supervisors and I decided to train it fully on the VSC supercomputer in order to see that everything worked properly. To make sure of that the idea was to generate some images using the validation set and compute a score to be compared with the one presented in the paper. This sanity check was in our opinion mandatory for ensuring that the code I was using was actually legit. As I discovered in a project for the course "Machine Learning and Deep Learning" followed at Politecnico di Torino [9], it is not uncommon to find the official code being underwhelming with respect to what is claimed in the published paper.
To limit the usage of VSC credits, I decided to use the pretrained DAMSM block they offered, and I trained only the generator for 600 epochs, as specified. It is important to let the reader know that the training time, even if I was using VSC, is enormous. For this and other training sessions I am presenting, I spent roughly 24 hours per 100 epochs, meaning that each training, from start to finish, takes almost a week. Part of the training time is taken by side operations such as model checkpoint saving and more importantly training examples like the ones that I show in Figure 4.2.

At the very beginning the model simply has no idea of what to do: the image is just a bunch of RGB colors that do not represent anything. However, in just 200 epochs we can see a realistic output. I must say that even though the example I have chosen here is quite good, at this early stage of training the model makes a lot of mistakes and many output images are unrealistic. From that stage to epoch 600 the model refines itself, ensuring higher quality for a the vast majority of cases.
Once trained, I put the model in evaluation mode to generate some images from the validation set and I used those to compute the inception score, a quantitative metric used in the original paper to automatically evaluate image quality. It measures two things simultaneously: image variety (in this

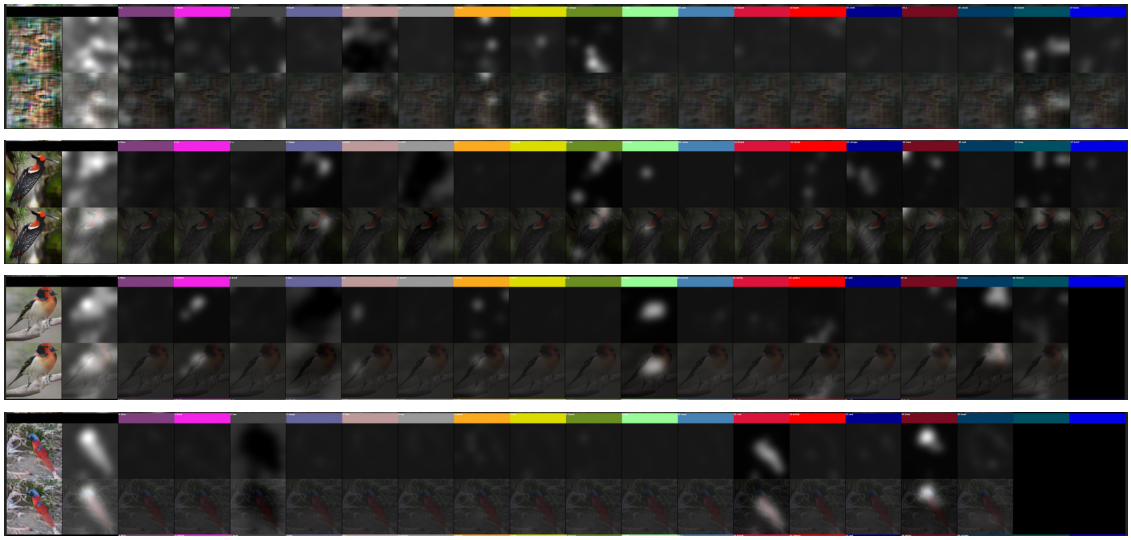

Figure 4.2. Training samples taken, from top to bottom at epoch 1, 200, 400, 600. Each image shows the output picture next to the attention maps for each input word.

context the various races of birds) and image distinction, meaning that it evaluates for each image if it represents something (each image is clearly a bird or not). If both things are true then the score will be high. The lowest possible value for this metric is 0 but it does not have upper bound as mathematically the highest possible score is infinity. The name comes from an inception model used for classification [32]. The idea behind this work and this score is that when it classifies similar objects the confidence sum gives a focused distribution while if we classify images containing different objects the resulting distribution will be more uniform.

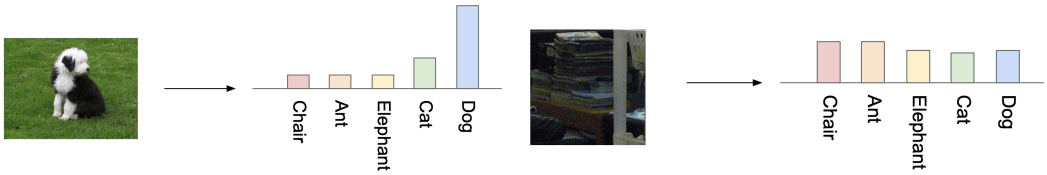

Figure 4.3. Basic idea behind inception score calculation. Source: [26]

In practice for each image we compute the label distribution and marginal

distribution and we see how far away they are from the ideal ones. It is clear to the reader that this metric has its limitations that mainly end up in the classifier network used and the training data it was trained on. These two elements define what features the classifier will consider and how. The one used for this evaluation is the official one that uses StackGAN inception model.
Luckily, generating images from the validation set and computing the inception score are two operations I was able to achieve locally. Therefore I did it multiple times so that I could compute not only the score but also its variance over three trials. Over 2920 images, 256x256 pixels RGB I obtained the following score:

	Inception score
Official	4.36 ± 0.03
Experimental	4.27 ± 0.20

Table 4.1. Inception score comparison between official and experimental value.

As we can observe, the obtained value is a bit lower than the presented one. However I do not think this result has to do with some mistake I made in the conversion of the code to a later Python version because other users that have used this implementation from the Git repository have experienced the same range of values [37]. In addition, other papers citing AttnGAN like in the work of Tian et al. [39], present a similar score at 4.23 ± 0.03.
Regardless of this quantitative score, when dealing with image generation it is also important to evaluate image results from a qualitative point of view. Figure 4.4 shows some examples.

Figure 4.4. Attention GAN output images from validation set.

Aside from some non realistic examples, most of the produced images are

quite believable and of good quality, stating that the true power of the model is actually here, meaning that the slightly lower than specified inception score is not that important.

4.4 RaGAN training

With the original code up and running, I went ahead implementing the blocks I discussed in chapter three. Without going into too much detail of every single implementation step, I will say that I tried to keep the original structure as much as possible, and I started building new blocks taking inspiration from their counterpart already present in the original code. In particular, F_{ref} blocks are quite similar to F_{attn}. What I did have to change was the input size of F_1 and F_2 in order to accept the additional attention vector generated by my blocks. In practice I concatenated the hidden feature vector with text and image attention, creating a bigger array: $h_i + f_{attn_i} + f_{ref_i}$. The same is true for the loss functions that now have to account for the additional conditioned term. Another thing that I did was to extend the initial script responsible for dealing with the dataset that now has to generate reference images and link them to the original text-image pair. I must point out that, in order to make the project more realistic I decided to allow for the association of reference images of different classes. This makes the whole task harder because there might be clashes between what the input text states in terms of colors and dimensions and what the reference image depicts, something that will prove to be a big disadvantage for the whole project, as I will present later in this chapter.

4.4.1 DAMSM training

For the training itself, I first trained the DAMSM block. As stated in chapter three, this block has not been changed at all, therefore I simply made sure the original code responsible for its training was good to run and then I trained it for 200 epochs, as specified in the in the original work. Even at this early stage I logged the loss terms associated with its training on Neptune.ai. Figure 4.5 shows them.
As we can expect from a tuned model like the original one, the loss functions are monotonic decreasing both on single terms (word and sentence loss) and overall. Word and sentence losses are computed by means of cross entropy loss that take as input either words and local features or sentence and global

features and compare them with image features as explained in chapter two. Indeed we have two pairs of losses for each type (sentence and word) because these vectors are compared in two different ways with image features, using either posterior probability or not.

4.4.2 Generator training

Once the DAMSM section was trained, I moved both text and image encoders to the correct folder of the main script containing the rest of the model, and I submitted the job for training the generator network. I have to say that due to a coding bug that I introduced, the first run failed: the generator loss diverged and I was not able to control the training process until I found and solved the issue that caused this behaviour. However, in order to avoid failures like that one and save some credits, since that divergence happened after some epochs I preferred to go extra careful with the other runs, and instead of training for 600 epochs straight, as specified for AttnGAN, I divided it into training sessions of 200 epochs. Every three epochs the model gets automatically saved into a checkpoint "pth" file. This allows for restarting training process from whichever checkpoint I want. To give the reader a better picture of the 600 epochs at once, I concatenated the various graphs locally using Pyplot library taking the data I logged to Neptune.ai.

Figure 4.6 shows the global trend of the whole generator and discriminator losses. I will refer to this first training trial as RaGAN V0 because here I did not implement the conditioned loss computed with respect to the reference image since as a first experiment I simply wanted to make sure that everything worked correctly before moving further. In any case, we can already see two important things from these two graphs. First and foremost, the generator loss is increasing while the discriminator one is decreasing. In addition the level of magnitudes at which they respectively rise or fall are quite different: the generator stays in the order of tens while the discriminator has a loss that is way lower, starting from roughly one and going closer and closer to zero. This difference in magnitude can be explained by the fact that discriminating images, aka classifying them is a much simpler task than actually producing them. For this reason the discriminators do not take more than a few epochs to learn how not to be fooled while the generators really have to struggle to let a fake image pass as real. In addition, the more the discriminators improve, and we have to underline the fact that at each iteration the discriminators are trained before the generators so are always a step ahead, the more it gets hard for the generators to keep up. The second thing

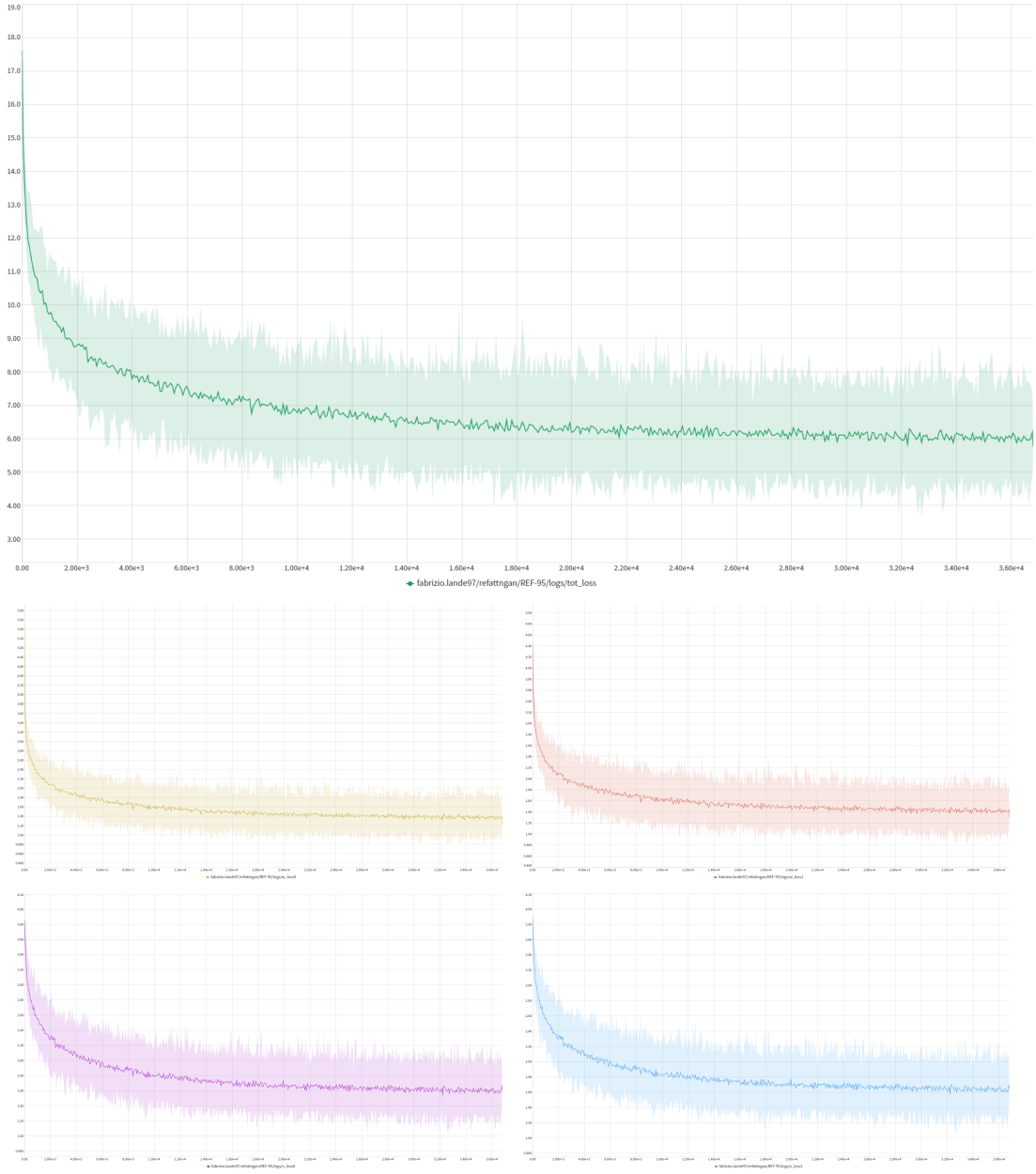

Figure 4.5. DAMSM loss graphs. Total loss (top), word losses (center) and sentence losses (bottom).

that is quite visible is the disturbance that the model has to face at every training re-initialization. Asking the supercomputer to reserve three shorter sessions costs less than requiring a long one. However this technique has

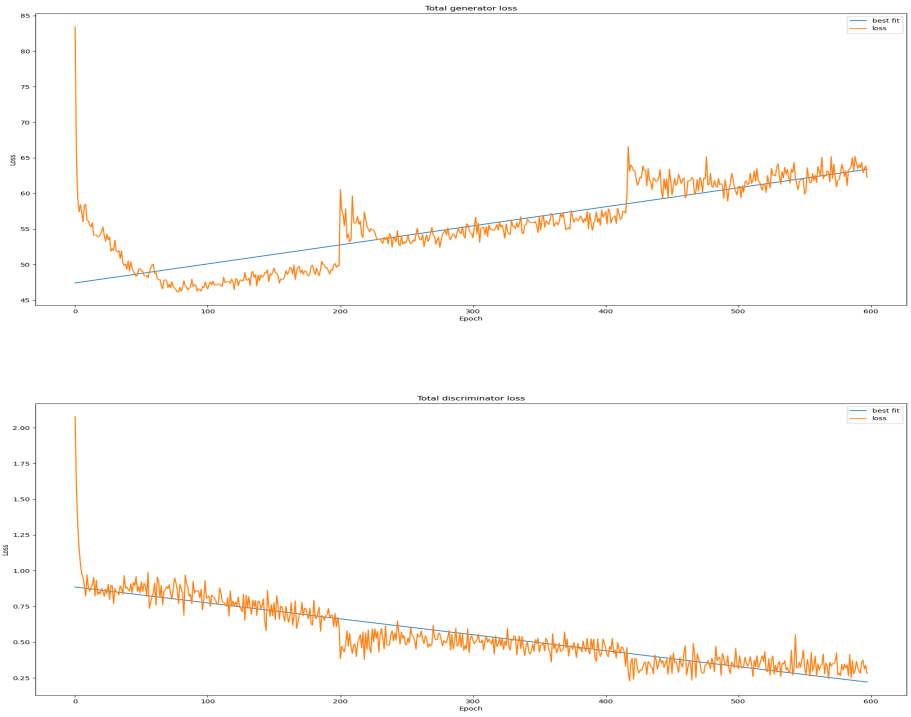

Figure 4.6. Generator (top) and discriminator (bottom) losses of RaGAN V0.

the disadvantage that the model has to spend a few epochs for re-obtaining balance and continuing its actual training. Indeed, around epoch 200 and 400 we can see these spikes of loss. In theory this should not happen, as the model is saved in its entirety and so it should be loaded back up exactly as it was left off. The reason why in practice this does not happen is given by an implementation detail that is not specified in the Attention GAN paper. Each encoder once loaded back up sees its weights being re initialized in a small range at random. This concept makes sense at the beginning, because it always ensures a new initial random state but definitely not when we restart the training from a checkpoint. Unfortunately, finding this reason took way longer than expected and in practice, I only managed to find it after both RaGAN v0 and v1 have been trained. At the time I was told that 2000 more credits would have been added to my account so I could have run these two experiments correcting this issue and running some more. However due to the expiration of my student account at KU Leuven I could not do

any of that.
It is interesting to take a closer look at the generators loss components and see if this increasing trend that we saw in Figure 4.6 is caused by the similarity loss or what else.

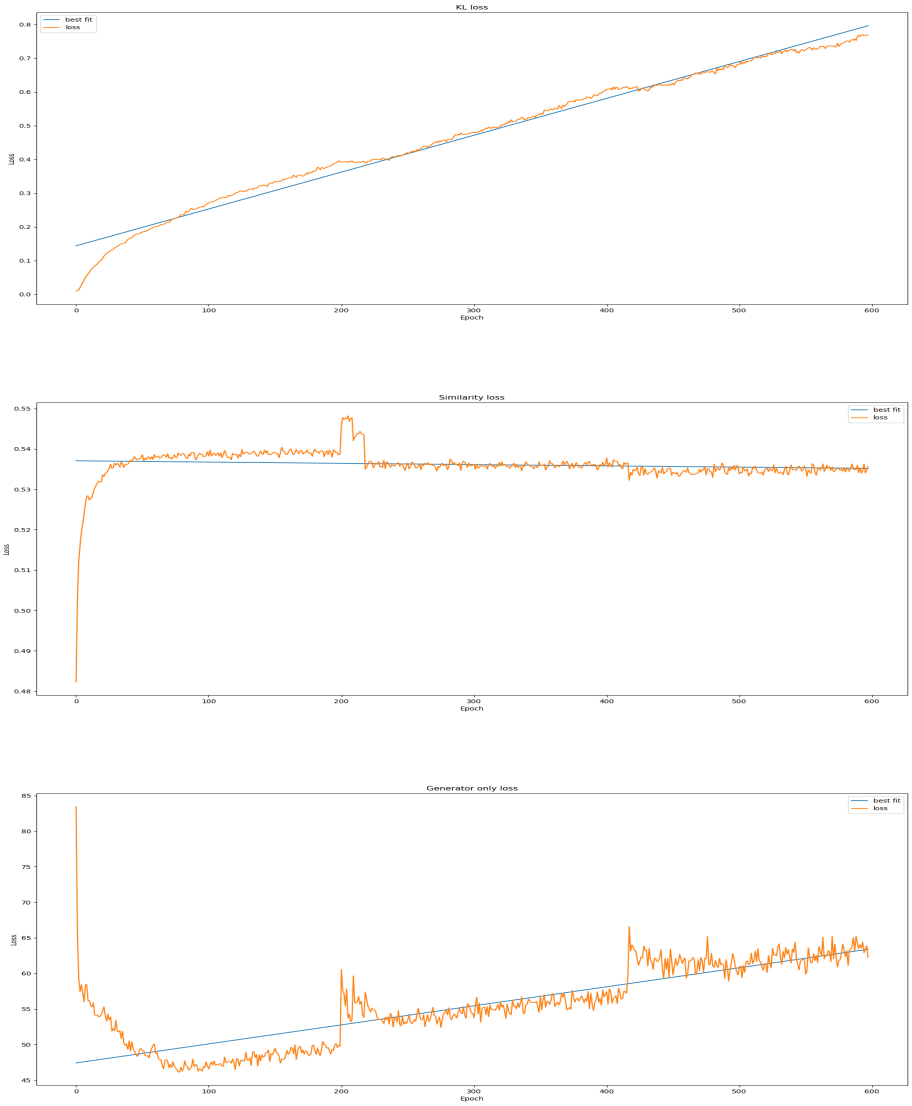

Figure 4.7. Generator loss disassembled into its components: KL loss (top), Similarity loss (center) and generators only loss (bottom).

The KL loss in Figure 4.7 is actually a measure of the distance of two

distributions, the input sentence distribution and the output sampled distribution. For the reasons explained in chapter three, this loss does have to slowly increase because it means that the sampling is offering more and more variance to the generator network. The similarity loss ideally has to decrease because it would mean that the generators are getting better and better at using the reference image in their synthetic pictures. However, even if it is technically decreasing, the level at which it does so is underwhelming. As we can see from the best fit, the decrease is so subtle that it is almost not noticeable. In any case these two pieces of the total generator loss are nothing in comparison with the loss produced by the generators themselves in terms of magnitude. Therefore, this increasing trend and its magnitude is mainly given by the generators, while the similarity loss has little to none guilt at this.
During training, I made sure to save training data results just like it was done for AttentionGAN. The images presented in Figure 4.8 and 4.9 use the following input sentences:

- The bird has a white belly with blue wings and head.
- This bird has a brown bill and crown and a white breast and belly.
- This small bird is brown and has a small brown beak.

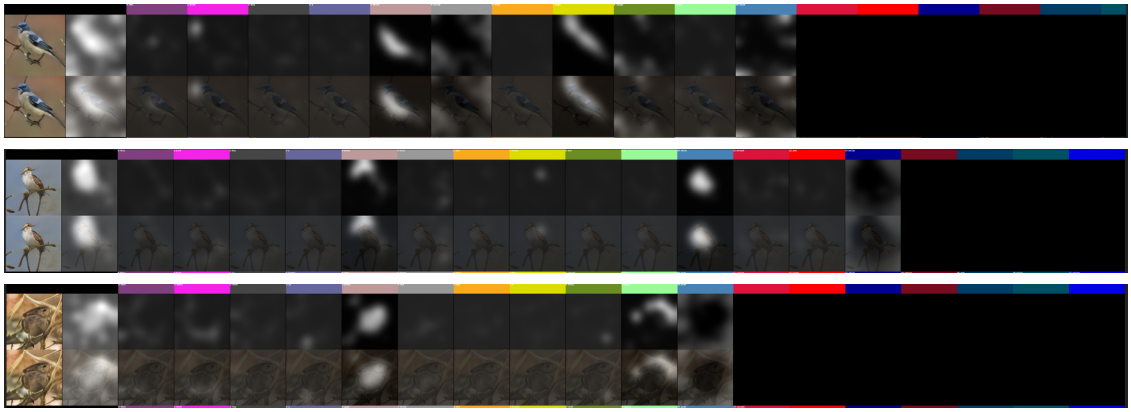

Figure 4.8. RaGAN output images with input sentences.

These are are a few examples of the output results the model achieves at the last training epochs (597-600). The final images are crisp, and realistic, keeping a strong correlation with the input text as we can see from the visual

results but also from the attention maps for each input word. In addition to this, now that the model has also all the blocks related to the reference image, I implemented a function that returned the generated image at training time next to the attention maps. Actually the attention map presented here is an average attention of all the maps produced by the model. This is caused by the architecture of the inception model: we consider 289 regions (17x17) and we are comparing them with as many regions extracted from the reference image so, instead of presenting them all I decided to take the average and show it next to the output image. I also showed one map at random to present how one attention map looks like.

The key note to take from Figure 4.9 is that at least in these three examples, the model takes particular attention on the bird shape as it is highlighted by the fact that the average attention maps are brighter in the area covered by the birds. In particular, in all three examples, extra attention (even brighter area) is given to the head of each bird, more specifically where the model has to represents their eye which is quite similar to the one given in the reference image. Generating an eye starting from a reference image of an eye is something the model is particularly good at and this trend will follow us in other sections of this chapter.
In order to push even further the concept of attention with respect to the reference image, I also modified the generator loss by adding the conditioned term as explained in chapter three. By doing this, the training behaviour changes a bit:

Even from the global perspective of Figure 4.10 we can already see a big difference in behaviour with respect to the previous version. If in the first case the generator loss had a first period in which it actually decreased, namely for the first 90 epochs, in this second case the total generator loss increases since the very beginning. The average loss value is also higher but this can be expected: we are adding additional terms to generators loss functions. Therefore it is normal that the total loss will have on average higher values. The graph of the discriminator loss is surprisingly quite similar to the one presented in Figure 4.6. This may be explained by the fact that this variation of the loss function only causes the generator to struggle more because it adds an additional constraint but the discriminators do not seem to see any worsening or improvement in quality.
Just like before, it is interesting to notice the various components that constitute the generator loss.

In this second scenario, what is important to notice is the fact that the similarity loss does suffer from this training in steps. At epoch 200 and

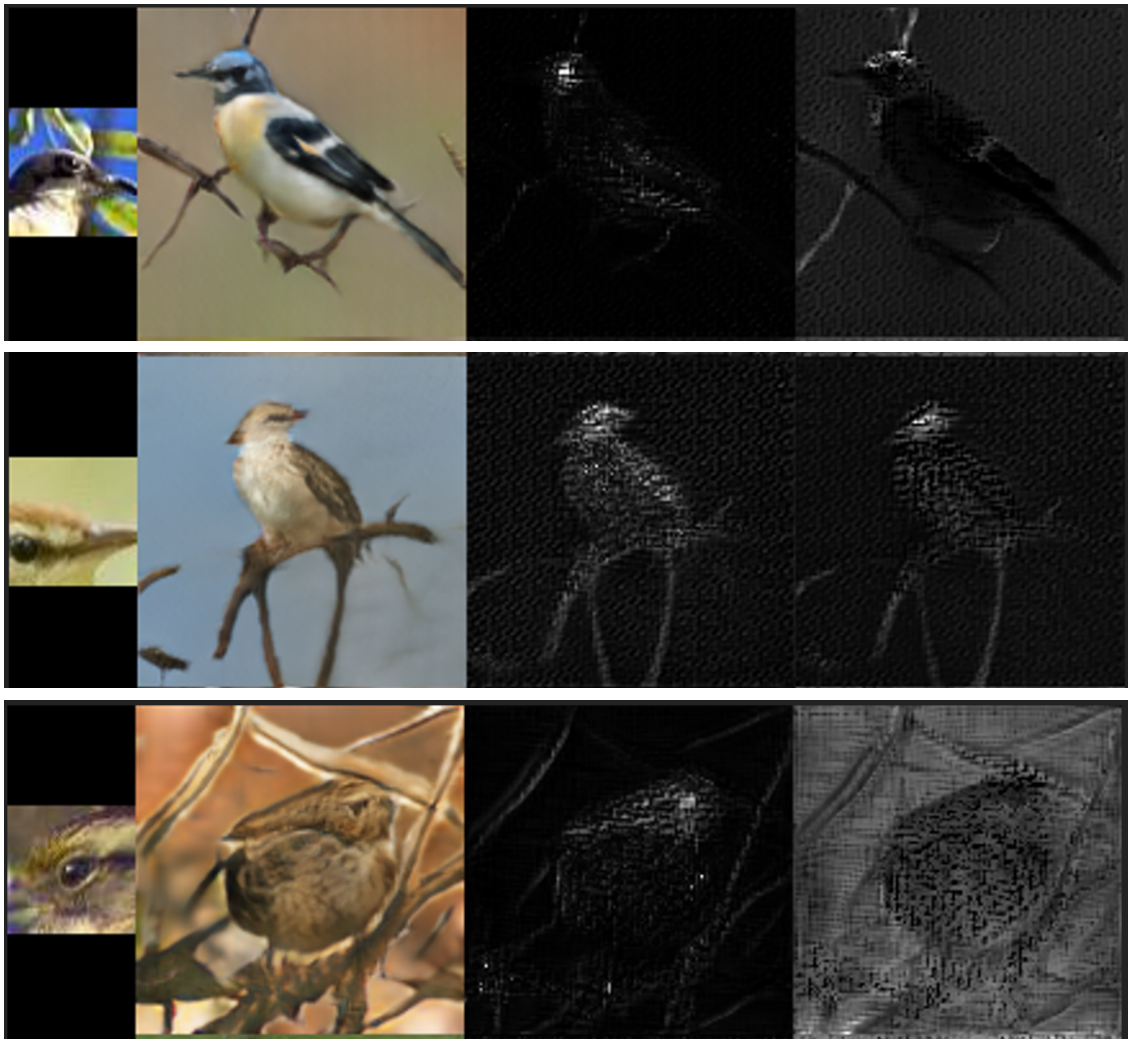

Figure 4.9. RaGAN reference image attention maps at epoch 600. Input reference image (left), output image (second left), average attention map (second right) and random attention map (right).

400 we can see the aforementioned spike in conjunction with the restart of the training that causes the loss to drop a bit in an unnatural way. This behaviour is definitely detrimental in the learning process, as it destabilizes the model.
Just like before it is important to show some attention maps associated with this second iteration of the model. In figure 4.12 and 4.13 we can appreciate some output images at training time next to attention maps related to input

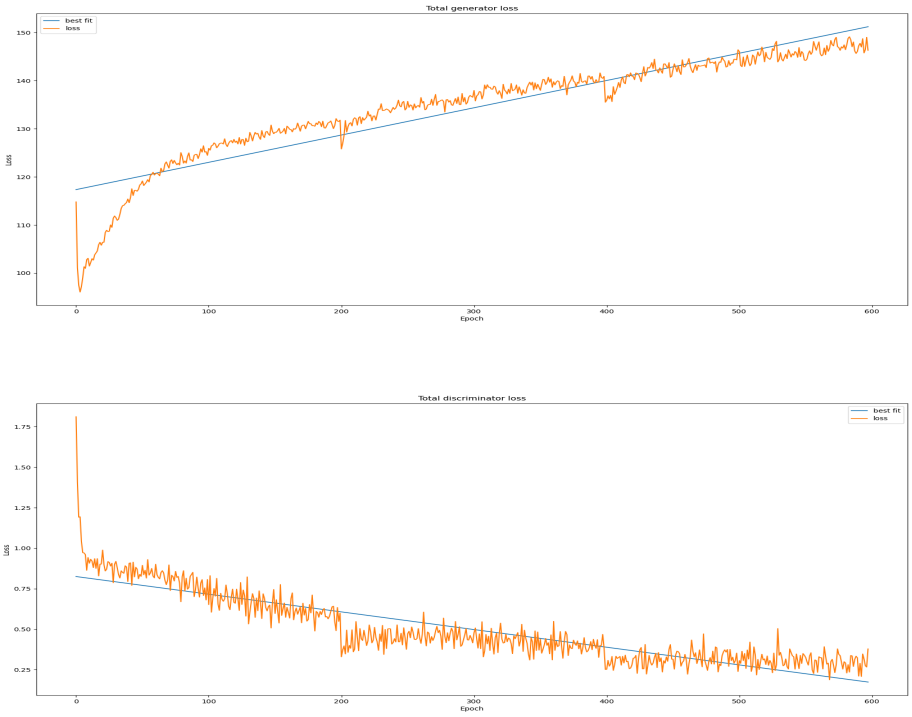

Figure 4.10. Generator (top) and discriminator (bottom) losses of RaGAN V1.

words and reference images respectively. For these examples the input texts were the following:

- A small yellow bird with a white rump and tail feather.
- A white chested bird with a blue head and a head proportional to the body.
- Small with small yellow. It is olive brown on its back. Its crown is black with cream color.

It is definitely worth noting the average attention maps generated with respect to the reference image. In the first case, it depicts an eye and indeed the most illuminated portion of the attention map is around the face of the bird. In the second case, a more generic reference image is given and for this reason the attention is quite generic, though mostly concentrated within the bird silhouette meaning that it still depicts something that should be

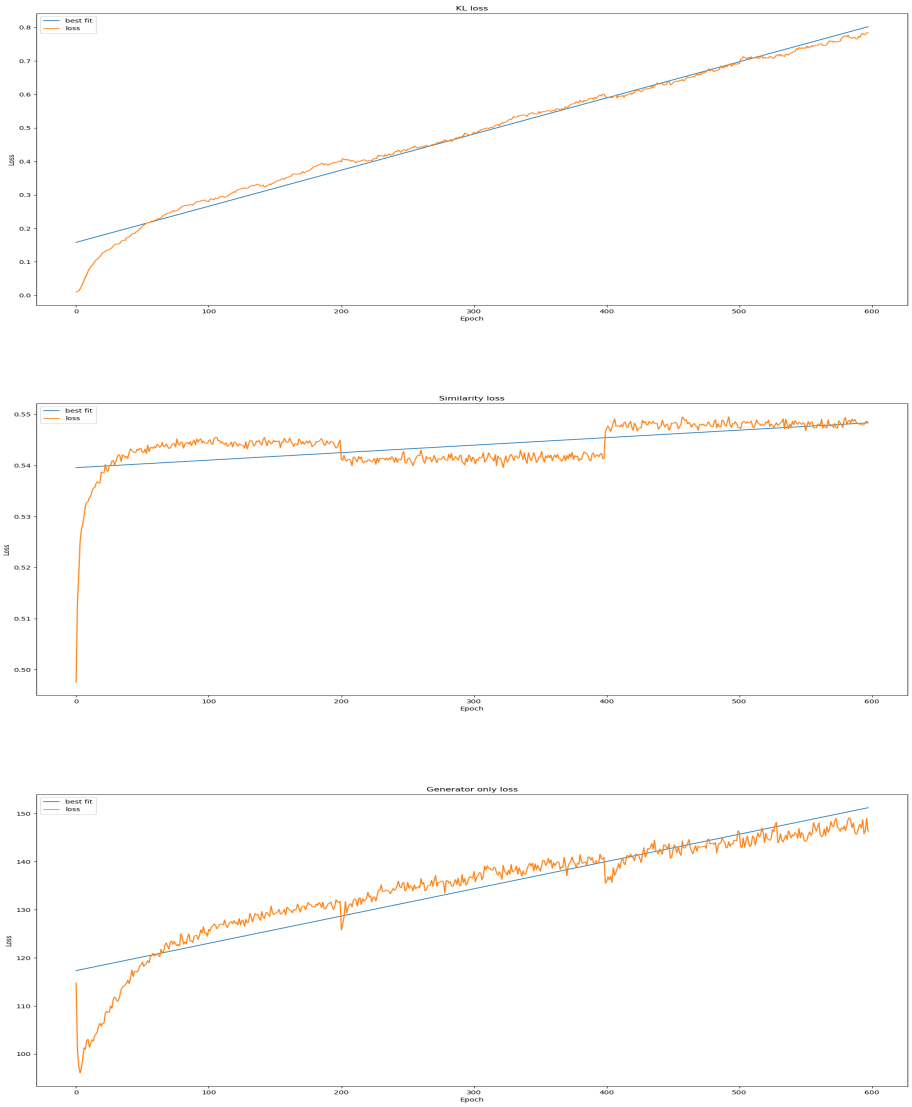

Figure 4.11. Generator loss disassembled into its components: KL loss (top), Similarity loss (center) and generators only loss (bottom).

used for the bird. The same goes for the third image where probably the only thing captured by the model is the fact that there is a strong edge that divides the background, probably some grass to the bird itself. Indeed the average attention map is highlighted specifically where we have an edge between wings and body and beak and background.

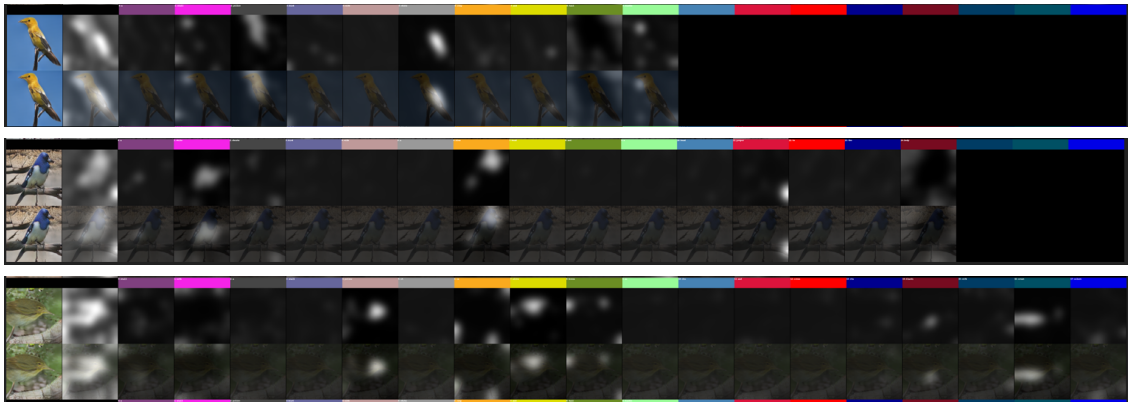

Figure 4.12. Output image samples next to text attention maps.

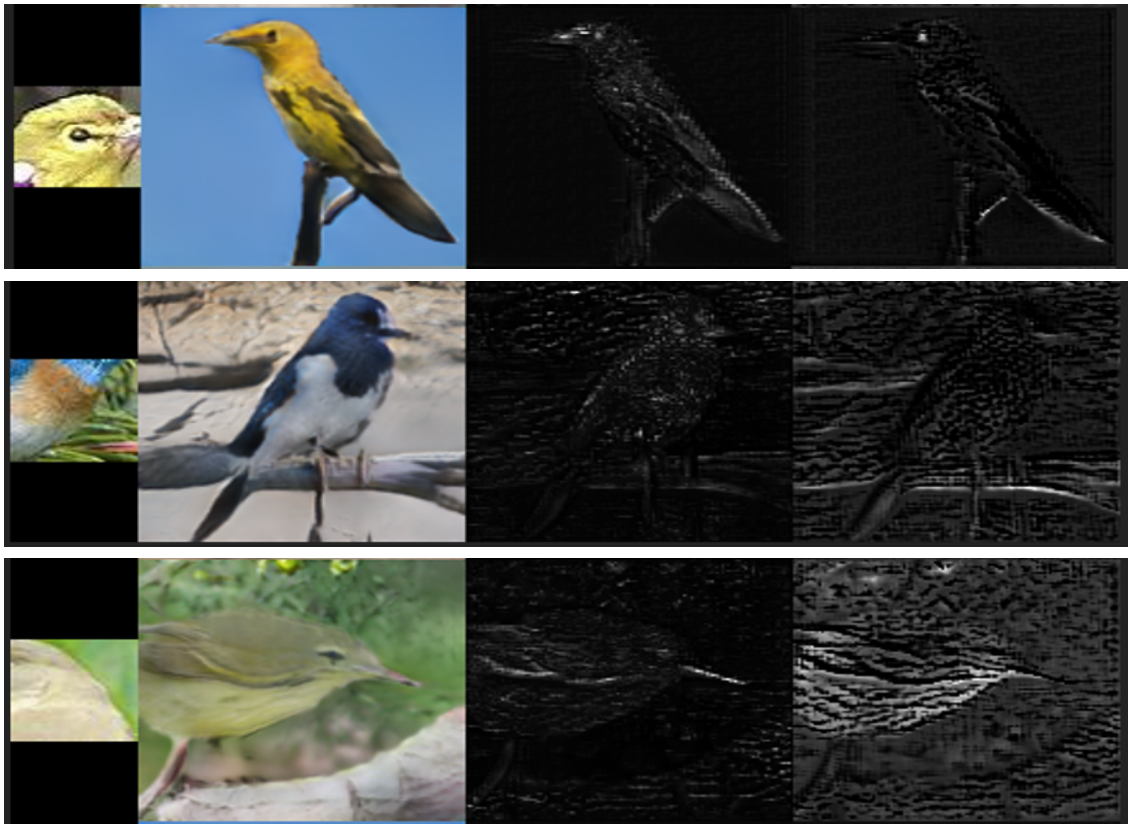

Figure 4.13. Output image samples next to reference images and attention maps.

4.5 Qualitative results

Regardless how the model behaves at training time, it is important to see what is able to produce once it has obtained knowledge.
As we saw before, Attention GAN returns good quality images in line with those presented on paper. It is however important to show some examples that do not clearly represent a bird. These examples are missing from the publication of Xu et al. but they are indeed a true part of the output results and need to be shown. As we can see in Figure 4.14, some animals are simply unreal, with body parts positioned in the wrong regions. Others are instead generated too close to the image frame, hiding some portions of the bird from the view.

Figure 4.14. Attention GAN bad quality output examples.

With respect to the two trained RaGAN models the results are quite similar. First of all, it is important to state that for some reason, that my supervisors and I have rooted down to a possible implementation bug, the setting on which we obtain the most realistic images is train mode, with gradient computation set to off of all model weights that can have a gradient. This means that layers like batch normalization and drop out are still working in training mode. However, what works is what actually gets used so even if it is not the canonical way of making run a trained deep learning model, I went ahead with this setting. Figure 4.15 and 4.16 at the end of this chapter show some output examples of the model trained without and with the additional conditioned loss term, presented in a customized fashion together with input sentence and reference image.

The image quality is overall qualitatively lower than the pictures obtained with the original model. Borders and in general the whole silhouette of each

bird is less crisp and actually almost blended with the background. In addition, it is quite hard to check whether or not the reference image has been used at all with the naked eye. Clearly though, there is a big difference in output results between what the model produces at training time and what happens at evaluation time. I tried to dive a bit deeper on the reason why this gap in quality happens, however I cannot seem to find a real motivation aside from an overfitting that the generator does at training time. It is possible that the model has discovered how to produce a small set of higher quality images (maybe one for each bird race) and that uses those for fooling the discriminator. Indeed, improved versions of GAN (like WGAN, that I will discuss briefly in chapter five) do train the discriminator more than the generator in order to make sure that this type of overfitting does not occur. Another possible and definitely more probable reason is the "short" training sessions I make the model go through. This task is more complicated than text to image generation so I should have given more training time in order to let the model better figure out how to deal with multiple input sources. In addition, as we saw, some epochs are not used for actually training the model but simply for returning to the training level right before the stop, in conjunction with epoch 200 and 400. So in practice I am not actually using all 600 epochs but a bit less.
Comparing the two RaGAN versions to one another, it is interesting to observe a variation in quality between the two. Even thought this cannot be really appreciated in the presented examples, which have specifically been chosen as better quality examples, the second version of the model generates lower quality images, often quite hard to understand, with colors that are simply off. The added conditioned loss term probably makes the problem even harder because it adds an additional constraint for the generator to endure.
Another way to compare the two models that I think is interesting, is to give as input a black reference image, that is, an image that does not contain information, and see how they behave. This not only allows for a sort of ablation study of each version but it also shows where and if the reference image is actually used. In order to do this, I created a 3x50x50 matrix with all values set to 0 that I passed as reference image. With that done I went ahead and run the model just like before. In this case, because we are interested in seeing how and where the reference image is used to produce the final output, I decided to also present the average attention map next to the output in order to better appreciate the most attended regions. Figure 4.17 and 4.18 show the results in this scenario for both RaGAN v0 and v1

respectively. It is interesting to notice something that I will go more in depth in the last chapter that holds true for the second version of the model: based on the attention maps we can see that the reference image is particularly used in the regions containing beaks and eyes. So even though the content of the input image is none, the model still tries to use it for generating these portions of the final images. This result is particularly interesting because it enforces the theory of having the model perform better on generating specific and clearly discriminated bird body parts.
Regarding RaGAN v0, the average attention maps have less of a focus than previously explained. This behaviour can be appreciated not only the ones presented in 4.17 but in many other examples that I managed to obtain from the validation set. Even though they look brighter in some regions more so than others but they are not that specific and in general look a bit more messy than in the second version of the model. This could be an indicator that the conditioned loss term I added is actually important to concentrate the information coming from the reference image to important and focused regions of the final image.

4.6 Quantitative results

The way Xu et al. evaluate the output quality is by means of inception score. However, this score does not consider how much of the reference image has been used for instance, how closely related input and output images are. In addition, I think that the inception value alone cannot really state whether or not the image quality of RaGAN is comparable to the one of the original model. Multi-modal image generation is arguably a harder task than text to image generation because, as we saw, the model also has to deal with the fact that the input is given in two different forms, hence, it has to first figure out what information to take from each source and then how to use it properly. With that being said, it is reasonable to consider good lower inception scores, in order to cope with the intrinsic difficulty RaGAN has to deal with.
By using the same model for computing the inception score when evaluating AttnGAN I produced the results visible in Table 4.3. As we can see, the results show a lower score for RaGAN if compared to AttnGAN due to the fact that the image quality is overall worse for my model and this probably affects the image diversity as different species of birds tend to look quite similar to one another due to the less prominent details. It is however surprising how version 0 is better then version 1 score wise. As we saw, the qualitative

results are indeed quite similar though slightly better on the second case which would suggest an opposite trend of scores. This inversion could be explained by the fact that maybe RaGAN V1 creates more defined images but that are overall more similar. It could be possible that the second model has learned slightly better how to create details, maybe due to the fact that has to take more consideration over reference image input, thus presenting better quality results from a qualitative point of view but when processed by the classifier adopted for computing the metric, the extracted features do not consider as much this difference in details over the general appearance therefore resulting as more similar.

	Inception score
AttnGAN	4.27 ± 0.20
RaGAN V0	3.51 ± 0.11
RaGAN V1	3.20 ± 0.10

Table 4.2. Inception Scores of each model in comparison.

I would like to let the reader notice that the presented values have a variance. This is because in this case I managed to run the evaluation locally, even if they took up to one hour to complete each time, and so I obtained more values that I could compare in a more scientific way, which is something that I simply could not do when training my model. Though, as I will discuss in chapter five, even if I had more resources to do so I would have prioritized other experiments.
As I was explaining, inception score cannot say anything about the relationship between input reference image and output. It also cannot say anything about the correlation that there is with respect to the input text. In the AttnGAN paper they try to measure the latter case using R-precision, a metric commonly used in the context of information retrieval. The idea is to check the percentage of relevant documents r in the top R ranked ones. The R-precision is then computed for each query as:

$$R_{precision} = \frac{r}{R} \tag{4.1}$$

In our particular case the query is the output image, encoded into a vector that contains the global features using the DAMSM image encoder. The documents to retrieve are the sentence feature vectors of a list of one hundred text sentences, where one is the ground truth and the other 99 are randomly

selected mismatched descriptions. By means of cosine similarity they measure the similarity between the query vector, the image, and the document vectors, the texts. Each text is then ranked so that the most similar ones are positioned at the top and then the position of the ground truth in this classification is noted. Ideally for each query the associated text should always be ranked first giving a total R-precision of 1. However in real scenarios the ground truth document is not the first, meaning that the R-precision of that particular query will be lower than 1 and so will the total R-precision. With this concept of metric in mind it could be possible to evaluate the relationship between reference and output image. However, as I mentioned when describing the reason why I have not trained the reference image encoder in conjunction with the others, the global feature vectors that I can extract from both type of images would not retain the same information. In one case the reference image features describe regions of an image that defines a region of what will become the final picture while in the other we are embedding the whole picture itself. Therefore doing this type of information retrieval experiment would be like asking to retrieve all the possible reference images associated with the output, as one output ideally could contain similarities with more than one reference image. I try to give an example to better explain this. Suppose that we have a perfect output, with a crisp and sharp picture of a bird. When we extract the global features from this picture we can expect the feature vector to retain information of each region of the image, hence, all the visible bird parts. Still in an ideal case this would mean that we could find strong similarities with multiple reference images because they can represent all the possible body parts. Therefore we would have multiple very high similarity scores that would cause an artificially lower R-precision score simply because of the false similarities that have been found. All of this taking for granted that it is possible to compare feature vectors of these two different types of images because again, one is a global image, the output, while the other is local, the reference. One way to solve this is to ensure that the reference images used as documents are representations of the same body part, so if the ground truth image is a beak then all the other mismatched images have to contain a beak. Nevertheless, the problem of comparing these two semantically different global feature vectors would still remain.
Finally I try here to discuss about a more modern and arguably better evaluation metric that does not appear in the work of Xu et al. simply because it was published in the very same year [13]. Heusel et al. propose Fréchet

inception distance, or FID score, a novel way to define the goodness of synthetic images generated with GAN models that consists of comparing the synthetic and real data distributions. This metric claims to *capture the similarity of generated images to real ones better than the Inception Score* and it has the advantage of comparing the output data with statistics of real data, which is something that is missing from the inception score. Still citing from this math intense paper the metric is computed as a Wasserstein-2 distance between two Gaussian distributions, namely the one of the synthetic data and the one of the real data. In practice a pretrained inception v3 model is used to encode real data and synthetic data and we extract mean vector and co-variance matrix of the last layers of this encoder for each group of data obtaining $\mathcal{N}(\mu, \mathbf{E})$ and $\mathcal{N}(\mu_r, \mathbf{E}_r)$, with foot note r to state the real data distribution. These Gaussian distributions are compared as follows:

$$FID = ||\mu - \mu_r||_2^2 + tr(\mathbf{E} + \mathbf{E}_r - 2(\mathbf{E}\mathbf{E}_r)^{\frac{1}{2}})^2 \tag{4.2}$$

In order to actually compute this score, I used an implemented version available to download [34] that takes two folders containing two different image datasets, one containing real validation data and the other containing the output of whichever model, and returns the score. After resizing the real images to a standard value and fixing a few bugs that did not allow the script to run properly I computed the following FID scores:

	FID score
AttnGAN	27.13 ± 0.45
RaGAN V0	29.69 ± 0.16
RaGAN V1	35.38 ± 0.23

Table 4.3. FID Scores of each model in comparison.

Once again these scores do not define the relationship between input and output but they arguably better define the image quality that each model is able to produce. In order to interpret these results we have to keep in mind that FID scores define a distance, therefore the lower the value is the better quality of images it represents. As we can see AttnGAN manages to obtain a better score that matches the image quality that is capable of synthesize. In addition, even with this metric we can observe the same behaviour between

[2]tr: trace linear algebra operation.

RaGAN V0 and V1: the first version achieves a better score which is arguably the opposite of what we could expect since the addition of the conditioned loss term should in theory improve the image results. However, for both metrics, we have to keep in mind that the training process was actually performed only once for each model, thus these results should be interpreted with care, since most deep learning models are subject to significant stochasticity due to random initialization, random batches and so on. GAN models especially are known to be unstable which translates to high variance on results. A more solid discussion could be made if the training session would have been carried multiple times with different random seeds and average over the results. It is very well possible that the presented scores would be quite different if I did more runs for each of the settings.

Figure 4.15. RaGAN output without using the conditioned loss term.

Figure 4.16. RaGAN output using the conditioned loss term.

Figure 4.17. RaGAN v0 output using a black reference image.

Figure 4.18. RaGAN v1 output using a black reference image.

Chapter 5

Going further

5.1 Issues and limitations

It is definitely worth noting that the model really struggles when the input text and reference image contradict each other. The example in Figure 5.1, taken from both versions, tries to highlight this issue. Whenever the input reference image colors do not match the ones we are asking the model to generate in the text, we get bad pictures not only in terms of shapes but also in terms of pigments used.

Figure 5.1. Contradicting inputs results. RaGAN v1 top and v2 bottom.

Another thing to highlight is that the image quality really drops if the input reference image has an unclear content. If for example the visual input contains background cropped from the original image, feathers without a clear definition of which part of the body we are referring to or similar situations where what is given is actually not informative, then the result will be poor.

On the other hand, if the reference image is not ambiguous, it seems that

Figure 5.2. Non informative reference image results. RaGAN v1 top and v2 bottom.

the model is more capable of making good use off of it. In particular, I found out that reference images of beaks and eyes are associated with better quality images that actually resemble features from this input.

Figure 5.3. Output when using reference images of beaks. RaGAN v1 top and v2 bottom.

The problem, and indeed the biggest limitation of the model is that through the reference image channel too much information is passed. The models sees coming from here very different images, whether it is some water or a beak or feathers or branches and tries to makes sense out of it, clearly not managing to do so. Probably, if I had to go back in time and redo this project I would limit myself to use only a subset of bird parts and then maybe adding more and more.

5.2 Further work

I do not consider this work a model that is not able to evolve in the future. Indeed there are many sections of it that, given more time and more importantly more computational resources I would have liked to improve myself. There are quite a few experiments that I simply could not try due to the lack of credits in the VSC supercomputer. Among those there is of course the entire section related to parameter tuning. As mentioned in chapter three, I originally decided to incorporate my custom similarity loss using a weighting term λ_2 in 3.10 that should balance it with the rest of the terms and should also, whenever and if needed, give more or less importance to this portion of the function responsible for defining how much the reference image to consider and where. Technically, the only value I tried for this parameter is 1 but I would have liked to try values like 2, 5 and 10, in order to see if increasing this portion of the loss would have caused some changes in the quality of the output and in the usage of the reference image itself. In practice, I would have tried the model for 600 epochs from scratch and checked the results and the total loss graph. Speaking of number of epochs to use, as I presented in this work I decided to stick with 600 just like they do in Attention GAN. However, multi-source image generation is arguably a more complicated task than text to image generation. Therefore a longer training session would have possibly enhanced the quality of the synthetic images as the model would have had more time to learn and "fit" this more complex scenario. Due to the aforementioned limitations I preferred to limit myself to at least 600 epochs in order to ensure a level of training in line with the original model.
In chapter three I discussed how I added my custom blocks to what was already present and how I tried to do it in a smooth, harmonic way. This line of work was pushed by the fact that GAN architectures are pretty famous for being quite hard to train, often facing problems with divergence [14]. In reality I started implementing a slightly different version of the architecture called WGAN following a tutorial shared with my by my thesis supervisors [5]. WGAN is a variant of the canonical GAN architecture that claims to be more stable and less prone to divergence. It uses a few tricks to get the job done like a linear activation function instead of sigmoid, different labels, different loss and so on. However, if simple to implement on paper, I found it quite difficult to update my code in order to have it function properly. For this reason, and because there was no insurance of this technique to work as advertised, I preferred to stop its implementation in favor of not changing

too much the architecture itself unless necessary for fitting the new input and attention vectors.
In many papers there is often a section related to the so called "ablation studies". These studies consists of removing a portion of the model and see how it reacts to it, in terms of quantitative and qualitative results. In complex models that behave like a black box, and deep learning models are quite good at this, it is often hard to determine what is used for what and why everything works or not. By doing this we can study better the behaviour of each component. In my case I wanted to remove the F_{ref} blocks and train the model without adding the attention mechanism, leaving RaGAN working only with the reference image and nothing more (aside from the input text and everything related to it). The way I wrote the code actually already allows for this experiment: by simply toggling the value of a flag variable it is possible to deactivate these blocks all together. Unfortunately the lack of credits shut the experiment down.
Another thing that could be studied would be how the model behaves if multiple reference images were given at once. Feeding 5 images as input for example could be a matter of encoding those with the reference image encoder, returning for each local and global features. Then summing up the global features and passing that single vector as input and using the region features in the reference blocks each returning multiple attention vectors that could be summed up and normalized. Optionally it could be possible to leave all these vectors separated, creating multiple reference blocks each responsible of one specific part of the birds body. This could be a solution for overcoming the aforementioned limitation of having one single "path" of the reference image features regardless the type of body part given as input.
Finally, it would be interesting to investigate if this model could work with a different text encoder trained on Italian input sentences. In theory, a pretrained bidirectional LSTM model that has only seen Italian sentences could be simply swapped with the current one, trained in conjunction with the final image encoder during the DAMSM training and in theory could still allow for a working RaGAN model capable of receiving Italian text as input.

5.3 Conclusions

The study behind the implementation of RaGAN has been a journey that I carried in a time span of almost one year. Throughout this period I discovered many works related to synthetic image generation and the GAN architecture,

learning about the difficulties and achievements that this field of research has brought to the table of deep learning models. Even if I have ultimately took the implementation decisions behind the work I presented, I have to admit that some issues and limitations have clearly showed up. In addition, the limited resources left me with a fervent desire of implementing and testing many things that unfortunately never saw light. Nonetheless, I hope to have left the reader with some new ideas, suggestions and things to reflect on.
With this work I tried to add another perspective to the problem of image generation and possibly another hint on a new direction this field could go to. I do believe that multi-source image generation using deep learning is possible but that requires a lot of computational power to be fully studied. Further experiments, parameter tuning, and a longer training phase are in my opinion what could really push this work further, and I hope someone in the future will decide to take this work from where I left off, enhancing it.

Bibliography

[1] Neptune.ai official site. https://neptune.ai/. Accessed: 2021-10-21.

[2] Pycharm inceptionv3 repository. https://download.pytorch.org/models/inception_v3_google-1a9a5a14.pth.

[3] Vsc - flamish super computer centrum. https://www.vscentrum.be/. Accessed: 2021-10-21.

[4] Winscp official download site. https://winscp.net/eng/download.php. Accessed: 2021-10-21.

[5] Jason Brownlee. How to develop a wasserstein generative adversarial network (wgan) from scratch. https://machinelearningmastery.com/how-to-code-a-wasserstein-generative-adversarial-network-wgan-from-scrat Accessed: 2021-10-25.

[6] Guillem Collell, Luc Van Gool, and Marie-Francine Moens. Acquiring common sense spatial knowledge through implicit spatial templates, 2020.

[7] Guillem Collell and Marie-Francine Moens. Learning representations specialized in spatial knowledge: Leveraging language and vision. *Transactions of the Association for Computational Linguistics*, 6, 2018.

[8] Google developers page. Training gan networks. https://developers.google.com/machine-learning/gan/training. Accessed: 2021-10-05.

[9] Politecnico di Torino. Machine learning and deep learning information page. https://didattica.polito.it/pls/portal30/gap.pkg_guide.viewGap?p_cod_ins=01TXFSM&p_a_acc=2021&p_header=S&p_lang=EN. Accessed: 2021-10-14.

[10] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[11] Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra. Draw: A recurrent neural network for image generation,

2015.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition, 2015.

[13] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium, 2018.

[14] Jonathan Hui. Gan — why it is so hard to train generative adversarial networks! https://jonathan-hui.medium.com/gan-why-it-is-so-hard-to-train-generative-advisory-networks-819a86b3750b Accessed: 2021-10-25.

[15] Justin Johnson, Agrim Gupta, and Li Fei-Fei. Image generation from scene graphs, 2018.

[16] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2014.

[17] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. 2017.

[18] Tejas D. Kulkarni, Will Whitney, Pushmeet Kohli, and Joshua B. Tenenbaum. Deep convolutional inverse graphics network, 2015.

[19] KU Leuven. Information retrieval and search engines information page. https://onderwijsaanbod.kuleuven.be/syllabi/e/H02C8AE.htm#activetab=doelstellingen_idp1981296. Accessed: 2021-10-14.

[20] KU Leuven. Natural language processing course at ku leuven. https://onderwijsaanbod.kuleuven.be/syllabi/e/H02B1AE.htm#activetab=doelstellingen_idp73184. Accessed: 2021-10-14.

[21] Mike Lewis, Marjan Ghazvininejad, Gargi Ghosh, Armen Aghajanyan, Sida Wang, and Luke Zettlemoyer. Pre-training via paraphrasing, 2020.

[22] Bowen Li, Xiaojuan Qi, Thomas Lukasiewicz, and Philip H. S. Torr. Manigan: Text-guided image manipulation, 2020.

[23] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft coco: Common objects in context, 2015.

[24] Xihui Liu, Guojun Yin, Jing Shao, Xiaogang Wang, and Hongsheng Li. Learning to predict layout-to-image conditional convolutions for semantic image synthesis, 2020.

[25] Danquing Lui. A practical guide to relu. https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7. Accessed: 2021-10-14.

[26] David Mack. A simple explanation of the inception score. https://medium.com/octavian-ai/a-simple-explanation-of-the-inception-score-372dff6a8c7a. Accessed: 2021-10-29.

[27] Tiago Miguel. How the lstm improves the rnn. https://towardsdatascience.com/how-the-lstm-improves-the-rnn-1ef156b75121. Accessed: 2021-10-14.

[28] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization, 2019.

[29] Bharath Raj. A simple guide to the versions of the inception network. https://towardsdatascience.com/a-simple-guide-to-the-versions-of-the-inception-network-7fc52b863202. Accessed: 2021-10-14.

[30] Scott Reed, Zeynep Akata, Santosh Mohan, Samuel Tenka, Bernt Schiele, and Honglak Lee. Learning what and where to draw, 2016.

[31] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[32] Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans, 2016.

[33] Ian Sample. What are deepfakes – and how can you spot them? https://www.theguardian.com/technology/2020/jan/13/what-are-deepfakes-and-how-can-you-spot-them. Accessed: 2021-10-07.

[34] Maximilian Seitzer. pytorch-fid: FID Score for PyTorch. https://github.com/mseitzer/pytorch-fid, August 2020. Version 0.2.1.

[35] Tristan Sylvain, Pengchuan Zhang, Yoshua Bengio, R Devon Hjelm, and Shikhar Sharma. Object-centric image generation from layouts, 2020.

[36] Bryan Tan. Generate novel artistic artworks with deep learning. https://towardsdatascience.com/generate-novel-artistic-artworks-with-deep-learning-f2f61da69e6e. Accessed: 2021-10-07.

[37] Qiuyuan Huang Han Zhang Zhe Gan Xiaolei Huang Xiaodong He Tao Xu, Pengchuan Zhang. Attention gan official git repository. https://github.com/taoxugit/AttnGAN. Accessed: 2021-10-21.

[38] THINKML TEAM. Cpu vs gpu in machine learning algorithms: Which is better? https://thinkml.ai/cpu-vs-gpu-in-machine-learning-algorithms-which-is-better/. Accessed: 2021-10-21.

[39] Anjie Tian and Lu Lu. Attentional generative adversarial networks with representativeness and diversity for generating text to realistic image. *IEEE Access*, PP:1–1, 01 2020.

[40] James Vincent. Artificial intelligence is helping old video games look like new. https://www.theverge.com/2019/4/18/18311287/ai-upscaling-algorithms-video-games-mods-modding-esrgan-gigapixel. Accessed: 2021-10-07.

[41] Antonia Vojtekova, Maggie Lieu, Ivan Valtchanov, Bruno Altieri, Lyndsay Old, Qifeng Chen, and Filip Hroch. Learning to denoise astronomical images with u-nets. *Monthly Notices of the Royal Astronomical Society*, 503(3):3204–3215, Nov 2020.

[42] P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona. Caltech-UCSD Birds 200. Technical Report CNS-TR-2010-001, California Institute of Technology, 2010.

[43] Wikipedia contributors. Kullback–leibler divergence — Wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Kullback%E2%80%93Leibler_divergence. Accessed: 2021-10-14.

[44] Papers with code. Leakyrelu. https://paperswithcode.com/method/leaky-relu. Accessed: 2021-10-14.

[45] Tao Xu, Pengchuan Zhang, Qiuyuan Huang, Han Zhang, Zhe Gan, Xiaolei Huang, and Xiaodong He. Attngan: Fine-grained text to image generation with attentional generative adversarial networks, 2017.

[46] Shaowei Yao and Xiaojun Wan. Multimodal transformer for multimodal machine translation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4346–4350, Online, July 2020. Association for Computational Linguistics.

[47] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, and Dimitris Metaxas. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks, 2017.