ABSTRACT

Any work done will be appreciated only when it has the uniqueness, which is not there among the already existing ones. Individuals opt for the best possible choice in almost all aspects of their life, and they search and choose a product that suits them best. To achieve this , we need to provide them with various set of possibilities as per their expectations, and sometimes also based on the requirement of the particular aspect. In this world, which habituated on seeing revolutionary services extending from day to day, one of the field which runs only on uniqueness is fashion technology. This project is focused on producing some notable changes in designing handbags, one of the fashion product, with the help of Generative Adversarial Networks (GAN).

In this project, a framework, namely GAN for estimating generative models via an adversarial process, during which we simultaneously train two models, where one is a generative model that captures the information of distribution, and other is a discriminative model that estimates the probability that a sample came from the training data rather than generator. The training procedure for generator is to maximize the probability of discriminator making an error. An open data set namely Shirts, handbags corresponding to a data of 740 handbag images, taken from kaggle, is fed into the generator network. And the generator is trained with some fake images and tries to generate some images from that information. This data is further sent to discriminator which discriminates the images generated with the real images, thus learning the features on working over a learning rate of 0.0003 and trained over for 50 epochs.

From this analysis, a set of generated images are produced, exhibiting some sort of uniqueness, having the same form as of original data and thus providing nearly desired results, i.e., a new item which may not actually existed before, is generated. This project is executed in python 3.7 through Google Colab. This concept can be further extended to the generation of any item and in any field.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF ABBREVIATIONS

GAN	Generative Adversarial Network

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

The main purpose of this project is focused on producing changes in designing handbags, one of the fashion accessories, with the help of Generative Adversarial Network (GAN).

1.2 MOTIVATION

It would be really awesome, if we can generate new products with the existing ones alone, without any need of modifications. This exactly is one of the major applications of Generative Adversarial Network (GAN) and we feel really fascinated to work on this.

1.3 PROBLEM DEFINITION

We here use the GAN (Generative Adversarial Networks) to design the hand bag model which is already fed to the discriminator and its. We use Adam optimizer with certain configurations and we train the entire network with a learning rate of 0.003 and for over 50 epochs. And finally, we display loss of discriminator, generator for each epoch along with real loss and fake.

1.4 GOALS AND OBJECTIVES

The basic GAN model is composed of the input vector, generator, and discriminator. Among them, the generator and discriminator are implicit function expressions, usually implemented by deep neural networks. GAN can learn the generative model of any data distribution through adversarial methods with excellent performance. It has been widely applied to different areas since it was proposed in 2014. In this review, we introduced the origin, specific working principle, and development history of GAN, various applications of GAN in digital image processing, Cycle-GAN, and its application in medical imaging analysis, as well as the latest applications of GAN in medical informatics and bioinformatics.

1.5 SCOPE AND APPLICATION

To generate new handbag images by creating and training a Convolutional Neural Network namely discriminator and a Deconvolutional Neural Network namely generator, with the shirts-handbags dataset of 740 handbag images, which altogether forms a Generative Adversarial Network (GAN).

CHAPTER 2
LITERATURE REVIEW

2.1 INTRODUCTION

In this chapter, each of the previous work related to this project will be discussed by papers which are summarized as below.

Impact of Fashion Industry on World

According to the statistics, In 2019, global retail sales of apparel and footwear reached 1.9 trillion U.S. dollars, and were expected to rise to above three trillion U.S. dollars by 2030. The fashion industry continues to have positive growth, especially in emerging markets within the Asia-Pacific and European regions.

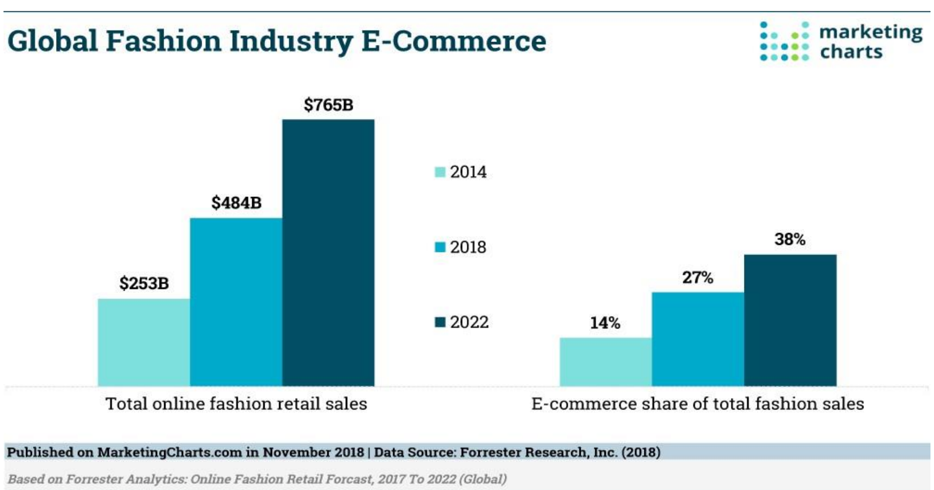

Figure 2.1 E Commerce growths in Fashion Industry

India is becoming increasingly a focal point for the fashion industry, reflecting a rapid growth in middle class and an increasingly powerful manufacturing sector. Indeed, it is to be noted that India's ascent is one of ten trends the fashion industry should watch in

2019, highlighted in our latest State of Fashion` report, written in partnership with the Business of Fashion (BoF).

India's apparel market will be worth $59.3 billion in 2022, making it the sixth largest in the world, comparable to the United Kingdom's ($65 billion) and Germany's ($63.1 billion), according to data from McKinsey's Fashion Scope.

Coming to the priority given by the women on fashion accessories apart from garments, on focusing upon one of the most used accessories, the handbags, the statistics gave shocking infact some interesting results.

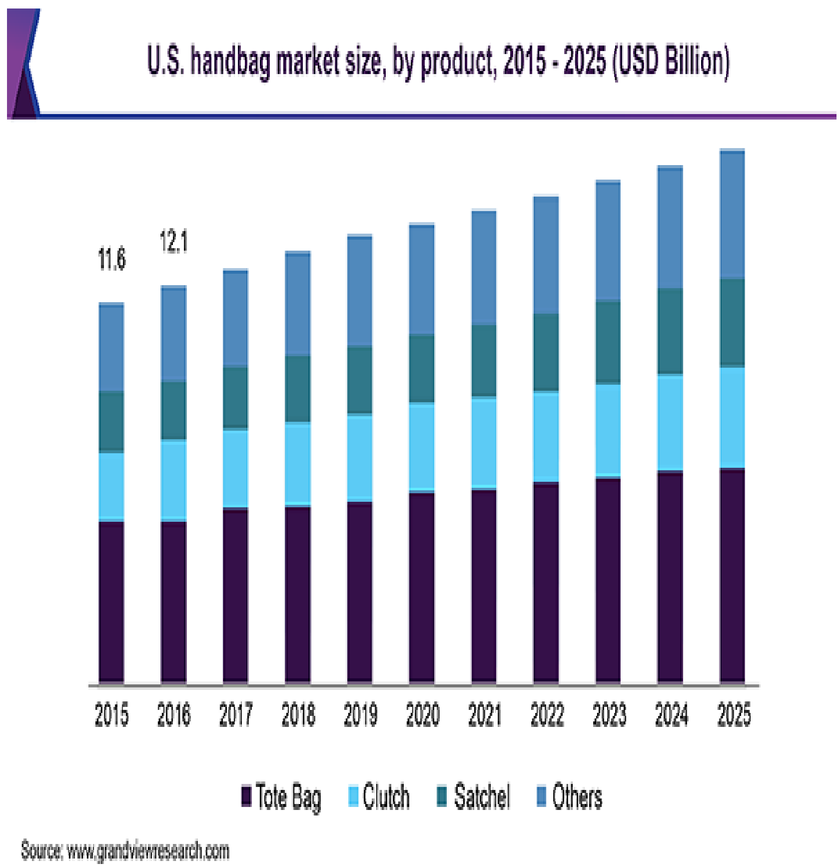

Figure 2.2 Statistics on handbags-1

For most women their handbag is a multi-tasking device that combines the virtues of practicality and utility, along with showing off personal taste. It suggests a certain economic prosperity and acts as a soupcon of the childhood security blanket. While women may often play it safe, either due to workplace convention or simply their lack of interest in clothes and so choose to dress unremarkably in their daily life, these same women will frequently have a standout handbag.

2.2 LITERATURE REVIEW

2.2.1Fashion GAN: Display your fashion design using Conditional Generative Adversarial Nets

An end-to-end virtual garment display method based on Conditional Generative Adversarial Networks was proposed in this paper. Virtual garment display plays an important role in the fashion design technology, for it can directly show the design effect of the garment without having to make a sample garment like traditional clothing industry.[1]

Different from existing 3D virtual, this method only need users to input a desired fashion sketch and a specified fabric image then the image of the virtual garment whose shape and texture are consistent with the input fashion sketch and fabric image can be shown out quickly and automatically.

This FashionGAN is workable for single-color and regular (like stripe) fabric pattern, but it cannot map an irregular one.. Compared with the existing image-to-image methods, the quality of images generated by this method is better in terms of color and shape. This FashionGAN is workable for single-color and regular (like stripe) fabric pattern, but it cannot map an irregular one.

2.2.2 Inclusive GAN: Improving Data and Minority Coverage in Generative Models

In this paper, the problem of minority inclusion as one of data coverage and improved data coverage using a novel paradigm that harmonizes adversarial training (GAN) with reconstructive generation was formalized [2]. The method discussed in this paper outperforms state-of-the-art methods in terms of Precision, Recall, and some other things, and the improvement generalizes well on unseen data. And also the method to ensure explicit inclusion for minority subgroups at little or no compromise on overall full-dataset performance was further extended. It was concluded here that the improvement of all our minority models comes at little or no compromise from their performance on the overall dataset.

2.2.3 A Review on Generative Adversarial Networks: Algorithms, Theory, and Applications

Generative adversarial networks (GANs) are a hot research topic currently. GANs have been widely studied since 2014, and a large number of algorithms have been proposed. However, there is a few comprehensive study explaining the connections among different GANs variants, and how they have evolved [3]. An attempt to provide a review on various GANs methods from the perspectives of algorithms, theory, and applications is made in this paper. Firstly, the motivations, structure of most GANs algorithms and mathematical representations, were in details. Furthermore, GANs have been combined with other machine learning algorithms for specific applications, such as semi-supervised learning, transfer learning, and reinforcement learning. This paper compares the commonalities and differences of these GANs methods.

Secondly, theoretical issues related to GANs are investigated. Thirdly, typical applications of GANs in image processing and computer vision, natural language processing, music, speech and audio, medical field, and data science are illustrated. Finally, the future open research problems for GANs are pointed out.

2.2.4 GENERATIVE ADVERSARIAL NETS

A new framework for estimating generative models via an adversarial process, in which two models are simultaneously trained namely: a generative model G that captures the data distribution, and a discriminative model D that estimates the probability that a sample came from the training data rather than G was produced [4]. The training procedure for G is to maximize the probability of D making a mistake. This framework corresponds to a minimax two-player game. Here, backpropagation alone is used to train the entire system and there is no usage of any Markov chains or unrolled approximate inference networks during either training or generation of samples. Experiments demonstrate the potential of the framework through qualitative and quantitative evaluation of the generated samples. This new framework comes with advantages and few notable disadvantages relative to previous modelling frameworks. The primary disadvantage is that there is no explicit representation of pg(x), and that D must be synchronized well with G during training.

2.2.5 Toward AI fashion design: An Attribute-GAN model for clothing match

This paper investigated the clothing match problem under the cGAN framework. Unlike the extant fashion mining literature, in which style is usually classified according to similarity, this paper investigates clothing match rules based on semantic attributes according to the generative adversarial network (GAN) model [5]. Specifically, an Attribute-GAN to generate clothing-match pairs automatically was proposed. To implement the Attributed-GAN, which constitutes training a generator, supervised by an adversarial trained collocation discriminator and attribute discriminator, a large-scale outfit dataset was built by themselves and annotated clothing attributes manually. Besides the advantage of higher image quality generation, Attribute-GAN achieved the best diversity of synthetic images and a matching degree of generated clothing outfits, owing to the generalization capability.

CHAPTER 3

REQUIREMENT ANALYSIS

3.1 SOFTWARE REQUIREMENT

- Python 3.7
- Google Colab

3.2 HARDWARE REQUIREMENT

- 4Gb RAM
- WINDOWS 8

CHAPTER 4
METHODOLOGY AND IMPLEMENTATION

4.1Working of GAN

GANs contains two models: a generator and a discriminator. These two models are typically implemented by neural networks, but they will be implemented with any form of differentiable system that maps data from one space to the opposite. The generator tries to capture the distribution of true examples for brand spanking new data example generation.

The discriminator is typically a binary classifier, discriminating generated examples from truth examples as accurately as possible.

Given a training set, this system learns to get new data with an equivalent statistics because the training set, for instance, a GAN trained on photographs can generate new photographs that check out least superficially authentic to human observers, having many realistic characteristics. Though originally proposed as a sort of generative model for unsupervised learning, GANs have also proven useful for semi-supervised learning, fully supervised learning and reinforcement learning. The core idea of a GAN is predicated on the "indirect" training through the discriminator, [clarification needed] which itself is additionally being updated dynamically. This basically means the generator isn't trained to attenuate the space to a selected image, but rather to fool the discriminator. this permits the model to find out in an unsupervised manner.

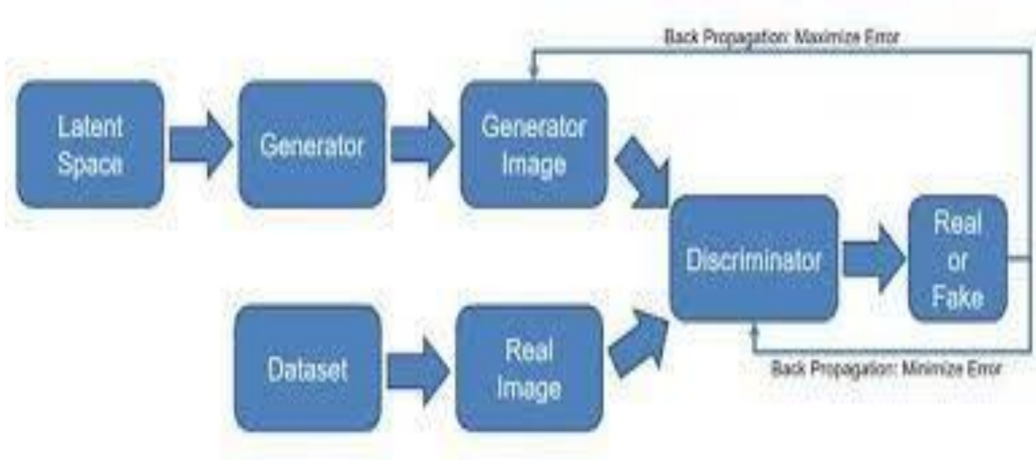

Figure 4.1 Block diagram of GAN

A known dataset is the initial training data for the discriminator. Training it involves presenting it with samples from the training dataset, until it achieves acceptable accuracy. The generator trains supported whether it succeeds in fooling the discriminator. Typically the generator is seeded with randomized input that's sampled from a predefined latent space (e.g. a multivariate normal distribution). Thereafter, candidates synthesized by the generator are evaluated by the discriminator. Independent back propagation procedures are applied to both networks in order that the generator produces better samples, while the discriminator becomes more skilled at flagging synthetic samples. When used for image generation, the generator is usually a deconvolutional neural network, and therefore the discriminator may be a convolutional neural network

Deep neural networks are used mainly for supervised learningssuch as classification or regression. Generative Adversarial Networks or GANs, however, uses neural networks for a very different and interesting purpose namely Generative modeling.

Generative Adversarial Networks or GANs as for short, are an approach to generative modeling using deep learning methods, such as convolutional neural networks.

Generative modeling is an unsupervised learning task in machine learning which involves automatically discovering and learning the regularities or patterns in input data in such a way that the model can be used to generate or output new examples that plausibly could have been drawn from original dataset.

GANs are a clever way of training the generative model by framing the problem as a supervised learning problem with two sub-models: the generator model that we train to generate new examples, and the discriminator model which tries to classify examples as either real (from the domain) or fake (generated).

GANs are really an exciting and rapidly changing field, delivering on the promise of generative models in their ability to generate realistic examples across a range of problem domains, most notably in image-to-image translation tasks such as translating photos of summer to winter or day to night, and in generating photorealistic photos of objects, scenes, and people that even humans cannot tell are fake. So now we are using this same approach for generating new designs in the fashion domain.

There are two neural networks called a Generator and a Discriminator. The generator generates a "fake" sample which is given a random vector/matrix, and the discriminator attempts to detect whether a given sample is "real" (picked from the training data) or "fake" (generated by the generator). Training happens in random and we train the discriminator for a few epochs, then train the generator for a few epochs, and repeat. This way both the generator and the discriminator gets gradually better at doing their jobs.

GANs however, may be notoriously difficult to train, and are extremely sensitive to hyper parameters, activation functions and regularization. So, we need to make careful choices on the right parameters.

So, as GAN comes under the process of unsupervised learning, we don’t keep any labels on that data and as told earlier we are using a kaggle data set namely Shirtshandbags and we take those data in the form of batches and we train the network with these batches.

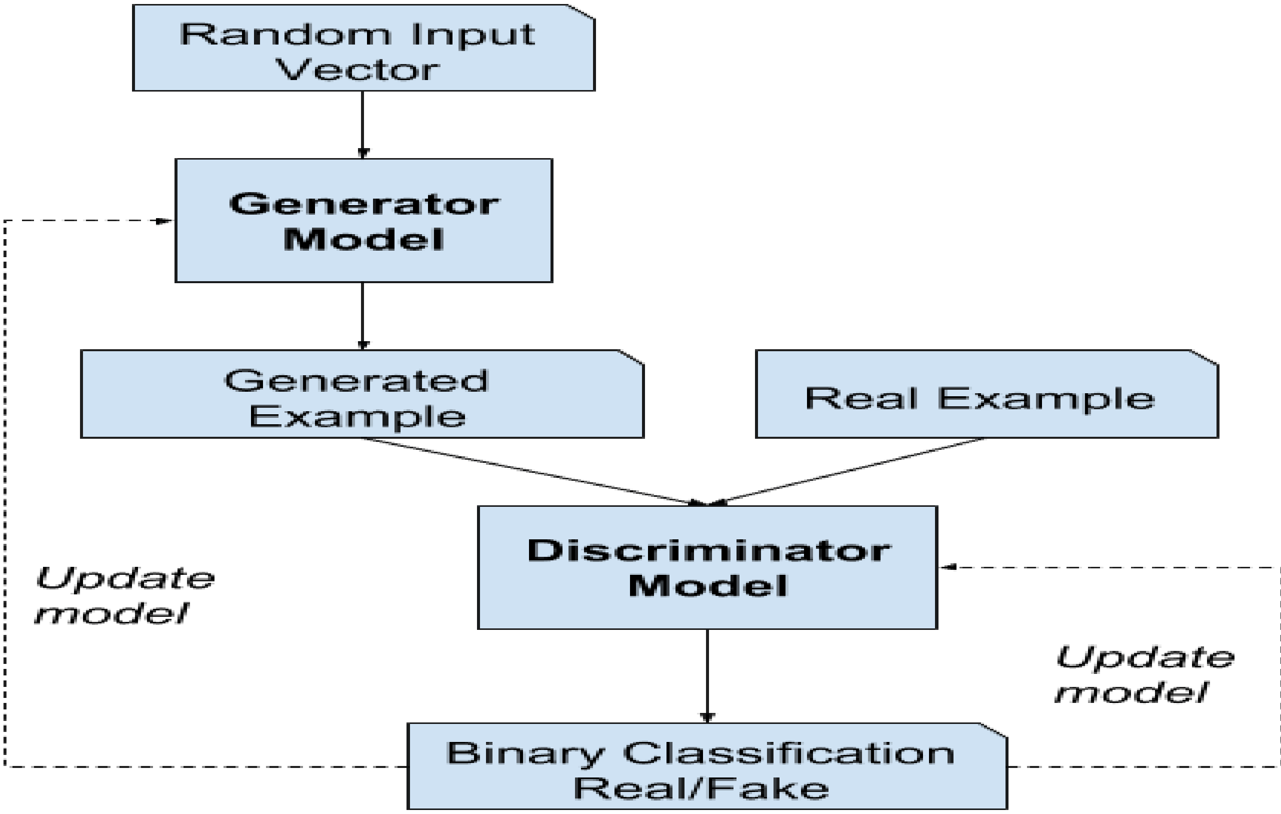

Figure 4.2 Architecture of GAN

4.2GANs and Convolutional Neural Network

GANs typically work with image data and use Convolutional Neural Networks, or CNNs, for the generator and discriminator models.

The reason for this is that may be both because the first description of the technique was in the field of computer vision and used CNNs and image data, and because of the remarkable progress that has been seen in recent years using CNNs more generally to achieve state-of-the-art results on a suite of computer vision tasks such as object detection and face recognition.

Modelling image data means that the latent space, the input to the generator, provides a compressed representation of the set of images or photographs used to train the network model. It also means that the generator generates new images or photographs, providing an output that can be easily viewed and assessed by the developers or users of the model.

It may be this fact above others, that the ability to visually assess the quality of generated output, has both led to the focus of computer vision applications with CNNs and on the massive leaps in the capability of GANs as compared to other generative models, deep learning based or otherwise.

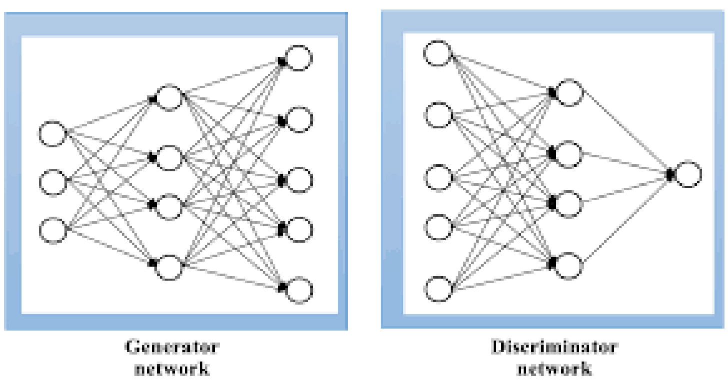

Figure 4.3 Generator and Discriminator Neural Networks

4.3 Data Pre-processing

The start of the process is done by naming the project and downloading the data set from kaggle using open data sets library into google colab. The images from data set will then be resized into 64x64 pixels which makes the task slightly easier while dealing with larger images and we are also going to normalize the pixel values with a mean and standard deviation of 0.5.

Here the purpose is to chose the images with pixel intesnities of 0 to 1 and to translate them to the range of -1 to 1 for symmetrical distribution around origin which helps to easily train the discriminator . And now a data loader is being created with batch size of 128, also using torch vision transforms we will resize the images and center crop them and take them to the tensors of range (-1,1).

Figure 4.4 Shirtshandbags Data set used for training

Now some helper functions are used here to denormalize the images. And some more helper functios to take the batch of images from dataloader and display them in the form of a grid. Now, using device data loader we will wrap the data and use it for training.

4.4 Discriminator Neural Network

The loaded data is then fed into the discriminator first. The discriminator takes an image as input, and it tries to classify it as "real" or "generated". In this sense, it's like some other neural network. We'll use convolutional neural networks (CNN) which outputs a single number output for each and every image.

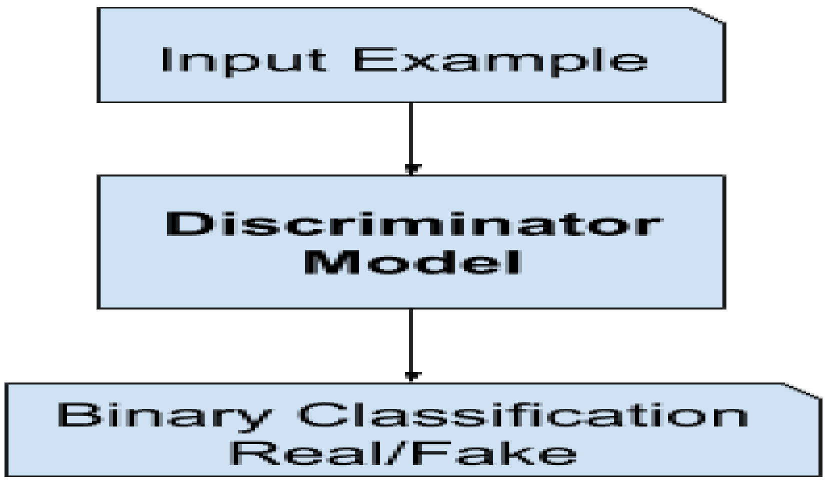

Figure 4.5GAN Discriminator Network

This image will be a 3 channel 64 pixel image. The number which comes as an output from the discriminator, will be regarded as the probability that the discriminator thinks the image is drawn from the actual data set. This comes under binary classification. And then we'll use stride of 2 to progressively reduce the size of the output feature map.

The discriminator model takes example from domain as input (real or generated) and predicts a binary class label of real or fake (generated).The real example comes from training dataset. The generated examples are output by generator model.

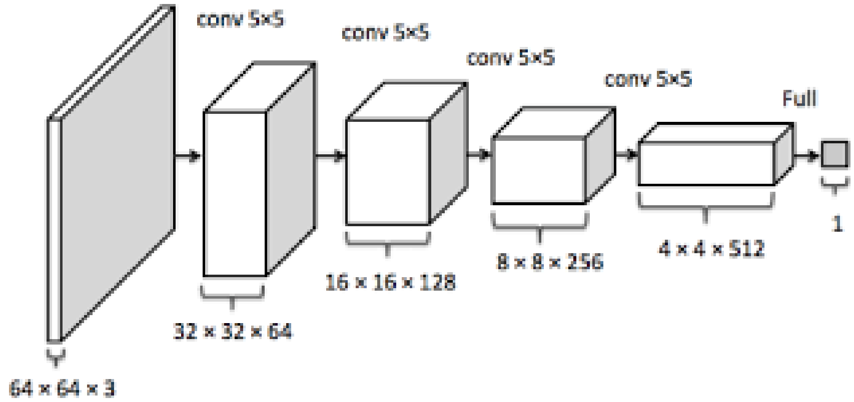

Figure 4.6 Layers of GAN Discriminator Network

Here, we are choosing a sequential model for both generator and discriminator. First of all, a batch of images of dimension 3x64x64 is fed as an input to first convolution layer of kernel size of 4 and stride 2 with zero padding and using leaky rectified linear unit activation function,is then fed to max pool layer. And the output of this layer will be reduced to half and becomes 32x32 feature map with doubling the channels to 64. So, this same process of reducing images size to half and doubling the channels will continue from 3x64x64 to 512x4x4. Finally after the 4th convolution layer where the size is of 4x4 and consists of 512 channels , we will convert into the size of 1x1 using kernel size of 4 and using zero padding the channels will be reduced from 512 to 1.

$3 \times 64 \times 64 \rightarrow 64 \times 32 \times 32 \rightarrow 128 \times 16 \times 16 \rightarrow 256 \times 8 \times 8 \rightarrow 512 \times 4 \times 4 \rightarrow 1 \times 1 \times 1$

Now, the output image of dimensions 1 x 1 x 1 tensor is flattened to a single vector of size 1 and finally sigmoid activation function is applied to it.

Leaky ReLU is a slightly modified version of ReLU, as it allows small negative region to pass through it. Different from the regular ReLU function, Leaky ReLU allows the pass of a small gradient signal for negative values. As a result, it makes the gradients from the discriminator flows stronger into the generator. Instead of passing a gradient (slope) of 0 in the back-prop pass, it passes a small negative gradient.

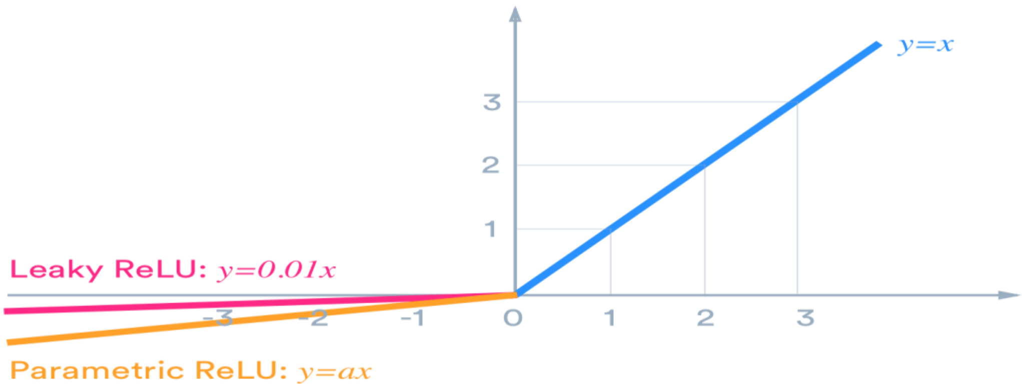

Figure 4.7 Leaky ReLU Activation Function

The discriminator is a normal (and well understood) classification model. After the training process, the discriminator model is discarded as we are interested in the generator. Sometimes, the generator can be repurposed as it has learned to effectively extract features from examples in problem domain. Some or all of the feature extraction layers can be used in transfer learning applications using the same or similar input data.

4.5 Generator Neural Network

The generator model is the one which takes a fixed-length random vector as input and generates a sample in the domain. The vector is drawn randomly from a Gaussian distribution, and the vector is used to seed the generative process. After training, points in this multidimensional vector space will correspond to points in problem domain, forming a compressed representation of the data distribution.

This vector space is referred to as a latent space, or a vector space which is comprised of latent variables. Latent variables, or hidden variables, are those variables that are so important for a domain but that are not directly observable.

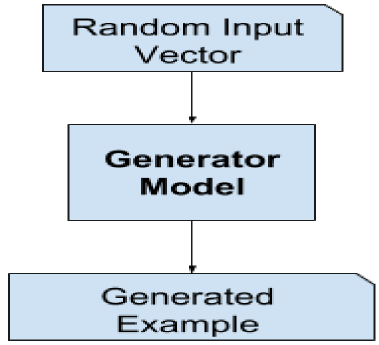

Figure 4.8.GAN Generator Network

Often we refer to latent variables, or a latent space, as a projection or compression of a data distribution. That is, a latent space provides a compression or high-level concepts of observed raw data such as the input data distribution. In the case of GANs, the generator model applies meaning to points in a chosen latent space, such that new points drawn can be provided to the generator model as input and used to generate new and different output examples from the latent space.After training, the generator model is kept and used to generate new samples.

The input to the generator is typically a vector or a matrix of random numbers (referred to as a latent tensor) which is used as a seed for generating an image. The generator will convert a latent tensor of shape (128, 1, 1) into an image tensor of shape 3 x 28 x 28. To achieve this, we'll use the ConvTranspose2d layer from PyTorch, which performs as a transposed convolution (also referred to as a deconvolution).

Even here as discussed earlier, a generator network is also a sequential neural network starting with a latent size of 128.The basic idea behind the latent size is that , if there is just one vector , the controlling will be only on the one aspect of the output. Roughly, Some combinations of the latent vectors controls the specific attribute of the generated image, once the generator has been trained. So the more latent vectors we have the more variety we can get for the images. Typically 64 or 128 is enough for latent size and gor bigger detailed images more higher values are preferred.

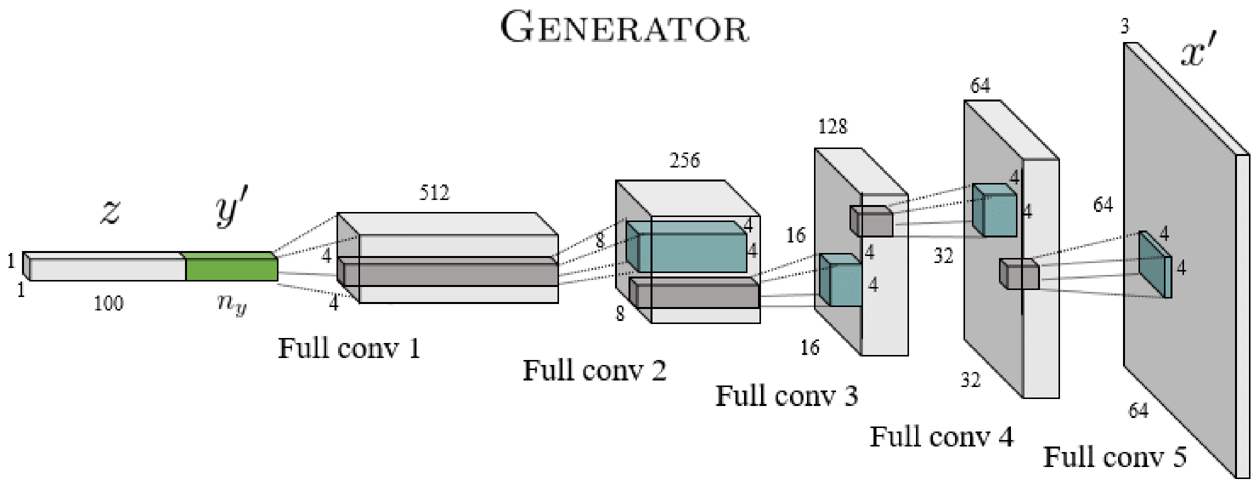

Figure 4.9 Layers of GAN Generator Network

Now the generator takes a latent vector which is of size 128 x 1 x 1.This ia a batch of latent tensors not just a single latent tensor. So it takes a latent tensor and uses a convolution 2D layer to read transpose. And we go from latent size channels to 512 channels, and then using kernel size of 4 and stride of 1, we convert the image shape from 1 x 1 to 4 x 4. For the generator we are using Relu activation function and this process continues for all the other next upcoming layers. So we will once again send this to another convolution 2D layer, where the channel size is halved from 512 to 256 and the image size gets multiplied by 2 for each layer. And this same process with kernel size of 4 and stride of 2 and with padding 1 is continued once again from 256 to 128 and image size of 16 x 16 from 8 x 8. And this will continue for a few times and will end up with channel size of 3 and image shape of 64 x 64←using kernel size of 4 and using zero padding the channels will be reduced from 512 to 1.

1 x 1 x 1 → 512 x 4 x 4 → 256 x 8 x 8 → 128 x 16 x 16 → 64 x 32 x32 → 3 x 64 x 64

And another thing we notice is that we have here Tanh or hyperbolic tangent is used as activation function. The hyperbolic tangent function reduces the values into the range -1 to 1.So the outputs of the generator are going to be images with the pixel values of range -1 to 1 and of the shape

3 x 64 x 64. So we have setup the generator in such a way that it is going to generate the images that atleast have same ranges of pixel intensities as the discriminator.

4.6 Training the Generator, Discriminator and the whole Network

For the given data, the discriminator network may be a standard convolutional network which will categorize the pictures that are fed there to a binomial classifier labeling images as real or fake. The generator is an inverse convolutional network, during a sense, while a typical convolutional classifier takes a picture and down samples it to supply a probability, the generator takes a vector of random noise and up samples it to a picture, the primary one throws away data through down sampling techniques like max pooling, and therefore the second generates new data.

We use torch.randn to create random tesnsors of batch size 128 with tensors of latent size 128 and we pass this batch of data into generator. This generator converts each latent sensor into an image of size 3 x 64 x 64. This batch of data gets converted and gets denormalized. And when we display them we get random noise as the networks are not trained yet.

First we train the discriminator and then with the help of that we will train generator. Since the discriminator is binary classification model , we can use binary cross entropy loss function to quantify how well it differentiate between real and generated images. Generally we use this function in case of multiple classes in this case, for a single class it is used.

Now, we are going to train discriminator with one batch of real images. Firstly we clear out the gradients in the discriminator and then real images are passed to the discriminator. For each image present in the batch , a single number will come as output in the range of (0,1). The target for the real image is 1.

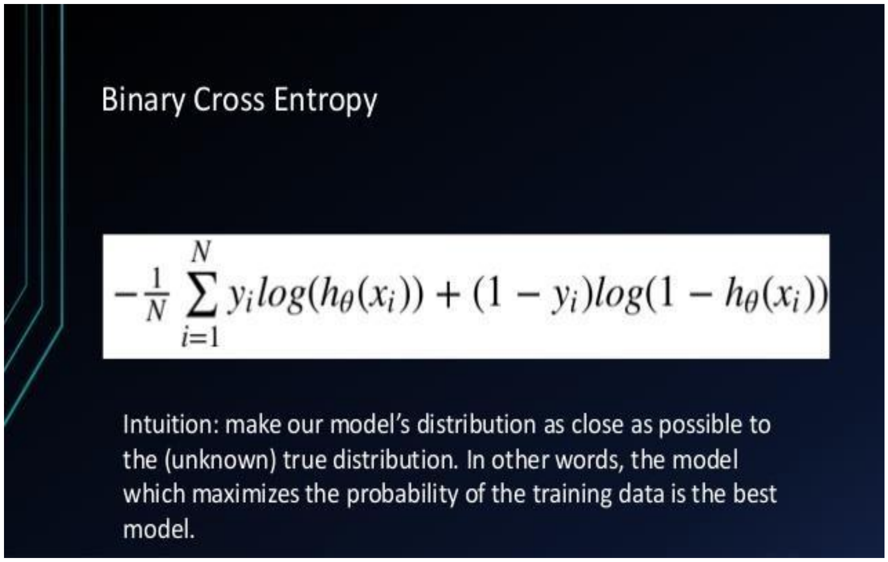

Figure 4.10 Binary Cross Entropy Loss Function

So the discriminator has to target those which are real images. Now we pass in the real predictions made by discriminator and the target values as 1 for the cross entropy loss function and the loss from this function .While training the discriminator , the discriminator is expected to give output 1 if the image was picked from the real dataset, and 0 if it was generated using the generator network.

We first pass a batch of real images, and compute the loss, setting the target labels to 1.Then we pass a batch of fake images (generated using the generator) pass them into the discriminator, and compute the loss, setting the target labels to 0. Finally the two losses are added and the overall loss to perform gradient descent to adjust the weights of the discriminator is used. It's important to note that the weights of the generator model is not changed while training the discriminator.

And we will take the average of the real predictions to get the real score. And the other part is that , the same process is repeated with the fake images and the targets here for these images is 0 and the predictions will be taken on passing these fake images into the discriminator and passing these into the loss function we get fake loss and on taking mean for these values, we get fake score. The fake score is to be as low as possible. This acts as another evaluation metric here. And the overall loss is obtained by adding real loos obtained by passing batch of real images and fake loss by passing a batch of fake images. And then we perform gradient descent to modify the weights of discriminator.

Now coming to the generator, which generates the input to the discriminator. Since the outputs of the generator are images, it's not obvious how we can train the generator. So we generate batch of images using generator by creating some random latent tensors and put them back into generator This is where we employ a rather elegant trick, which is to use the discriminator as a part of the loss function. We generate a batch of images using the generator, pass the into the discriminator. We calculate the loss by setting the target labels to 1 i.e. real. We do this because the generator's objective is to "fool" the discriminator. We use the loss to perform gradient descent i.e. change the weights of the generator, so it gets better at generating real-like images to "fool" the discriminator.

And before going to train the full loop for the intermediate checking we will be going with storing the generated images at each epoch. So in a overview, first the discriminator is trained for a batch of real and generated examples whose targets are 0 and 1 respectively and then the loss will be calculated using binary cross entropy and update discriminator, so that it gets better at making this differentiation .And then we take random inputs and put them into generator and these generated images are fed to discriminator and then try to train the generator such that the discriminator predicts 1 i.e., to fool the discriminator. We use Adam optimizer with certain configurations and we train the entire network with a learning rate of 0.003 and for over 50 epochs. And finally we display loss of discriminator, generator for each epoch along with real loss and fake loss. However we need the generator loss to be low. And for each epoch the output is displayed.

CHAPTER 5

RESULTS AND DISCUSSIONS

So after the network got trained, as told earlier generated images at each epoch is taken. Some of those are attached to gain the inference about changes and developments of the output after each epoch. Here generated images from the GAN are given after 1st,10th ,20th,30th,40th,45th and 50th epochs.

When we observe the images at each epoch , we can find the betterment in the output produced as epochs increased. Here after experimenting with different learning rates , depending upon the clarity of the output, I have chosen learning rate as 0.003 and trained the network for over 50 epochs.

From the 1st epoch's output to 50th epoch output , we can observe that noise level is also decreased along with the increment in the clarity. But also we need to chose the hyperparameter, learning rate in this case, wisely and also the training epochs as it leads to some time overfitting in the model, when we chose large values and also gives raise to more noise and underfitting, in case of smaller values. Especially in GAN, hyperparameters plays a key role.

```
Image('./generated/generated-images-0001.png')
```

```
Image('./generated/generated-images-0010.png')
```

```
Image('./generated/generated-images-0020.png')
```

```
Image('./generated/generated-images-0030.png')
```

Figure 5.1 Generated Output images at different Epochs

Comparison between original input and output from GAN

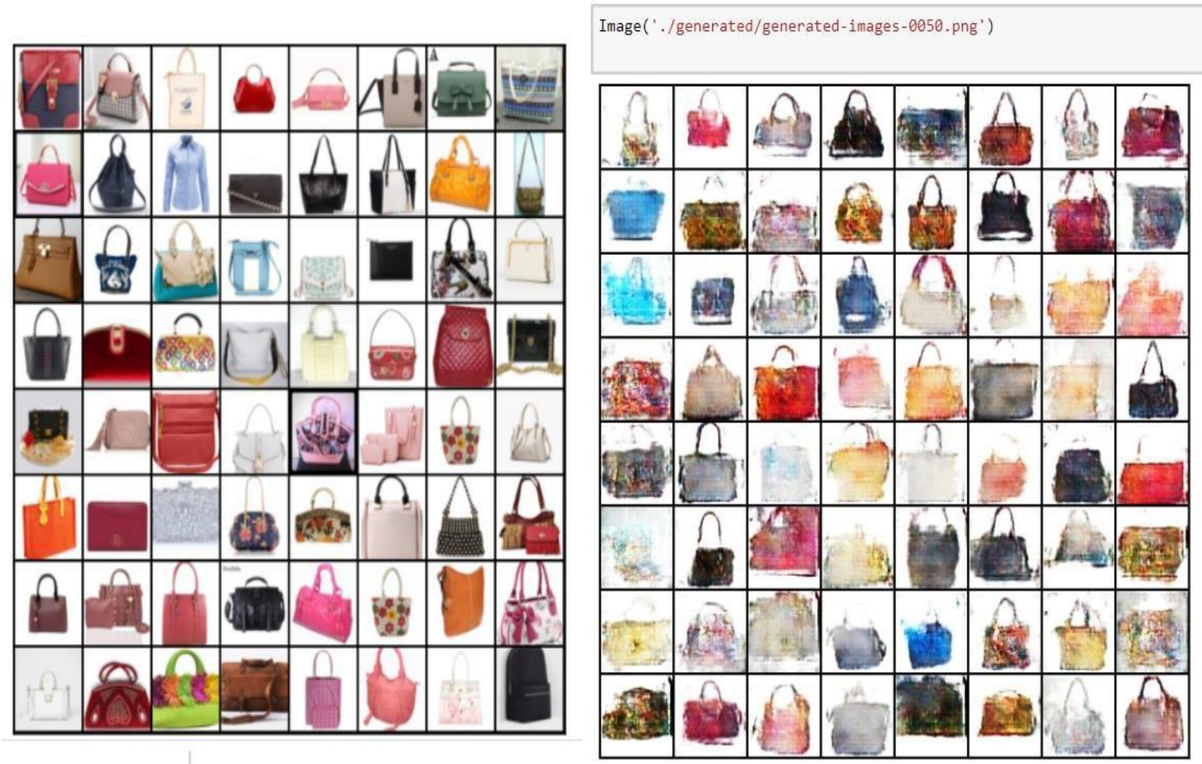

Figure 5.2 Comparison between Input and Final output

We can note that the output here is almost similar to that of input but has different models than that of the input. So in a nut shell, the GAN network we trained has learnt the features of handbags from the unlabelled data consisting of different types of handbags given, and also from those features it has produced new batch of images which may or may not be exactly in the original dataset.

When we train this model further for different learning rate, different number of epochs and for different activation functions after thorough experimentation, we may also get a better output than this. So we can also improve the output here upon proper choices of all these factors.

The details of the outputs at some epochs is as shown below.

Epoch [1/50], loss_g: 5.3326, loss_d: 0.1942, real_score: 0.9274, fake_score: 0.0979

Epoch [10/50], loss_g: 5.8759, loss_d: 0.0908, real_score: 0.9403, fake_score: 0.0256

Epoch [20/50], loss_g: 3.8008, loss_d: 1.5762, real_score: 0.3327, fake_score: 0.0010

Epoch [30/50], loss_g: 5.2242, loss_d: 0.0603, real_score: 0.9696, fake_score: 0.0276

Epoch [40/50], loss_g: 7.0624, loss_d: 0.4008, real_score: 0.7514, fake_score: 0.0049

Epoch [45/50], loss_g: 4.9674, loss_d: 0.0855, real_score: 0.9496, fake_score: 0.0243

Epoch [50/50], loss_g: 4.9264, loss_d: 0.0942, real_score: 0.9232, fake_score: 0.0086

CHAPTER 6

CONCLUSION AND FUTURE ENHANCEMENT

Evaluation of GAN's performance will be an important area to explore. Over a few years, applications of the Generative Adversarial Networks (GANs) have seen an astounding growth. This technique has been successfully used for high-fidelity natural image synthesis, data augmentation tasks, improving image compressions, and many more. From emoting super- realistic expressions to exploring deep space, and from bridging the human-machine empathetic disconnect to introducing new art forms, GANs have it all covered enough.

There are numerous domains where GANs are used such as generating examples for Image Datasets, Generate Photographs of Human Faces, Generate Realistic Photographs, Generate Cartoon Characters, Image-to-Image Translation, Text-to-Image Translation, Semantic-Image- to-Photo Translation, Face Frontal View Generation, Generate New Human Poses, Photos to Emojis, Photograph Editing, Face Aging, Photo Blending, Super Resolution, Photo Impainting, Clothing Translation, Video Prediction and 3D Object Generation and many more.

The same method I used in my work, can be extended over any kind of dataset and especially in the domains which needs new innovations and unique items and luckily one of the coolest things is that we can generate innovations using GANs with the ones existing. I have chosen fshion technology as it is one if the domains which need innovations and I have chosen one of the items in it i.e., handbags.

GANs are powerful tools that help to give the power of imagination to AI. GAN's are crucial to many different state of the art image generation and manipulation systems, and have the potential to enable many other applications in the future.

CHAPTER 7

CONTRIBUTIONS

S. Priyanka has made the report describing about the project, the reason for producing the project, the goals that the project tries to achieve, the areas where the project can be applied and the problem that it tries to solve including requirement analysis with the system architecture and the terms needed to be explained and has also added screenshots of the output to the appendix also added the code in the report and also added the references to the papers that are described in the literature review section.

B. Naga Rajeswari has added the drawings of the system design and described the parts of the architecture of the project and their functionality. The flowchart was added about the project and provided the description of how the project works and the list of URLs of webpages of the project. The conclusions and the future scope for the project was also added to the report.

B. Greeshmanth Reddy has added the summary of the five research papers in the area of usage of Web Applications and machine learning to fight the COVID-19 pandemic using various methods such as displaying information, prediction, discovering medicines, etc.

REFERENCES

[1] [1]"FashionGAN: Display your fashion design using Conditional Generative Adversarial Nets"https://www.researchgate.net/publication/328505097_FashionGAN_Display_your_fashion_design_using_Conditional_Generativ e_Adversarial_Nets . Accessed 20 June, 2021

Yirui, Cui & Liu, Q. & Gao, C. & Su, Zhuo. (2018). FashionGAN: Display your fashion design using Conditional Generative Adversarial Nets. Computer Graphics Forum. 37. 109-119. 10.1111/cgf.13552

[2] [2]"Inclusive GAN: Improving Data and Minority Coverage in Generative Models" . 23 Aug 2020 https://arxiv.org/pdf/2004.03355.pdf Accessed 20 June, 2021

[3] [3]"A Review on Generative Adversarial Networks: Algorithms, Theory, and Applications" .20 Jan 2020 https://arxiv.org/pdf/2001.06937.pdf Accessed 20 June, 2021

[4] [4]"Generative Adversarial Nets". 23 Aug 2020https://arxiv.org/pdf/1406.2661.pdf Accessed 10 June, 2014

[5] Toward AI fashion design: An Attribute-GAN model for clothing match" .12 March 2019 https://openreview.net/pdf/b718118254f91dbb5fb5f578dbf2496724ff010d.pdf Accessed 20 June, 2021

APPENDIX A

CODE SNIPPETS

```
project_name = 'hand-bag-new'
!pip install opendatasets --upgrade –quiet
import opendatasets as od
od.download('https://www.kaggle.com/bashturtle/shirtshandbags')
import os
data_dir='./shirtshandbags'
print(os.listdir(data_dir))
from torch.utils.data import DataLoader
from torchvision.datasets import ImageFolder
import torchvision.transforms as T
import torch
from torchvision.utils import make_grid
import matplotlib.pyplot as plt
%matplotlib inline
im_sz = 64
batch_sz = 128
new_stats = (0.5, 0.5, 0.5), (0.5, 0.5, 0.5)
train1 = ImageFolder(data_dir, transform=T.Compose([
    T.Resize(im_sz),
    T.CenterCrop(im_sz),
    T.ToTensor(),
    T.Normalize(*new_stats)]))

train2 = DataLoader(train1, batch_sz, shuffle=True, num_workers=3, pin_memory=True)
def denormalize(im_tensors):
    return im_tensors * new_stats[1][0] + new_stats[0][0]
def show_img(images, nmax=64):
    fig, ax = plt.subplots(figsize=(8, 8))
    ax.set_xticks([]); ax.set_yticks([])
    ax.imshow(make_grid(denormalize(images.detach()[:nmax]), nrow=8).permute(1, 2, 0))

def show_batch(dl, nmax=64):
    for images, _ in dl:
        show_img(images, nmax)
        break
show_batch(train2)

def get_device():
```

```
    if torch.cuda.is_available():
        return torch.device('cuda')
    else:
        return torch.device('cpu')

def to_device(data, device):
    if isinstance(data, (list,tuple)):
        return [to_device(x, device) for x in data]
    return data.to(device, non_blocking=True)

class DeviceDataLoader():
    def __init__(self, dl, device):
        self.dl = dl
        self.device = device

    def __iter__(self):
        for b in self.dl:
            yield to_device(b, self.device)

    def __len__(self):
        return len(self.dl)
device=get_device()
train2 = DeviceDataLoader(train2, device)
import torch.nn as nn
#discriminator network
discriminator = nn.Sequential(
    nn.Conv2d(3, 64, kernel_size=4, stride=2, padding=1, bias=False),  # in: 3 x 64 x 64
    nn.BatchNorm2d(64),
    nn.LeakyReLU(0.2, inplace=True),

    nn.Conv2d(64, 128, kernel_size=4, stride=2, padding=1, bias=False),  # out: 64 x 32 x 32
    nn.BatchNorm2d(128),
    nn.LeakyReLU(0.2, inplace=True),

    nn.Conv2d(128, 256, kernel_size=4, stride=2, padding=1, bias=False),   # out: 128 x 16 x 16
    nn.BatchNorm2d(256),
    nn.LeakyReLU(0.2, inplace=True),

    nn.Conv2d(256, 512, kernel_size=4, stride=2, padding=1, bias=False),   # out: 256 x 8 x 8
    nn.BatchNorm2d(512),
    nn.LeakyReLU(0.2, inplace=True),

    nn.Conv2d(512, 1, kernel_size=4, stride=1, padding=0, bias=False), # out: 512 x 4 x 4
```

```
    nn.Flatten(), # out: 1 x 1 x 1
    nn.Sigmoid())
discriminator = to_device(discriminator, device)
latent_size=128
#generator network
generator = nn.Sequential(
    # in: latent_size x 1 x 1

    nn.ConvTranspose2d(latent_size, 512, kernel_size=4, stride=1, padding=0, bias=False),   # out:
512 x 4 x 4
    nn.BatchNorm2d(512),
    nn.ReLU(True),

    nn.ConvTranspose2d(512, 256, kernel_size=4, stride=2, padding=1, bias=False),        # out: 256
x 8 x 8
    nn.BatchNorm2d(256),
    nn.ReLU(True),

    nn.ConvTranspose2d(256, 128, kernel_size=4, stride=2, padding=1, bias=False),        # out: 128
x 16 x 16
    nn.BatchNorm2d(128),
    nn.ReLU(True),

    nn.ConvTranspose2d(128, 64, kernel_size=4, stride=2, padding=1, bias=False),        # out: 64 x
32 x 32
    nn.BatchNorm2d(64),
    nn.ReLU(True),

    nn.ConvTranspose2d(64, 3, kernel_size=4, stride=2, padding=1, bias=False),        # out: 3 x 64
x 64
    nn.Tanh()

)
xb = torch.randn(batch_sz, latent_size, 1, 1) # random latent tensors
fake_images = generator(xb)
print(fake_images.shape)
show_img(fake_images)
generator = to_device(generator, device)
def train_discriminator(real_images, opt_d):
    opt_d.zero_grad()
    real_preds = discriminator(real_images)
    real_targets = torch.ones(real_images.size(0), 1, device=device)
```

```
    real_loss = F.binary_cross_entropy(real_preds, real_targets)
    real_score = torch.mean(real_preds).item()
    latent = torch.randn(batch_sz, latent_size, 1, 1, device=device)
    fake_images = generator(latent)
    fake_targets = torch.zeros(fake_images.size(0), 1, device=device)
    fake_preds = discriminator(fake_images)
    fake_loss = F.binary_cross_entropy(fake_preds, fake_targets)
    fake_score = torch.mean(fake_preds).item()
    loss = real_loss + fake_loss
    loss.backward()
    opt_d.step()
    return loss.item(), real_score, fake_score
def train_generator(opt_g):
    opt_g.zero_grad()
    latent = torch.randn(batch_sz, latent_size, 1, 1, device=device)
    fake_images = generator(latent)
    preds = discriminator(fake_images)
    targets = torch.ones(batch_sz, 1, device=device)
    loss = F.binary_cross_entropy(preds, targets)
    loss.backward()
    opt_g.step()

    return loss.item()
from torchvision.utils import save_image
sample_dir = 'generated'
os.makedirs(sample_dir, exist_ok=True)
def save_samples(index, latent_tensors, show=True):
    fake_images = generator(latent_tensors)
    fake_fname = 'generated-images-{0:0=4d}.png'.format(index)
    save_image(denormalize(fake_images), os.path.join(sample_dir, fake_fname), nrow=8)
    print('Saving', fake_fname)
    if show:
        fig, ax = plt.subplots(figsize=(8, 8))
        ax.set_xticks([]); ax.set_yticks([])
        ax.imshow(make_grid(fake_images.cpu().detach(), nrow=8).permute(1, 2, 0))
fixed_latent = torch.randn(64, latent_size, 1, 1, device=device)
save_samples(0, fixed_latent)
from tqdm.notebook import tqdm
import torch.nn.functional as F
def fit(epochs, lr, start_idx=1):
    torch.cuda.empty_cache()

    # Losses & scores
```

```
    losses_g = []
    losses_d = []
    real_scores = []
    fake_scores = []

    # Create optimizers
    opt_d = torch.optim.Adam(discriminator.parameters(), lr=lr, betas=(0.5, 0.999))
    opt_g = torch.optim.Adam(generator.parameters(), lr=lr, betas=(0.5, 0.999))

    for epoch in range(epochs):
        for real_images, _ in tqdm(train2):
            # Train discriminator
            loss_d, real_score, fake_score = train_discriminator(real_images, opt_d)
            # Train generator
            loss_g = train_generator(opt_g)

        # Record losses & scores
        losses_g.append(loss_g)
        losses_d.append(loss_d)
        real_scores.append(real_score)
        fake_scores.append(fake_score)

        # Log losses & scores (last batch)
        print("Epoch [{}/{}], loss_g: {:.4f}, loss_d: {:.4f}, real_score: {:.4f}, fake_score: {:.4f}".format(
            epoch+1, epochs, loss_g, loss_d, real_score, fake_score))

        # Save generated images
        save_samples(epoch+start_idx, fixed_latent, show=False)

    return losses_g, losses_d, real_scores, fake_scores
lr = 0.0003
epochs =50
history = fit(epochs, lr)
losses_g, losses_d, real_scores, fake_scores = history
torch.save(generator.state_dict(), 'G.pth')
torch.save(discriminator.state_dict(), 'D.pth')
from IPython.display import Image
Image('./generated/generated-images-0001.png')
```

APPENDIX B

SCREENSHOTS

```
Image('./generated/generated-images-0050.png')
```

Figure 5.3 Final Output