⟨Investigation of text-based image manipulation of human face using StyleGAN and CLIP: mitigating gender bias from StyleCLIP using DCNN⟩

⟨Suhaibur Rehman⟩

MSc ⟨Master of Science⟩ in ⟨Data Science⟩
The University of Bath
⟨2021-22⟩

⟨Investigation of text-based image manipulation of human face using StyleGAN and CLIP: mitigating gender bias from StyleCLIP using DCNN⟩

Submitted by: ⟨Suhaibur Rehman⟩

Copyright

Declaration

This dissertation is submitted to the University of Bath in accordance with the requirements of the degree of Master of Science in the Department of Computer Science. No portion of the work in this dissertation has been submitted in support of an application for any other degree or qualification of this or any other university or institution of learning. Except where specifically acknowledged, it is the work of the author.

Abstract

Text-based image synthesis is a promising technique for advancing the field of image manipulation. In this dissertation, we investigate the implementation of text-based manipulation of human faces using StyleGAN and Contrastive Language-Image Pre-training (CLIP). We combined the generative power of StyleGAN and the image-text mapping ability of CLIP using global direction technique. We identified 2 major challenges in the StyleCLIP method, non-target feature manipulate and gender bias in the generated face image. We explore the problem of gender bias in the generated image and combine the discriminator of StyleGAN with a deep convolutional network capable of classifying gender to mitigate the gender bias. We propose this new combination as "FaceGAN", a novel method that is capable of manipulating human face images using text prompts from user without incorporating any gender bias.

Contents

List of Figures

List of Tables

Acknowledgements

I would like to use this opportunity to express my gratitude to my supervisor, **Professor James Davenport**, for his guidance and insightful suggestions during the planning and implementation of this project. I would also like to thank my supervisor, **Joe Goodier**, for organising regular session and supporting with the technical and coding aspect of this project. Finally, I want to express my gratitude to my family for providing much needed moral support throughout my studies.

Chapter 1

Introduction

This chapter provides a brief introduction to the problem statement and the motivation to address it. This is followed by the research question being answered, and then we outline the structure of the dissertation.

1.1 Motivation

This dissertation investigates the text-based image manipulation of human faces using StyleGAN and CLIP. We explore the challenges posed by the current model and combine it with a neural network capable of transforming the wide array of human face image without these challenges. We call this resulting method focused solely on human face image as FaceGAN. This dissertation extends the work of Patashnik et al. (2021) who combined CLIP model with StyleGAN to develop a text-based interface for image manipulation and thus reduced the effort to collect an annotated collection of images for each desired image manipulation.

Patashnik et al. (2021) removed issues such as speed and disentanglement raised by combining StyleGAN and CLIP using a latent optimization approach to modify the latent mapper input according to the user-provided text prompt and the latent mapper approach to allow faster image manipulation. However as demonstrated by Woolf (2021), the current model is still not able to address challenges like non-target feature manipulation and gender bias raised within StyleGAN and CLIP.

Woolf (2021) shows that if a feature corresponding to the target text vector is missing from the face, then the generator tries to cheat by manipulating the latent mapper other than the target vector to obtain an image similar to the desired image, which sometimes results in the transformation away from the target text vector, generating an image completely opposite of the prompt. Another challenge is of model bias where the AI assigns gender to gender neutral terms, like giving prompts like makeup and teacher shifts the image features of male towards that of female.

Motivated by works like [Schwemmer (n.d.), Tang et al. (2020)] to remove gender bias from different machine learning methods, we will investigate an unexplored approach to mitigate gender bias from the face image generated using StyleCLIP. We will combine the StyleGAN and CLIP models using global direction technique to transform the textual direction in CLIP space into a direction in StyleGAN space. As StyleGAN uses latent vector to generate the images, we will use the encoder designed by Tov et al. (2021) to invert the images from FFHQ

dataset to latent mappers. Then, we will train the model on the above data. Finally, we will study the efficacy of the Deep-CNN method in mitigating the gender bias of the generated image.

Given the result from the above study, we will combine the CNN with StyleGAN to propose a novel method called "FaceGAN" capable of manipulating human face images using text prompts from user without incorporating any gender bias. With the advent of social media apps such as Instagram, Face-App and Snapchat, people's curiosity to experiment and share the effect of new features on their face has increased (Smink et al. (2020)). This model if incorporated with these apps can provide the users an effective face styling feature without the need of specialized editing skills.

1.2 StyleCLIP challenges

As Woolf (2021) pointed out the cons in the StyleCLIP, we will test the current model to give an overview of these challenges. For testing purpose, we will use random images from `https://thispersondoesnotexist.com/` which generates a high quality portraits of non-existent humans using StyleGAN2 (Karras et al. (2019)). This will help to maintain the ethics and privacy of the dissertation.

1.2.1 Non-target feature manipulation

The figure 1.1 represents a non-existent female generated from the website `https://thispersondoesnotexist.com/`. As shown in 1.1, the ears of the female is covered by her hair, due to which the StyleCLIP is not able to successfully manipulate the target vector for the given text prompt "Face with big ears" and instead manipulates the features other than target vector by reducing the density of hair on the sides to depict as if the ear size has increased. This way the generator cheats with the discriminator and is able to convince the latter that the generated image is the one desired.

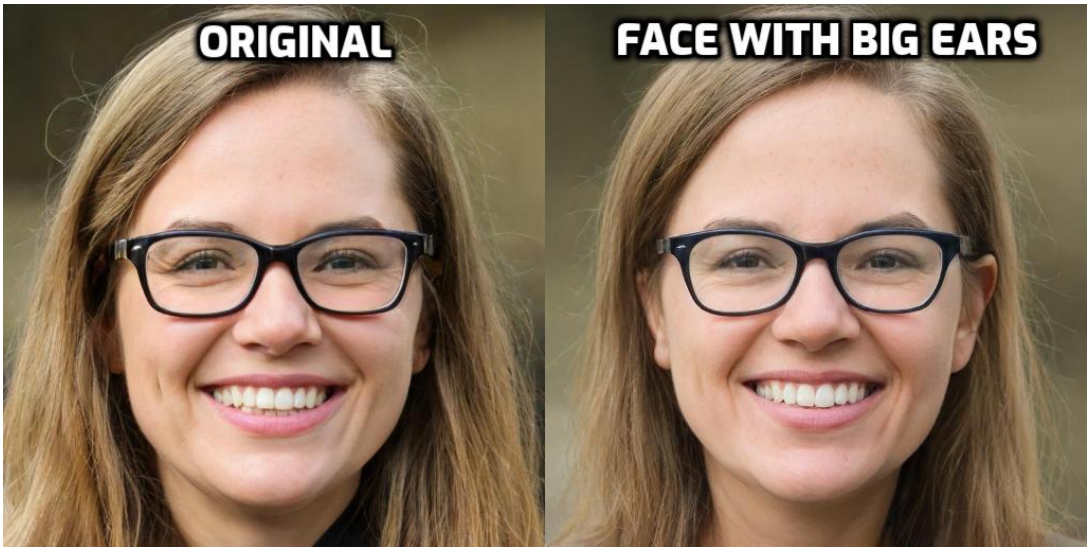

Figure 1.1: Text prompt: "Face with big ears", Param: strength - 4.3, disentanglement - 0.11. Input image source: `https://thispersondoesnotexist.com/`

However, not all the negative manipulation is due to StyleGAN, some of these opposite alphas are due to CLIP as well. In the figure 1.2, when given text prompt "face with closed mouth", "Face with mouth shut" or "remove smile", the generated image was completely opposite of what was desired. The encoder in the CLIP model, instead of mapping the complete text prompt, mapped only keywords "mouth" or "smile" with the image into a common latent space. Any text prompt containing the similar words was giving the similar transformation even though it was complete opposite of the desired manipulation.

Figure 1.2: Text prompt: "closed mouth", param: strength - 4.44, disentanglement - 0.15. Input image source: `https://thispersondoesnotexist.com/`

1.2.2 Gender Bias

The strength of CLIP model trained on web scraped data is inextricably linked to the manner in which the data was curated, demonstrated by Gal (2021). Due to the disproportionate representation in the millions of image-text pairs on which the CLIP model was pre-trained, the generated images shows bias for cases where these image-text pairs was dominated by one class. In figure 1.3, the input text prompt "Bodybuilder face" shifts the facial features of the female towards the male. The possible explanation for the mustache, beard hair, and strong manly feature in the generated facial image is the domination of males in the bodybuilding field, thus the increased representation of image-text pairs of males than females in the dataset. Similar results were obtained when prompts such as "police officer" and "sailor" were given.

1.2.3 Proposed approach to remove gender bias

In this dissertation, we will focus on mitigating the gender bias of the current model. Since the CLIP model is already trained on a million image-text pairs, it will be manually intensive to annotate new images for it to train, so we will try to remove this issue via StyleGAN. We will compare the performance of the DCNN model trained on the UTKFace dataset in predicting the gender of the original and generated image. If the performance is satisfactory, we will combine it with StyleGAN discriminator to verify if the gender is similar in the original and generated image. To quote Brownlee (2020), Founder of `https://machinelearningmastery.com/`,

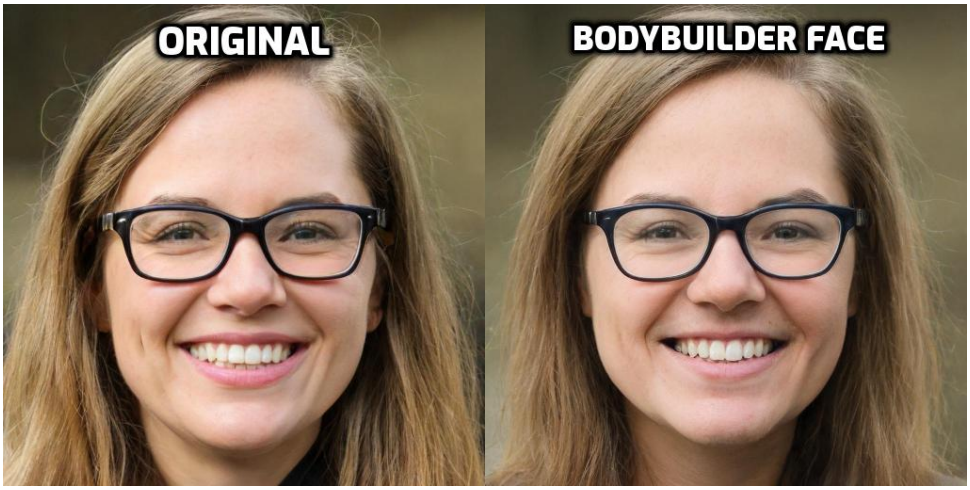

Figure 1.3: Text prompt: "Bodybuilder Face", param: strength - 4.58, disentanglement - 0.15. Input image source: `https://thispersondoesnotexist.com/`

"Many improvements to the GAN architecture have been made possible by modifications to the discriminator model".

These modifications are introduce with the belief that a better discriminator model will force the generator to produce gender accurate image.

1.3 Dissertation structure

The structure of the dissertation is as follows. Chapter 2 focuses on the literature and technology survey, Chapter 3 specifies the requirement of the project, Chapter 4 explains the method used in this project, Chapter 5 provides the detail about the results, and Chapter 5 concludes this dissertation with scope of possible future work.

Chapter 2

Literature and Technology Survey

In this chapter, we present a literature survey describing GANs and their applications in text-image synthesis. Related work using CLIP, StyleGAN, and work related to mitigate gender bias from machine learning model will be discussed.

2.1 From Auto-Encoder to Generative Models

In 2013, Diederik Kingma and Max Welling introduced generative encoders that were able to generate new instances that look similar to the training sample (Kingma and Welling, 2013). These encoders were known as variational autoencoders (VAE) as they performed variational Bayesian inference, an efficient way to perform the approximate Bayesian inference technique (Beal, 1970). The latent loss in VAE is given by:

$$\mathscr{L} = -\frac{1}{2}\sum_{i=1}^{n}\left[1 + \log\left({\sigma_i}^2\right) - {\sigma_i}^2 - {\mu_i}^2\right]$$

Figure 2.1: Equation from (Kingma and Welling, 2013)

L is the latent loss, n is the dimensionality of the codings, and μ_i and σ_i are the mean and standard deviation of the i^{th} component of the codings. The encoder outputs the vectors μ and σ.

Although the VAE were easier to train and the sampling process was faster than the restricted Boltzmann machine, the generative adversarial networks eventually took the lead, as they were able to generate more realistic and high-quality images (Bond-Taylor et al., 2021).

2.1.1 Generative Adversarial Networks

The Generative Adversarial Network was proposed by Goodfellow et al. (2014). GAN comprises of two neural networks: a Generator and a Discriminator. The generator takes the latent representation of the image to be generated as input while the discriminator takes the image from generator or training set as an input and predict if the image is fake or real. The loss function in GANs is given by:

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

Figure 2.2: Equation from Goodfellow et al. (2014)

The resolution of images generated by GAN has improved a lot [Dhawan and Kumar (2020), Brock, Donahue and Simonyan (2018)] but the difficulty in training GAN due to 'instability, modal collapse, and non-convergence" still remains (Saxena and Cao, 2021).

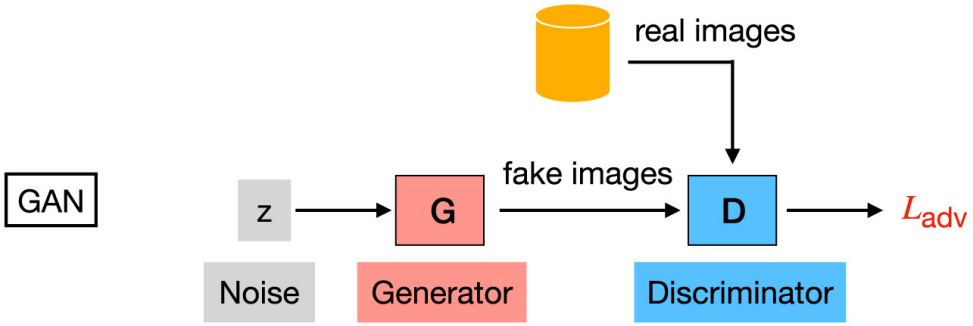

Figure 2.3: Figure 1.3: Architecture of GAN (Frolov et al., 2021). Given a random sample of noise from a normal distribution, the generator is trained to generate images that mislead the discriminator. The discriminator is trained to recognize actual and produced images.

Recent Application of GANs

The current application of GAN includes image to image translation ((Isola et al., 2017), Zhu et al. (2017a)), super-resolution (Ledig et al. (2017)), video prediction (Kwon and Park, 2019), creative AI art (Mazzone and Elgammal, 2019), text-based image translation (Tatariants, 2022), data augmentation (Frid-Adar et al., 2018), representation learning (Bengio, Courville and Vincent (2013), Donahue and Simonyan (2019)), and style transfer (Gatys, Ecker and Bethge (2016), Jing et al. (2019)). In this chapter, we will focus on application of GANs in text-based image manipulation.

2.2 Text-based image manipulation

Picture captioning is a common and difficult subject in Natural Language Processing and Computer Vision: given an image, a written description of the image must be provided. Text to picture synthesis is the inverse problem: given a text description, an image must be generated that fits that description.

From a high level, these issues are not different from language translation issues. Images and text are two separate languages" to encode related information, just as equivalent semantics might be represented in two different languages.

Nonetheless, they are fundamentally separate difficulties since text-image or image-text conversions are highly multi-modal. If one tries to translate a basic line like "This is a magnificent red flower" into French, there aren't many possible translations. If one tries to conjure up a mental image of this description, there are a plethora of images that could be conjured up.

Though similar multi-modal behavior is prevalent in image captioning challenges, the problem is simplified because language is typically sequential. This structure is used to conditioning the production of fresh words on previously created words. As a result, word to image synthesis is a more difficult task than image captioning.

Once the technology is ready for commercial applications, image synthesis from natural language has a wide range of potential applications. Instead of spending hours searching for the desired design, people might manufacture custom furniture for their homes by simply describing it to a computer. Material makers could work more closely with a machine to create content using natural language (Bodnar, 2018).

2.2.1 Review of Text-image manipulation

Initial work in text-image synthesis

The initial work in text-based image manipulation was done by Reed et al. (2016). It enhanced conditional GANs to create natural images from text descriptions and was demonstrated to function on limited datasets (e.g., Oxford-102 Flowers (Nilsback and Zisserman, 2008) and CUB-200 Birds (Wah, n.d.)) of limited image resolution (64 $\times$ 64 pixels). Mansimov et al. (2015) introduced AlignDRAW, a variational recurrent autoencoder based on DRAW introduced in Gregor et al. (2015), to create the image in several phases by responding to relevant words. While the AlignDRAW approach is not adversarial, the authors utilised a GAN in post-processing to sharpen the output images. Recently, this discipline has achieved significant gains in picture quality (based on qualitative and quantitative evaluation criteria), dataset complexity (Lin et al., 2014), and image resolution (e.g., 256 x 256 and higher). Approaches such as enhanced text encodings, loss-terms developed expressly for text-to-image synthesis, and unique architectures are examples of these developments (e.g., stacked networks and attention). Furthermore, quantitative evaluation metrics (e.g., R-precision, Visual-Semantic similarity, and Semantic Object Accuracy) have been introduced specifically to evaluate the quality of text-to-image synthesis models.

Conditional GANs in text-image synthesis

To generate images of birds and flowers from specific text instructions, researchers trained a deep convolutional generative adversarial network (DC-GAN) conditioned on text attributes recorded by a hybrid character-level convolutional recurrent neural network. The discriminator and the generator both used feed-forward inference based on the text feature. Dong et al. (2017) built an end-to-end conditional GAN framework on image and text descriptions to perform image synthesis on CalTech's bird and flower datasets by outputting realistic images corresponding to input text descriptions while retaining features of input images irrelevant to text descriptions. The generator used an encoder and decoder architecture, while the discriminator conducted a distinguishing task based on text semantic properties and used an unique loss function to maximise the generator's learning for image synthesis.

RNN, LSTM and Reinforcement learning for text-image synthesis

FashionGAN in (Zhu et al., 2017b) proposed splitting the generative process into two phases to enhance text-based picture synthesis on the DeepFashion dataset. A semantic segmentation map was created in the first step to obey the wearer's pose as a latent spatial arrangement, while a generative model with a composition mapping layer was used in the second step to

render the final image with precise regions and textures conditioned on the segmentation map. In Chen et al. (2017), recurrent attentive models were used to fuse image and language features in an automatic language-based image system. To perform language based image segmentation and picture colorization, the suggested framework includes a convolutional image encoder, an LSTM text encoder, a fusion network, a deconvolutional network, and an optional convolutional discriminator. Kim et al. (2019) demonstrated a novel use of text-guided image alteration by creating a collaborative picture-Drawing game between two agents using natural language processing, neural networks, and reinforcement learning.

Novel methods for text-image generation application

Using a deep neural network built of language short-term memory as a language encoder and variational auto-encoder, GAN, and pixelCNN as an image encoder, an interactive image-based manipulation was produced on the MNIST dataset in (Shinagawa et al., 2018). Bahng et al. (2018) proposed a model called Text2Colors that consists of two conditional generative adversarial networks: text-to-palette generation networks and palette-based colorization networks, to colourize a grayscale image according to the colour palette generated from the input texts. The former extracts the semantics of the text and generates appropriate colour palettes. The latter uses the created colour palette to colourize a grayscale image. On the Caltech flower dataset, a multi conditional GAN was constructed in (Park, Yoo and Kwak, 2018), and high-resolution images were obtained by adding stackGAN. This work was expanded in (Liu et al., 2018) by implementing a novel model for semantic picture synthesis called Conditional Cycle-Generative Adversarial Network (CCGAN). Based on word descriptions, this model can produce a photo-realistic image. This model considers both text descriptions and original photos, in contrast to previous text-to-image algorithms, which generally disregard information from the original images. Han et al. (2018) published an improved technique to synthesis images based on text description by employing a novel loss function and several input pairs running through a resolution refinement sub-network to generate high-resolution images.

Text-based manipulation of human images

Zhou et al. (2019) released a text-guided person image modification that changed the stance and attribute of a person's image in two stages based on the input text description. A text-guided pose generator infers a plausible pedestrian pose from the description in the first stage, and a visual appearance transferred image approach synthesises a realistic and appearance transferred person image from the text in combination with the target pose in the second stage. In Li et al. (2019a), a novel method called controlGAN was proposed that effectively synthesises high-quality images while also controlling sections of the image synthesis based on natural language descriptions. To do this, the model incorporated a word-level spatial and channel-wise attention-driven generator that can separate multiple visual features and allow the model to focus on creating and manipulating subregions corresponding to the most relevant words. The ManiGAN approach Li et al. (2019b), which comprises of two important components: text-image affine combination module (ACM) and detail correction module, was used to do text-based image editing (DCM). The ACM chooses image regions that are related to the given text and then correlates them with associated semantic words for effective manipulation storing original image attributes to aid in the reconstruction of text-irrelevant contents. The DCM corrects mismatched properties and fills in gaps in the synthetic image's information.

Text-based manipulation of human face

Text-based image manipulation on the human face has recently been accomplished, as seen in (Xia et al., 2020), where a novel framework called TediGAN presented multi-modal picture production and manipulation using textual descriptions on human faces. StyleGAN inversion module, visual-linguistic similarity learning, and instance-level optimization are the three components of the proposed technique. The inversion module converts real images to the latent space of a StyleGAN that has been properly trained. By mapping the image and text into a common embedding space, the visual-linguistic similarity learns text-image matching. Identity preservation is the goal of the instance-level optimization.

Overcoming challenges for text-based manipulation of human face

Issues including inaccuracy in spatial description, ambiguity in the description of appearance, and incompleteness are handled with image text shared space in (Xu et al., 2021). Both appearance (colour, for example) and spatial editing are included in the proposed framework (e.g., pose and expression modification). Textual guidance achieves the former, whereas structural input, such as landmarks, determines the structure of picture content for the latter. Another study on facial editing was published in (Jiang et al., 2021), which proposed a Talk-to-Edit system that consists of three major parts: user request understanding, semantic field manipulation, and system feedback, and performs fine-grained attribute manipulation through dialogue between the user and the system.

2.3 StyleGAN and its progression

In 2018, an Nvidea team updated the GAN architecture utilising style transfer approaches to boost high-resolution image creation (Huang and Belongie, 2017) and introduced StyleGAN in their work (Karras, Laine and Aila, 2018). Two networks, a mapping network and a synthesis network, make up StyleGAN. A mapping network is a multilayer perceptron with eight layers that maps the latent representation to numerous style vectors, whereas a synthesis network generates visuals by processing input through several convolutional and up-sampling layers.

2.3.1 StyleGAN-2

Later an improved version of StyleGAN was introduced named StyleGAN2 capable of generating improved quality of images and also simplifying the inversion process of images by introducing a path length regularizer (Karras et al., 2019). The new model addressed the issue by normalising the mean and variance of each feature map separately, potentially erasing any information discovered in the magnitudes of the features relative to each other. The droplet artefact is caused by the generator purposefully slipping signal strength information past instance normalisation: by establishing a powerful, localised spike that dominates the statistics, the generator can effectively scale the signal.

2.3.2 StyleGAN-3

StyleGAN-3 improves on StyleGAN-2 by addressing the "texture sticking" issue (Karras et al., 2021b). They used the Nyquist-Shannon sampling theorem to evaluate the problem and

concluded that the layers in the generator learned to exploit the high-frequency signal in the pixels they operate on, as demonstrated in (Karras et al., 2021a).

They advocated that putting rigorous low pass filters between each generator's layers force the generator to function on the pixels in a way that is loyal to the continuous signals they represent, rather than operating on them as purely discrete signals. They used more signal filters to impose rotational and translational invariance (Alaluf et al., 2022). The resulting StyleGAN-3 is capable of resolving the texture sticking issue as well as producing images that rotate and translate smoothly.

Figure 2.4: Figure 2.3: A demonstration of StyleGAN on Flickr-Faces-HQ Dataset (FFHQ). Sources A and B generated two sets of images from their respective latent codes; the rest of the images were made by replicating a predetermined subset of styles from source B and getting the rest from source A.

2.4 CLIP: Connecting Text and Images

Contrastive Language–Image Pre-training (CLIP) introduced by Open-AI in Radford et al. (2021) is a neural network trained on a million image-text pairs and can be instructed in natural language to predict caption corresponding to the input image. This section will explore the approach and work related to CLIP.

2.4.1 Approach

CLIP measures the similarity between the text and image. It consists of two encoder, one for image and other for text. The similarity score of CLIP is calculated by inner product of the encodings of text and images. CLIP is trained on web text-image pairings. This data is used to generate the following CLIP proxy training task: Predict which of 32,768 randomly picked word samples was truly linked with an image on the dataset. CLIP models learn to

detect a wide range of visual concepts in images and correlate them with their names in order to complete this task. As a result, CLIP models can be applied to a wide range of visual categorization tasks. For example, if a dataset's aim is to classify images of dogs against photos of cats, the CLIP model predicts the text description "a photo of a dog" or "a photo of a cat" for the input image (Bremms, 2021).

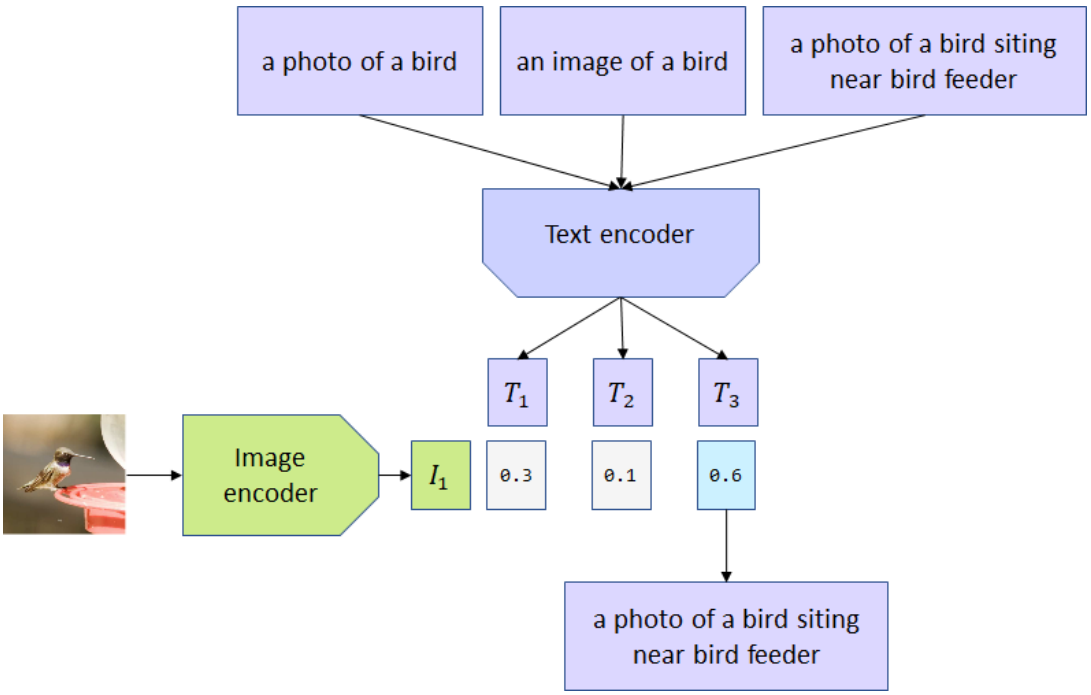

Figure 2.5: Figure 2.4: A demonstration of working of CLIP. When matched with the same image, the texts "a photo of a bird," "a photo of a bird sitting near a bird feeder," and "an image of a bird" all produce different probabilities. (Frolov, 2021)

2.4.2 Related work of CLIP

CLIP extends previous research on zero-shot transfer, natural language supervision, and multi-modal learning. Larochelle, Erhan and Bengio (2008) introduced concept of zero-data learning, but until recently it was largely used in computer vision as a method of generalising unknown object categories (Ba et al., 2015). The use of natural language as a flexible prediction space to facilitate generalisation and transfer was a significant finding. Socher et al. (2013) created a proof of concept by training a model using CIFAR-10 to predict in a word vector embedding space and demonstrating that this model could predict two previously undiscovered classes. A new model named "DeVise" scaled this approach the same year, demonstrating that it was possible to fine-tune an ImageNet model so that it could properly identify items outside of the original 1000 training set (Frome et al., 2013).

An important work in CLIP was done by Li et al. (2016) as they demonstrated leveraging natural language supervision to enable zero-shot transfer to numerous current computer vision classification datasets, including the ImageNet dataset. They achieved this by fine-tuning an ImageNet CNN to predict a considerably broader set of visual concepts (visual n-grams) using the text of 30 million Flickr photographs' titles, descriptions, and tags, and were able to achieve 11.5% accuracy on ImageNet zero-shot.

Finally, CLIP is one of several publications that have recently looked at the topic of learning visual representations via natural language supervision. This line of research makes use of more contemporary architectures, such as the Transformer, and includes the studies VirTex, ICMLM, and ConVIRT, which looked at the same contrastive objective as CLIP but in the context of medical imaging [Desai and Johnson (2020),Vaswani et al. (2017), Sariyildiz, Perez and Larlus (2020),Zhang et al. (2020)]. VirTex investigated autoregressive language modelling; ICMLM, masked language modelling; and ConVIRT, masked language modelling.

2.5 Mitigation of gender bias: Related work

The inclusion of algorithmic bias in search and recommendation systems has the potential to have a significant impact on society. One study showed evidence of gender bias in recommendation algorithms that target job-related advertisements (Lambrecht and Tucker, 2019). Personal assistant technologies' gendering as female is also being contested as indirect discrimination, possibly in violation of international law governing women's rightsada (n.d.).

The major reason for any bias in AI and machine learning is the skewed training data. While one of the way is to expand the diversity of these collected data, a manually intensive task, another way is to leverage AI only to tackle these biases Leavy et al. (2020). To quote Andrew McAfee, a research scientist at MIT,

"If you want the bias out, get the algorithms in."

Zhao et al. (2017) developed an approach for preventing general bias during training. A protected variable (such as gender) is trained to be predicted by an adversarial network, and the model learns to stop it from doing so. Two proof-of-concept tasks, income prediction on the UCI Adult dataset and analogy completion, were subjected to adversarial bias mitigation. An adversarial framework for text classification was independently developed by Kumar et al. (2019) to prevent confounding factors, such as the mention of a specific country, from adversely influencing classification, such as language identification. Xia, Field and Tsvetkov (2020) used tweets that were pre-annotated with the likely race of the author to draw from this body of research to reduce racial bias in a small LSTM-based hate speech detection model.

CNN-based classifiers typically use a variety of techniques to handle data imbalance, including pre-processing (Celis, Keswani and Vishnoi, 2019), in-processing to address discrimination during the model training phase (d'Alessandro, O'Neil and LaGatta, 2017), and post-processing (Awasthi, Kleindessner and Morgenstern, 2019), which deals with the data after the model has been trained. These de-biasing techniques are frequently employed, particularly when classifying images, such as when estimating gender Kamarulzalis, Razali and Moktar (2018).

Chapter 3

Requirements

This chapter describes the resources that will be used to implement this project.

3.1 Dataset Overview

The section provides details of the datasets that will be used to train the models. All these datasets are public datasets and can be used without any conflict.

3.1.1 Flickr-Faces-HQ (FFHQ)

FFHQ is a collection of 70,000 high-quality PNG images at 1024 X 1024 resolution, with a wide range of age, ethnicity, and image background (Karras, Laine and Aila (2018)). It also covers a wide range of accessories such as eyeglasses, sunglasses, hats, and so on. Images were crawled from Flickr, inheriting all biases of the website, and automatically aligned and cropped with dlib. Only images with permissive licences were gathered. Several automatic filters were used to prune the set, and finally Amazon Mechanical Turk was used to remove the odd statue, painting, or photo of a photo. This dataset will be used to train the StyleGAN model.

Figure 3.1: Visualization of FFHQ sample images

3.1.2 UTKFace dataset

This dataset consists of over 20,000 face images with annotations of age, gender, and ethnicity. The images cover a large variation in pose, facial expression, illumination, occlusion, resolution, etc. The labels of each image are embedded in the file name, formatted as [age] [gender] [race] [datetime].jpg. The DCNN model will be trained on this dataset (Zhang and Qi (2017)).

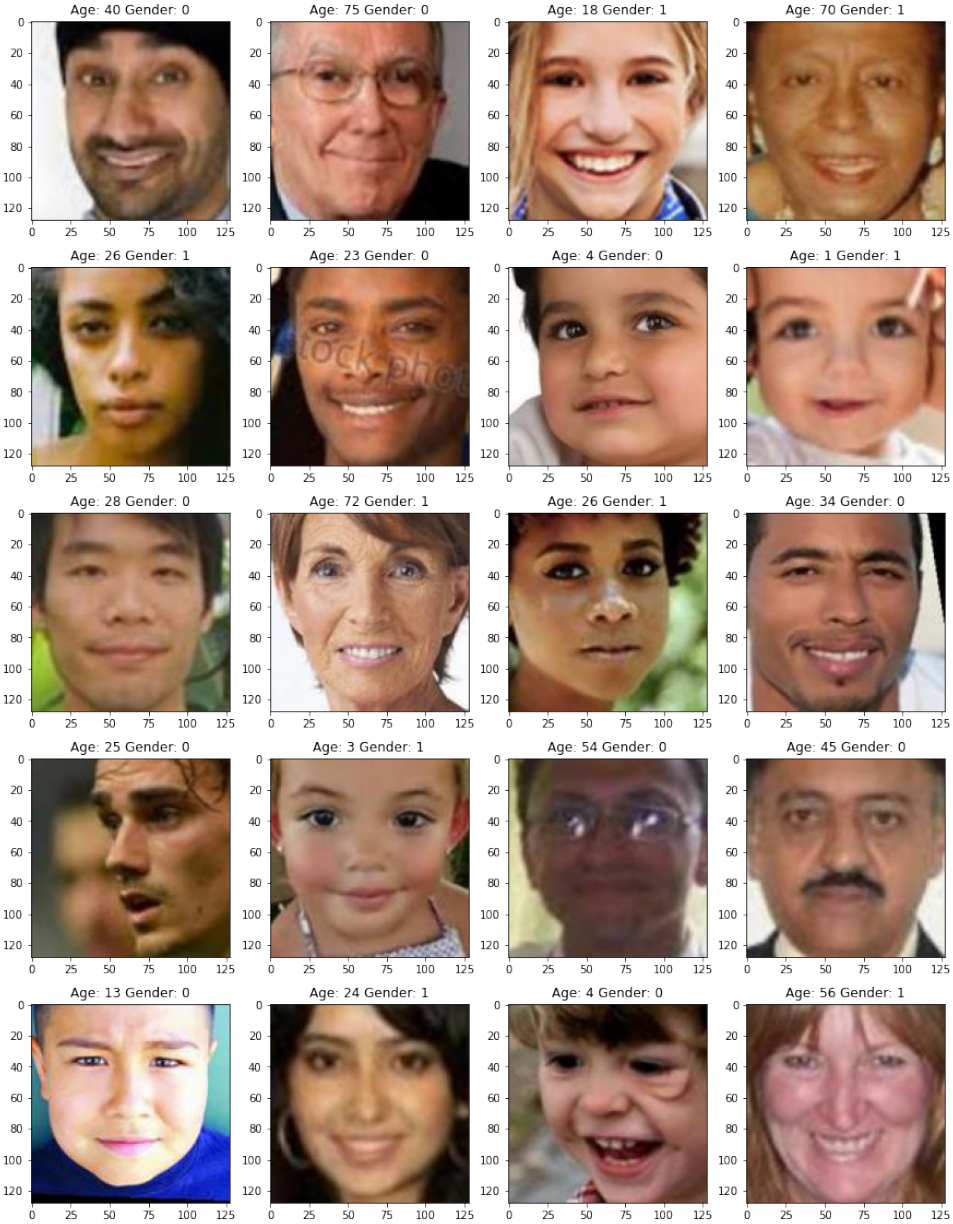

Figure 3.2: Visualization of UTKFace sample images. 0 represent gender "Male" while 1 represents gender "Female". Code for the visualization is provided in Appendix.

3.2 Python Libraries

The programming language for the implementation of the project will be Python.

1. **Numpy** (Harris et al., 2020) is a fundamental scientific Python library helpful for quick matrix and vector computations.

2. **Matplotlib** (Hunter, 2007) is a library for the Python programming language and its NumPy numerical mathematics extension that allows for data visualisation and graphical plotting.
3. **Sklearn** (Pedregosa et al., 2011) is a free machine learning package for Python that includes different clustering, regression, and classification techniques.
4. **Keras** (Chollet et al., 2015) is an open-source software library that provides a Python interface for artificial neural networks like pre-trained CNNs.
5. **Pytorch** (Paszke et al., 2019) is capable of accelerating tensor operations on cuda cores and is particularly designed for deep learning jobs.
6. **Tensorflow** (Abadi et al., 2015) is a Google-developed open-source library intended primarily for deep learning applications.

3.3 Computing resources

The image inversion, data augmentation and training of CNN model will be done on personal computer. The University of Bath HEX GPU will be used for training the StyleGAN. Google Colab will be used to compile all the models and test the integeration of StyleGAN and CNN.

Chapter 4

Methodology

In this chapter, we discuss about the implementation of StyleGAN, image inversion encoder, CLIP, integrating StyleGAN and CLIP using global direction approach and pre-trained CNN on UTKFace dataset.

4.1 Implementation of StyleGAN

This section provides the detail implementation of StyleGAN method. We implemented the StyleGAN based on the Rosinality implementation of StyleGAN in PyTorch (Seonghyeon, 2019). This method was fine-tuned by freezing the lower layer of discriminator Mo, Cho and Shin (2020).

4.1.1 Style Transfer

The quality of the generated images by StyleGAN has significantly improved as a result of the style transfer techniques in the generator to guarantee that every scale of the generated images has the same local structure as the training photos. Only the generator was changed; the discriminator and loss function remained unchanged. The architecture of StyleGAN comprises of two networks, mapping network and synthesis network.

4.1.2 Progressive Growing of GAN

The progressive growing GAN training method is used to train the StyleGAN generator and discriminator models. This means that both models are trained with low resolution images in the begining, 4 x 4 in our case. Fittng the model with this resolution of image continues till it gets stable, then the area of the images is quadrupled,i.e, 8 X 8. Once the model gets stable with 8 X 8 images, then the resolution of the images is increased further. This process is repeated till the desired target of image is met, 1024 x 1024 in our case. (Karras et al., 2017).

4.1.3 Bilinear Sampling

Instead of the transpose convolutional layers used in other generator models, the progressive expanding GAN uses closest neighbour layers for up-sampling. The first divergence in the StyleGAN is the use of bilinear up-sampling layers instead of nearest neighbour. As quoted

by Karras, Laine and Aila (2018) in a Style-Based Generator Architecture for Generative Adversarial Networks paper:

"In both networks, we use bilinear sampling instead of nearest-neighbor up/downsampling, which we accomplish by lowpass filtering the activations with a separable 2nd order binomial filter after each upsampling layer and before each downsampling layer."

4.1.4 Noise mapping Network

Mapping network is a multilayer perceptron (MLP) with eight layers that translates latent representations z (codings) to vectors w. This vector is then subjected to a sequence of affine transformations (i.e., Dense layers with no activation functions, illustrated in Figure 4.1 by the "A" boxes), giving a number of vectors. These vectors have varying degrees of influence on the style of the resulting image, ranging from fine-grained texture to high-level characteristics. The mapping network, in a nutshell, turns the codings into different style vectors.

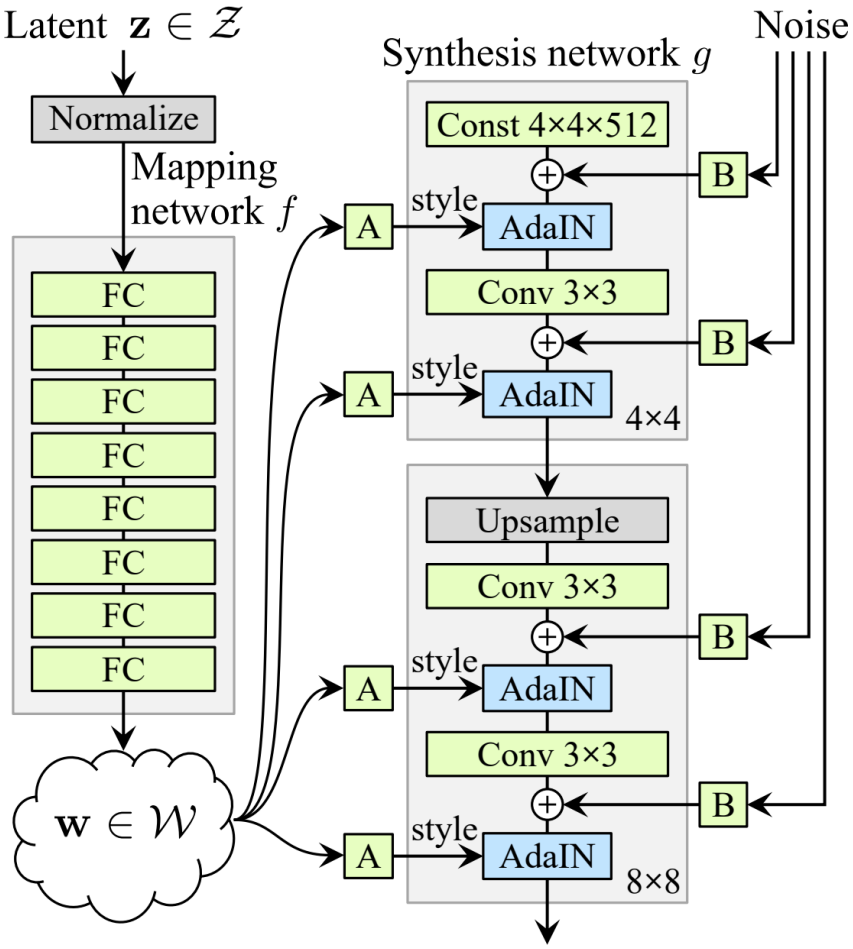

Figure 4.1: StyleGAN architecture consisting of mapping network and synthesis network. Architecture Source: (Karras, Laine and Aila, 2018)

4.1.5 Synthesis Network

The images are generated by the synthesis network. It is made up of constant learnt input. This input will be constant after training, but it will be modified via backpropagation during

training. This input is processed by various convolutional and upsampling layers. Following that, noise is injected to the convolutional layers' inputs and outputs (before the activation function).

Addition of Gaussian noise

Each convolutional layer in the synthesis network produces a block of activation maps. Prior to the AdaIN operations, Gaussian noise is applied to each of these activation maps. For each block, a different sample of noise is generated and evaluated using per-layer scaling factors.

Adaptive Instance Normalization (AdaIN)

Each noise layer is followed by an Adaptive Instance Normalization (AdaIN) layer, which standardises each feature map separately by removing the mean of the feature map and dividing by the standard deviation, and then utilises the style vector to define the scale and offset of each feature map. For each feature map, the style vector has one scale and one bias term.

$$\mathrm{AdaIN}(\boldsymbol{x}_i, \boldsymbol{y}_i) = \boldsymbol{y}_{s,i}\left(\frac{\boldsymbol{x}_i - \mu(\boldsymbol{x}_i)}{\sigma(\boldsymbol{x}_i)}\right) + \boldsymbol{y}_{b,i}$$

Figure 4.2: Adaptive Instance Normalization. Equation source: (Huang and Belongie, 2017)

4.1.6 Mixing Regularisation

StyleGAN employs a technique known as mixing regularisation (or style mixing), in which a percentage of the generated images use two different codings. The codings, say c_1 and c_2, are specifically routed via the mapping network, yielding two style vectors, say w_1 and w_2. The synthesis network then constructs a picture using styles w_1 for the first levels and styles w_2 for the remaining levels. The cutoff level is chosen at random and remains constant during training. This stops the network from assuming that styles at neighbouring levels are associated, which promotes localization in the GAN, which means that each style vector only impacts a subset of features in the output image (Kim, Choi and Uh, 2021).

4.2 Fine-tuning StyleGAN

There are different approaches for fine-tuning the StyleGAN like fine-tuning only scale and shift parameters, generating latent optimization, and freezing the generator or discriminator layer. Our approach is inspired by Mo, Cho and Shin (2020) where we fine-tune the model by freezing the lower 3 layers of discriminator. Fine-tuning of GAN is done to overcome the issues due to over-fitting and low-fidelity samples. Before fine-tuning, following function were included:

4.2.1 Exponential Moving Average (EMA) of Model Weights

For inference, we employed an exponential moving average of the generator's weights. This aids in the stabilisation of the converged model, in the same way that optimizers like Adam

and layers like batch normalisation use exponential moving averages to make their statistics stable across numerous updates.

4.2.2 Gradient Penalty

Gradient Penalty is used to enforce a constraint that requires the gradients of the critic's output with respect to the inputs to have unit norm. The gradient was employed using inbuilt function 'grad' of PyTorch. The gradient penalty (GP) was calculated by flattening the first layer and normalizing second layer.

4.3 Implementation of CLIP

This section describes the architecture of CLIP, implementation of CLIP and calculation of cosine similarity between text and image pairs.

4.3.1 Architecture

CLIP is a neural network consisting of two encoders: Text encoder and Image encoder. CLIP uses a Vision transformer to get visual features and a causal language model to get the text features. DistilBERT was used as an text encoder. The general architecture of DistilBert is same as Bert. BERT is essentially a transformer architecture Encoder stack. An encoder-decoder network with self-attention on the encoder side and attention on the decoder side is referred to as a transformer architecture. The Encoder stack of BERT has 12 layers. BERT's feedforward networks are larger, with 768 hidden units and 18 attention heads. DistillBERT includes additional network of embedding and transformer layer connected to Bert base layer by distillation technique (Sanh et al., 2019).

We used Resnet-50 as our image encoder which is a 50 layers deep convolutional neural network. The pre-trained version of this model is trained on more than a million images from ImageNet database. It has 48 Convolution layers along with 1 MaxPool and 1 Average Pool layer.

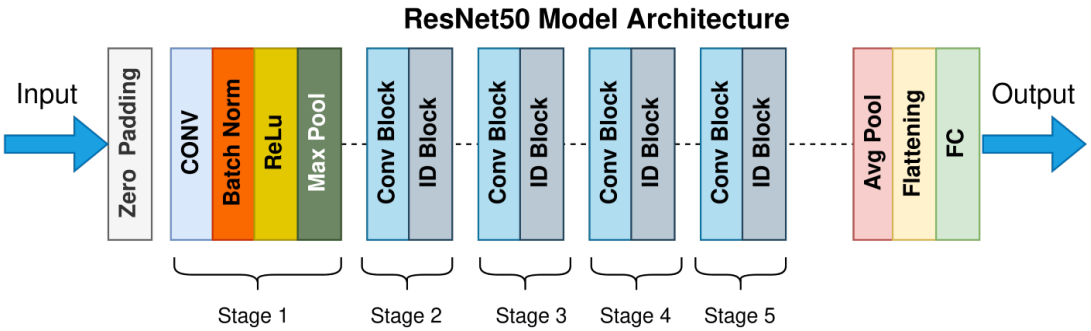

Figure 4.3: Resnet-50. Architecture Source: (He et al., 2015)

The text encoder take the captions as an input while the image encoder takes the images as an input. Both the text and visual features are then projected to a latent space with identical dimension. The dot product between the projected image and text features is then used as a similar score.

Working

In the figure 4.4, "pepper the aussie cup" is passed as an input to the text encoder and are encoded as series of numbers:

$$(T_1, T_2, T_3, ..., T_N)$$

Similarly, the images are encoded as:

$$(I_1, I_2, I_3, ..., I_N)$$

In an ideal world, these encoding would be equivalent. In reality, this should be the case everywhere: the text's number sequence should be extremely similar to the image's number sequence. How close the embedded representation (series of numbers) for each text is to the embedded representation for each image is one way for us to quantify the "quality" of our model. To calculate the similarity between these two, we use cosine similarity.

Cosine similarity

The light blue squares in the illustration 4.4 represent the inner product of text and image encoding. T_1 is the embedded representation of the first text, and T_2 is the embedded representation of the first picture. The model is regarded accurate if the cosine similarity for these encoding is high.

The grey squares represent areas where the text and image do not align. While the high cosine similarity score in blue squares shows the accurateness of the model, the grey squares with low cosine similarities, represents loss in the model.

Fitting text to images

The text encoder and picture encoder are fit simultaneously by maximising the cosine similarity of the blue squares while reducing the cosine similarity of the grey squares over all of our text-image pairs. After fitting the model, the image is processed through the image encoder to extract the text description that best fits the image.

4.4 Integrating StyleGAN with CLIP

Patashnik et al. (2021) introduced three methods to combine StyleGAN and CLIP. These three methods are compare in the table 4.1:

The optimizer and mapper deduce the latent step based on the input image, but training is only done once per text prompt. The global direction approach necessitates one-time pre-processing before being applied to distinct (picture, text prompt) pairs.

Though the latent mapper provides for swift inference, it occasionally falls short when fine-grained disentangled manipulation is required. Furthermore, the directions of many manipulation steps for a given text prompt are frequently similar. Motivated by these conclusions, we applied the global direction approach to map a text prompt into a single style space S, that is considered to be more disentangled than other latent spaces.

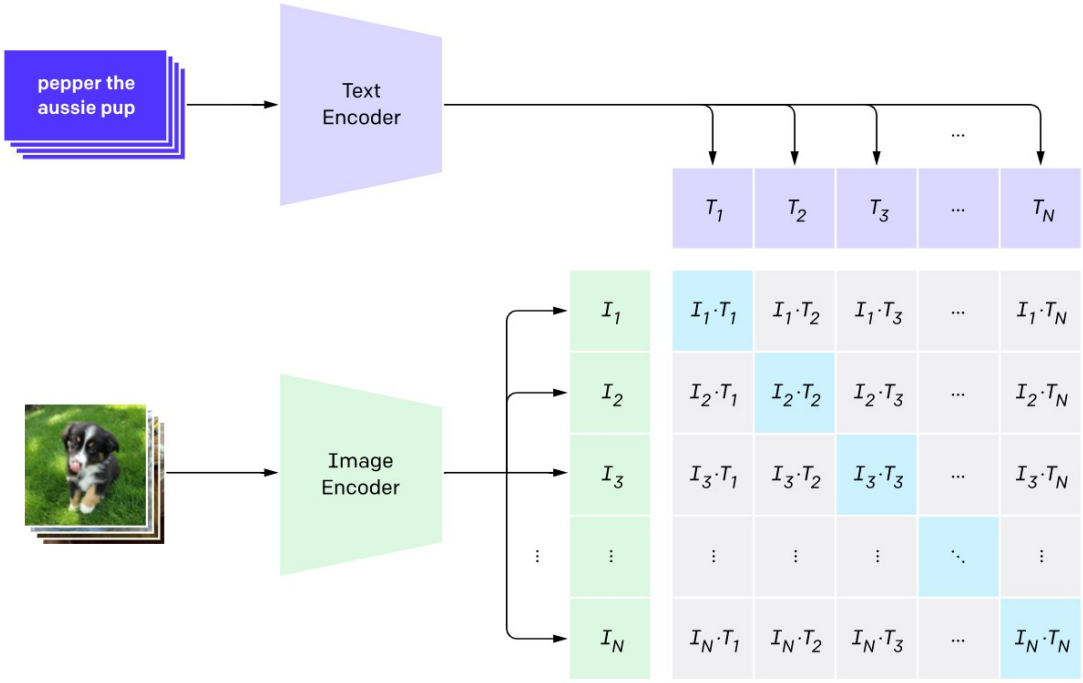

Figure 4.4: Demonstration of calculating cosine similarity in CLIP. Architecture Source: (Radford et al., 2021)

Table 4.1: Comparision of latent mapper, latent optimizer and global direction techniques for StyleCLIP. These comparision are with respect to the computing resources used by original author . Table source: (Patashnik et al., 2021)

Method	Pre-processing	Training time	Inference time	Input image independent
Latent Mapper	n/a	12 hr	75 ms	No
Latent Optimizer	n/a	n/a	98 sec	No
Global direction	n/a	n/a	98 sec	Yes

4.4.1 Global Direction Working

In Style space, the global direction is found by trans-versing along the direction such that the target attribute is adjusted for an arbitrary image, as shown in figure 4.5, which is the global direction of "Curly Hair." Because the same direction is applied to map the image to latent space, we referred it as the global direction. CLIP is used to map this direction to style space.

4.4.2 Manipulation and Disentanglement strength

The manipulation strength and disentanglement threshold are two important parameters of generating images using StyleGAN and CLIP. The function of manipulation strength is that the stronger the manipulation strength, the closer the resulting image is to the desired description. Negative values of this parameter distance the created image from the goal description. While the disentanglement controls the area intensity of these changes. The greater the disentanglement value, the greater the specificity of the changes to the target property. Lower numbers indicate that more extensive alterations are applied to the input image.

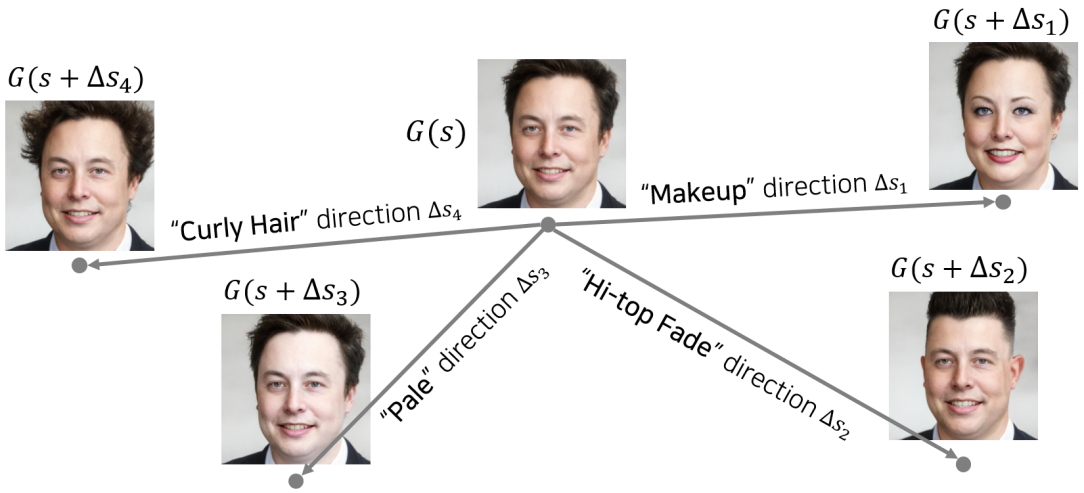

Figure 4.5: Demonstration of Global direction approach. Image Source: (Patashnik et al., 2021)

4.5 Encoder for inverting image

Tov et al. (2021) inspired the encoder, named, "encoder for editing" (e4e) used to invert the image into a latent vector that StyleGAN may utilise to reconstruct the image. StyleGAN interprets this vector, known as noise, as data to generate the desired target image.

4.5.1 Architecture and Working

The e4e encoder is an extension of the Pixel-2-Style-2-Pixel (pSp) encoder (Shen et al., 2019) that is designed exclusively for editing. It produces a single base style code, designated by w, as well as a sequence of $N-1$ offset vectors (shown in yellow in figure 4.6). The offsets are then added together with the base style code W to get the final N style codes, which are subsequently sent into a fixed, pre-trained StyleGAN generator to produce the reconstructed image.

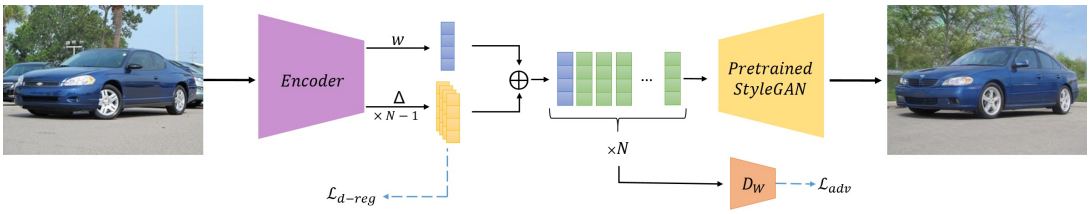

Figure 4.6: Demonstration of Global direction approach. Image Source: Tov et al. (2021)

The encoder takes an input image and outputs a single style code ω as well as a set of offsets $\Delta_1,\ldots,\Delta_{N-1}$ where N is the number of StyleGAN's style modulation layers. By replicating the ω vector N times and adding each Δ_i to its associated entry, the final latent representation is achieved. During training, the L_{d-reg} regularization encourages tiny variations between the final representation's entries, allowing it to stay near to W_*. Each latent code is guided by L_{adv} towards the range of StyleGAN's mapping network, resulting in a final representation that is closer to W^k. The encoder's final learnt representation is near to W as a result of using both regularisation terms.

4.6 Very Deep Convolutional Networks for Large-Scale Image Recognition: VGG-16

For classifying the gender of the images, we implemented the VGG-16 CNN model introduced by (Simonyan and Zisserman, 2014). This model won 1^{st} and 2^{nd} in the localisation and image classification categories respectively in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2014 (Russakovsky et al., 2015). It has 13 convolutional layer and 3 fully connected layer, that is why known as VGG-16.

4.6.1 Data Augmentation

The process of generating synthetic data from existing training examples by supplementing the samples with a variety of random modifications is known as data augmentation. The image pixels were re-scaled to [0,1] instead of (0,255).

Image Rotation

Image rotation is a popular augmentation technique that allows the model to become insensitive to object orientation. The image was rotated by 40 degrees. When the image is rotated, some pixels travel outside the image, leaving an empty space that must be filled. To fill in these values, we utilised the default value "nearest," which simply replaces the empty area with the nearest pixel values.

Image Shift and Flip

To place the object in the middle of the image, we adjusted the image's pixels horizontally or vertically by assigning a constant value to each pixel. To do this, the arguments, width shift range = 0.2 and height shift range = 0.2, were utilised. The parameter horizontal shift was used to flip the images along the horizontal axis.

4.6.2 Architecture

The network's input is a image of dimension (150, 150, 3) (Figure 4.9). The first two layers have 64 channels with the same padding and a 3*3 filter size. Following a max pool layer of stride (2, 2), two layers with convolution layers of 128 filter size and filter size are added (3, 3). This is followed by a stride (2, 2) max-pooling layer that is the same as the preceding layer. There are then two convolution layers with filter sizes of (3, 3) and 256 filters. Following that, there are two sets of three convolution layers and a max pool layer. Each has 512 filters of the same size (3, 3) with the same padding. This image is then sent into a convolution layer stack of two. We obtained a (4, 4, 512) feature map by stacking convolution and max-pooling layers. This output is flattened to become a (1, 8192) feature vector. "ReLU" serves as the activation function for all hidden layers. "ReLU" is more computationally efficient since it allows for faster learning and reduces the risk of vanishing gradient problems. We selected sigmoid activation for the final dense layer since gender classification is a binary classification problem.

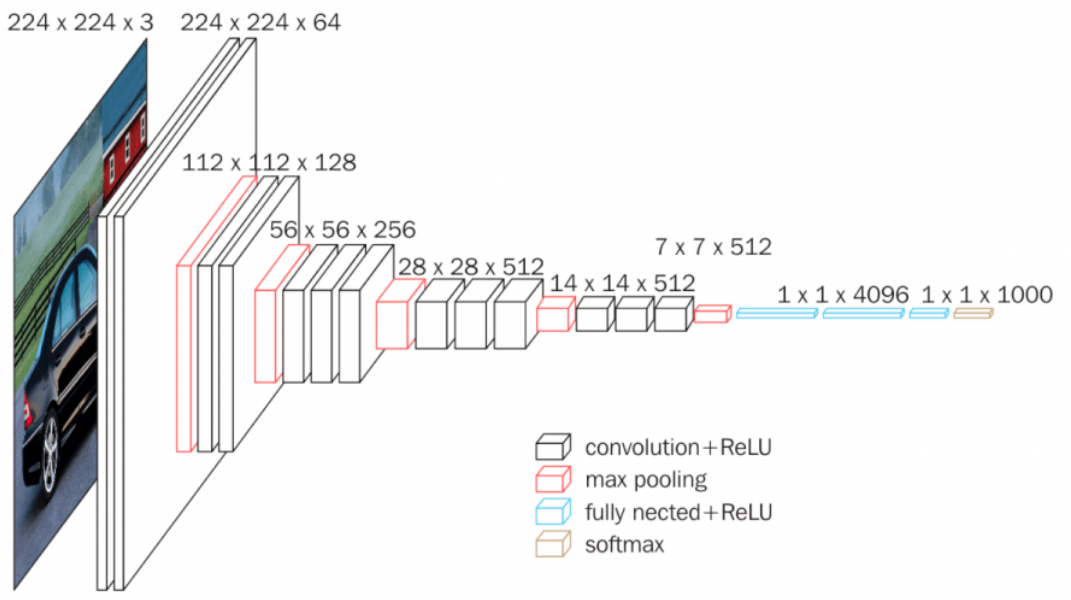

Figure 4.7: VGG-16 Model Architecture. Image Source: (Simonyan and Zisserman, 2014)

4.6.3 Training

The model was trained for a batch size of 128 and number of epochs as 25. The loss function used is "binary cross entropy", optimizer as "stochastic gradient descent" and the metric "accuracy" used to evaluate the performance model. The below table shows the distribution of dataset for training, validation and testing.

Table 4.2: UTKFace dataset distribution

Purpose	Number of Images	Percentage
Training	16000	80%
Validation	2000	20%
Testing	2000	20%

4.7 FaceGAN: Propose Method

We combine the above trained VGG-16 model with the discriminator of StyleGAN to mitigate the gender bias from the generated image. This section provied the working of propose FaceGAN method.

4.7.1 Working

The working of the propose method is very similar to the above StyleCLIP method.

The image is inverted by the inversion coder into the latent mapper. The text and image input are mapped into the global direction of style space. The only difference is that the input image is passed to VGG-16 CNN model which classify the gender of the person. This gender (0 for male and 1 for female) is saved as "gender ouput A" as shown in the figure 4.8.

Meanwhile, on the basis of the latent mapper and image-text mapping passed to the StyeGAN, the generator reconstructs the image as per the input caption. This generated image is passed

to the discriminator and VGG-16. The discriminator checks for the image if it is fake or real, while the VGG-16 classify the gender of the image.

The gender classified is saved as "gender output B". If the "gender output A" is equal to the "gender output B", then conditional statement added within CNN outputs "True" for real and "False" for fake. This "True" or "False" is passed to the discriminator which consider the output by VGG-16 while classifying the image as "Real" or "Fake".

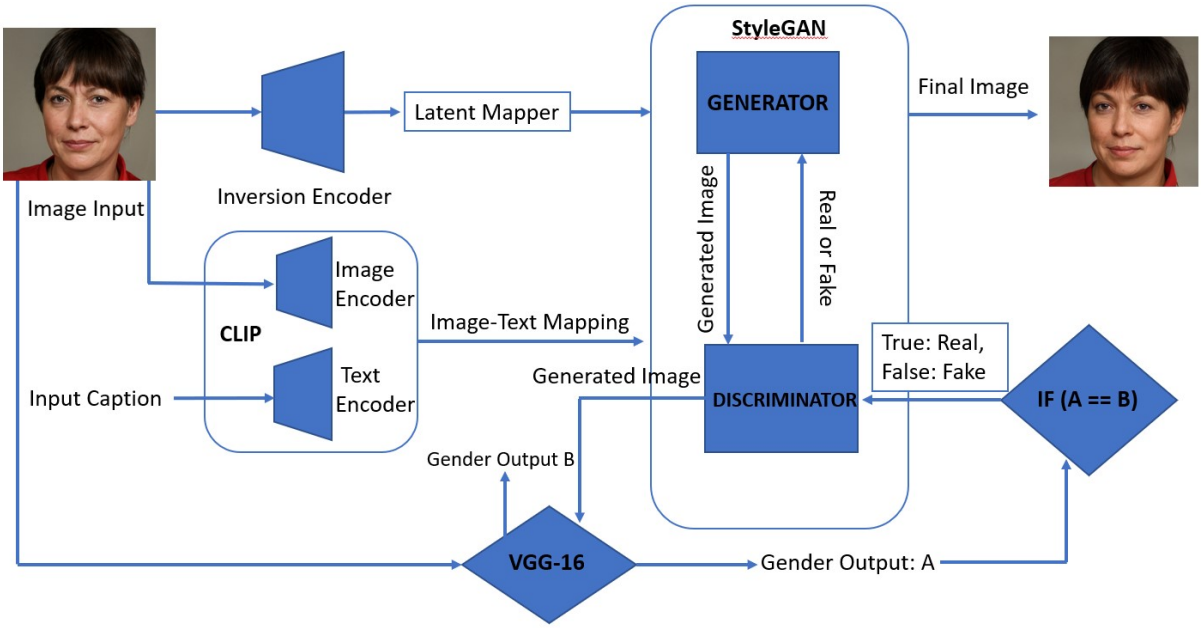

Figure 4.8: FaceGAN: Architecture of proposed method

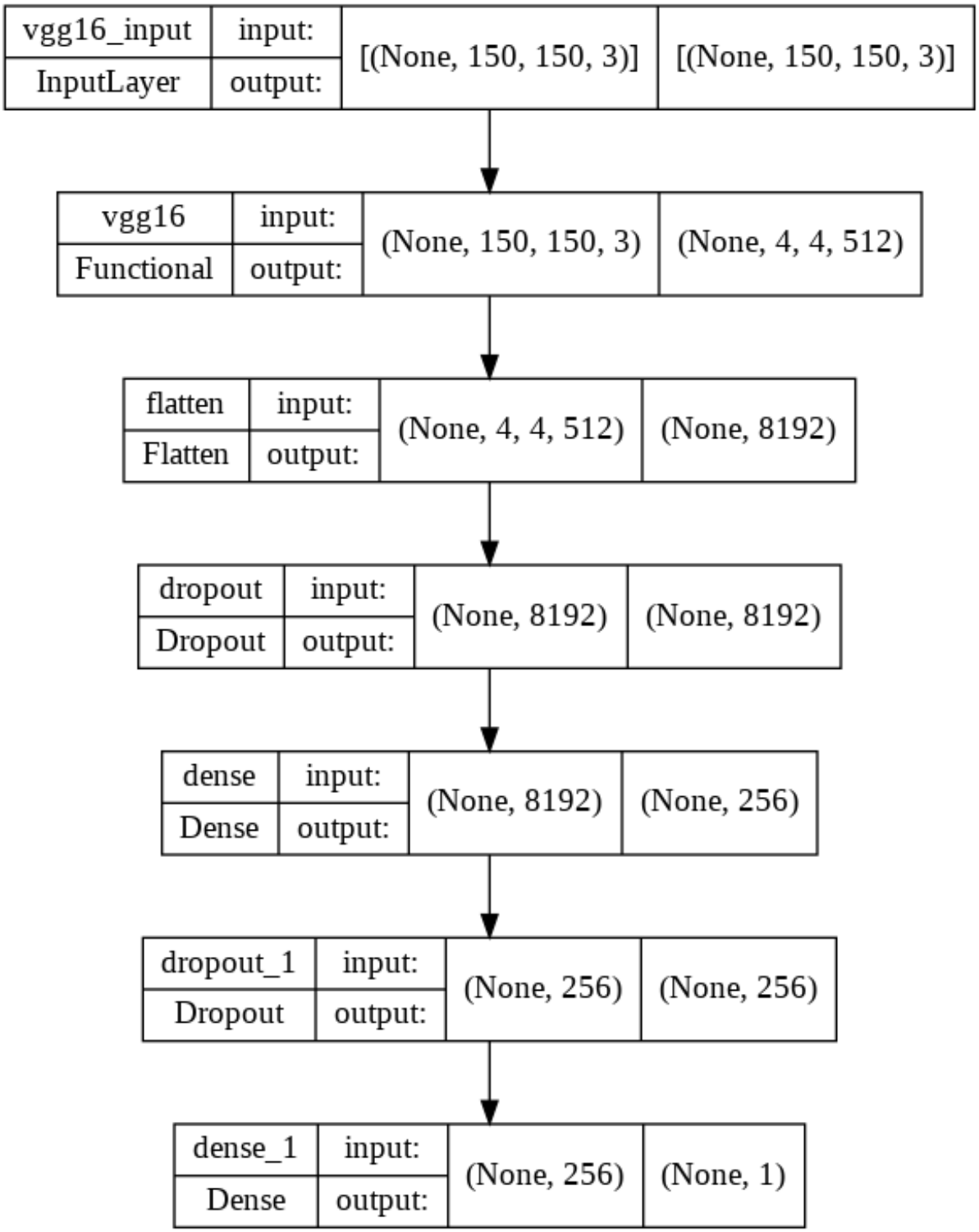

Figure 4.9: Implemented VGG-16 Model Summary. The input dimension of our image is (150,150,3) instead of the (224,224,3) in the above model

Chapter 5

Result

This chapter discusses about the gender classification performance by VGG-16 model and comparison of performance of FaceGAN and styleCLIP in generating gender-bias free image.

5.1 VGG-16 performance in detecting the gender

As the VGG-16 model was already pre-trained on the Imagenet, it was expected to get a high training and validation accuracy (accuracy details shown by fig in Appendix). The model recorded a test accuracy of 95.192% which is lower than the accuracy of 99.9% achieved by hussain (2021). hussain (2021) combined the VGG-16 with the a custom CNN of 5 convolutional layer with "relu" activation with each layer followed by a dropout layer and batch normalization. As we used high quality images from `https://thispersondoesnotexist.com/`, we didn't added any further layer as that could have increased the execution time.

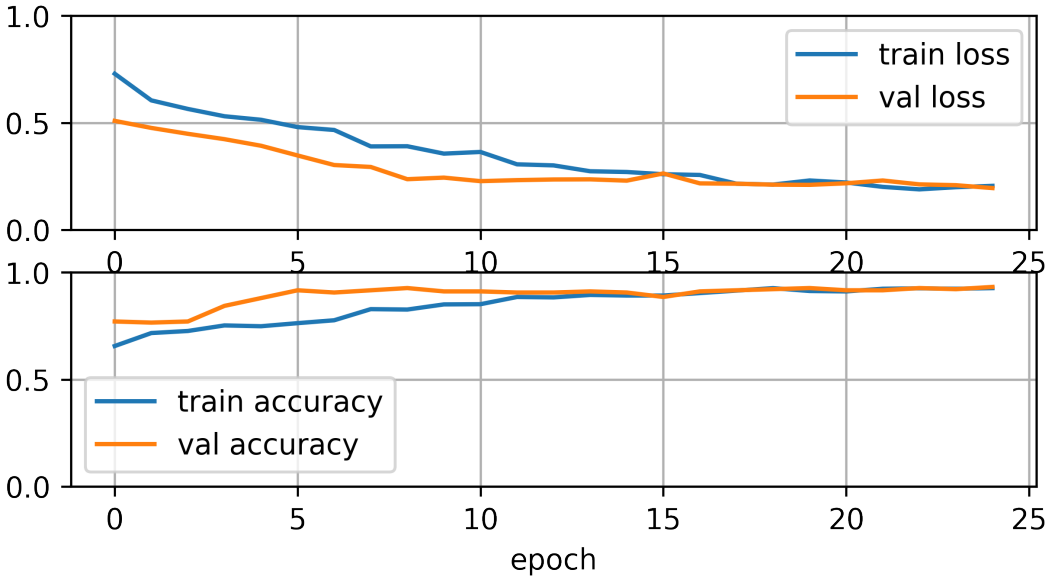

Figure 5.1: VGG-16: Comparision of training accuracy and Validation accuracy for gender classification

Before integrating VGG-16 with the StyleGAN, we further tested the CNN model on a number

of images from `https://thispersondoesnotexist.com/` to confirm the final accuracy of model. An accuracy of more than 90% by VGG-16 on gender classification tasks is more than enough at this stage. The performance result is shown in the figure below:

Table 5.1: VGG-16 Gender classification on `https://thispersondoesnotexist.com/` images. As these person does not exist, the base gender of these people was assumed on the basis of my judgement, what I saw from naked eye.

Number of Images	Correct Gender	Wrong Gender	Performance
45	41	04	91.11%

Sample of this testing is shown in the figure 5.3:

Figure 5.2: Sample of VGG-16 testing. Here BG represents base gender, PG represents predicted gender, 0 for male, 1 for female, X represents incorrect prediction

5.2 Comparision of FaceGAN: proposed method and StyleCLIP in gender bias free text-image synthesis

Though the accuracy of the VGG-16 in classifying the gender was good, the accuracy of the proposed method, "FaceGAN", was not as expected. In this section, we discuss about that in detail:

5.2.1 FaceGAN failed when the parameter manipulation strength and disentanglement had low or high values

FaceGAN failed to generate any image when the values of the manipulation strength and disentanglement was either low or high. The exact range of these low and high values cannot be given as it was different in-case of each image. These error was unexpected and it affected

the further performance of our method as we had to limit the working of our method to a limited range of these parameters affecting the intensity of the changes applied to input image. This error also affected the result described below.

5.2.2 Undesired manipulation where changes of opposite gender was desired

A blunder was made as we failed to include the possibility of a situation where user wanted to manipulate the input image with the characteristics of opposite gender.

These changes were discarded as VGG-16 classified the "gender output B" dissimilar to "gender output A" due to which the conditional statement gave the feedback "Fake" to the StyleGAN, even though these manipulation was desired by user. These manipulation were only possible if the intensity of changes were kept low by controlling the manipulation strength and disentanglement parameter but not too low as the FaceGAN will not generate image as mentioned in above case.

For example, in the figure 5.3, the input text prompt was "Elon Musk Face." The faceGAN was not able to manipulate the image as desired. It was only able to apply minimal feature of Elon Musk, which yielded the result as shown below. If we increase the intensity further, then the faceGAN classified the image as "male" resulting in passing argument "Fake" to the generator by discriminator. While, the image generated by StyleCLIP was able to accommodate a lot of facial feature related to Elon Musk.

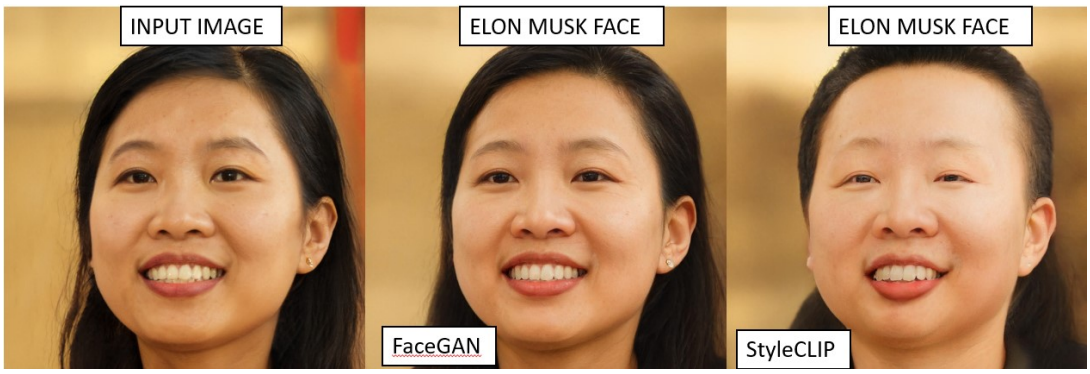

Figure 5.3: Comparision of manipulation for text prompt "Elon Musk Face" by FaceGAN and StyleCLIP. Input image source: `https://thispersondoesnotexist.com/`

5.2.3 Positive but limited gender characteristics preserved when gender bias prompt given

While the proposed method did not performed well than the current StyleCLIP method in the above cases, a slight achievement was noticed when a user provided a prompt bias towards a opposite gender.

The FaceGAN was able to preserve the original gender characteristic of the image but this achievement was limited to a very specific intensity as the manipulation strength and disentanglement parameter can't be change to high and low values as described above.

We use the image shown in figure 1.1 to describe this further. As shown in figure 5.4, user provided a text prompt "Bodybuilder face", the image generated by the StyleCLIP shifted the gender attributes of the input image towards male (like moustaches and strong jaw like male) while the FaceGAN was able to preserve some females characteristics in the generated image.

Though it should be noted the images were generated at different values of manipulation strength and disentanglement intensity as our method failed to generate image for a higher value of these parameters as discussed above.

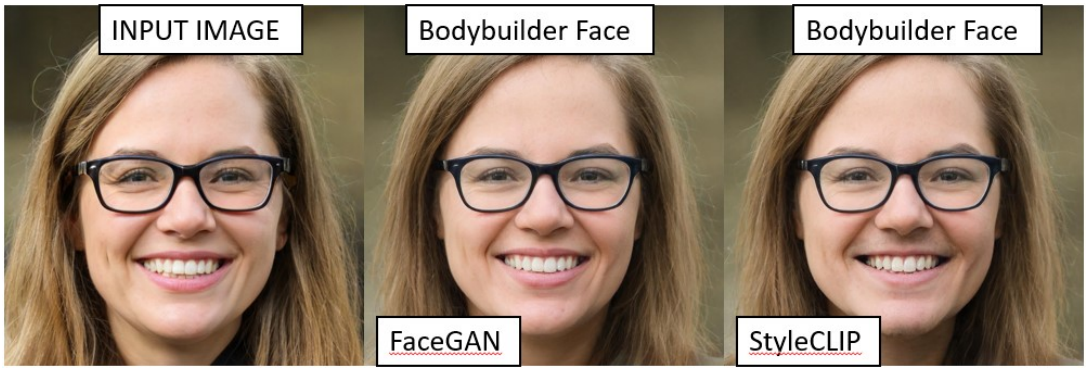

Figure 5.4: Comparision of manipulation for text prompt "Bodybuilder Face" by FaceGAN and StyleCLIP. Input image source: `https://thispersondoesnotexist.com/`

Chapter 6

Conclusion

In this thesis, we investigated the text-based image manipulation using StyleGAN and CLIP. We combined the generative power of StyleGAN and the image-text mapping ability of CLIP using global direction technique. We identified 2 major challenges in the StyleCLIP method, non-target feature manipulate and gender bias in the generated image. We focused on mitigating the gender bias from the above model by combining the StyleGAN with VGG-16 CNN model. The VGG-16 was trained to classify the gender of the input image and generated image. We included a conditional model in the VGG-16 that if the gender of input image is different from the generated image, then it will pass the feedback "Fake" to the discriminator of StyleGAN, in a belief that this will force the generator not to introduce the gender bias in the generated image. We introduce the combination of these method as a new method called "FaceGAN". Though our method did not performed as expected but in one area it achieved slight improvement than the current method. The FaceGAN was able to preserve the original gender characteristic of the image in some cases. This led us to the foundation of our future work that the biases from these text-based image manipulation methods can be removed by introducing neural or adversarial network within these method.

6.0.1 Personal Learning from this project

Through this project, I got an opportunity to learn about an advance field of machine learning. The learning stage was quite difficult and project implementation phase gave me valuable lesson on planning project. I have identified a number of shortcoming in myself through these project and will try to overcome them in future from experience that I gained during this dissertation.

6.1 Future Work

Our proposed method "FaceGAN" was able to preserved positive but limited gender characteristics when a gender bias prompt was given by user. At the same time, FaceGAN failed to generate an image when the parameter manipulation strength and disentanglement had low or high values. We also made a blunder as we failed to include the possibility of a situation where user wanted to manipulate the input image with the characteristics of opposite gender. All these postive and negative achievement of our result has guided us to an unexplored approach where we use machine learning to limit the bias from the existing methods to leverage their

capabilities in a wide array of application. Some possible future work on the basis of our results are:

6.1.1 Investigate the reason of FaceGAN failure when the parameter manipulation strength and disentanglement had low or high values

This error was unexpected and due to the time complexity of this project, we could not investigate the possible reason behind this. New methods for integrating the StyleGAN and CLIP can be explored to see if this challenge continues with the new approaches.

6.1.2 Include the possibility of case where user wants to include features of opposite gender

This was a blunder from our side as we failed to include this possibility in our method. But this method can be overcome by combining the CLIP model with a NLP model. This model must be trained to identify the gender of text prompt. This can be done by training it on the dataset of text caption related to different gender similar to what we do in sentimental analysis. This model can then map these gender sentiment (male, female or neutral) with the StyleGAN.

6.1.3 Optimizing and reducing the training time

Like every GAN, FaceGAN also faced difficulty when training it with the limited computation time. As we integrated multiple encoders, neural and adversarial network together, the trainig of our model became very difficult. In a way, it reduced the period of this dissertation by half.

There is already a number of work going in this field, one of which is increasing the performance of the GAN on a very small dataset. Similar approaches can be explored to reduce the dependency of our method on high-intensive GPU.

Bibliography

n.d.

n.d.

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y. and Zheng, X., 2015. TensorFlow: Large-scale machine learning on heterogeneous systems [Online]. Software available from tensorflow.org. Available from: `https://www.tensorflow.org/`.

Alaluf, Y., Patashnik, O., Wu, Z., Zamir, A., Shechtman, E., Lischinski, D. and Cohen-Or, D., 2022. Third time's the charm? image and video editing with stylegan3 [Online]. Available from: `https://doi.org/10.48550/ARXIV.2201.13433`.

Awasthi, P., Kleindessner, M. and Morgenstern, J., 2019. Equalized odds postprocessing under imperfect group information [Online]. Available from: `https://doi.org/10.48550/ARXIV.1906.03284`.

Ba, J., Swersky, K., Fidler, S. and Salakhutdinov, R., 2015. Predicting deep zero-shot convolutional neural networks using textual descriptions [Online]. Available from: `https://doi.org/10.48550/ARXIV.1506.00511`.

Bahng, H., Yoo, S., Cho, W., Park, D.K., Wu, Z., Ma, X. and Choo, J., 2018. Coloring with words: Guiding image colorization through text-based palette generation. *Eccv*.

Beal, M.J., 1970. Variational algorithms for approximate bayesian inference. Available from: `https://discovery.ucl.ac.uk/id/eprint/10101435/`.

Bengio, Y., Courville, A. and Vincent, P., 2013. Representation learning: A review and new perspectives. *Ieee transactions on pattern analysis and machine intelligence*, 35(8), pp.1798–1828.

Bodnar, C., 2018. Text to image synthesis using generative adversarial networks [Online]. [Online]. Available from: `https://doi.org/10.13140/RG.2.2.35817.39523`.

Bond-Taylor, S., Leach, A., Long, Y. and Willcocks, C.G., 2021. Deep generative modelling: A comparative review of VAEs, GANs, normalizing flows, energy-based and autoregressive models. *IEEE transactions on pattern analysis and machine intelligence* [Online], pp.1–1. Available from: `https://doi.org/10.1109/tpami.2021.3116668`.

Bremms, M., 2021. A beginner's guide to the clip model. Available from: https://www.kdnuggets.com/2021/03/beginners-guide-clip-model.html.

Brock, A., Donahue, J. and Simonyan, K., 2018. Large scale gan training for high fidelity natural image synthesis [Online]. Available from: https://doi.org/10.48550/ARXIV.1809.11096.

Brownlee, J., 2020. A gentle introduction to stylegan the style generative adversarial network. Available from: https://machinelearningmastery.com/introduction-to-style-generative-adversarial-network-stylegan/.

Celis, L.E., Keswani, V. and Vishnoi, N.K., 2019. Data preprocessing to mitigate bias: A maximum entropy based approach [Online]. Available from: https://doi.org/10.48550/ARXIV.1906.02164.

Chen, J., Shen, Y., Gao, J., Liu, J. and Liu, X., 2017. Language-based image editing with recurrent attentive models [Online]. Available from: https://doi.org/10.48550/ARXIV.1711.06288.

Chollet, F. et al., 2015. Keras. https://keras.io.

d'Alessandro, B., O'Neil, C. and LaGatta, T., 2017. Conscientious classification: A data scientist's guide to discrimination-aware classification. *Big data* [Online], 5(2), pp.120–134. Available from: https://doi.org/10.1089/big.2016.0048.

Desai, K. and Johnson, J., 2020. Virtex: Learning visual representations from textual annotations [Online]. Available from: https://doi.org/10.48550/ARXIV.2006.06666.

Dhawan, S. and Kumar, S., 2020. Improving resolution of images using generative adversarial networks [Online]. *2020 4th international conference on electronics, communication and aerospace technology (iceca)*. pp.880–887. Available from: https://doi.org/10.1109/ICECA49313.2020.9297414.

Donahue, J. and Simonyan, K., 2019. Large scale adversarial representation learning. *Advances in neural information processing systems*, 32.

Dong, H., Yu, S., Wu, C. and Guo, Y., 2017. Semantic image synthesis via adversarial learning [Online]. Available from: https://doi.org/10.48550/ARXIV.1707.06873.

Frid-Adar, M., Diamant, I., Klang, E., Amitai, M., Goldberger, J. and Greenspan, H., 2018. Gan-based synthetic medical image augmentation for increased cnn performance in liver lesion classification. *Neurocomputing*, 321, pp.321–331.

Frolov, S., Hinz, T., Raue, F., Hees, J. and Dengel, A., 2021. Adversarial text-to-image synthesis: A review. *Neural networks* [Online], 144, pp.187–209. Available from: https://doi.org/https://doi.org/10.1016/j.neunet.2021.07.019.

Frolov, V., 2021. Clip from openai: What is it and how you can try it out yourself. Available from: https://habr.com/en/post/537334/.

Frome, A., Corrado, G.S., Shlens, J., Bengio, S., Dean, J., Ranzato, M.A. and Mikolov, T., 2013. Devise: A deep visual-semantic embedding model [Online]. In: C. Burges, L. Bottou, M. Welling, Z. Ghahramani and K. Weinberger, eds. *Advances in neural information processing systems*. Curran Associates, Inc., vol. 26. Available from: https://proceedings.neurips.cc/paper/2013/file/7cce53cf90577442771720a370c3c723-Paper.pdf.

Gal, Y., 2021. Oatml. Available from: `https://oatml.cs.ox.ac.uk/blog/2021/06/27/web-scraped-harmful.html`.

Gatys, L.A., Ecker, A.S. and Bethge, M., 2016. Image style transfer using convolutional neural networks. *Proceedings of the ieee conference on computer vision and pattern recognition*. pp.2414–2423.

Goodfellow, I.J., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. and Bengio, Y., 2014. Generative adversarial networks [Online]. Available from: `https://doi.org/10.48550/ARXIV.1406.2661`.

Gregor, K., Danihelka, I., Graves, A., Rezende, D.J. and Wierstra, D., 2015. Draw: A recurrent neural network for image generation [Online]. Available from: `https://doi.org/10.48550/ARXIV.1502.04623`.

Han, J., Zhang, Z. da, Mao, A. and Zhou, Y., 2018. Semantics images synthesis and resolution refinement using generative adversarial networks. *Csps*.

Harris, C.R., Millman, K.J., Walt, S.J. van der, Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N.J., Kern, R., Picus, M., Hoyer, S., Kerkwijk, M.H. van, Brett, M., Haldane, A., Río, J.F. del, Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C. and Oliphant, T.E., 2020. Array programming with NumPy. *Nature* [Online], 585(7825), pp.357–362. Available from: `https://doi.org/10.1038/s41586-020-2649-2`.

He, K., Zhang, X., Ren, S. and Sun, J., 2015. Deep residual learning for image recognition [Online]. Available from: `https://doi.org/10.48550/ARXIV.1512.03385`.

Huang, X. and Belongie, S., 2017. Arbitrary style transfer in real-time with adaptive instance normalization [Online]. Available from: `https://doi.org/10.48550/ARXIV.1703.06868`.

Hunter, J.D., 2007. Matplotlib: A 2d graphics environment. *Computing in science & engineering* [Online], 9(3), pp.90–95. Available from: `https://doi.org/10.1109/MCSE.2007.55`.

hussain, Y., 2021. Gender classification using vgg16+cnn. Available from: `https://www.kaggle.com/code/yasserhessein/gender-classification-using-vgg16-cnn`.

Isola, P., Zhu, J.Y., Zhou, T. and Efros, A.A., 2017. Image-to-image translation with conditional adversarial networks. *Proceedings of the ieee conference on computer vision and pattern recognition*. pp.1125–1134.

Jiang, Y., Huang, Z., Pan, X., Loy, C.C. and Liu, Z., 2021. Talk-to-edit: Fine-grained facial editing via dialog [Online]. Available from: `https://doi.org/10.48550/ARXIV.2109.04425`.

Jing, Y., Yang, Y., Feng, Z., Ye, J., Yu, Y. and Song, M., 2019. Neural style transfer: A review. *Ieee transactions on visualization and computer graphics*, 26(11), pp.3365–3385.

Kamarulzalis, A.H., Razali, M.H.M. and Moktar, B., 2018. Data pre-processing using smote technique for gender classification with imbalance hu's moments features.

Karras, T., Aila, T., Laine, S. and Lehtinen, J., 2017. Progressive growing of gans for improved quality, stability, and variation [Online]. Available from: `https://doi.org/10.48550/ARXIV.1710.10196`.

Karras, T., Aittala, M., Laine, S., Härkönen, E., Hellsten, J., Lehtinen, J. and Aila, T., 2021a. Alias-free generative adversarial networks. *Proc. neurips*.

Karras, T., Aittala, M., Laine, S., Härkönen, E., Hellsten, J., Lehtinen, J. and Aila, T., 2021b. Alias-free generative adversarial networks [Online]. Available from: https://doi.org/10.48550/ARXIV.2106.12423.

Karras, T., Laine, S. and Aila, T., 2018. A style-based generator architecture for generative adversarial networks [Online]. Available from: https://doi.org/10.48550/ARXIV.1812.04948.

Karras, T., Laine, S., Aittala, M., Hellsten, J., Lehtinen, J. and Aila, T., 2019. Analyzing and improving the image quality of stylegan [Online]. Available from: https://doi.org/10.48550/ARXIV.1912.04958.

Kim, J., Choi, Y. and Uh, Y., 2021. Feature statistics mixing regularization for generative adversarial networks [Online]. Available from: https://doi.org/10.48550/ARXIV.2112.04120.

Kim, J.H., Kitaev, N., Chen, X., Rohrbach, M., Zhang, B.T., Tian, Y., Batra, D. and Parikh, D., 2019. CoDraw: Collaborative drawing as a testbed for grounded goal-driven communication [Online]. *Proceedings of the 57th annual meeting of the association for computational linguistics*. Florence, Italy: Association for Computational Linguistics, pp.6495–6513. Available from: https://doi.org/10.18653/v1/P19-1651.

Kingma, D.P. and Welling, M., 2013. Auto-encoding variational bayes [Online]. Available from: https://doi.org/10.48550/ARXIV.1312.6114.

Kumar, S., Wintner, S., Smith, N.A. and Tsvetkov, Y., 2019. Topics to avoid: Demoting latent confounds in text classification [Online]. Available from: https://doi.org/10.48550/ARXIV.1909.00453.

Kwon, Y.H. and Park, M.G., 2019. Predicting future frames using retrospective cycle gan [Online]. *2019 ieee/cvf conference on computer vision and pattern recognition (cvpr)*. pp.1811–1820. Available from: https://doi.org/10.1109/CVPR.2019.00191.

Lambrecht, A. and Tucker, C., 2019. Algorithmic bias? an empirical study of apparent gender-based discrimination in the display of stem career ads. *Management science* [Online], 65(7), p.2966–2981. Available from: https://doi.org/10.1287/mnsc.2018.3093.

Larochelle, H., Erhan, D. and Bengio, Y., 2008. Zero-data learning of new tasks. *Aaai*.

Leavy, S., Meaney, G., Wade, K. and Greene, D., 2020. Mitigating gender bias in machine learning data sets [Online]. Available from: https://doi.org/10.48550/ARXIV.2005.06898.

Ledig, C., Theis, L., Huszár, F., Caballero, J., Cunningham, A., Acosta, A., Aitken, A., Tejani, A., Totz, J., Wang, Z. et al., 2017. Photo-realistic single image super-resolution using a generative adversarial network. *Proceedings of the ieee conference on computer vision and pattern recognition*. pp.4681–4690.

Li, A., Jabri, A., Joulin, A. and Maaten, L. van der, 2016. Learning visual n-grams from web data [Online]. Available from: https://doi.org/10.48550/ARXIV.1612.09161.

Li, B., Qi, X., Lukasiewicz, T. and Torr, P.H.S., 2019a. Controllable text-to-image generation [Online]. Available from: https://doi.org/10.48550/ARXIV.1909.07083.

Li, B., Qi, X., Lukasiewicz, T. and Torr, P.H.S., 2019b. Manigan: Text-guided image manipulation [Online]. Available from: https://doi.org/10.48550/ARXIV.1912.06203.

Lin, T.Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C.L. and Dollár, P., 2014. Microsoft coco: Common objects in context [Online]. Available from: https://doi.org/10.48550/ARXIV.1405.0312.

Liu, X., Meng, G., Xiang, S. and Pan, C., 2018. Semantic image synthesis via conditional cycle-generative adversarial networks. *2018 24th international conference on pattern recognition (icpr)*, pp.988–993.

Mansimov, E., Parisotto, E., Ba, J.L. and Salakhutdinov, R., 2015. Generating images from captions with attention [Online]. Available from: https://doi.org/10.48550/ARXIV.1511.02793.

Mazzone, M. and Elgammal, A., 2019. Art, creativity, and the potential of artificial intelligence. *Arts* [Online], 8(1). Available from: https://doi.org/10.3390/arts8010026.

Mo, S., Cho, M. and Shin, J., 2020. Freeze the discriminator: a simple baseline for fine-tuning gans [Online]. Available from: https://doi.org/10.48550/ARXIV.2002.10964.

Nilsback, M.E. and Zisserman, A., 2008. Automated flower classification over a large number of classes. *Indian conference on computer vision, graphics and image processing*.

Park, H., Yoo, Y. and Kwak, N., 2018. Mc-gan: Multi-conditional generative adversarial network for image synthesis [Online]. Available from: https://doi.org/10.48550/ARXIV.1805.01123.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J. and Chintala, S., 2019. Pytorch: An imperative style, high-performance deep learning library [Online]. In: H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox and R. Garnett, eds. *Advances in neural information processing systems 32*. Curran Associates, Inc., pp.8024–8035. Available from: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Patashnik, O., Wu, Z., Shechtman, E., Cohen-Or, D. and Lischinski, D., 2021. Styleclip: Text-driven manipulation of stylegan imagery [Online]. Available from: https://doi.org/10.48550/ARXIV.2103.17249.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. and Duchesnay, E., 2011. Scikit-learn: Machine learning in Python. *Journal of machine learning research*, 12, pp.2825–2830.

Radford, A., Kim, J.W., Hallacy, C., Ramesh, A., Goh, G., Agarwal, S., Sastry, G., Askell, A., Mishkin, P., Clark, J., Krueger, G. and Sutskever, I., 2021. Learning transferable visual models from natural language supervision [Online]. Available from: https://doi.org/10.48550/ARXIV.2103.00020.

Reed, S., Akata, Z., Yan, X., Logeswaran, L., Schiele, B. and Lee, H., 2016. Generative adversarial text to image synthesis [Online]. Available from: https://doi.org/10.48550/ARXIV.1605.05396.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C. and Fei-Fei, L., 2015. ImageNet Large Scale Visual Recognition Challenge. *International journal of computer vision (ijcv)* [Online], 115(3), pp.211–252. Available from: https://doi.org/10.1007/s11263-015-0816-y.

Sanh, V., Debut, L., Chaumond, J. and Wolf, T., 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter [Online]. Available from: https://doi.org/10.48550/ARXIV.1910.01108.

Sariyildiz, M.B., Perez, J. and Larlus, D., 2020. Learning visual representations with caption annotations [Online]. Available from: https://doi.org/10.48550/ARXIV.2008.01392.

Saxena, D. and Cao, J., 2021. Generative adversarial networks (gans): Challenges, solutions, and future directions. *Acm comput. surv.* [Online], 54(3). Available from: https://doi.org/10.1145/3446374.

Schwemmer, C., n.d. Diagnosing gender bias in image recognition systemscarsten schwemmer. Available from: https://journals.sagepub.com/doi/full/10.1177/2378023120967171.

Seonghyeon, K., 2019. Rosinality/style-based-gan-pytorch: Implementation a style-based generator architecture for generative adversarial networks in pytorch. Available from: https://github.com/rosinality/style-based-gan-pytorch.

Shen, Y., Gu, J., Tang, X. and Zhou, B., 2019. Interpreting the latent space of gans for semantic face editing [Online]. Available from: https://doi.org/10.48550/ARXIV.1907.10786.

Shinagawa, S., Yoshino, K., Sakti, S., Suzuki, Y. and Nakamura, S., 2018. Interactive image manipulation with natural language instruction commands. *Arxiv*, abs/1802.08645.

Simonyan, K. and Zisserman, A., 2014. Very deep convolutional networks for large-scale image recognition [Online]. Available from: https://doi.org/10.48550/ARXIV.1409.1556.

Smink, A.R., van Reijmersdal, E.A., van Noort, G. and Neijens, P.C., 2020. Shopping in augmented reality: The effects of spatial presence, personalization and intrusiveness on app and brand responses. *Journal of business research* [Online], 118, pp.474–485. Available from: https://doi.org/https://doi.org/10.1016/j.jbusres.2020.07.018.

Socher, R., Ganjoo, M., Sridhar, H., Bastani, O., Manning, C.D. and Ng, A.Y., 2013. Zero-shot learning through cross-modal transfer [Online]. Available from: https://doi.org/10.48550/ARXIV.1301.3666.

Tang, R., Du, M., Li, Y., Liu, Z., Zou, N. and Hu, X., 2020. Mitigating gender bias in captioning systems [Online]. Available from: https://doi.org/10.48550/ARXIV.2006.08315.

Tatariants, M., 2022. Gan technology: Use cases for business applications. Available from: https://mobidev.biz/blog/gan-technology-use-cases-for-business-application.

Tov, O., Alaluf, Y., Nitzan, Y., Patashnik, O. and Cohen-Or, D., 2021. Designing an encoder

for stylegan image manipulation [Online]. Available from: https://doi.org/10.48550/ARXIV.2102.02766.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L. and Polosukhin, I., 2017. Attention is all you need [Online]. Available from: https://doi.org/10.48550/ARXIV.1706.03762.

Woolf, M., 2021. Easily transform portraits of people into ai aberrations using styleclip. Available from: https://minimaxir.com/2021/04/styleclip/.

Xia, M., Field, A. and Tsvetkov, Y., 2020. Demoting racial bias in hate speech detection [Online]. *Proceedings of the eighth international workshop on natural language processing for social media*. Online: Association for Computational Linguistics, pp.7–14. Available from: https://doi.org/10.18653/v1/2020.socialnlp-1.2.

Xia, W., Yang, Y., Xue, J.H. and Wu, B., 2020. Tedigan: Text-guided diverse face image generation and manipulation [Online]. Available from: https://doi.org/10.48550/ARXIV.2012.03308.

Xu, X., Chen, Y.C., Tao, X. and Jia, J., 2021. Text-guided human image manipulation via image-text shared space. *Ieee transactions on pattern analysis and machine intelligence*, PP.

Zhang, Zhifei, S.Y. and Qi, H., 2017. Age progression/regression by conditional adversarial autoencoder. *Ieee conference on computer vision and pattern recognition (cvpr)*. IEEE.

Zhang, Y., Jiang, H., Miura, Y., Manning, C.D. and Langlotz, C.P., 2020. Contrastive learning of medical visual representations from paired images and text [Online]. Available from: https://doi.org/10.48550/ARXIV.2010.00747.

Zhao, J., Wang, T., Yatskar, M., Ordonez, V. and Chang, K.W., 2017. Men also like shopping: Reducing gender bias amplification using corpus-level constraints [Online]. *Proceedings of the 2017 conference on empirical methods in natural language processing*. Copenhagen, Denmark: Association for Computational Linguistics, pp.2979–2989. Available from: https://doi.org/10.18653/v1/D17-1323.

Zhou, X., Huang, S., Li, B., Li, Y., Li, J. and Zhang, Z., 2019. Text guided person image synthesis. *Proceedings of the ieee/cvf conference on computer vision and pattern recognition (cvpr)*.

Zhu, J.Y., Park, T., Isola, P. and Efros, A.A., 2017a. Unpaired image-to-image translation using cycle-consistent adversarial networks. *Proceedings of the ieee international conference on computer vision*. pp.2223–2232.

Zhu, S., Fidler, S., Urtasun, R., Lin, D. and Loy, C.C., 2017b. Be your own prada: Fashion synthesis with structural coherence [Online]. Available from: https://doi.org/10.48550/ARXIV.1710.07346.

Appendix A

Training

```
loss: 0.3556 - accuracy: 0.8507 - val_loss: 0.2434 - val_accuracy: 0.9115
loss: 0.3634 - accuracy: 0.8516 - val_loss: 0.2269 - val_accuracy: 0.9115
loss: 0.3056 - accuracy: 0.8854 - val_loss: 0.2317 - val_accuracy: 0.9062
loss: 0.3006 - accuracy: 0.8834 - val_loss: 0.2347 - val_accuracy: 0.9062
loss: 0.2732 - accuracy: 0.8950 - val_loss: 0.2353 - val_accuracy: 0.9115
loss: 0.2697 - accuracy: 0.8921 - val_loss: 0.2293 - val_accuracy: 0.9062
loss: 0.2593 - accuracy: 0.8921 - val_loss: 0.2629 - val_accuracy: 0.8854
loss: 0.2559 - accuracy: 0.9037 - val_loss: 0.2165 - val_accuracy: 0.9115
loss: 0.2151 - accuracy: 0.9152 - val_loss: 0.2149 - val_accuracy: 0.9167
loss: 0.2109 - accuracy: 0.9268 - val_loss: 0.2105 - val_accuracy: 0.9219
loss: 0.2300 - accuracy: 0.9133 - val_loss: 0.2097 - val_accuracy: 0.9271
loss: 0.2206 - accuracy: 0.9114 - val_loss: 0.2169 - val_accuracy: 0.9167
loss: 0.2002 - accuracy: 0.9239 - val_loss: 0.2300 - val_accuracy: 0.9167
loss: 0.1885 - accuracy: 0.9258 - val_loss: 0.2119 - val_accuracy: 0.9271
loss: 0.1986 - accuracy: 0.9239 - val_loss: 0.2080 - val_accuracy: 0.9219
loss: 0.2050 - accuracy: 0.9268 - val_loss: 0.1945 - val_accuracy: 0.9323
```

Figure A.1: Sample accuracy of VGG-16 model. It includes accuracy and loss of both training and validation of few epochs

Appendix B

Code

B.1 File: part1.py

```
# -*- coding: utf-8 -*-
"""Dissertation.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/11GwE19ruJZiG8Lq4cXiN_pBAEglofZE-

# Libraries
"""

# Importing Libraries

from pandas.core.frame import DataFrame
from IPython.core.pylabtools import figsize
import pandas as pd
import numpy as np
import os
import glob
import matplotlib.pyplot as plt
from matplotlib.image import imread
from PIL import Image
from keras.preprocessing.image import ImageDataGenerator
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
import tensorflow.keras.applications.resnet50 as ResNet50

import torch
import torch.nn as nn
import torch.nn.functional as F

from torchvision.utils import make_grid

from PIL import Image
Image.MAX_IMAGE_PIXELS = None

import keras
from keras.applications.vgg16 import VGG16
from PIL import Image

from keras.models import Sequential
from keras.layers import Dropout, Dense, Flatten
from keras.optimizers import gradient_descent_v2

from keras.utils.vis_utils import plot_model

from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array
from keras.models import load_model

import torchvision.transforms as transforms


from torch.nn import init
from torch.autograd import Function

from math import sqrt

import random

from torch.autograd import grad

import config as CFG
from modules import ImageEncoder, TextEncoder, ProjectionHead

from transformers import DistilBertModel, DistilBertConfig

# Data Visualization
folder = '/content/drive/MyDrive/dataset/'

# ****************File Names*****************************

# path of the datasets, dataset stored in google drive

data_path = os.path.join(folder,'*utk')
files = glob.glob(data_path)
plt.figure(figsize=(15,20))


# Creating a dataframe of images
def image_dataframe(filepath):
  age = []
  gender = []
  for i in range(len(filepath)):
    filename = filepath[i]
    new_filename = filename.replace(folder,"")
    #new_filename.append()
    img_size = Image.open(filepath[i])
    width, height = img_size.size
    img_pixels = str(width) + " x " + str(height)
    info = new_filename.split("_")
```

```
    age.append(info[0])
    gender.append(info[1])


  data = {"age": age, "gender": gender, "files": filepath, "Image␣
      Size": img_pixels}
  df = pd.DataFrame.from_dict(data)
  df["age"] = df["age"].astype(str).astype(int)
  df["gender"] = df["gender"].astype(str).astype(int)

  return df

# Visualization function for dataset

def visualization(dataframe):
  for i in range(len(dataframe)):
    plt.subplot(5,4,i+1)
    image = imread(dataframe["files"][i])
    label = "Age:␣"+ str(int(dataframe["age"][i])) + "␣" +
        "Gender:␣" + str(int(dataframe["gender"][i]))
    plt.title(label)
    plt.imshow(image)

img_df = image_dataframe(files).head(20)
visualization(img_df)

"""#␣StyleGAN"""

#Reference: https://github.com/sangwoomo/freezeD

def display_tensor(image_tensor, num_images=32, size=(3, 150,
    150)):

    image_tensor = (image_tensor + 1) / 2
    image_unflat = image_tensor.detach().cpu().clamp_(0, 1)
    image_grid = make_grid(image_unflat[:num_images], nrow=4,
        padding=0)
    plt.imshow(image_grid.permute(1, 2, 0).squeeze())
    plt.axis('off')
    plt.show()

 class Dataset(torch.utils.data.Dataset):

    def __init__(self, root, n_classes=10, resolution=256):
        super().__init__()

        self.n_classes = n_classes

        # List of paths to training examples
        self.examples = []
        self.load_examples_from_dir(root)

        # Initialize transforms
        self.transforms = transforms.Compose([
            transforms.Resize((resolution, resolution),
                Image.LANCZOS),
            transforms.RandomHorizontalFlip(),
            transforms.Lambda(lambda x: np.array(x)),
            transforms.ToTensor(),
            transforms.Normalize((0.5, 0.5, 0.5), (0.5, 0.5, 0.5)),
        ])

    def load_examples_from_dir(self, abs_path):

        assert os.path.isdir(abs_path)

        img_suffix = '.png'

        n_classes = 0
        for root, _, files in os.walk(abs_path):
            if n_classes == self.n_classes:
                break
            for f in files:
                if f.endswith(img_suffix):
                    self.examples.append(root + '/' + f)
            n_classes += 1

    def __getitem__(self, idx):
        example = self.examples[idx]
        img = Image.open(example).convert('RGB')
        return self.transforms(img)

    def __len__(self):
        return len(self.examples)

#Reference: https://github.com/rosinality/style-based-gan-pytorch

def init_linear(linear):
    init.xavier_normal(linear.weight)
    linear.bias.data.zero_()


def init_conv(conv, glu=True):
    init.kaiming_normal(conv.weight)
```

```
    if conv.bias is not None:
        conv.bias.data.zero_()


class EqualLR:
    def __init__(self, name):
        self.name = name

    def compute_weight(self, module):
        weight = getattr(module, self.name + '_orig')
        fan_in = weight.data.size(1) * weight.data[0][0].numel()

        return weight * sqrt(2 / fan_in)

    @staticmethod
    def apply(module, name):
        fn = EqualLR(name)

        weight = getattr(module, name)
        del module._parameters[name]
        module.register_parameter(name + '_orig',
            nn.Parameter(weight.data))
        module.register_forward_pre_hook(fn)

        return fn

    def __call__(self, module, input):
        weight = self.compute_weight(module)
        setattr(module, self.name, weight)


def equal_lr(module, name='weight'):
    EqualLR.apply(module, name)

    return module


class FusedUpsample(nn.Module):
    def __init__(self, in_channel, out_channel, kernel_size,
        padding=0):
        super().__init__()

        weight = torch.randn(in_channel, out_channel, kernel_size,
            kernel_size)
        bias = torch.zeros(out_channel)

        fan_in = in_channel * kernel_size * kernel_size
        self.multiplier = sqrt(2 / fan_in)

        self.weight = nn.Parameter(weight)
        self.bias = nn.Parameter(bias)

        self.pad = padding

    def forward(self, input):
        weight = F.pad(self.weight * self.multiplier, [1, 1, 1, 1])
        weight = (
            weight[:, :, 1:, 1:]
            + weight[:, :, :-1, 1:]
            + weight[:, :, 1:, :-1]
            + weight[:, :, :-1, :-1]
        ) / 4

        out = F.conv_transpose2d(input, weight, self.bias,
            stride=2, padding=self.pad)

        return out


class FusedDownsample(nn.Module):
    def __init__(self, in_channel, out_channel, kernel_size,
        padding=0):
        super().__init__()

        weight = torch.randn(out_channel, in_channel, kernel_size,
            kernel_size)
        bias = torch.zeros(out_channel)

        fan_in = in_channel * kernel_size * kernel_size
        self.multiplier = sqrt(2 / fan_in)

        self.weight = nn.Parameter(weight)
        self.bias = nn.Parameter(bias)

        self.pad = padding

    def forward(self, input):
        weight = F.pad(self.weight * self.multiplier, [1, 1, 1, 1])
        weight = (
            weight[:, :, 1:, 1:]
            + weight[:, :, :-1, 1:]
            + weight[:, :, 1:, :-1]
            + weight[:, :, :-1, :-1]
        ) / 4
```

```
        out = F.conv2d(input, weight, self.bias, stride=2,
            padding=self.pad)

        return out


class PixelNorm(nn.Module):
    def __init__(self):
        super().__init__()

    def forward(self, input):
        return input / torch.sqrt(torch.mean(input ** 2, dim=1,
            keepdim=True) + 1e-8)


class BlurFunctionBackward(Function):
    @staticmethod
    def forward(ctx, grad_output, kernel, kernel_flip):
        ctx.save_for_backward(kernel, kernel_flip)

        grad_input = F.conv2d(
            grad_output, kernel_flip, padding=1,
                groups=grad_output.shape[1]
        )

        return grad_input

    @staticmethod
    def backward(ctx, gradgrad_output):
        kernel, kernel_flip = ctx.saved_tensors

        grad_input = F.conv2d(
            gradgrad_output, kernel, padding=1,
                groups=gradgrad_output.shape[1]
        )

        return grad_input, None, None


class BlurFunction(Function):
    @staticmethod
    def forward(ctx, input, kernel, kernel_flip):
        ctx.save_for_backward(kernel, kernel_flip)

        output = F.conv2d(input, kernel, padding=1,
            groups=input.shape[1])

        return output

    @staticmethod
    def backward(ctx, grad_output):
        kernel, kernel_flip = ctx.saved_tensors

        grad_input = BlurFunctionBackward.apply(grad_output,
            kernel, kernel_flip)

        return grad_input, None, None


blur = BlurFunction.apply


class Blur(nn.Module):
    def __init__(self, channel):
        super().__init__()

        weight = torch.tensor([[1, 2, 1], [2, 4, 2], [1, 2, 1]],
            dtype=torch.float32)
        weight = weight.view(1, 1, 3, 3)
        weight = weight / weight.sum()
        weight_flip = torch.flip(weight, [2, 3])

        self.register_buffer('weight', weight.repeat(channel, 1,
            1, 1))
        self.register_buffer('weight_flip',
            weight_flip.repeat(channel, 1, 1, 1))

    def forward(self, input):
        return blur(input, self.weight, self.weight_flip)
        # return F.conv2d(input, self.weight, padding=1,
            groups=input.shape[1])


class EqualConv2d(nn.Module):
    def __init__(self, *args, **kwargs):
        super().__init__()

        conv = nn.Conv2d(*args, **kwargs)
        conv.weight.data.normal_()
        conv.bias.data.zero_()
        self.conv = equal_lr(conv)

    def forward(self, input):
```

```
        return self.conv(input)


class EqualLinear(nn.Module):
    def __init__(self, in_dim, out_dim):
        super().__init__()

        linear = nn.Linear(in_dim, out_dim)
        linear.weight.data.normal_()
        linear.bias.data.zero_()

        self.linear = equal_lr(linear)

    def forward(self, input):
        return self.linear(input)

class ConvBlock(nn.Module):
    def __init__(
        self,
        in_channel,
        out_channel,
        kernel_size,
        padding,
        kernel_size2=None,
        padding2=None,
        downsample=False,
        fused=False,
    ):
        super().__init__()

        pad1 = padding
        pad2 = padding
        if padding2 is not None:
            pad2 = padding2

        kernel1 = kernel_size
        kernel2 = kernel_size
        if kernel_size2 is not None:
            kernel2 = kernel_size2

        self.conv1 = nn.Sequential(
            EqualConv2d(in_channel, out_channel, kernel1,
                padding=pad1),
            nn.LeakyReLU(0.2),
        )

        if downsample:
            if fused:
                self.conv2 = nn.Sequential(
                    Blur(out_channel),
                    FusedDownsample(out_channel, out_channel,
                        kernel2, padding=pad2),
                    nn.LeakyReLU(0.2),
                )

            else:
                self.conv2 = nn.Sequential(
                    Blur(out_channel),
                    EqualConv2d(out_channel, out_channel, kernel2,
                        padding=pad2),
                    nn.AvgPool2d(2),
                    nn.LeakyReLU(0.2),
                )

        else:
            self.conv2 = nn.Sequential(
                EqualConv2d(out_channel, out_channel, kernel2,
                    padding=pad2),
                nn.LeakyReLU(0.2),
            )

    def forward(self, input):
        out = self.conv1(input)
        out = self.conv2(out)

        return out

class Discriminator(nn.Module):

    def load_VGG-16(img, gender_output_A):

      model = load_model(model_folder+'/model_vgg␣(2)')
      result = model.predict(img)
      if result == gender_output_A:
        return True

    def __init__(self, fused=True, from_rgb_activate=False):
        super().__init__()

        self.progression = nn.ModuleList(
            [
                ConvBlock(16, 32, 3, 1, downsample=True,
                    fused=fused),  # 512
                ConvBlock(32, 64, 3, 1, downsample=True,
```

```
                fused=fused),  # 256
            ConvBlock(64, 128, 3, 1, downsample=True,
                fused=fused),  # 128
            ConvBlock(128, 256, 3, 1, downsample=True,
                fused=fused),  # 64
            ConvBlock(256, 512, 3, 1, downsample=True),  # 32
            ConvBlock(512, 512, 3, 1, downsample=True),  # 16
            ConvBlock(512, 512, 3, 1, downsample=True),  # 8
            ConvBlock(512, 512, 3, 1, downsample=True),  # 4
            ConvBlock(513, 512, 3, 1, 4, 0),
        ]
    )

    def make_from_rgb(out_channel):
        if from_rgb_activate:
            return nn.Sequential(EqualConv2d(3, out_channel,
                1), nn.LeakyReLU(0.2))

        else:
            return EqualConv2d(3, out_channel, 1)

    self.from_rgb = nn.ModuleList(
        [
            make_from_rgb(16),
            make_from_rgb(32),
            make_from_rgb(64),
            make_from_rgb(128),
            make_from_rgb(256),
            make_from_rgb(512),
            make_from_rgb(512),
            make_from_rgb(512),
            make_from_rgb(512),
        ]
    )

    # self.blur = Blur()

    self.n_layer = len(self.progression)

    self.linear = EqualLinear(512, 1)

def forward(self, input, step=0, alpha=-1):
    for i in range(step, -1, -1):
        index = self.n_layer - i - 1

        if i == step:
            out = self.from_rgb[index](input)

        if i == 0:
            out_std = torch.sqrt(out.var(0, unbiased=False) +
                1e-8)
            mean_std = out_std.mean()
            mean_std = mean_std.expand(out.size(0), 1, 4, 4)
            out = torch.cat([out, mean_std], 1)

        out = self.progression[index](out)

        if i > 0:
            if i == step and 0 <= alpha < 1:
                skip_rgb = F.avg_pool2d(input, 2)
                skip_rgb = self.from_rgb[index + 1](skip_rgb)

                out = (1 - alpha) * skip_rgb + alpha * out

    out = out.squeeze(2).squeeze(2)
    # print(input.size(), out.size(), step)
    out = self.linear(out)

    return out

class ConstantInput(nn.Module):
    def __init__(self, channel, size=4):
        super().__init__()

        self.input = nn.Parameter(torch.randn(1, channel, size,
            size))

    def forward(self, input):
        batch = input.shape[0]
        out = self.input.repeat(batch, 1, 1, 1)

        return out


class StyledConvBlock(nn.Module):
    def __init__(
        self,
        in_channel,
        out_channel,
        kernel_size=3,
        padding=1,
        style_dim=512,
        initial=False,
        upsample=False,
```

```
    fused=False,
):
    super().__init__()

    if initial:
        self.conv1 = ConstantInput(in_channel)

    else:
        if upsample:
            if fused:
                self.conv1 = nn.Sequential(
                    FusedUpsample(
                        in_channel, out_channel, kernel_size,
                            padding=padding
                    ),
                    Blur(out_channel),
                )

            else:
                self.conv1 = nn.Sequential(
                    nn.Upsample(scale_factor=2,
                        mode='nearest'),
                    EqualConv2d(
                        in_channel, out_channel, kernel_size,
                            padding=padding
                    ),
                    Blur(out_channel),
                )

        else:
            self.conv1 = EqualConv2d(
                in_channel, out_channel, kernel_size,
                    padding=padding
            )

    self.noise1 = equal_lr(NoiseInjection(out_channel))
    self.adain1 = AdaptiveInstanceNorm(out_channel, style_dim)
    self.lrelu1 = nn.LeakyReLU(0.2)

    self.conv2 = EqualConv2d(out_channel, out_channel,
        kernel_size, padding=padding)
    self.noise2 = equal_lr(NoiseInjection(out_channel))
    self.adain2 = AdaptiveInstanceNorm(out_channel, style_dim)
    self.lrelu2 = nn.LeakyReLU(0.2)

def forward(self, input, style, noise):
    out = self.conv1(input)
    out = self.noise1(out, noise)
    out = self.lrelu1(out)
    out = self.adain1(out, style)

    out = self.conv2(out)
    out = self.noise2(out, noise)
    out = self.lrelu2(out)
    out = self.adain2(out, style)

    return out


class Generator(nn.Module):
    def __init__(self, code_dim, fused=True):
        super().__init__()

        self.progression = nn.ModuleList(
            [
                StyledConvBlock(512, 512, 3, 1, initial=True),  # 4
                StyledConvBlock(512, 512, 3, 1, upsample=True),  #
                    8
                StyledConvBlock(512, 512, 3, 1, upsample=True),  #
                    16
                StyledConvBlock(512, 512, 3, 1, upsample=True),  #
                    32
                StyledConvBlock(512, 256, 3, 1, upsample=True),  #
                    64
                StyledConvBlock(256, 128, 3, 1, upsample=True,
                    fused=fused),  # 128
                StyledConvBlock(128, 64, 3, 1, upsample=True,
                    fused=fused),  # 256
                StyledConvBlock(64, 32, 3, 1, upsample=True,
                    fused=fused),  # 512
                StyledConvBlock(32, 16, 3, 1, upsample=True,
                    fused=fused),  # 1024
            ]
        )

        self.to_rgb = nn.ModuleList(
            [
                EqualConv2d(512, 3, 1),
                EqualConv2d(512, 3, 1),
                EqualConv2d(512, 3, 1),
                EqualConv2d(512, 3, 1),
                EqualConv2d(256, 3, 1),
                EqualConv2d(128, 3, 1),
                EqualConv2d(64, 3, 1),
```

```
            EqualConv2d(32, 3, 1),
            EqualConv2d(16, 3, 1),
        ]
    )

    # self.blur = Blur()

def forward(self, style, noise, step=0, alpha=-1,
    mixing_range=(-1, -1)):
    out = noise[0]

    if len(style) < 2:
        inject_index = [len(self.progression) + 1]

    else:
        inject_index = sorted(random.sample(list(range(step)),
            len(style) - 1))

    crossover = 0

    for i, (conv, to_rgb) in enumerate(zip(self.progression,
        self.to_rgb)):
        if mixing_range == (-1, -1):
            if crossover < len(inject_index) and i >
                inject_index[crossover]:
                crossover = min(crossover + 1, len(style))

            style_step = style[crossover]

        else:
            if mixing_range[0] <= i <= mixing_range[1]:
                style_step = style[1]

            else:
                style_step = style[0]

        if i > 0 and step > 0:
            out_prev = out

        out = conv(out, style_step, noise[i])

        if i == step:
            out = to_rgb(out)

            if i > 0 and 0 <= alpha < 1:
                skip_rgb = self.to_rgb[i - 1](out_prev)
                skip_rgb = F.interpolate(skip_rgb,
                    scale_factor=2, mode='nearest')
                out = (1 - alpha) * skip_rgb + alpha * out

            break

    return out


class StyledGenerator(nn.Module):
    def __init__(self, code_dim=512, n_mlp=8):
        super().__init__()

        self.generator = Generator(code_dim)

        layers = [PixelNorm()]
        for i in range(n_mlp):
            layers.append(EqualLinear(code_dim, code_dim))
            layers.append(nn.LeakyReLU(0.2))

        self.style = nn.Sequential(*layers)

    def forward(
        self,
        input,
        noise=None,
        step=0,
        alpha=-1,
        mean_style=None,
        style_weight=0,
        mixing_range=(-1, -1),
    ):
        styles = []
        if type(input) not in (list, tuple):
            input = [input]

        for i in input:
            styles.append(self.style(i))

        batch = input[0].shape[0]

        if noise is None:
            noise = []

            for i in range(step + 1):
                size = 4 * 2 ** i
                noise.append(torch.randn(batch, 1, size, size,
                    device=input[0].device))
```

```
        if mean_style is not None:
            styles_norm = []

            for style in styles:
                styles_norm.append(mean_style + style_weight *
                    (style - mean_style))

            styles = styles_norm

        return self.generator(styles, noise, step, alpha,
            mixing_range=mixing_range)

    def mean_style(self, input):
        style = self.style(input).mean(0, keepdim=True)

        return style

class AdaptiveInstanceNorm(nn.Module):
    def __init__(self, in_channel, style_dim):
        super().__init__()

        self.norm = nn.InstanceNorm2d(in_channel)
        self.style = EqualLinear(style_dim, in_channel * 2)

        self.style.linear.bias.data[:in_channel] = 1
        self.style.linear.bias.data[in_channel:] = 0

    def forward(self, input, style):
        style = self.style(style).unsqueeze(2).unsqueeze(3)
        gamma, beta = style.chunk(2, 1)

        out = self.norm(input)
        out = gamma * out + beta

        return out


class NoiseInjection(nn.Module):
    def __init__(self, channel):
        super().__init__()

        self.weight = nn.Parameter(torch.zeros(1, channel, 1, 1))

    def forward(self, image, noise):
        return image + self.weight * noise
```

B.2 File: part3.py

```
# -*- coding: utf-8 -*-
"""Part3.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1SpbjBUVJ2Jghk6YfLCDB-zKgJlQ0ix5y
"""

train_dir = os.path.join(base_dir,'train')
val_dir   = os.path.join(base_dir,'val')
test_dir  = os.path.join(base_dir,'test')
if not os.path.exists(train_dir):
  print(train_dir +' does not exist.')
if not os.path.exists(val_dir):
  print(val_dir +' does not exist.')
if not os.path.exists(test_dir):
  print(test_dir +' does not exist.')

n_train_per_class = 1600
n_val_per_class = 2000
n_test_per_class = 2000

from keras.preprocessing.image import ImageDataGenerator

train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range = 40,
    width_shift_range = 0.2,
    height_shift_range = 0.2,
    shear_range = 0.2,
    zoom_range = 0.2,
    horizontal_flip = True,
    fill_mode = 'nearest')
test_datagen = ImageDataGenerator(rescale=1./255)

batch_size = 32
# this is a generator that will read pictures found in
# subfolers of training data, and indefinitely generate
# batches of training image data
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size = (150,150),# all images will be resized to 150x150
    batch_size=batch_size,
    class_mode='binary')# since we use binary_crossentropy loss,
        we need binary labels

# this is a similar generator, for validation data
val_generator = test_datagen.flow_from_directory(
    val_dir,
    target_size = (150,150),
    batch_size=batch_size,
    class_mode='binary')
import keras
from keras.applications.vgg16 import VGG16
from PIL import Image

Image.MAX_IMAGE_PIXELS = None


# build the convolutional base of VGG16 network
VGG16_conv_base = VGG16(weights = 'imagenet', # the model if
    pre-trained on ImageNet
                  include_top = False,
                  input_shape = (150,150,3))

# set the early layers (up to the 'block5_conv1')
# to non-trainable (weights will not be updated)
for layer in VGG16_conv_base.layers[:14]:
    layer.trainable = False

VGG16_conv_base.summary()

sgd = gradient_descent_v2.SGD(lr=1e-4, momentum=0.9)

top_model = Sequential()
top_model.add(VGG16_conv_base)
top_model.add(Flatten())
top_model.add(Dropout(0.5))#
top_model.add(Dense(256, activation='relu'))
top_model.add(Dropout(0.5))
top_model.add(Dense(1, activation='sigmoid'))

top_model.summary()
from keras.utils.vis_utils import plot_model
plot_model(top_model, show_shapes=True)

top_model.compile(loss='binary_crossentropy',
              optimizer= sgd,
              metrics=['accuracy'])
```

```
history=top_model.fit(
        train_generator,
        steps_per_epoch=33,
        epochs=25,
        validation_data=val_generator,
        validation_steps= 6)

import matplotlib.pyplot as plt
fig = plt.figure(figsize = [6,3], dpi = 480)
fig.add_subplot(2,1,1)
plt.plot(history.history['loss'], label='train loss')
plt.plot(history.history['val_loss'], label='val loss')
plt.legend()
plt.grid(True)
plt.ylim([0,1.0])
plt.xlabel('epoch')

fig.add_subplot(2,1,2)
plt.plot(history.history['accuracy'], label='train accuracy')
plt.plot(history.history['val_accuracy'], label='val accuracy')
plt.legend()
plt.grid(True)
plt.ylim([0,1.0])
plt.xlabel('epoch')

model_folder = '/content/gdrive/My Drive/Colab Notebooks/models'
if not os.path.exists(model_folder):
    os.mkdir(model_folder)
top_model.save(model_folder+'/CNN_vgg(3).h5')

test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size = (150, 150),
    batch_size = batch_size,
    class_mode = 'binary'
)
test_loss, test_acc = top_model.evaluate(test_generator,
    steps=len(n_test_per_class) // batch_size)
print('Test accuracy: ', test_acc)
```