B.TECH PROJECT REPORT

Generative face completion

Submitted in partial fulfillment of the requirements
of the degree of
Bachelor of Technology

By

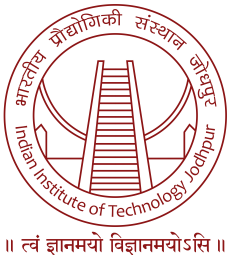
Sohail Khan (B16CS036)

Vishakh S (B16CS038)

Zaid Khan (B16CS040)

Under the guidance of

Dr. Gaurav Harit



Dept. of Computer Science and Engineering

Indian Institute of Technology Jodhpur

April 2019

Abstract

Inpainting, a common image processing operation, aims to fill the missing regions of an image with synthesized components. It can prove beneficial to a multitude of applications ranging from restoration of damaged paintings and photographs to replacement of selected objects in images. Generating semantically valid pixels to fill sections of a human face can be particularly challenging, due to the presence of large variations in appearance. A copy and paste strategy may perform well for background completion, but may end up producing invalid results when used to complete facial images. A recently developed technology that can prove to be useful in such situations, is Generative Adversarial Networks (GAN). Using generative networks for face completion gives promising results.

Acknowledgements

We would like to express our sincere gratitude to our BTP supervisor Dr. Gaurav Harit for providing us invaluable guidance and suggestions throughout the course of this project. This report would be incomplete without thanking him for keeping us motivated, while giving us an opportunity to work at our own pace.

We offer our sincere thanks to the researchers at MMLAB, The Chinese University of Hong Kong for making the CelebA dataset available open-source.

Contents

List of Figures

1 Introduction

Image completion is a reconstruction operation that fills the lost regions of an image with generated components. Being able to fill in the masked regions can be very useful for several applications, from replacing the lost blocks in the transmission of images to restoring deteriorated sections in a valuable mural. Consider for example, the removal of red eyes from an image. The concept of inpainting can be employed to generate visually consistent pixels to replace the red eyes. Removal of logos from videos can also be done along the same lines.

Most of the existing algorithms employ a simple copy and paste strategy for image completion. This works by searching for patches of the image that appear similar to the one to be synthesized. If the search is successful, the existing patterns and structures in the known patch are used for the generation. This strategy works particularly well for background completion. For example, replacing a patch of grass in an image with another doesn't compromise the visual consistency.

However, the assumption that we would be able to encounter similar patches in the same image is far from the reality. Many objects (like a human face) contain unique patterns that cannot be matched with any other pattern in the same image. So, an attempt to synthesize components by a copy-and-paste strategy is futile. An alternative would be to search for similar patches in a large external database. Although patches that match the unique patterns may not be found, it is possible to generate contents that fit well within the context such that the completed image appears to be visually realistic.

A particularly promising technology is GAN (Generative Adversarial Networks). GAN is a kind of neural network with two separate deep neural networks competing each other: a generator (G) and a discriminator (D). The generator fills the missing regions of the masked image to produce an image that would look like an original one. The discriminator on the other hand, distinguishes between a real and fake (reconstructed) image. When the discriminator classifies an image as fake, the generator produces a new one. This process continues in a loop until the generator is able to fool the discriminator into classifying a reconstructed image as real. Figure 1.1 shows the architecture of a generative adversarial network.

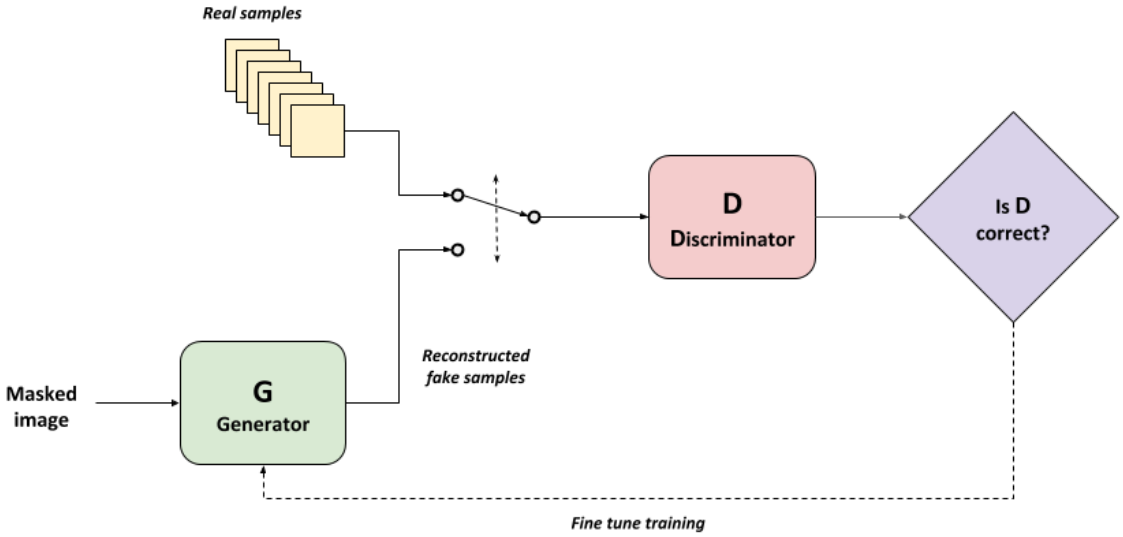


FIGURE 1.1: Architecture of GAN

In this project, a neural network has been trained to synthesize pixels for the masked regions. The deep generative completion model used, consists of an encoding-decoding generator and two adversarial discriminators - a local discriminator ensuring local consistency (texture) and a global discriminator ensuring the global consistency (tonality) [2]. The model has been trained with a combination of a reconstruction loss and two adversarial losses. It has been observed that the model is able to tackle the challenging face completion task by generating semantically valid patterns based on learned representations. Further, the idea of semantic parsing has been explored in the light of making the reconstructed patches consistent with the surrounding contexts. The results, show that use of GAN for face completion tasks gives promising results.

2 Dataset used

The open-source CelebFaces Attributes (**CelebA**) dataset has been utilized for the purpose of learning and evaluating the model. The images in this dataset were originally collected by researchers at MMLAB, The Chinese University of Hong Kong [3]. CelebA is a large-scale facial attributes dataset with:

- **10,177** unique identities
- **2,02,599** face images covering large pose variations and background clutter
- **40 binary attributes per image**, namely:
  - 5_o_Clock_Shadow
  - Arched_Eyebrows
  - Attractive
  - Bags_Under_Eyes
  - Bald
  - Bangs
  - Big_Lips
  - Big_Nose
  - Black_Hair
  - Blond_Hair
  - Blurry
  - Brown_Hair
  - Bushy_Eyebrows
  - Chubby
  - Double_Chin
  - Eyeglasses
  - Goatee
  - Gray_Hair
  - Heavy_Makeup
  - High_Cheekbones
  - Male
  - Mouth_Slightly_Open
  - Mustache
  - Narrow_Eyes
  - No_Beard
  - Oval_Face
  - Pale_Skin
  - Pointy_Nose
  - Receding_Hairline
  - Rosy_Cheeks
  - Sideburns
  - Smiling

- Straight_Hair
- Wavy_Hair
- Wearing_Earrings
- Wearing_Hat
- Wearing_Lipstick
- Wearing_Necklace
- Wearing_Necktie
- Young

- Each face image is cropped and roughly aligned by the positions of the two eyes and reshaped to 128x128x3 pixels

FIGURE 2.1: Sample images from the CelebA dataset

Figure 2.1 depicts a few image samples from the dataset with attributes: (a) Eyeglasses (b) Bangs (c) Pointed nose (d) Oval face (e) Wearing hat (f) Wavy hair (g) Mustache (h) Smiling

Each of the 40 attributes is associated with a label. Hence, each image is associated with 40 labels, one for each attribute. An attribute label of **1** indicates the presence of the feature specified by that attribute, in the image. On the other hand, an attribute value of **-1** indicates its absence in the image.

Figure 2.2 shows the labels corresponding to a few attributes (13 out of 40) for the first 20 images in the dataset. The first column corresponds to the index while the second column depicts the image id. The subsequent columns accomodate the labels corresponding to each attribute.

	image_id	5_o_Clo...	Arched...	Attracti...	Bags_U...	Bald	Bangs	Big_Lips	Big_Nose	Black_...	Blond_...	Blurry	Brown_...	Bushy_...
1	000001.jpg	-1	1	1	-1	-1	-1	-1	-1	-1	-1	-1	1	-1
2	000002.jpg	-1	-1	-1	1	-1	-1	-1	1	-1	-1	-1	1	-1
3	000003.jpg	-1	-1	-1	-1	-1	-1	1	-1	-1	-1	1	-1	-1
4	000004.jpg	-1	-1	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
5	000005.jpg	-1	1	1	-1	-1	-1	1	-1	-1	-1	-1	-1	-1
6	000006.jpg	-1	1	1	-1	-1	-1	1	-1	-1	-1	-1	1	-1
7	000007.jpg	1	-1	1	1	-1	-1	1	1	1	-1	-1	-1	1
8	000008.jpg	1	1	-1	1	-1	-1	1	-1	1	-1	-1	-1	-1
9	000009.jpg	-1	1	1	-1	-1	1	1	-1	-1	-1	-1	-1	-1
10	000010.jpg	-1	-1	1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
11	000011.jpg	-1	-1	1	-1	-1	-1	-1	-1	1	-1	-1	-1	-1
12	000012.jpg	-1	-1	1	1	-1	-1	-1	-1	1	-1	-1	-1	1
13	000013.jpg	-1	-1	-1	-1	-1	-1	-1	-1	-1	1	-1	-1	-1
14	000014.jpg	-1	1	-1	-1	-1	-1	-1	1	1	-1	-1	-1	1
15	000015.jpg	1	-1	-1	1	-1	-1	-1	1	-1	-1	-1	-1	-1
16	000016.jpg	1	-1	-1	1	-1	-1	-1	-1	-1	-1	-1	-1	-1
17	000017.jpg	-1	-1	-1	-1	-1	-1	-1	-1	1	-1	-1	-1	-1
18	000018.jpg	-1	1	-1	-1	-1	-1	-1	1	-1	1	-1	-1	-1
19	000019.jpg	-1	1	1	-1	-1	-1	-1	-1	-1	1	-1	-1	-1
20	000020.jpg	-1	-1	-1	-1	-1	-1	-1	-1	1	-1	-1	-1	-1

FIGURE 2.2: Snapshot of attribute labels of 20 images from the CelebA dataset

3 Approach

This section elaborates on the approach used to reconstruct images using generative adversarial nets. Given a masked image, our objective is to synthesize missing contents that are both **semantically consistent** and **visually realistic**. Section 3.1 introduces GAN. Section 3.2 and 3.3 elucidate the model used to synthesize the missing components. The subsequent sections expound each component in the used model.

3.1 Technology used

Often hailed as "*the most interesting idea in the last 10 years in Machine Learning*", **generative adversarial networks** (GANs in short) have enjoyed huge success ever since they were introduced in 2014 by Ian J. Goodfellow and co-authors in the article Generative Adversarial Nets [1]. GANs are deep neural net architectures comprised of two nets, competing with each other (hence the term adversarial). This very simple setup results in both of the nets coming up with increasingly complex results to fool the other. This situation is analogous to a minimax game in Game Theory.

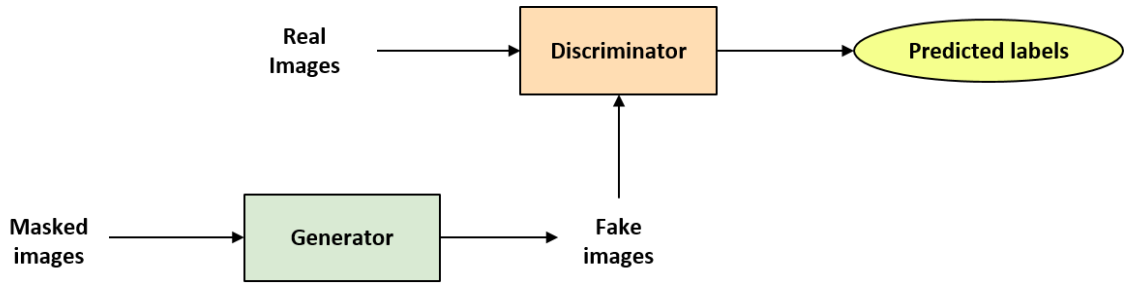


FIGURE 3.1: Architecture of a conventional GAN

In the case of a GAN, we have a Generator which takes as input, random noise and tries to synthesize data very close to the dataset we have. The other network (the adversary) is called the Discriminator. It takes as input, the generated data and tries to discriminate the generated data and real data. This process of generation and rejection (by the discriminator)

would continue until the generator is able to produce images that are close to the real images .i.e the discriminator is fooled. Figure 3.1 depicts the architecture of a conventional GAN.

Formally, the objective function for a GAN can be written along the lines of a minimax game as [1]:

$$\min_G \max_D V(D,G) = E_{x\ in\ p_{data}(x)}(\log(D(x))) + E_{z\ in\ p_z(z)}(\log(1 - D(G(z))))$$

where

- x - training sample or x_{real}
- z - noise vector
- G and D are differentiable functions modelling the generator and discriminator
- G(z) - generator's output or x_{fake}
- D(x) - discriminator's output for x_{real}
- D(G(z)) - discriminator's output for x_{fake}
- the first term models the recognition of real images
- the second term models the recognition of generated images

So, the discriminator tries to maximize D(x) and minimize D(G(z)). On the other hand, the generator attempts to maximize D(G(z)).

As the discriminator changes its behavior, so does the generator, and vice versa. Put another way, as time passes, the generator gets better and better at synthesizing pixels for the missing components. With time, the discriminator also gets better at separating fake images from real ones. To sum it up, both networks are trying to optimize a different and opposing objective function, or loss function. Their losses push against each other.

3.2 Generator

The generator **G** is designed as an autoencoder to synthesize new contents for the masked regions in the input image. The masked input (along with filled noise), is first mapped to latent representations through the encoder. The latent representation captures the relationships between the known and unknown regions. This hidden representation is then fed to the decoder for generating the desired contents.

We use the architecture from conv1 to pool3 of the VGG-19 network, stack two more convolutional layers on top of it and add a fully-connected layer after that to complete the encoder. The decoder is symmetric to the encoder, but with unpooling layers. Figure 3.2

depicts the architecture of the generator G.

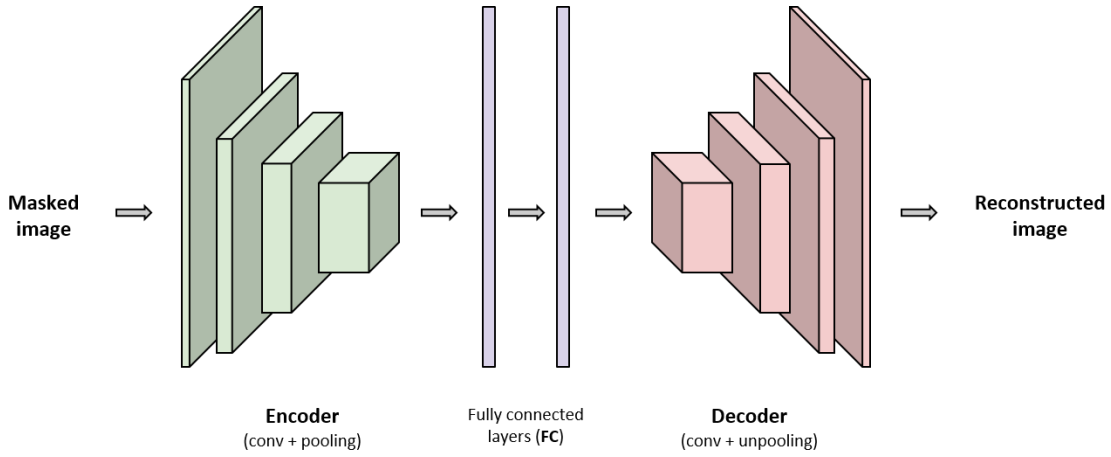


FIGURE 3.2: Architecture of the generator

Both GANs and autoencoders are generative models, .i.e. they learn a given data distribution rather than its density. The key difference is the manner in which they do it. Auto encoders learn a given distribution by comparing its input and output. This is good for learning hidden representation of data, but is pretty bad for generating new data. This is because we learn an averaged representation of the data and thus the output becomes pretty blurry. Generative adversarial networks on the other hand, take an entirely different approach. They use a second network called the discriminator to measure the distance between the generated and the real data. It receives some data as an input and returns a number between 0 and 1. 0 meaning the data is fake and 1 meaning 1 it is real. The generators goal then is learning to convince the discriminator into believing it is generating real data.

The main advantage of GANs over autoencoders in generating data is that they can be conditioned by different inputs. For example, you can use it to learn the mapping between two domains or to reproduce several classes of data. Hence, they can be used for different tasks. An autoencoder, on the other hand compresses its input down to a vector - with much fewer dimensions than its input data, and then transforms it back into a tensor with the same shape as its input over several neural net layers. They're trained to reproduce their input, so it's kind of like learning a compression algorithm for that specific dataset. Put another way, it compresses data to lower dimension and generating semantic vectors from it. Figure 3.3 provides a visualization of the above mentioned compression. The hidden layer

in between, acts as a bottleneck, constraining the amount of information that can traverse the full network. Hence, it forces a learned compression of the input data.

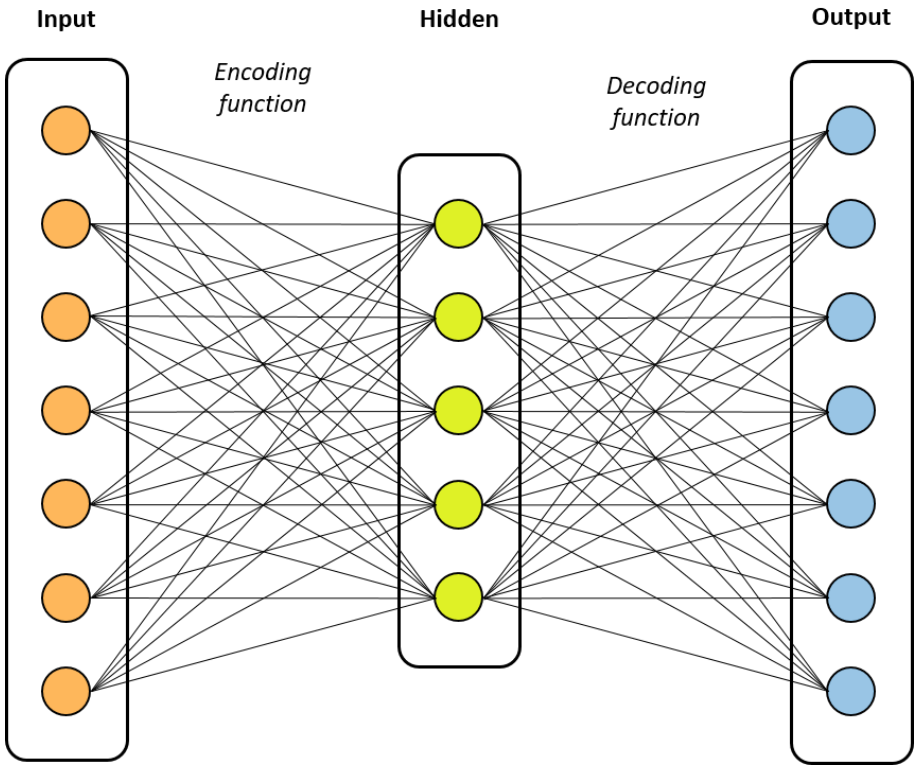


FIGURE 3.3: Autoencoder

For generative face completion, our network requires to understand the semantics of image. It has to then generate semantically valid contents for the masked image. Hence, we are using an autoencoder as the generator.

3.3 Discriminator

While the generator fills the masked section of the input image, it does not ensure that the reconstructed pixels are coherent and visually realistic. The pixels generated by G may be quite blurry and only capturing the coarse shape of missing face components. To encourage the production of realistic results with sharp facial features, we use a discriminator D that serves as a binary classifier to distinguish between real (images in the dataset) and fake (generated) images. The primary objective of this discriminator is to help improve the quality of the synthesized results.

Two discriminators have been employed to guarantee local as well as global consistency. We first use a **local D** for the missing region, which determines whether the contents synthesized for the missing region are real or not. The local D helps generate details of the missing contents with sharper boundaries. In addition, it encourages the generated object parts to be semantically valid.

However, the limitations of local D are also obvious due to locality. First, the local loss can neither regularize the global structure of a face, nor guarantee the statistical consistency within and outside the masked regions. For example, the tonality (fair/dark complexion) inside and outside the mask may be different. Second, while the generated new pixels are conditioned on their surrounding contexts, a local D can hardly have a direct impact outside the masked regions during the back propagation, due to the unpooling structure of the decoder. Consequently, the inconsistency of pixel values along region boundaries becomes obvious.

Therefore, we use another **global D** to determine the faithfulness of an entire image. The fundamental idea is that the newly generated contents should not only be realistic, but also consistent with the surrounding context. The global D greatly alleviates the inconsistency issue and further enforce the generated contents to be more realistic. To sum it up, while the local D ensures proper texture, the global D guarantees proper tonality so that the output seems visually realistic.

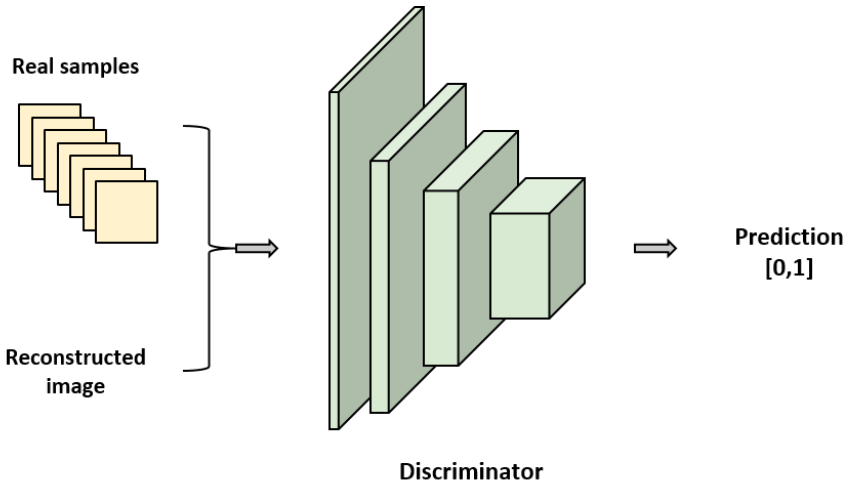


FIGURE 3.4: Architecture of the discriminator

Figure 3.4 shows the architecture of the discriminator used. It uses four convolution layers to predict a label (real/fake) corresponding to the image provided to it. Four convolution

layers have been used instead of one. This ensures that the discriminator is strong enough, to encourage the generator to produce facial contents which are semantically consistent.

3.4 Objective function

We first introduce a reconstruction loss L_r to the generator, which is the L_2 distance between the network output and the original image. With the L_r only, the generated contents tends to be blurry and smooth. The reason is that, the L_2 loss penalizes outliers heavily. The network in an attempt to avoid large penalties, ends up synthesized contents that appear blurry.

We employ adversarial loss, which is reflection of how the generator can maximally fool the discriminator and how well the discriminator can distinguish between fake and real images. It is defined as:

$$L_{a_i} = \min_G \max_D E_{x\ in\ p_{data}(x)}(\log(D(x))) + E_{z\ in\ p_z(z)}(\log(1 - D(G(z))))$$

where

- L - **the adversarial loss**
- a_i **- the two discriminators**
- x - training sample or x_{real}
- z - noise vector
- G and D are differentiable functions modelling the generator and discriminator
- G(z) - generator's output or x_{fake}
- D(x) - discriminator's output for x_{real}
- D(G(z)) - discriminator's output for x_{fake}
- the first term models the recognition of real images
- the second term models the recognition of generated images

- D tries to :
  - Maximize the probability of real images being labelled as real
  - Minimize the probability of generated images being labelled as real
- G tries to minimize the probability of the generated images being labelled by the discriminator as fake.

The local and global discriminators, a_1 and a_2 share the same definition of loss function. The only difference is that the local discriminator just provides training signals (loss gradient) for the missing regions while the global discriminator back-propogates loss gradient across the entire image. Figure 3.5 depicts the same.

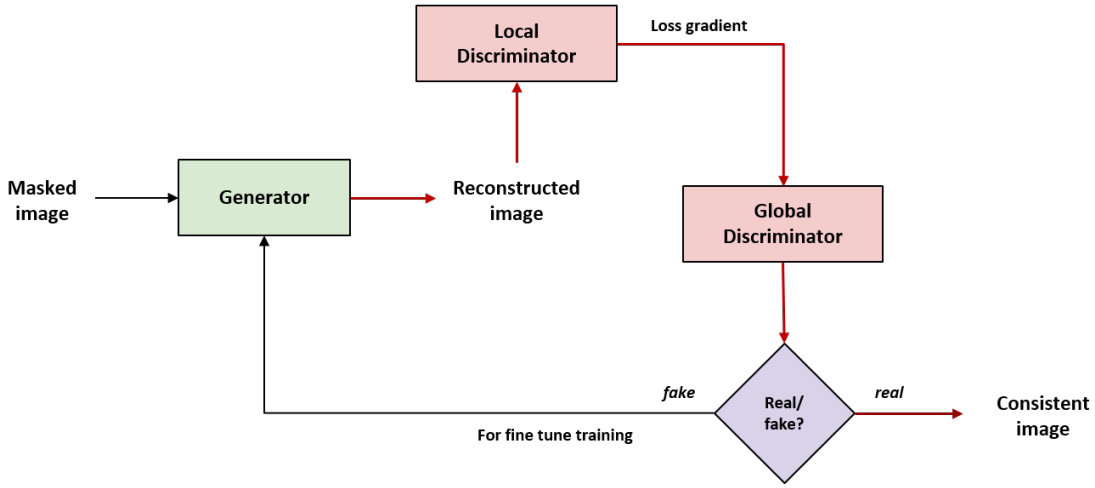


FIGURE 3.5: Backpropogation

Section 4 describes in detail, the results obtained by using the above model. (A short note on autoencoders can be found in Appendix A.)

4 Implementation and results

4.1 A sneak peek into the implementation

For the purpose of implementing the model, we have made use of the PyTorch package. PyTorch is a python package that provides two high level features:

- Tensor computation (like numpy) with strong GPU acceleration
- Deep Neural networks built on a tape-based autograd system

To put it simply, PyTorch is essentially a deep learning platform that provides maximum flexibility and speed. Other packages can also be used along with PyTorch like Numpy, Scipy and Cython.

4.2 Output obtained

Figures 4.1, 4.2, 4.3 and 4.4 depict the output obtained by our model. The first row corresponds to the real image. The second and third row display the masked image (input to the model) and the generated output respectively.

The results show that our model generates visually pleasing contents for the key missing facial components. The model is robust and quite flexible. It can handle a variety of maskings and occlusions. Further, the output generated is of high perceptual quality, reiterating the fact that the use of GAN for inpainting tasks provides promising outputs.

FIGURE 4.1: Output-1

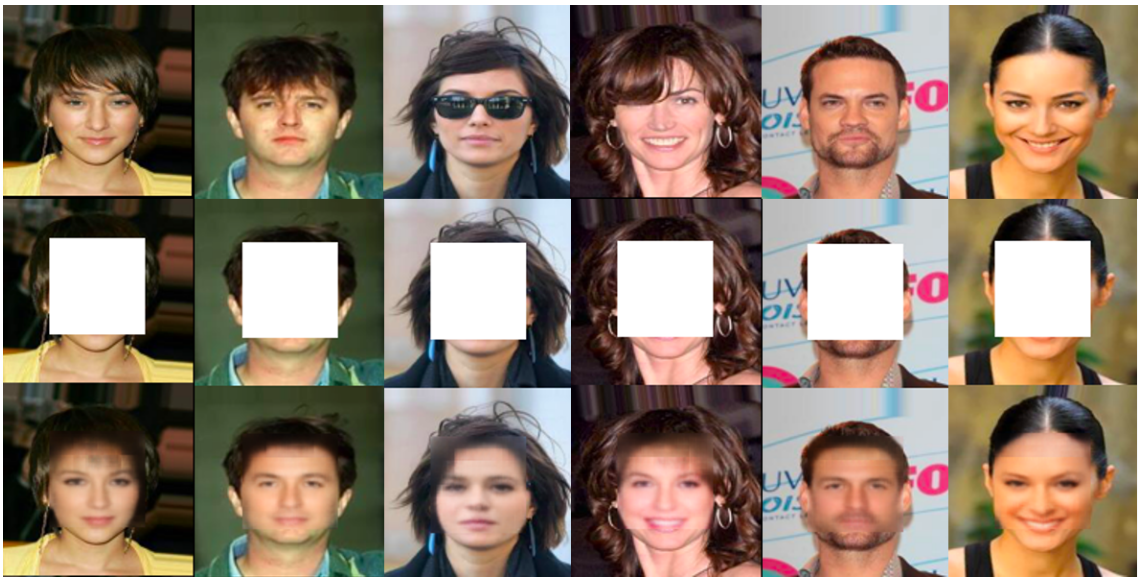


FIGURE 4.2: Output-2

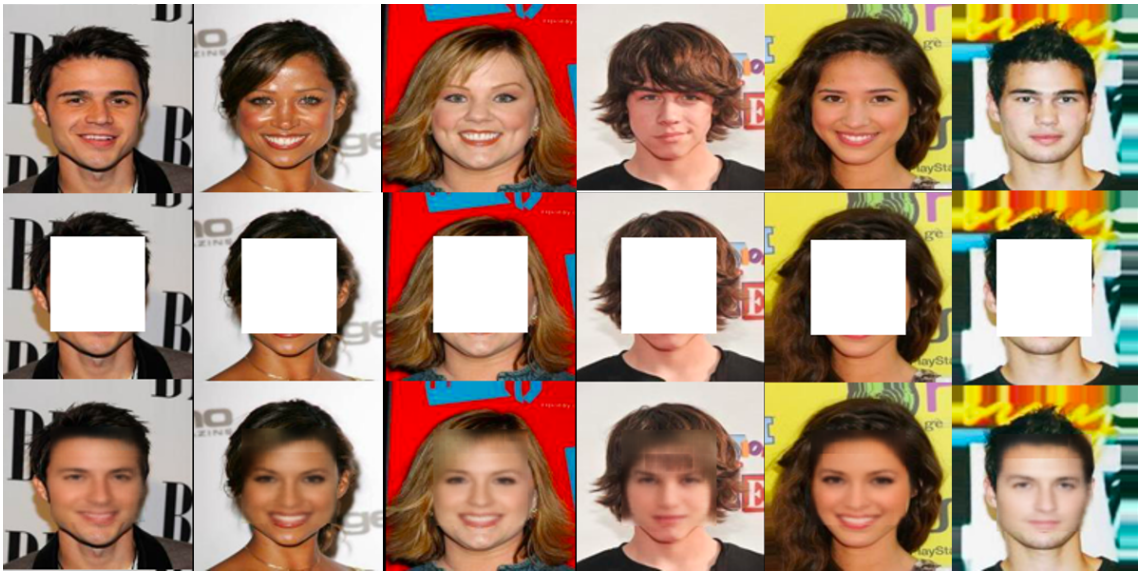


FIGURE 4.3: Output-3

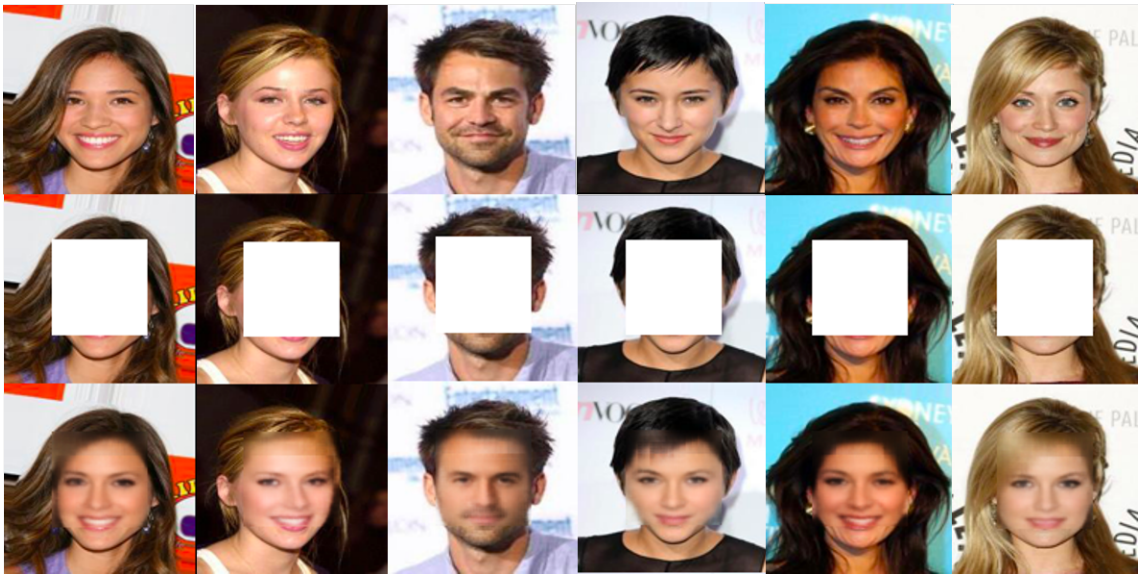


FIGURE 4.4: Output-4

5 Limitations

Although our model is able to generate semantically plausible and visually pleasing contents, it has some limitations. Firstly, the faces in the CelebA dataset are roughly cropped and not properly aligned. Our model cannot handle some unaligned faces. We show a few failure cases in the figure 5.1. The unpleasant synthesized contents indicate that the network does not recognize the position/orientation of the face and its corresponding components. This issue can be alleviated with 3D data augmentation.

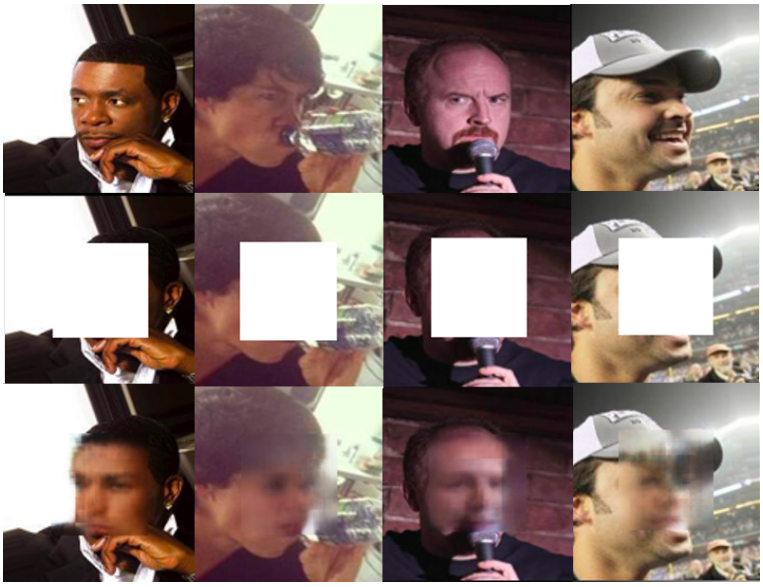


FIGURE 5.1: Failure cases

Our model does not fully exploit the spatial correlations between adjacent pixels. As a result, if one half of the lips is masked, there is no guarantee that the generated components match in color with the unmasked ones. This can be alleviated by the use of pixel-level recurrent neural networks [4].

Further, if one eye is masked, it may so happen that the generated iris colour is not same as that of the unmasked one. This problem can be solved by the use of a semantic parsing network.

While the contents generated by our model are semantically consistent, there is no guarantee that the output generated would resemble a person of the same gender as the input. In some cases, the output generated by our model more closely resembled a female even when the input image was that of a male. This can be attributed to the fact that the dataset has far fewer images of males than females.

6 Conclusions and future work

In this project, a deep generative network has trained for face completion. The network is based on a GAN, with an autoencoder as the generator and two adversarial loss functions - local and global. The model is successful in its attempt to synthesize semantically valid contents for the missing facial parts. The results provide a convincing evidence for employing a generative adversarial network for completing missing regions of the face.

Our GAN model may generate independant facial components that are not suitable to the original subjects with respect to facial expressions and part's shapes. Say for example, if only one eye is masked, the model may end up generating an eye that does not fit well with the other one. Put another way, it may so happen that the generate eye is asymmetric to the unmasked one or the generated eye is wierdly big. This is because the global D is not able to ensure the finer details related to the generated image. In future work, we plan to introduce a semantic parsing network to enhance the harmony between the generated contents and the existing pixels [4].

A Note on autoencoders

An autoencoder is a type of artificial neural network used to learn efficient data encodings in an unsupervised manner. The aim of an autoencoder is to learn a representation (encoding) for a set of data, typically for dimensionality reduction, by training the network to ignore signal "noise". Along with reduction side, a reconstructing side is learnt, where the autoencoder tries to generate from the reduce encoding, a representation as close as possible to its original input, hence its name. Recently, the autoencoder concept has become more widely used for learning generative models of data. An autoencoder learns to compress data from the input layer into a short code, and then uncompress that code into something that closely matches the original data. This forces the autoencoder to engage in dimensionality reduction, for example by learning how ignore noise. Some architecture use stacked sparse autoencoder layer for image recognition.

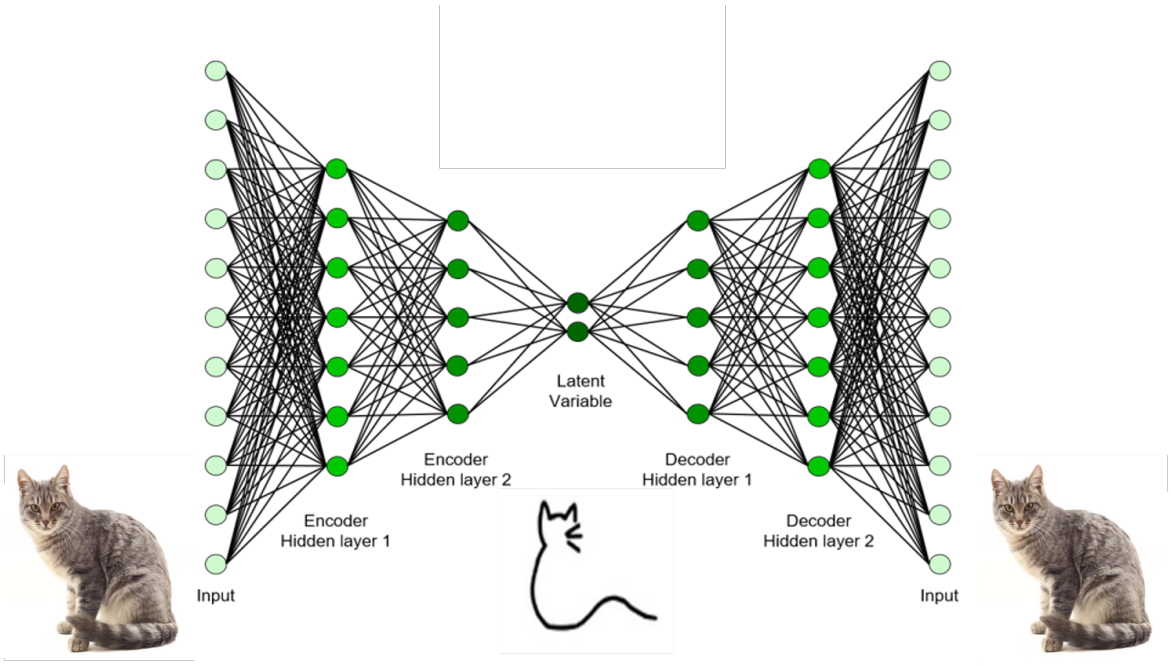


FIGURE A.1: Autoencoder on an image of a cat

For example, consider the example of using an autoencoder on an image of a cat. The first encoding layer might learn to encode easy features like corners, the second to analyze

the first layer's output and then encode less local features like the tip of a nose, the third might encode a whole nose, until the final encoding layer encodes the whole image into code that matches the concept of cat. The decoding layers will learn to decode the learnt code representation back into its original form as close as possible. Figure A.1 depicts the above scenario.

An alternative use a generative model: for example if a system is manually fed the codes it has learned for bird and swimming, it may attempt to generate an image of a swimming bird, even if it has never seen it before.

References

[1] Ian J. Goodfellow et al. “Generative Adversarial Nets”. In: *NIPS*. 2014 (cit. on pp. 7, 8).

[2] Yijun Li et al. “Generative Face Completion”. In: *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (2017), pp. 5892–5900 (cit. on p. 2).

[3] Ziwei Liu et al. “Deep Learning Face Attributes in the Wild”. In: *Proceedings of International Conference on Computer Vision (ICCV)*. 2015 (cit. on p. 3).

[4] Xintao Wang et al. “ESRGAN: Enhanced Super-Resolution Generative Adversarial Networks”. In: (Sept. 2018) (cit. on pp. 19, 21).