Synthesising Images from Text Descriptions

Sergi García Sarroca

Abstract

In the last years, image synthesis from text descriptions has become one of the most important research fields of computer vision. The aim of this field is to understand spatial relations between described objects and their position in the image, composing realistic images from these relationships. In this project, state-of-the-art text-to-image synthesis methods are studied and evaluated using the FashionGen dataset in order to study their behaviour, creating a fashion generator application as a result. We propose a text-to-image synthesis method to automatically generate clothing fashion images from natural language descriptions. The image generation is based on a Generative Adversarial Network trained using the FashionGen dataset, while the input based on natural language descriptions processed by a text encoder with an attention mechanism and the data fusion performed by the use of Conditional Batch Normalization in order to condition the image generation by text features. We purpose a new methodology as a benchmark for text-to-image synthesis by testing our method into three different GAN architectures and making it accessible and extendable to the rest of GANs by just a small change in their architectures.

Index Terms

Natural Language Processing, Generative Adversarial Networks, Image Generation, Computer Vision, Transformers, Multimodal, AI, Text-to-Image, Conditional Batch Normalization

I. Introduction

Humans perceive the world as a multimodal source of information that is constantly stimulating and conditioning our understanding, learning and the decisions and actions we take because of it. We can categorize this information in different channels of transmission, such as visual, audio, text, tactile... Each of the channels will be perceived by a different sense and then all the information will be processed by the brain, fusing the different sources of information to understand and learn the given stimulus and then being able to take a decision conditioned by it.

Similarly, Artificial Intelligence (AI) is the field of computing that aims to process digital information to perform different tasks. In AI, most of the developed projects are unimodal, that means they are trained for realizing tasks focused on one information channel i.e., object detection (vision), image classification (vision), speech recognition (audio), text descriptions (text), sentiment analysis (text)... During the evolution of AI we have seen how it outperforms humans when solving tasks such as classification, playing games, looking for defects, etc... but, it is still struggling when we try to solve multimodal tasks with it.

In the last decade, we have seen a raise into the different artificial intelligence (AI) fields. On the one hand, we find Computer Vision which is the field of AI in charge of extracting meaningful information from visual inputs such as images, videos, etc... This meaningful information is understood as the several patterns and different patterns that compose the inputs and how they are related in order to make sense for composing the images.

In 2014, Ian Goodfellow et. all presented *Generative Adversarial Networks* [3] (GANs), a paper describing an adversarial architecture that understands and learn information from a data distribution and generates new data based on that same distribution. This model was a benchmark and began a state-of-the-art on the image generation task by making possible to learn the features that compose an image data distribution and generating images that are similar to that distribution. To compute this task, GANs are built as two architectures, discriminator *D* and generator *G*, that will *"compete"* between them during training. Discriminator will be trained on maximizing the distance between real and fake images that will be synthesized by *G*. This allows the model to learn to distinguish between them. Meanwhile, we ask the generator to create a set of images from a random latent vector. These images will be passed along D and the result will be used to minimize the distance of the real and fake data on the generator. All in all, the training procedure is similar to a two-player game, where *player 2* draws images and *player 1* classify them between real and fake.

On the other hand, *Natural Language Processing* (NLP) is the field of AI that translates human language to machines, making they able to understand/process the information in it. The objective of NLP is not only to understand the meaning of each word from a given text, but to understand how these words relate and make sense to the text by understanding the context.

Author: Sergi García Sarroca, sergi.garciasa@e-campus.uab.cat
Advisor 1: David Vázquez, Computer Vision, Universitat Autonoma de Barcelona
Advisor 2 : Rafael Pardinas, ElementAI
Thesis dissertation submitted: July 2022

NLP is a challenging task that has been studied since the beginning of computing and AI. This is due to the fact that computers are not able to process characters and need a numerical representation of them in order to be processed. Knowing that, the first intuition of how to solve the problem is to manually label each of the characters with numbers to achieve this representation. If we extend this to a word level, we would need a person who labels each of the existing words. The same will happen when we want to represent phrases and the meaning of each word in every situation, and so on, so for... At this point, we can understand that compute this task is time and computationally expensive. To leverage with the number labelling task, several solutions were proposed, i.e., one-hot encoding.

One-hot encoding is a solution where categorical labels are transformed into vectors of zeros and ones. The length of these vectors is equal to the number of categories/words we want to encode. One-hot encoding allows us to represent each of the words as a position inside the one-hot vector and access it by its index.

In our paper, concatenation, will be useful to make the model understand which category correspond to which image and will also help the model to create clusters with more information than the image patterns extracted during the training process. The problem with this methodology is that when your data has a long number of categories the one-hot vectors became bigger and bigger and this will be computationally expensive and will lead to very long training process. Due to this fact, other methodologies were studied in order to extend the performance of the model from word conditioning to text conditioning, such as, tokenization mechanisms [9, 20, 19], and Conditional Batch Normalization [24].

Nowadays, recent progress in Deep Learning researches related to generative models from text descriptions provide new tools to designers to exploit their creativity, develop new ideas and helping to obtain start-points. In the world of art and design it is really common seen figures, pictures, products, etc... that are always influenced from previous works. Usually, these influences come from most liked works of certain old artists that new artist take into account. This helps artists and designers to have benchmarks to develop their new ideas, but it is still hard for them to create start-points from zero.

This paper presents a combination of generative adversarial networks with transformers in order to create a text-to-image synthesis tool that help designers to create faster new ideas.

II. State of the art

A. Related Models

Image generation and natural language processing has been highly researched in the last years. From these researches a hundred of papers with different experiments, have appeared, with different ways of training these models and obtaining results. In this section are going to present the most relevant previous works for our experiments.

1) DCGAN:

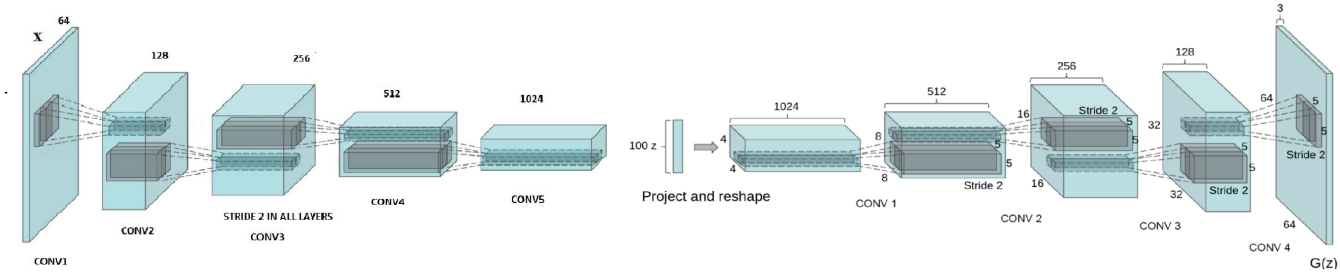


Fig. 1: Image representation of the architecture of DCGAN and its feature dimension evolution.

Discrete-GAN [2] fig. 1, is a reconditioning of the original GAN [3] implemented by Ian Goodfellow et. all, but in this case DCGAN applying convolutional layers to extract image features and convolutional-transpose layers in order to up-sample feature maps and convert them into 3 channel images. The discriminator will take as an input a batch of images with size 3x64x64 (same output as the generator) and by using convolutions will try to learn the pattern that compose the dataset. At every convolution the size of the images will be reduced, but the depth of the feature maps will increase.

The architecture of the Discriminator is assembled as:

- 4 blocks composed by:
  - Convolutions: that will compress the learned patterns into an increasing feature vector.
  - BatchNorm: As in the generator it will normalize the data.
  - Leacky ReLU: This is the activation function is usually used for nets where we can have spare gradients as GANs.
- 1 final block block with:
  - Convolution
  - Sigmoig: This activation function will output a probability determining if the image is real or not. (Binary output between 0 and 1).

The architecture of the Generator is assembled as:

- 4 blocks containing:
  - Transpose Convolution 2D: These convolutions will up-sample the input feature map to a desired dimension of feature maps. In the case of the first block, the input is the z = 100 (latent vector) which will be transformed into a feature map of $64 * 8 = 512$ feature map.
  - After the convolution, a BatchNorm layer is applied to the result. This will normalize the data by maintaining the mean output close to 0 and the output standard deviation close to 1.
  - Finally, ReLU activation function is used. This function will output the output directly if it is positive, otherwise it will output zero.
- 1 final block with:
  - Transpose Convolution.
  - Tanh activation function, this function will map the data between [-1, 1].

All in all, the Generator takes the latent vector z and maps it into the real distribution data, trying to fit 100 feature dimensions by using the learned weights in order to make them similar to the real data.

2) Wasserstein GAN: One of the problems of DCGAN [2] is that the loss function does not describe a useful output in order to understand if the model is learning properly how to generate images, and a visual inspection is needed to determine that the quality of the images is correct and the GAN actually converges. Therefore, research on how to train a GAN and actually understand if the training is converging was needed.

In WGAN [13] the authors' purpose a new way to achieve stabilization during the training. This means that the loss function now has a significant, and we can understand if the GAN is learning properly. This generative adversarial network minimizes an approximation of the Earth-Mover's distance (EM) that finally looks like,

$$\max_{\|f\|_L \leq 1} \mathbb{E}_{x \sim \mathbb{P}_r}[f(x)] - \mathbb{E}_{x \sim \mathbb{P}_\theta}[f(x)] \quad , \tag{1}$$

where $\mathbb{E}_{x \sim \mathbb{P}_r}[f(x)]$ is the distribution of the real data, $\mathbb{E}_{x \sim \mathbb{P}_\theta}[f(x)]$ is the distribution of the generated data and $\max_{\|f\|_L \leq 1}$ is a constrain on the discriminator in order to output results between $[-1, 1]$.

In order to use this loss function, the authors changed the architecture of the Discriminator by removing the last layer, the Sigmoid layer. Now, this architecture will not be able to output a classification between real or fake, but it will continue understanding which distribution is real and which is fake. In other words, the Discriminator will not be anymore a classifier and can be understood as a feature extractor. This is because the aim of this function is to maximize the distance between the real data distribution and the fake data distribution. Contrary to the Discriminator, the learning of the Generator should be based on minimizing the distance between the real data distribution and the synthetic data distribution, this will make synthetic images look similar to the real distribution.

Finally, the authors of WGAN [13] add a change during the training loop. This change is based on how many times the discriminator will look into the dataset. During the training of DCGAN [2] the *Critic* (named before as *Discriminator*) will look at the batch of images just once. Contrary to DCGAN [2], during the training of the Discriminator, in WGAN [13] the authors decided to look the batch of images 5 times per epoch. That will make the architecture learn better the features of the batch and lead to a better understanding of the data. Besides, in terms of time, this may lead to a longer training.

3) Wasserstein-GAN with Gradient Penalty: Wasserstein Gan [13] introduces a method that makes the training process much more stable, but sometimes still leads to poor generated samples or fails to converge. [4] studies these problems and determine that they are due to the use of weight clipping to enforce Lipschitz constraint on the Critic. Therefore, the authors' purpose an alternative way to clipping weights based on penalize the norm of the gradient of the critic with respect to the input,

$$L = \underbrace{\mathop{\mathbb{E}}_{\tilde{x}\sim\mathbb{P}_g}[D(\tilde{x})] - \mathop{\mathbb{E}}_{x\sim\mathbb{P}_r}[D(x)]}_{\text{Original critic loss}} + \underbrace{\lambda \mathop{\mathbb{E}}_{\hat{x}\sim\mathbb{P}_{\hat{x}}}\left[\left(\|\nabla_{\hat{x}} D(\hat{x})\|_2 - 1\right)^2\right]}_{\text{Our gradient penalty}} \quad , \tag{2}$$

where $\mathbb{P}_{\hat{x}}$ is defined to sampling uniformly along straight lines between pairs of points sampled from the data distribution $\mathbb{P}_r$ and the generator distribution $\mathbb{P}_g$. λ is the penalty coefficient, usually set to 10 as it generally lead to good results.

Since [4] penalizes the norm of the critic's gradient with respect to each input independently and not to the entire batch of inputs, the architecture of the Critic includes layer normalization (Instance Norm layers) instead of batch-normalization.

4) StyleGAN: This architecture [1] leads to an automatically learned, unsupervised separation of high-level attributes and stochastic variation in the generated images and it enables intuitive, scale-specific control of the synthesis.

The generator starts from a learned constant input and adjust the "style" of the image at each convolution layer based on the latent code, therefore directly controlling the strength of image features at different scales. Combined with noise injected directly into the network, this architectural change leads to automatic, unsupervised separation of high-level attributes from stochastic variation in the generated images, and enables intuitive scale-specific mixing and interpolation operations. The discriminator and the loss function are not modified.

The generator embeds the input latent code into an intermediate latent space, which has a profound effect on how the factors of variation are represented in the network. The input latent space must follow the probability density of the training data. The intermediate latent space is free from this restriction and therefore, is allowed to be disentangled.

[1] Presents two automated metrics for qualifying these aspects of the generator:

- Perceptual path length.
- Linear separability.

5) Transformers: The Transformer [17] is an architecture eschewing recurrence and instead relying entirely on an attention mechanism to draw global dependencies between the input and output. It follows an architecture using stacked self-attention and point-wise, fully-connected layers for both then encoder and decoder.

- Encoder: The encoder is composed of a stack of N=6 identical layers. Each layer has two sub-layers. The first is a multi-head self-attention mechanism, and the second is a simple, position-wise fully connected feed-forward network.
- Decoder: The decoder is also composed of a stack of N=6 identical layers. In addition to the two sub-layers in each encoder layer, the decoder inserts a third sub-layer, which performs multi-head attention over the output of the encoder stack. Similar to the decoder, residual connections are employed around each of the sub-layers, followed by layer normalization.

In this paper we take the ability of the Transformers to generate the text embeddings from the descriptions of each of the images from the dataset. We use a BERT [5] pre-trained model to extract those embeddings and then we feed the embeddings to the *batch normalization* layer for normalizing the image feature map with the text feature map into a single combined multimodal feature map. We will repeat this process along all the network in order to *remember* the network how the multimodal feature maps look like.

6) Dall·E: Dall·E [21] is a text-to-image task based on a transformer that auto-regressively models the text and image tokens as a single stream of data.

In this paper, the authors train a 12-billion parameter auto-regressive transformer on 250 million image-text pairs resulting this into a flexible, high fidelity generative model of images that is controllable through natural language.

The resulting system achieves high quality image generation on MS-COCO dataset [10] zero-shot, without using any of the training labels.

The goal is to train a transformer to auto-regressively model the text and image tokens as single stream of data. However, using pixel directly as image tokens would require an inordinate amount of memory for high-resolution images. Likelihood objectives tend to prioritize modeling short-range dependencies between pixels, so much of the modeling capacity would be spent capturing high-frequency details instead of the low-frequency structure that makes objects visually recognizable to us.

Two stage training procedure:

- Stage 1: Train a discrete variational autoencoder (dVAE) to compress each 256x256 RGB image into a 32x32 grid of image tokens. In [21] it is explained that by this is possible to reduce the context size of the transformer by a factor of 192 without a large degradation in visual quality.
- Stage 2: Concatenate up to 256 BPE-encoded text tokens with the 32x32=1024 image tokens, and train an auto-regressive transformer for fusing the image tokens with text tokens.

B. The Datasets:

Text-to-image synthesis is one of the most studied branch of Deep Learning. Several models are appearing with large amount of parameters able to synthesize images from text descriptions between other tasks. To do so, special datasets should be used with text descriptions of the image contents. In this paper we present the datasets used as benchmark for image generation and for transitioning to text-to-image generation.

1) MNIST: This dataset is composed by a large set of binary images with handwritten numbers. All the images inside of this dataset were size-normalized and centered in a fixed-size image. It contains 70.000 images divided on::

- Training set: 60.000.
- Test set: 10.000.

MNIST is the perfect dataset to test the built generative architectures due to its nature. As it is a grey-scale image dataset, with high difference between the black background and the white represented number, patterns and features can be easily extracted and learned from the using model.

2) Fashion-MNIST: Created by Zalando's article images, this dataset contains 70.000 grayscale images. Each sample is a 28x28 image labeled with one of the 10 classes that compose the dataset. Fashion-MNIST contains 70.000 images divided on:

- Training set: 60.000.
- Test set: 10.000.
- 10 different labels: T-Shirt/Top, Trouser, Pullover, Dress, Coat, Shandal, Shirt, Sneaker, Bag, Ankle boot.

The authors of this dataset aims to substitute the MNIST dataset for benchmarking machine learning algorithms. For this reason it shares the same image size and split structure for training and test sets.

3) CelebA: CelebFaces Attributes Dataset (CelebA) is a large-scale face attributes dataset composed by more than 200K images of face celebrities:

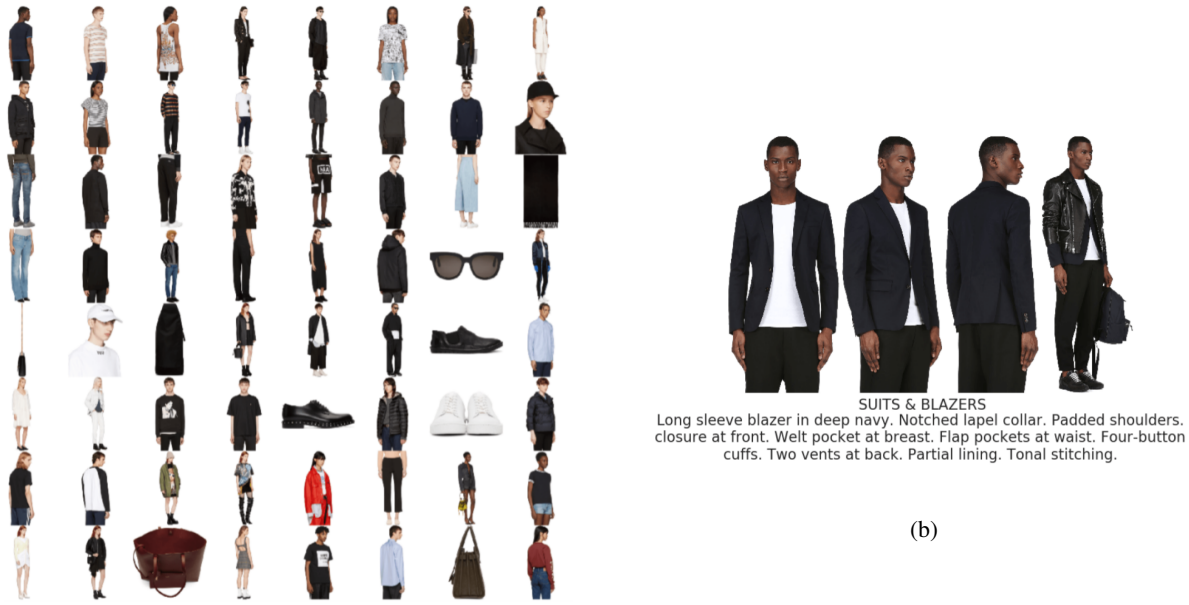
- 202599 number of face images of various celebrities.
- 10177 identities, marked with number labels.
- 40 binary attribute annotations per image.
- 5 landmark locations.

CelebA is a good benchmark for color image generation. This is due to the nature of the images composed by the faces of the celebrities and unfocused backgrounds. This makes easier GANs to focus on learning face features.

4) FashionGEN: This dataset is created aiming to assist the task of fashion designers to share ideas with others by translating verbal descriptions to images. Thus, given a description of a particular item, images of clothes and accessories are generated matching the description. This dataset is composed by:

- It consist of 293.008 images:
  - 260.480 images for training
  - 32.528 for validation
  - 32.528 for test

- Full HD images photographed under consistent studio conditions.
- All fashion items are photographed from 1 to 6 different angles depending on the category of the item.
- Each product belongs to a main category and a more fine-grained category.
- Each fashion item is paired with paragraph-length descriptive captions sourced from experts.
- Metadata is provided for each item. Also, the color distribution extracted from the text description presented.



(a) (b)

Fig. 2: (a) Batch of FashionGEN samples and (b) One concrete example from different angles.

Fig. 2a is an example of how many items have been photographed and how they where photographed from different angles. Furthermore, fig. 2b shows a concrete item photographed from four different angles and with a proper description written by a professional designer.

C. Conditional Batch Normalization

1) Batch Normalization:
Batch Normalization is a common technique used in deep learning in order to normalize the data as we compute the feature extractions on a network. Given a batch of N examples, batch normalization normalizes the feature maps as show in equation 3. e Is the numerical parameter that will give stability to the equation avoiding division by 0. Gamma Channels and Beta Channels correspond to trainable parameters that will compute the mean and variance of the feature maps along batch and spatial dimensions.

$$BN(F_{i,c,h,w}|\gamma_c, \beta_c) = \gamma_c * \frac{F_{i,c,h,w} - E_B[F_{\cdot}, c, ., .]}{\sqrt{Var_B[F_{\cdot}, c, ., .] + e}} \tag{3}$$

2) Conditioning:
Conditional Batch Normalization is a technique used on [24] to remember a network the conditional features after each convolutional layer of a network. The idea is to predict the mean and the variance of the conditioning features. To do so, conditional batch norm study the change on gamma and beta from the frozen original scalars, for which is straightforward to initialize a neural network to produce an output with zero-mean and small variance.
Therefore, an MLP is used to predict these variations:

$$\Delta\beta = MLP(e_q), \quad \Delta\gamma = MLP(e_q)$$

These MLPs will encode the variations and will output the number of features corresponding to the number of features extracted from the principal data as each layer:

$$\hat{\beta_c} = \beta_c + \Delta\beta_c, \quad \hat{\gamma_c} = \gamma_c + \Delta\gamma_c$$

These predictions will be added then to the *beta* and *gamma* trainable parameters. Finally, those calculus are used as parameters for the batch normalization.

D. Loss Functions

1) Binary Cross Entropy (BCE): For the experiments implemented on DCGAN [2] and conditional [14] DCGAN, BCE was selected, as we can understand the process as a classification problem where we need to make the model differ between real and fake images. On our particular case, the problem can be seen as asking the model *"Is this data real?"* or rephrase it as *"What is the probability of the data to be real?"*. Ideally, the probability of being real will be **1** and fake **0**. This can be understood by analyzing the nature of the BCE equation which was taken from the implementation provided by Pytorch:

$$l_n = -[y_n * log(x_n) + (1 - y_n) * log(1 - x_n)], \qquad where: \tag{4}$$

y_n is either 1 or 0, depending if the batch is real or fake, And x_n is the current image of the batch being analyzed. From this point, it is possible to differentiate three different situations:

- y_n **is equal to 1 and** D **is fitted with a real batch** : If we substitute this value into the equation we arrive to:

  $$l_n = -[1 * log(x_n) + (1 - 1)] * log(1 - x_n)],$$
  $$l_n = -[1 * log(x_n) + 0],$$

  First, we calculate the probability of a real batch x fitted into the discriminator D which should give high values and then we substitute this result on equation 4:

  $$l = -log(D(x))$$

  Now, it is possible to calculate the mean of the batch, which will give the probability of the batch of being real expressed in a logarithmic way. The result will give low values, that will certify that the discriminator is classifying properly the batch as a real batch.

- y_n **is equal to 0 and** D **is fitted with a fake batch:** If we substitute on the equation:

  $$l_n = -[0 * log(x_n) + (1 - 0)] * log(1 - x_n)],$$
  $$l_n = -[0 + (1 - 0)] * log(1 - x_n)],$$

  When a fake batch x is fitted to the discriminator D, the discriminator $D(G(X))$ should return low values, corresponding to the low probability of this batch of being real. If we substitute this result into the equation 4:

  $$l = -log(1 - D(G(x)))$$

  Now, it is possible to calculate the mean of the batch, which will give the probability of the batch of being fake expressed in a logarithmic way. This result should also return low values certifying with them that the fake batch was properly classified by the discriminator. When we back-propagate this result along the networks will make them understand how a fake batch distribution looks like.

- y_n **is equal to 1 but** D **is fitted with a fake batch:** This is the last step of the process. In this case equation 4 is developed like:

  $$l_n = -[1 * log(x_n) + (1 - 1)] * log(1 - x_n)],$$
  $$l_n = -[1 * log(x_n) + 0],$$

  This case is similar to the previous one, as we need to fit the discriminator with a fake batch of images x, and $D(G(x))$ should return low values but now we will label this batch as a real batch what will make the equation 4 be again:

  $$l = -log(D(G(x)))$$

  In this case, if $D(G(x))$ outputs high values, means that the Generator is fooling the Discriminator and when the loss function is calculated by targeting this batch as real it will return low values. On the other hand, if $D(G(x))$ outputs low values, means that G is not performing properly, and when calculating the loss function targeting this batch as a real batch it will return high values.

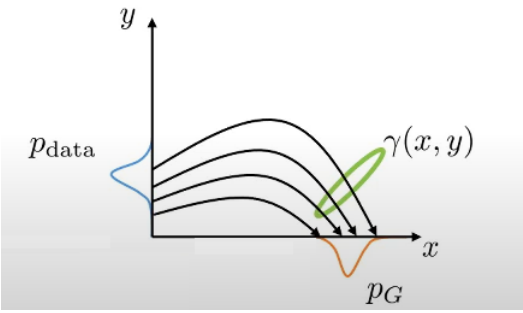


Fig. 3: Visual representation of the corresponding distances between two distributions. γ represents the joint distribution between p_{data} and p_G. By calculating the Wasserstain loss we are trying to find the minimum γ that minimizes the expected value of the distances.

2) Wasserstain Loss: Also known as Earth Moving Distance, is a loss function that aims to find the function f that will map as close as possible the target distribution by minimizing the distance between the generated distribution and the target distribution. When applied to GANs, the objective is to maximize, on the discriminator, the real distribution from the fake distribution. On the other hand, the idea on the generator is to minimize the distance of the fake distribution and the real distribution. In general terms, we can think about Wasserstain distance as the distance we need to *travel* in order to move from one distribution to the destination distribution. In figure 3 we can see a visual representation of this, where we can find a distribution on the y axis and another on x axis and γ is the joint distribution of both. In order to validate the Wasserstain distance, we need to find the γ that minimizes the expected value of these distance. The equation of Wasserstain Loss is provided by 1. In section II-A we explain the evolution from Wasserstain loss to Wasserstain loss with gradient penalty and how the network should be adapted in order to use these loss functions.

E. Evaluation Methods

1) Inception Score:
IS is a measure of how realistic a a GAN's output is. It measures two attributes simultaneously:

- The variety of the generated sample.
- If each image distinctly looks like something.

In order to study this attributes the inception score is helped by the Inception-V3 [23] classifier, from which the score takes its name. This classifier will be used to detect if the generated image contains a one distinct object or not:

- On the one hand if the generated image contains one well-formed object, then the classifier outputs a narrow distribution focused on one peak.
- On the other hand if the generated image is a jumble, or contains multiple objects, it is closer to a 'uniform' distribution with many similar height bars.

Furthermore, it is needed to evaluate the variety of the samples. To do so, the label distributions of the generated images is summed, obtaining with that a marginal distribution. Finally, a combination of these measures is needed. As both measures will be probability distributions they are comparable to each other. This comparison is done by using the Kull-back Leiber (KL) [7] divergence which was created to compare how similar/different are two probability distributions.

Knowing this, if both attributes are true, the score of the inception score will be high. If either or both are false, the score will be low.

2) Frechet Inception Distance: Similar to the Inception Score, FID [6] is a metric for evaluating the quality of generated images and specifically developed to evaluate GANs. The goal of this score is to evaluate synthetic images based on the statistics of a collection of the synthetic images compared to the statistics of a collection of real images from the target domain.
Following the steps of IS, the FID score also uses the Inception-V3 model. In this case, a specific layer of this model is selected in order to extract the convolutional features generated when a batch of real and batch of generated images are fitted into the model. The output are two feature distributions that would be used to calculate the Frechet Distance, also called Wasserstain-2

distance between them. The procedure to calculate FID score is then:

- First, the last layer (output layer) of the *Inception-V3* model is removed and the activation's from the last pooling layer are taken as output.
- The output of this layer is a 2048 activation feature vector. Therefore, each image is predicted as a 2048 feature vector called the 'the coding vector'
- In order to obtain a reference from the batch of real images, it is needed to calculate the coding vector from them by fitting them into the model. After this, we repeat the same step but this time using a batch of synthetic images generated by our GAN. We now have two coding vectors containing the reference features of both real and fake images.
- Then the FID score could be calculated by applying the formula from the paper:

$$FID = || \mu_x - \mu_y ||^2 + Tr(\Sigma_x + \Sigma_y - 2\sqrt{\Sigma_x \Sigma_y}) \quad , \tag{5}$$

  where μ_x and μ_y refer to the feature-wise mean of the real and generated images, Σ_x and Σ_y are the covariance matrix for the real and synthetic images and Tr refers to the trace linear algebra equation.

Finally, the result of this equation will be the FID score for the evaluated model. The lower the number is, the better results you will obtain while training. If the numbers of the FID are high but still decrease is due to the fact that Inception V3 does not contain images similar to your dataset but it still performs its function of evaluating the generated images.

III. Method

This paper presents a benchmark of GANs and how to condition their performance by the use of embeddings or one hot vectors followed by a conditional batch normalization. Furthermore, an evolution from images of one channel to three channels was implemented at the same time as an increase of the difficulty of the datasets. The methodology for these experiments is the following:

- Implementation of DCGAN [2]: This architecture was the start point for the experiments. First of all it was implemented for processing one channel images. For that, the MNIST dataset was used. After, the architecture was tested on a more extensive dataset, the FashionMNIST dataset, which is a dataset of one channel images but much more similar to the final dataset, FashionGEN [16].
  After these tests, the architecture was modified to process three channel images and tested, first of all with the CelebA [12] dataset, as it is an easy dataset to easy generate faces and make the network converge. With that working, the networks then were ready for the final test that was to test the architecture on the FashionGEN [16] and obtain the results.

- Implementation of WGAN [13]: From the implementation of DCGAN [2], some modifications were implemented in order to obtain the architecture for WGAN [13]. First of all, the sigmoid layer at the end of the DCGAN discriminator architecture was retired. After this, the loss function was changed from Binary Cross Entropy to Wasserstain Loss 1 and weight clipping. Furthermore, the training loop was modified following the instructions of WGAN [13] to perform five more iterations per batch in order to increase the discriminator's performance. Following the same methodology as before, this architecture was tested on all four datasets.

- Implementation of WGAN with Gradient Penalty [4]: Finally, on the implementation of the basic GANs, the last modification was performed on the Wasserstain Loss in order to implement gradient penalty. As mentioned in [13], *weight clipping is a terrible way to enforce a Lipschitz constraint. If the clipping parameter is large, then it can take a long time for any weights to reach their limit, thereby making it harder to train the critic till optimally. If the clipping parameter is small, this can easily lead to vanishing gradients when the number of layers is big, or batch normalization is not used*, that is why the idea gradient penalty appear. In [4], the changes are applied on the critic's training loop and on the loss function. On the critic's training loop, a fake sample is generated, with the generator, from the latent vector z and apply an interpolation between the real images and the generated images, and with that interpolation and the norm of the gradient is now possible to satisfy the Lipschiz constraint. in the loss function, now appears a new parameter, that is a regularization parameter, which contains a lamda parameter (usually set to 10) and the norm of the gradient of the interpolated images minus one, in order to be as close as one as possible. In terms of the generator's update, it will remain the same but the fact that now there is no weight clipping, there is this new regularization term added at the end of the loss function.
- Implementation of Conditioning: After the implementation of the basic GANs, the next step was to implement the first methodology of conditioning. In order to achieve the conditioning, the methodology was by creating an embedding in order to fit the different classes. Therefore, the first change in the architectures was to create an embedding with the dimensions of the number of classes we have in our dataset (for the FashionGEN dataset this number is 48) and with the

depth corresponding to the input of our network. In the case of the Discriminator this depth is the *image size* x *image size* because it will be reshaped on the forward pass in order to have the same dimensions as an image and being able to be concatenated as an extra channel of the inputs. Similar to the Discriminator, in the Generator, an embedding with the dimensions of the number of classes is created but in this case with the same depth as the latent vector that will be the input of the network. The reason of this is because as we want to concatenate the embedding to the noise and convert them progressively into the shape of an image we need to math the dimensions. Furthermore, on both networks is needed to adapt the input. In case of D, we need to add an extra input channel as now the images will contain the embedding of the labels. In G we need to add the depth of the embedding as similar to D, the input now will contain the label embedding.

- Implementation of Conditional Batch Normalization: This step is inspired in the same idea mentioned on [24] where after developing the conditional batch normalization they substitute each batch normalization layer by their conditional implementation. Following this idea, we substituted in each part of the GAN (Discriminator and Generator) every batch normalization layer by a conditional batch normalization layer, leading this to a new architecture that will be used in order to fuse the text features extracted by a third network. Furthermore, in order to fit these new layers we had to take into account the dimensions of the MLPs that will estimate the new mean and standard deviation that will establish the updates of the fused feature vectors.

IV. Experiments

The project was planned in order to follow the same structure for the training of the three different networks that are being used. Therefore, the same image transformations and corresponding dimensionality adjustment are performed the same on the three architectures. All the experiments were performed on a cluster of maximum 5 Tesla P100 GPUs of 12GB. Decreasing the time of the experiments from 5 days and 12h for trainings with 1 GPU, to 1 day and 12h for trainings with 5 GPUs.

1) Image Pre-processing:
Regarding to the image transformations performed previously to the training, a normal batch of images will contain 64 RGB images per batch, with width and height set to 64 and center cropped. Furthermore, mean and standard deviation are set to 0.5 for every channel, in order to normalize the images between 0 and 1, and following the recommendations of the paper.

2) Conditional GANs:

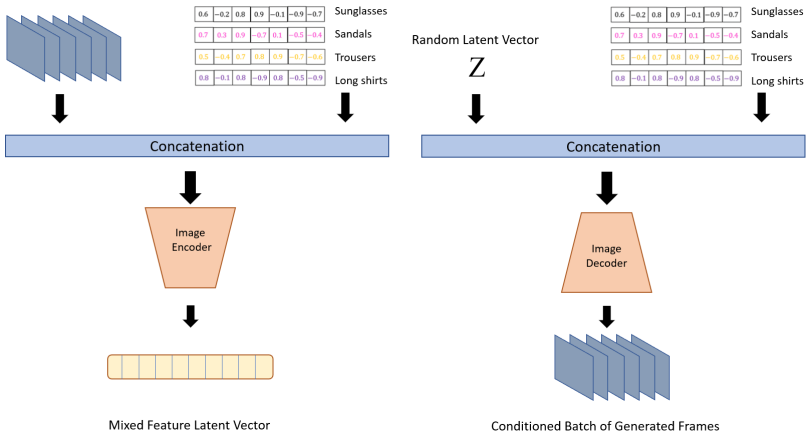


Fig. 4: Explanation of the structure of classic conditional GANs, using concatenation for text conditioning.

On a first step, the conditioning of the architectures was performed, on the three GANs with the same strategy, by concatenating a word embedding with the dimensions of the number of classes. The word embedding can be understood as a simple look up table that will store the information of how the classes of the dataset are distributed into a latent space. This word embedding is created then with shape *[size of the dictionary of embeddings, size of each embedding vector]*. Knowing this, we create our embedding to contain the information of the 48 classes of the dataset into a space set to 64*64 in order to match the number of dimensions of the images. As the embedding should be understood as an extra channel that will be added to each image containing the information of the desired class, the last step to perform before the concatenation of the embedding is to reshape the embedding, to finally look like a channel. The input of the discriminator network is now a batch with the dimensions set as [64, 3+1, 64, 64] and an embedding with dimensions [64, 1, 64, 64].

The concatenation is produced using the early fusion technique, that means, the word embedding is concatenated to the image batch before entering to the network. The embedding must follow some dimensionality setting in order to match the dimensions of the image batch.

For the Generator network, we follow the same idea, but with some little changes as the input is set to a latent vector z with depth 100. As we now, this latent vector is going to increase its dimensions progressively at each transpose convolution until we finally obtain the desired dimensions that are [64, 3, 64, 64] that refer to the batch generated RGB images. Because of that difference with the discriminator, the embedding will be created now as a vector with depth 100 with the information of how the 48 classes relate in the latent space. Notice that depth is set to 100 in order to match the shape of the latent vector z. As previously, before concatenating the embeddings, we need to perform the last transformation by increasing the dimensions of the embedding by 2, so finally, we will have an embedding with shape [64, 100, 1, 1].
After the forward step on the Generator, the batch of generated images will have a shape of [64, 4, 64, 64], noticing that the number 4 correspond to the 3, RGB, channels and the extra channel that contains the label that is conditioning our generated image.

3) Conditional Batch Normalization:

As mentioned before, this technique is used for the first time on [24]. The main idea of this technique is to, at each normalization

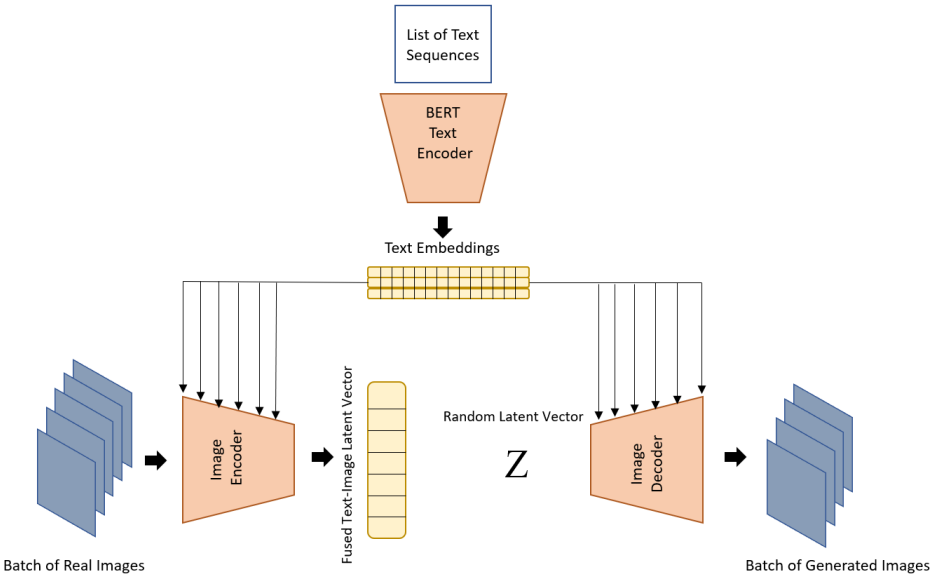


Fig. 5: Visual explanation of the final network mixing GANS and BERT text encoder to perform images synthesis from text embeddings.

layer introduced in a convolutional network, normalize vision features and text features in order to achieve a common feature vector that describes the features extracted from both modalities. To do so, the key point of this technique is to calculate the mean and the standard deviation from a statistical distribution (in our case, text sequences) and store them into a learnable variable in order to update this information at every step of the network. With this, we achieve to "remember", at each layer, the network which is the distribution that should follow during the creation of each image, arriving with that to condition the generation of the images.

Therefore, the experiments realized with this technique started by changing the batch normalization layers of original architectures by the conditional batch normalization layers. After this, some dimensionality adjustments should be performed in order to match these new layers to the conditions of the network.

As CBN uses two different MLPs to calculate the mean and the standard deviation, we need to adjust the input, the hidden layers, and the output of each of the MLPs. Either for mean or std. deviation we adjusted the dimension in the same way. First, the input is set to the number of classes that we are introducing to the network. Then the hidden layers are set to 64 (this number has no mathematical relation). Finally, the output of the MLPs will be the same size as the vision feature vector calculated on the last convolution layer. This will make the different features calculated to match in dimensionality and

Architecture	Experiment	Max Value	Min Value	Final Value (20 epochs)
DCGAN	W-CBN/Adam	311,4	174,7	218,1
	ST-CBN/Adam	249,9	185,1	240
WGAN	W-CBN/RMSprop	409,8	303,7	354,9
	ST-CBN/RMSprop	334,3	277,3	305,1
	ST-CBN/Adam	301,2	201,5	300,1
WGAN-GP	W-CBN/RMSprop	316,9	238,6	264,1
	ST-CBN/RMSprop	484,5	204	204
	ST-CBN/Adam	463,1	239,9	375,1

TABLE I: FID minimum, maximum and final values obtained for each experiment performed on CBN architectures. *W-CBN* and *ST-CBN* correspond to **word** or **sentence**, respectively, referring to how the text is presented, as a simple word corresponding to a certain class or a sentences that will describe the image.

therefore we can perform batch normalization on them.

Due to the nature of the networks, we can observe that the setting of the output features from the CBN layer will increase by the power of 2 in the Discriminator and decrease by the power of 2 in the Generator. Furthermore, in these experiments we introduced a new way of extracting text features. In this case we are no longer creating an embedding of labels, we are using the last hidden state of a pretrained BERT transformer enconder which has an output of 768. That means that now the number of classes we have is 768.

The first experiments performed with this technique were using just a word for conditioning the image generation. This means, that every word was encoded as a 768 embedding by a transformer and then normalized with the vision corresponding embedding.

Finally, we tested this same experiment but by the use of text descriptions that contain much more information of how the images should be generated. In this case the transformer had to encode each of the descriptions in a batch into a 768 embedding that contains the latent information of the description. Therefore, the input of the first normalization layer is [64, 128] from the vision modality and [64, 768] from the text modality and the output is a feature vector with shape [64, 128].

V. Results

Along the paper we discussed the different experiments we have performed for each architecture, coming from integral implementation of the architectures, going through classical feature conditioning and ending in the integration of *Conditional Batch Normalization* which is the final objective of the paper. Here we present the results obtained for the experiments realized with the final architectures that count with the conditional batch normalization on their structure. We have to remark that though we obtained really great results for DCGAN, WGAN and WGANGP were harder to train and will need of more hyperparameter tuning in order to improve the results. Therefore, in this section we are going to present the results obtained for those experiments that obtained a numerical value of the FID under 250 as we observed this would be a good threshold to differentiate from a difuse image or a recognisable image, we can observe the different FID values in table I. The rest of the results and further experiments will be commented below on the annexes.

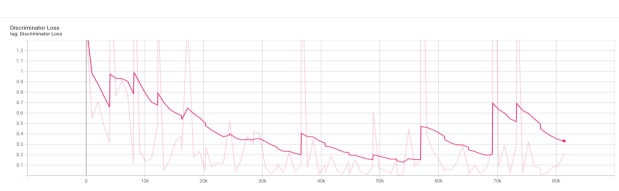
A. Quantitative and Visual Results

As discussed on the section where the explanation of the evaluation methods is done, we used the *Frechet Inception Distance* in order to follow the learning progression of our GANs. We know that this distance will then numerically represent if the distribution of the features of the generated images is similar to the distribution of the features of a real batch. In table I we present the different values obtained from the FID [6] along the different experiments performed and focusing on the minimum, maximum and final values of our trainings.

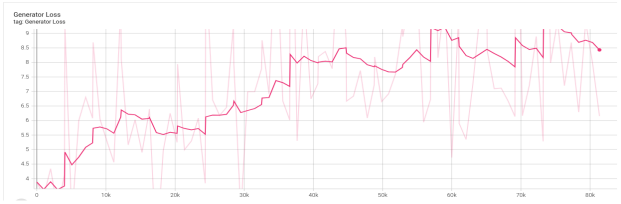
1) Experiments on DCGAN: Here we present the different graphics and generated images obtained from the experiments with single word classification and for the sentence conditioning of the architectures.

On 6 we can observe that the discriminator and generator preserve the equilibrium that the training should maintain for obtaining good results.

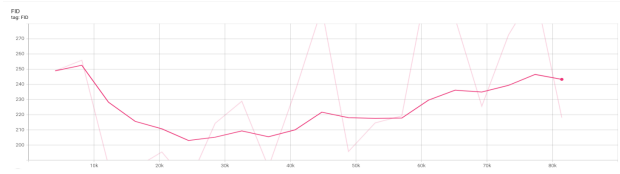
In figures 7a and 7b we can observe the results obtained by the Generator of DCGAN when achieving its best FID numeric value and when the training is completed as seen in tab I.



(a) Progression of the Discriminator



(b) Progression of the Generator



(c) Progression of FID

Fig. 6: Progression of the training and FID results on the experiments performed on DCGAN with CBN for single word embeddings. Step 4000 correspond to the minimum value and step 24250 correspond to the maximum value presented on I

(a) Images generated for the best FID result

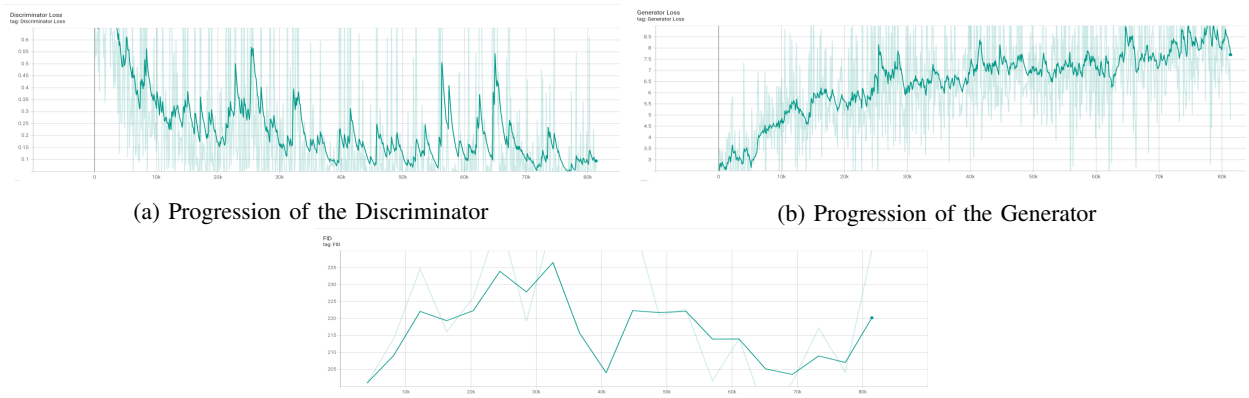
(b) Last images generated from the training

Fig. 7: Visual comparison from the best FID and final results on the experiment performed on DCGAN for single word conditioning from text embeddings.

We can observe that the network is able to perform the conditioning results from the very beginning of the training as figure 7a is extracted from approximately from the epoch 2 and half of the 3rd epoch. We can observe this conditioning visually as the last row of the generated images was set to generate images from the embedding corresponding to the word *BOOTS*. This is a simple but effective way of visually see if the network is achieving its objectives. From figure 7b we can see that the images generated preserve the conditioning and the images does not differ so much from 7a.

Similar to 6, in 8 the equilibrium between the discriminator and generator is preserved, leading that to good visual results evaluated by 8c. By comparing the FID graphs 6c and 8c we can notice that when conditioning with single words the graph is smoother than when training with descriptive sentences of the images. Furthermore, we see that the best results of the training are obtained earlier with single word conditioning.

Figure 9 show the results of text-to-image conditioning for the experiments performed on DCGAN by using descriptive sentences to condition the results.We can observe the conditioning if we look to the descriptions of the images, which is the text input for the image generator. For example image, if we treat the image as a grid of images, image in position (1, 1) correspond to description *"Long sleeve semi-sheer organza hooded jacket in 'fume' grey. Satin trim in green throughout. Half-zip closure at front. Kangaroo pocket at waist. Dropped shoulders. Elasticized cuffs. Tonal stitching."*



(a) Progression of the Discriminator

(b) Progression of the Generator

(c) Progression of FID

Fig. 8: Progression of the training and FID results on the experiments performed on DCGAN with CBN for sentence word embeddings

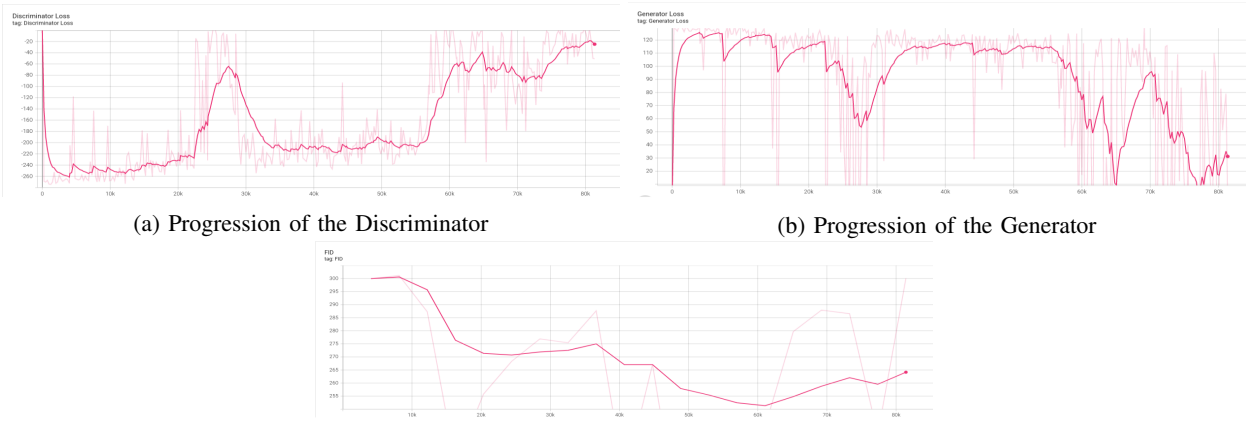
(a) Images generated for the best FID result

(b) Last images generated from the training

Fig. 9: Visual comparison from the best FID and final results on DCGAN conditioned by text descriptions.

If we compare both images 9a with 9b, we can observe that though the FID graph shows that 9a shows the images for the best FID numerical result, 9b present slightly better results arriving to represent some details in the images that are not seen in 9a. For example, image in position (8,8) show a person with a white T-shirt and some details inside of the T-shirt in 9b but not in 9a. Furthermore, if we compare the image in position (6,2), we can observe that a better detail of image creation is achieve with longer training. Other results can be remarked: (1,1), (1, 6), (3,5)...

2) Experiments on WGAN: Results on WGAN with conditional batch normalization were harder to obtain as the hyper-parameters set on [13] were for results obtained into a certain datasets. With the new dataset *FashionGEN* [16], we tested two different optimizers and learning rates. Figure 10 represents the progression of the experiment performed on WGAN with conditional batch normalization for sentence image descriptions, trained with Adam optimizer and with a learning rate of $2e-4$. First thing we notice is that the values of the FID 10c are much larger than in the training performed on DCGAN8c. Second thing to notice is that, as explained in WGAN [13], the output of the Discriminator (critic in the paper) and the Generator should be bounded between 1 and -1, but we can observe here this is not accomplished. Although this last statement it is not accomplished, we can notice that every time there is a drop on the Generator graph 10b, there is also an increase of the values of FID 10c and there is an increase too on the values of the Discriminator loss 10a, which demonstrates us, that Wasserstain loss is a more stable loss for training the GAN and obtaining meaningful information from the loss graphics.



(a) Progression of the Discriminator

(b) Progression of the Generator

(c) Progression of FID

Fig. 10: Progression of the training and FID results on the experiments performed on WGAN with CBN for sentence word embeddings

Figure 11 shows the progression achieved during the training of our modified WGAN on Adam optimizer. We can observe how in figure 11a, position (1,1) is starting to converge and understanding the information introduced by the text description "*Long sleeve semi-sheer organza hooded jacket in 'fume' grey. Satin trim in green throughout. Half-zip closure at front. Kangaroo pocket at waist. Dropped shoulders. Elasticized cuffs. Tonal stitching*" or position (1,5) with description "*Long sleeve buffed leather jacket in black. Ribbed knit trim at stand collar, sleeve cuffs, and hem. Contrasting linen twill sleeves in ivory white. Welt pockets at front. Fully lined. Tonal stitching*". These confirm us that the network is understanding the text embeddings. If we compare these results to the results obtained on 9 it is easy to notice that 9 obtained much better results, and this is also numerically proved by the FID results seen on both graphs 8c and 10c.
Figure 11a shows the final results obtained by the network after 20 epochs. We can see how the network still follows the guides of the text descriptions by trying to represent them, but it is not able to give good results in terms of image quality. This is numerically proven on 10c, where it possible to notice that the values of the FID increased during the last steps.

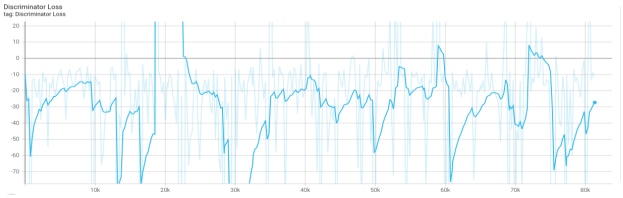
(a) Images generated for the best FID result

(b) Last images generated from the training

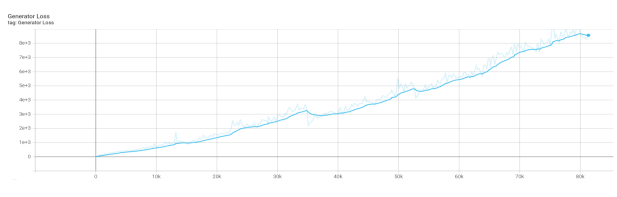
Fig. 11: Visual comparison from the best FID and final results on WGAN trained with Adam optimizer and 2e-4 learning rate.

3) Experiments on WGAN-GP: As explained in [13], WGAN presents a more stable way to train a generative adversarial network but in some cases it struggles on converging. Therefore we modified the architecture and implemented the extension architecture of WGAN which is WGAN with Gradient Penalty and introduce our new normalization layer for conditioning

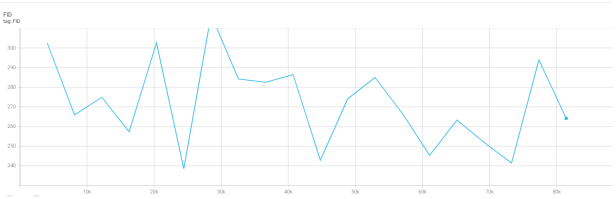
the images. Figure 12 represents the progression of the numerical results of the architecture conditioned on a single word representing a certain class. The training was performed following the steps of the algorithm set in [4]. We observe how in this experiment 12a has values closer to the bounding -1 set by WGAN, but still obtain results far from the conditions set by [13]. Furthermore, in 12a at step 20K there is a big rise of the values and compared to the FID values, gives us a significant meaning as the FID values increase too, and therefore the network is not generating good images.



(a) Progression of the Discriminator



(b) Progression of the Generator



(c) Progression of FID

Fig. 12: Progression of the training and FID results on the experiments performed on WGANGP with CBN for single word embeddings

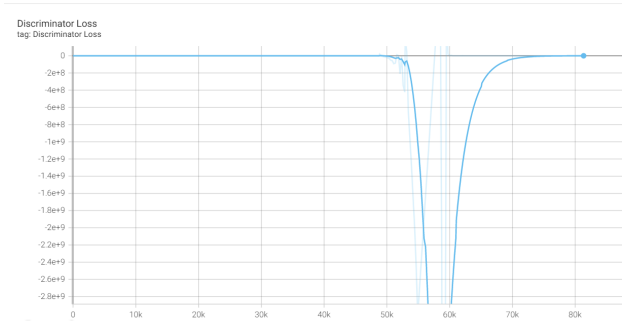
(a) Images sampled at step 47K, epoch 11

(b) Last images generated from the training

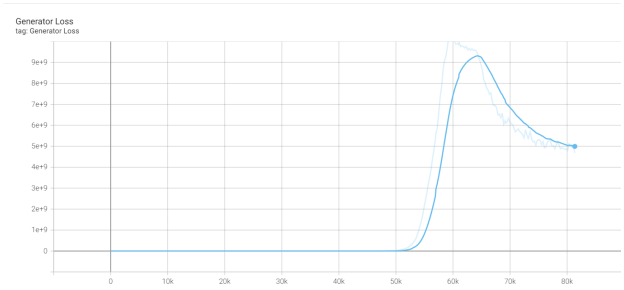
Fig. 13: Visual comparison from a sample on during the training and final results on WGANGP trained with RMSprop optimizer and 5e-5 learning rate.

From visual results, we can prove that the network is learning properly the text embeddings and leading the generated images to their corresponding label. We can prove that by attending, for example in figure 14a, in image (1,3) from is possible to interpret how the network generates and image conditioned from the work *HATS*. Another example is image (1,7) which stands for label *BRIEFCASES*, we can observe that the network is understanding the label but is still lacking in properly complete the image. Similar as the training of DCGAN with single word labels, in this experiment we set the last 17 labels to the word *BOOTS* in order to visually prove that network is properly understanding the conditioning. We can observe that most of the images correspond to a pair of drawings that correspond to a pair of boots.

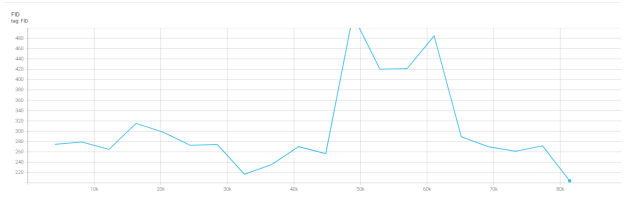
Following the line of the experiments, we tested our changed WGANGP in order to fit sentence descriptions as our text conditioning embeddings. In figure 14 we show the progression of the training realized on WGANGP for RMSprop optimizer and $5e-4$ learning rate, reducing the learning rate stated at [4], as we notice in the previous experiments on WGAN with RMSprop optimizer did not lead to good results. First thing we notice is the scale of the graphs, as on the final steps of our training there is a big raise on the Discriminator and on the Generator, and that is also followed by the FID results. Despite the big raise of numbers around step 60k, we looked over the previous steps and notice that the numbers are closer to the bounds (-1,1) that WGAN should be. This is also translated in better FID results, see 14c, and also in better visual results 15.



(a) Progression of the Discriminator



(b) Progression of the Generator



(c) Progression of FID

Fig. 14: Progression of the training and FID results on the experiments performed on WGANGP with CBN for sentence word embeddings and RMSprop optimizer

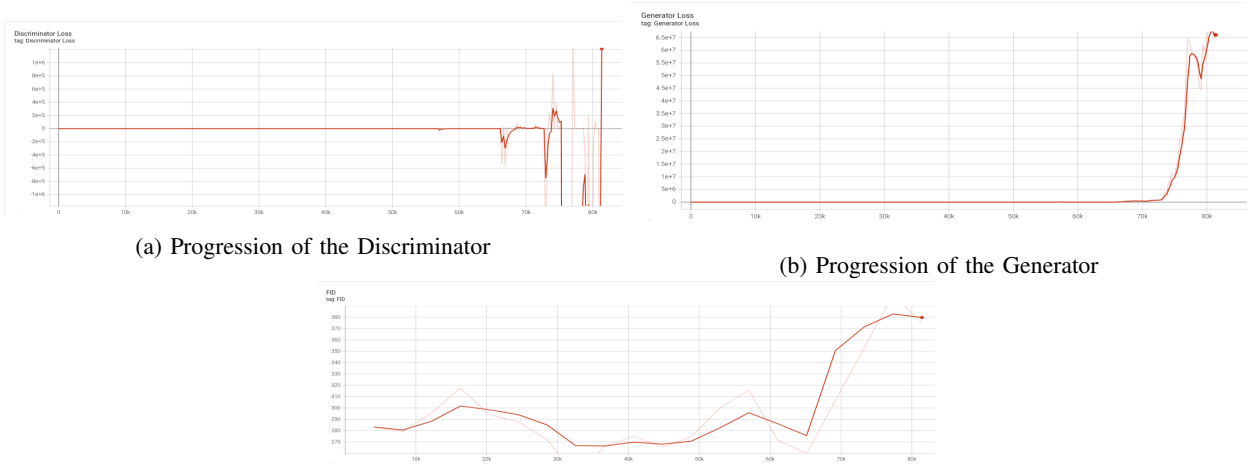
From visual results we can see that in this experiment the network learns properly the text embeddings and lead the images to their information. For example, we can see that image (4,6) in both 15a and 15b look similar, so we can understand there is a progression conditioned, both of them by the same labelling which is "*High-top suede sneakers in 'flame' red. Tonal buffed leather panelling throughout. Round toe. Tonal lace-up closure. Rubberized logo at padded bellows tongue. Signature zippered expansion panel at eyerows. Padded collar. Tonal textured rubber midsole. Tonal treaded rubber sole. Tonal hardware. Tonal stitching.*". Furthermore, on the final steps of the training the numbers of the graphics shown on 14, start to low down again, recovering their previous values. The visual results show us that the network is still remembering the text conditioning information. This is proved for example at image (3,2) where the labelling is "*Long sleeve knit sweater in royal blue. Ribbed crewneck collar, sleeve cuffs, and hem. Rubber logo patch at breast in white. Vented hem. Tonal stitching*". In this example we can see how the image is totally conditioned by the class of the clothe, its sleeve type and the colours mentioned on the description.

Finally, our last experiment on the dataset, was to train WGANGP with Adam optimizer and learning rate set to $2e-4$. We can notice that at the beginning of the training the slopes shown at 16 look similar to the ones obtained at 14 but this time around 70k, epoch 17, there is a big change on the progression of the slopes and we can clearly see how this affect to the FID results and directly to the final visual results, shown at 17b. Figure 17b shows clearly that at the last step of the training the gradients of the network vanishes and therefore the image generation drops to random images.

Even though the network is far from converging at the last steps of the training, we extracted some visual results from the earlier steps, shown at 17a, and we can notice that before the gradients vanishes the network was understanding the information given by the text embeddings. For example image (1,3) is conditioned by "*Slim-fit jeans in dark blue. Distressing throughout. Fading at front. Textured black leather logo patch at back waist. Silver-tone metal logo plaque at back pocket. Contrast stitching in tan. Red logo tab at button-fly.*", and it is possible to appreciate how the network generates the images properly by attending to the class, characteristics and color. This lead us to understand that the information given by the text embeddings generated is being understood by the network during the training.

(a) Images sampled at step 31.7k, epoch 7 (b) Last images generated from the training

Fig. 15: Visual comparison for sentence description conditioning from a sample extracted during the training and final results on WGANGP trained with RMSprop optimizer and 5e-4 learning rate.



(a) Progression of the Discriminator

(b) Progression of the Generator

(c) Progression of FID

Fig. 16: Progression of the training and FID results on the experiments performed on WGANGP with CBN for sentence description embeddings and Adam optimizer

4) Out of Context Experiments: Finally, to test the network that achieved best quantitative and visual results, we decided to fit the generator out of context descriptions. Meaning this that we took some descriptions from a real fashion website and by copying the descriptions of their products we generate images. By this, we visually demonstrate that the network is able to understand natural descriptions from real applications and that the network was learning and not copying the real data given for training. In figure 18 we can see 10 samples generated from these real descriptions.

We can notice that the network is able to understand what kind of product should draw and also the features that describe the product. For example if we focus on figures 18b and 18c, we clearly see that the network print the images conditioned on the word sunglasses and also for figure 18b is it possible to observe that the lenses are a bit rounded and finer than in figure 18c, what demonstrates us that the network is being conditioned by the full description.

Furthermore, on images 18a and 18d we can see that the network is capable to understand some shape conditions as in image 18a this shoulder bag should have a shape of width=7cm, height=16, length=23. It is possible to observe on the image

(a) Images sampled at step 34k, epoch 8 (b) Last images generated from the training

Fig. 17: Visual comparison for sentence description conditioning from a sample extracted during the training and final results on WGANGP trained with Adam optimizer and 2e-4 learning rate.

(a) Fabric shoulder bag with zippered top and removable adjustable shoulder strap with buckle and metal carabiners. One exterior compartment with hidden zipper and one interior mesh compartment. Lined. Width 7 cm. Height 16 cm. Length 23 cm.

(b) Round sunglasses with plastic frame, lenses and temples. Polarized tinted lenses with UV protection for color clarity and contrast while eliminating glare. Supplied in a fabric case with drawstring.

(c) Sunglasses with plastic frame. Plastic tinted lenses with UV protection.

(d) Synthetic leather sandals with frontal straps and bracelet with hook-and-loop fastener. Synthetic leather insole and thick sole with ribbed design. Thickness of the sole 6 cm.

(e) Cotton twill cap with adjustable metal clasp at the back. Cotton sweatband.

(f) Black leather belt with metal buckle. Width 3.3 cm

(g) Weekend bag in synthetic leather with two handles, removable shoulder strap, zipper on top and two compartments inside. Support studs on the base. Lined. Measures approx. 22x31x48 cm

(h) Five-pocket black jeans in washed cotton denim. High-waisted model with wide straight legs and zipper closure with button.

(i) Gold plated metal chain anklets in different designs. Adjustable length and spring ring closure.

(j) Synthetic leather boots with ankle shaft, laces in front and belt loop at the back. Synthetic leather insole and thick sole. Heel 3 cm.

Fig. 18: Image results generated from descriptions taken from a fashion website from the internet.

how width and height are perfectly respected. Even on image 18d the description indicates that the thickness of the sole is 6cm, and we can see how the sole is with thickness.

Finally, on image 18i we can see how the network is also able to understand material or color conditionigs as this image specifies *gold plated metal chain* and we can observe how the network arrived perfectly to the desired result.

VI. Conclusions

In this paper, we have discussed how humans perceive the information of the world as a multimodal source of information and why artificial intelligence should be adapted in order to understand similarly multimodal information. We have presented some state-of-the-art networks that started to describe task as image generation oriented to visual sources of information and networks that perform text oriented tasks by describing the patterns of natural language processing.

Furthermore, we have seen some of the most important techniques such as *One Hot Encoding* and *Tokenizers* that help us in order to perform these text tasks and also for starting multimodal text-to-image tasks by applying early fusion methodologies, such as concatenation. Moreover, we have explained the first image generation network and the next two following changes that this network suffered in order to improve the results and obtain a more stable training, and also we have introduced some of the state-of-the-art networks that are addressing the text-to-image generation task that were an inspiration for this paper.

Besides, we have introduced some public datasets that will help to introduce people into the image generation and text-to-image generation tasks, as these datasets are remarked as easy dataset for training, for example MNIST and FashionMNIST are a good benchmark as they are binary datasets with different classes that can be addressed by the use of *one hot encoding* or can be used to introduce NLP conditioning by using the words of the classes. Furthermore, we introduced two more datasets that were useful to jump from binary images to RGB images and to introduce longer sequences of conditioning in our network.

In order to introduce our innovative technique for text-to-image generation, we have presented where this technique comes from and how it can be implemented to address our purpose. To conclude with the theoretical part we have presented the different loss functions we used for our trainings and the different evaluation methodologies we tested in order to verify our results, concluding that FID is the one that should be used for all the trainings due to its better performance, although it sometimes lacks if the classes are not scene on the InceptionV3 network. Because of that, we conclude that FID is a good metric for checking the progression of our GANs but for better results and in order to compare them with state-of-the-art results a finetuning with part of the our dataset should be performed in order to let the network see the classes.

We have also explained how to address the image generation task and how to progressively transform this task into a text-to-image generation task by explaining all the steps taken during the development of the research and describing the final networks with detail. Besides, we demonstrate visually and quantitatively that the text-to-image generation task is performed perfectly with our methodology based on *Conditional Batch Normalization* and it is easy to reproduce.

From the different experiments performed, we can conclude that all networks were able to arrive to understand text descriptions and generate images from them. Our DCGAN approach was the one to obtain the lower FID number and also the best visual results, arriving also to generate images from text descriptions extracted from a real fashion website, being able to understand different features that conditioned the generation of the product, such as class, color, shape, etc…

All in all, we have presented a benchmark paper in order to address text-to-image synthesis by introducing *Conditional Batch Normalization* as new technique for fusing text features with image features to perform the task. Furthermore, we presented this technique on the very first image generation networks and seen that the networks arrive to perform the task perfectly and making the text-to-image synthesis available to everyone without the need of big clusters of GPUs to address the training of the networks. Therefore, we purpose this technique to be studied in modern GANs as a future work in this research and also hyperparameter tuning, and longer trainings should be performed in order to improve the results.

Acknowledgment

The authors would like to thank ElementAI and ServiceNow their time and the free access to their GPU servers for developing the project. Furthermore, special thanks to David Vázquez for the supervision of the project and for sharing his knowledge to increase researcher skills and AI understanding. In addition, thanks to Rafael Pardinas for helping on the supervision of the project and sharing his knowledge of virtual machines, docker and AI understanding, that were really useful to create a proper workflow in order to implement the different experiments performed. Finally, great thanks to the Computer Vision Master teachers of Barcelona for the teaching and good lessons that make possible the understanding of the technical concepts needed for the accomplishment of this paper.

REFERENCES

[1] Tero Karras Samuli Laine Timo Aila. "A Style-Based Generator Architecture for Generative Adversarial Networks". In: *Arxiv.org* (2019). DOI: https://arxiv.org/abs/1812.04948.

[2] Soumith Chintala Alec Radford Luke Metz. "Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks". In: *Arxiv.org* (2015). DOI: https://arxiv.org/abs/1511.06434.

[3] Ian J. Goodfellow Jean Pouget-Abadie Mehdi Mirza Bing Xu David Warde-Farley Sherjil Ozair Aaron Courville Yoshua Bengio. "Generative Adversarial Networks". In: *Arxiv.org* (2014). DOI: https://arxiv.org/abs/1406.2661.

[4] Ishaan Gulrajani Faruk Ahmed Martin Arjovsky Vincent Dumoulin Aaron Courville. "Improved Training of Wasserstein GANs". In: *Arxiv.org* (2017). DOI: https://arxiv.org/abs/1704.00028v3.

[5] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[6] Martin Heusel et al. *GANs Trained by a Two Time-Scale Update Rule Converge to a Local Nash Equilibrium*. 2018. arXiv: 1706.08500 [cs.LG].

[7] Taehyeon Kim et al. *Comparing Kullback-Leibler Divergence and Mean Squared Error Loss in Knowledge Distillation*. 2021. arXiv: 2105.08919 [cs.LG].

[8] A. Taylan Cemgil Sumedh Ghaisas Krishnamurthy Dvijotham Sven Gowal Pushmeet Kohli. "Autoencoding Variational Autoencoder". In: *Arxiv.org* (2020). DOI: https://arxiv.org/abs/2012.03715.

[9] Taku Kudo and John Richardson. *SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing*. 2018. arXiv: 1808.06226 [cs.CL].

[10] Tsung-Yi Lin et al. "Microsoft COCO: Common Objects in Context". In: *CoRR* abs/1405.0312 (2014). arXiv: 1405.0312. URL: http://arxiv.org/abs/1405.0312.

[11] Or Patashnik Zongze Wu Eli Shechtman Daniel Cohen-Or Dani Lischinski. *StyleCLIP: Text-Driven Manipulation of StyleGAN Imagery*. 2021. arXiv: 2103.17249 [cs.CV].

[12] Ziwei Liu et al. "Deep Learning Face Attributes in the Wild". In: *Proceedings of International Conference on Computer Vision (ICCV)*. 2015.

[13] Léon Bottou Martin Arjovsky Soumith Chintala. "Wasserstein GAN". In: *Arxiv.org* (2017). DOI: https://arxiv.org/abs/1701.07875.

[14] Simon Osindero Mehdi Mirza. "Conditional Generative Adversarial Nets". In: *Arxiv.org* (2014). DOI: https://arxiv.org/abs/1411.1784.

[15] Han Zhang Tao Xu Hongsheng Li Shaoting Zhang Xiaogang Wang Xiaolei Huang Dimitris Metaxas. "StackGAN: Text to Photo-realistic Image Synthesis with Stacked Generative Adversarial Networks". In: *Arxiv.org* (2016). DOI: https://arxiv.org/abs/1612.03242.

[16] Negar Rostamzade Seyedarian Hosseini Thomas Boquet Wojciech Stokowiec Ying Zhang Christian Jauvin Chris Pal. "Fashion-Gen: The Generative Fashion Dataset and Challenge". In: *Arxiv.org* (2018). DOI: https://arxiv.org/abs/1806.08317.

[17] Ashish Vaswani Noam Shazeer Niki Parmar Jakob Uszkoreit Llion Jones Aidan N. Gomez Lukasz Kaiser Illia Polosukhin. "Attention Is All You Need". In: *Arxiv.org* (2017). DOI: https://arxiv.org/abs/1706.03762.

[18] Yuzhen Lu Fathi M. Salem. "Simplified Gating in Long Short-term Memory (LSTM) Recurrent Neural Networks". In: *Arxiv.org* (2017). DOI: https://arxiv.org/ftp/arxiv/papers/1701/1701.03441.pdf.

[19] Mike Schuster and Kaisuke Nakajima. "Japanese and Korean voice search". In: *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2012, pp. 5149–5152. DOI: 10.1109/ICASSP.2012.6289079.

[20] Rico Sennrich, Barry Haddow, and Alexandra Birch. *Neural Machine Translation of Rare Words with Subword Units*. 2016. arXiv: 1508.07909 [cs.CL].

[21] Aditya Ramesh Mikhail Pavlov Gabriel Goh Scott Gray Chelsea Voss Alec Radford Mark Chen Ilya Sutskever. "Zero-Shot Text-to-Image Generation". In: *Arxiv.org* (2021). DOI: https://arxiv.org/abs/2102.12092.

[22] Alec Radford Jong Wook Kim Chris Hallacy Aditya Ramesh Gabriel Goh Sandhini Agarwal Girish Sastry Amanda Askell Pamela Mishkin Jack Clark Gretchen Krueger Ilya Sutskever. "Learning Transferable Visual Models From Natural Language Supervision". In: *Arxiv.org* (2021). DOI: https://arxiv.org/abs/2103.00020.

[23] Christian Szegedy et al. *Rethinking the Inception Architecture for Computer Vision*. 2015. arXiv: 1512.00567 [cs.CV].

[24] Harm de Vries Florian Strub Jérémie Mary Hugo Larochelle Olivier Pietquin Aaron Courville. "Modulating early visual processing by language". In: *Arxiv.org* (2017). DOI: https://arxiv.org/abs/1707.00683v3.

APPENDIX A
TRAINING ALGORITHMS

In this appendix we present the original algorithms used for creating the training loop for the different trainings performed on the architectures. The only change that was implemented on this algorithms was to add the text feature extraction from the pre-trained BERT architecture at the same time we sample a batch of random noise and a sample of real images.

Algorithm 1 Minibatch stochastic gradient descent training of generative adversarial nets. The number of steps to apply to the discriminator, k, is a hyperparameter. We used k = 1, the least expensive option, in our experiments.

Require: The gradient coefficient λ, the number of critic iteration n_{critic}, the batch size m, Adam hyperparameters α, β_1, β_2.
Require: initial critic parameters ω_0, initial generator parameters θ_0

for Number of training iterations **do**
  for k steps **do**
    Sample minibatch of m noise samples $z^1, ..., z^m$ from noise prior $p_g(z)$
    Sample minibatch of m samples $x^1, ..., x^m$ from data generating distribution $p_{data}(x)$
    Update the discriminator by ascending its stochastic gradient:
      $\nabla_{\theta_d} \frac{1}{m} \sum_{i=1} m[logD(x^i) + log(1 - D(G(z^i)))]$
  end for
  Sample minibatch of m noise samples $z^1, ..., z^m$ from noise prior $p_g(z)$
  Update the discriminator by ascending its stochastic gradient:
    $\nabla_{\theta_d} \frac{1}{m} \sum_{i=1} mlog(1 - D(G(z^i)))$
end for
The gradient-based updates can use any standard gradient-based learning rule. We used Adam in our experiments with $\alpha = 0.0002$, $\beta_1 = 0$ and $\beta_2 = 0.9$

Algorithm 2 Wasserstein GAN. All experiments in the paper used the default values α = 0.00005, c = 0.01, m = 64, n_{critic} = 5.

Require: α, the learning rate. c, the clipping parameter, m, the batch size. n_{critic}, the number of the critic's iterations per generator iteration.
Require: initial critic parameters ω_0, initial generator's parameters θ_0

while θ has not converged **do**
  for $t = 1$, ..., n_{critic} **do**
    Sample real data ${x^i}_{i=1}^m \sim \mathbb{P}_r$
    Sample ${z^i}_{i=1}^m \sim p(z)$ a batch of prior samples
    $g_\omega \leftarrow \nabla_\omega [\frac{1}{m} \sum_{i=1}^m f_\omega(x^i) - \frac{1}{m} \sum_{i=1}^m f_\omega(g_\theta(x^i))]$
    $\omega \leftarrow \omega + \alpha * RMSprop(\omega, g_\omega)$
    $\omega \leftarrow clip(\omega, -c, c)$
  end for
  Sample ${z^i}_{i=1}^m \sim p(z)$ a batch of prior samples.
  $\theta \leftarrow -\nabla_\theta \frac{1}{m} \sum_{i=1}^m f_\omega(g_\theta(z^i))$
  $\theta \leftarrow \theta - \alpha * RMSprop(\theta, g_\theta)$
end while

Algorithm 3 WGAN with Gradient Penalty

Require: The gradient coefficient λ, the number of critic iteration n_{critic}, the batch size m, Adam hyperparameters α, β_1, β_2.
Require: initial critic parameters ω_0, initial generator parameters θ_0

while θ has not converged **do**
  for $t = 1$, ..., n_{critic} **do**
    for $i = 1$, ..., m **do**
      Sample real data $x \sim \mathbb{P}_r$, latent variable $z \sim p(z)$, a random number $e \sim U[0, 1]$
      $\hat{x} \leftarrow G_\theta(z)$
      $\hat{x} \leftarrow \epsilon x + (1 - \epsilon)\hat{x}$
      $L^i \leftarrow D_\omega(\hat{x}) - D_\omega(x) + \lambda(||\nabla_{\hat{x}} D_{omega}(\hat{x})||_2 - 1)^2$
    end for
    $\omega \leftarrow$ Adam$(\nabla_m \frac{1}{m} \sum_{i=1}^{m}) L^i \omega, \theta, \alpha, \beta_1, \beta_2$
  end for
  Sample batch of latent variables ${z^i}_{i=1}^m \sim p(z)$.
  $\theta \leftarrow$ Adam$(\nabla_\theta \frac{1}{m} \sum_{i=1}^{m}) L^i - D_\omega(G_\theta(z)), \theta, \alpha, \beta_1, \beta_2$
end while