Paper implementation of Image-to-Image Translation with Conditional Adversarial Network: Pix2Pix

Aryaman Chauhan	2020B5A72006P
Devansh	2020B5A72001P
Harshvardhan Jouhary	2020B2A31623P

Birla Institute of Technology and Science, Pilani

Artificial Intelligence: CS F407

Submitted To : Dr. Aneesh Sreevallabh Chivukula

Abstract

Here, we have successfully implemented Pix2Pix paper, which was implemented using the Pytorch library. Throughout the process, we have followed the process called CRISP-DM, which stands for Cross-Industry Process for Data Mining. Through this lab project, we have successfully understood the working of GANs and their characteristics. Pix2Pix is a conditional GAN, which implements two GANs, namely a modified U-Net for Generator, and a PatchGAN for discriminator.

Keywords: GANs, Pytorch, Deep Learning, CRISP-DM, U-Net, PatchGAN, Conditional Adversarial Network.

Contents

Acknowledgement

Understanding and implementing a GAN in the short period of 2 months would’ve been difficult without the help of certain people. Firstly, we would like to thank our Institute, Birla Institute of Technology and Science, Pilani, which provided us with the wonderful opportunity to pursue the course of Artificial Intelligence.

We would like to express immense gratitude to our professor, **Dr. Aneesh Sreevallabh Chivukula,** who provided his guidance, and gave us his immense insights whenever we needed aid. Without his suggestions, this implementation would’ve been a uphill struggle.

Lastly, we would like to thank our friends for some quick insights, and our parents for providing moral support.

Image-to-Image Translation with Conditional Adversarial Network: Pix2Pix using PyTorch

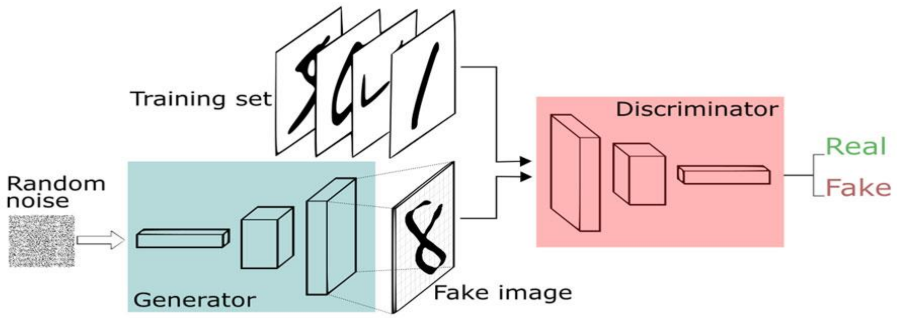
It is difficult to formulate a loss function, for image-to-image translation. But the paper suggests using GANs to learn this loss function, thus not needing to formulate this by hand. It uses a U-Net like architecture for Generator, and a PatchGAN for Discriminator. So, first we need to understand the model, and understand why do we want to implement this model.

1. Business Understanding

We will understand this paper by understanding a variety of models, to make our understanding concrete through a number of steps.

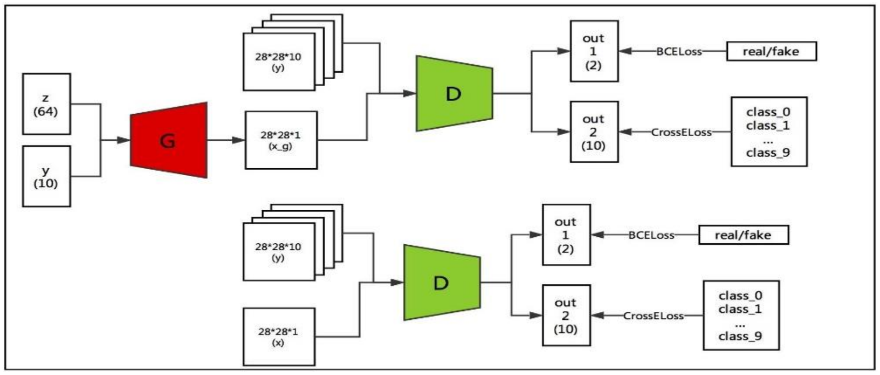
a. GANs:

GANs, or Generative Adversarial Networks, work on the principle of putting our machine against an adversary. They do this by training two models, one which is a Detective (aka Discriminator) and a Counterfeiter (aka Generator). The detective trains to easily be able to distinguish real from fake, while the counterfeiter tries to create Fake images to fool detective. This back and forth allows us to train a model which is able to create fake images similar to what a real image is being expected by the Generator.



b. Conditional GAN:

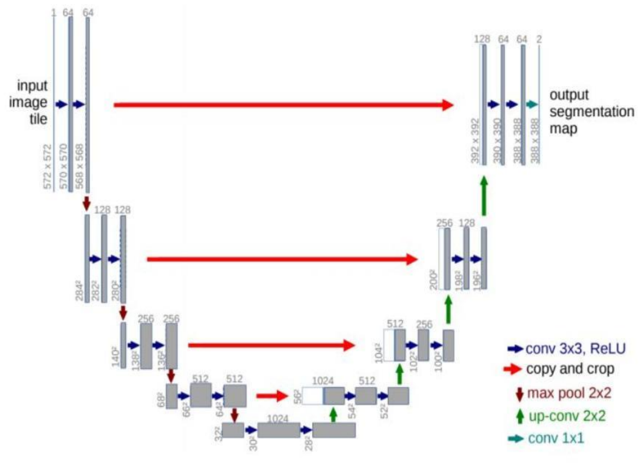
A conditional GAN can be understood as a GAN, where we not only input a image, but also some label associated with it. This makes the Discriminator robust and the Generator controlled. So now, we can generate image while passing in a label, this will allow us to know what output should be generated according to us.



This Photo by Unknown Author is licensed under CC BY-SA

c. U-Net:

This a network consisting of a bunch of Encoding steps, in a downward direction, and a bunch of Decoding steps, in upward direction. There is some of the data being passes from the encoder layer to decoder layer, called skip-connections, which allows mapping output to the input. The shape of the network appears like a U, thus granting it its name.

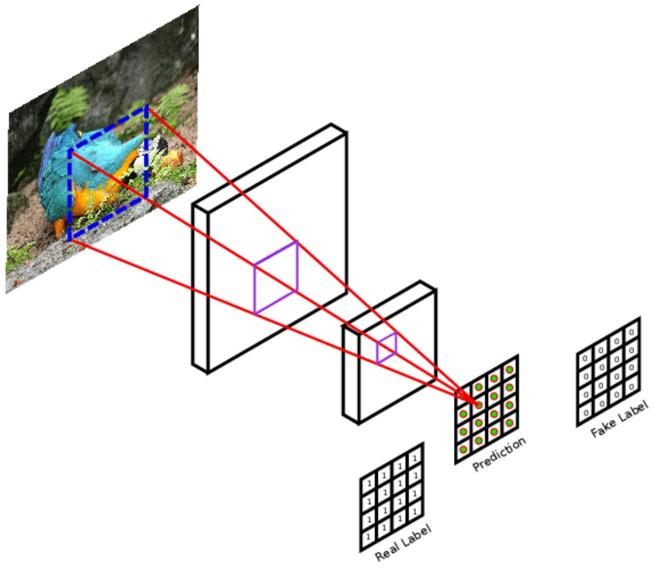


This Photo by Unknown Author is licensed under CC

d. PatchGAN:

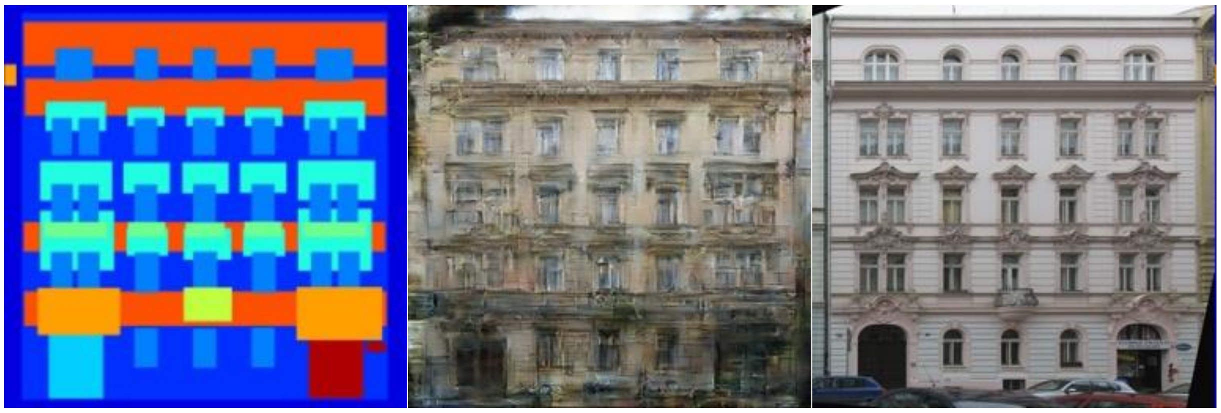
This was a new concept introduced in the paper, where the network discriminates a patch of Picture, and then moves on to the next patch. Paper suggests a patch of size 70x70 for 256x256 picture.

If increased to 256x256 patch, it'll behave like a normal CNN network.



e. Application:

The application of converting image to another image can be from a fun application like creating real images from comic image of making cats take place everywhere, to important and useful application like deblurring image, making actual image from sketches, or converting image from black and white to colored.



Input façade mask Generated by us Actual Image

We can now finally define our problem statement.

f. Problem Statement

We have a set of input images and expected output images generated from input images, which follow a particular pattern of conversion. The goal of our model is to learn this translation loss function, allowing us to generate a new set of target images from new images which might have unknown outcome. The goal is to learn in-depth about conditional GANs and Artificial Intelligence in general.

2. Data Understanding

Our data is a set of images, which consist of Input and Target Images in pairs. These pairs can either be separated into separate folders(split), or concatenated width-wise. The concatenation can either have Target image on the left, or on the right. Depending on the type of data we want to generate, Images can either be colored or black and white.

a. Initial Data Report

Data is set of paired Images, which can wither be concatenated into a single image or bifurcated into two folders. The pairs have the same name in case they're split.

b. Data Collection

Data was collected from Kaggle, where the actual dataset used in the original paper is also published. We have downloaded Façade dataset from here. Apart from this, we have also used Comic to Face dataset, which was also available in the form of split pairs. We could have also generated the images from Augmentation, generating blurred images, and using them as input to train from blurred to sharp images, which we couldn't actually implement due to time constraints. We could have also augmented images to Black and White, and done a B/W image to colored image conversion.

3. Data Preparation

Since the data is in the form of images, it needs to be preprocessed a lot make it fit for our training.

a. Selection of Data

We need to make sure that the data we select is available in pairs, and also these pairs should have some kind of pattern that can be learned by our model. It is preferred if the data is of dimensions 256x256 or higher, where the ratio should be 1:1.

b. Preprocessing the Data

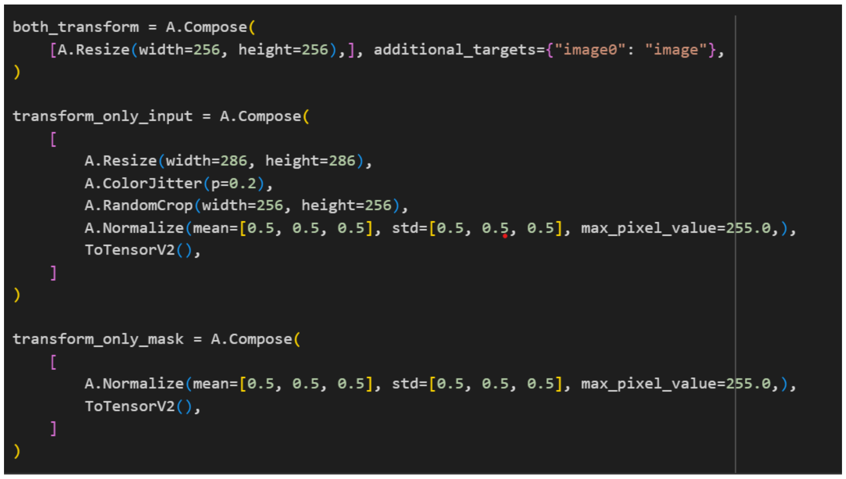
The data needs to be split if it is a concatenated pair of images, and it should be known whether the target is on the left or right. If it is separated into two folders, it still needs to be paired for further preprocessing. The data also needs to be resized to 256x256.

c. Adding Augmentations

The paper suggests that both the input and output images need to be normalized with both mean and standard deviation set to 0.5. The input image than should be resized to 286x286, followed by application of color jitter, followed by random crop back to 256x256. These allow in the removal of artifacts from the image, according to the paper.

d. Implementation

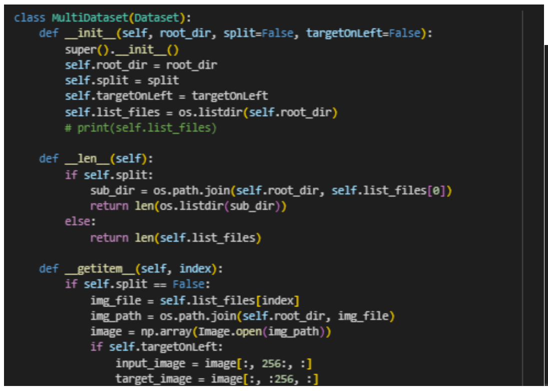
We have successfully implemented this by defining a class MultiDataset.



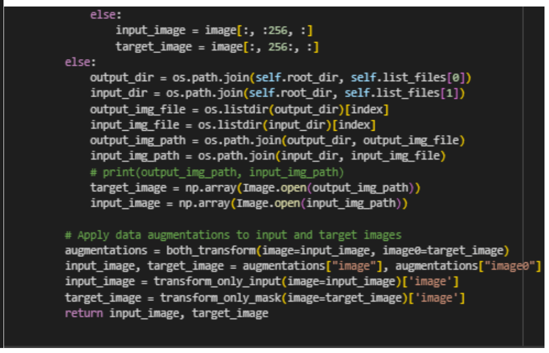
```
both_transform = A.Compose(
    [A.Resize(width=256, height=256),], additional_targets={"image0": "image"},
)

transform_only_input = A.Compose(
    [
        A.Resize(width=286, height=286),
        A.ColorJitter(p=0.2),
        A.RandomCrop(width=256, height=256),
        A.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5], max_pixel_value=255.0,),
        ToTensorV2(),
    ]
)

transform_only_mask = A.Compose(
    [
        A.Normalize(mean=[0.5, 0.5, 0.5], std=[0.5, 0.5, 0.5], max_pixel_value=255.0,),
        ToTensorV2(),
    ]
)
```

Applies augmentation

We use the Albumentations library to prepare these transforms. The MultiDataset in then defined, where it takes root directory of dataset, whether the data is split, and whether the concatenated target is on the left. It then processes the data accordingly, applying augmentations to give the final result.



```
class MultiDataset(Dataset):
    def __init__(self, root_dir, split=False, targetOnLeft=False):
        super().__init__()
        self.root_dir = root_dir
        self.split = split
        self.targetOnLeft = targetOnLeft
        self.list_files = os.listdir(self.root_dir)
        # print(self.list_files)

    def __len__(self):
        if self.split:
            sub_dir = os.path.join(self.root_dir, self.list_files[0])
            return len(os.listdir(sub_dir))
        else:
            return len(self.list_files)

    def __getitem__(self, index):
        if self.split == False:
            img_file = self.list_files[index]
            img_path = os.path.join(self.root_dir, img_file)
            image = np.array(Image.open(img_path))
            if self.targetOnLeft:
                input_image = image[:, 256:, :]
                target_image = image[:, :256, :]
```



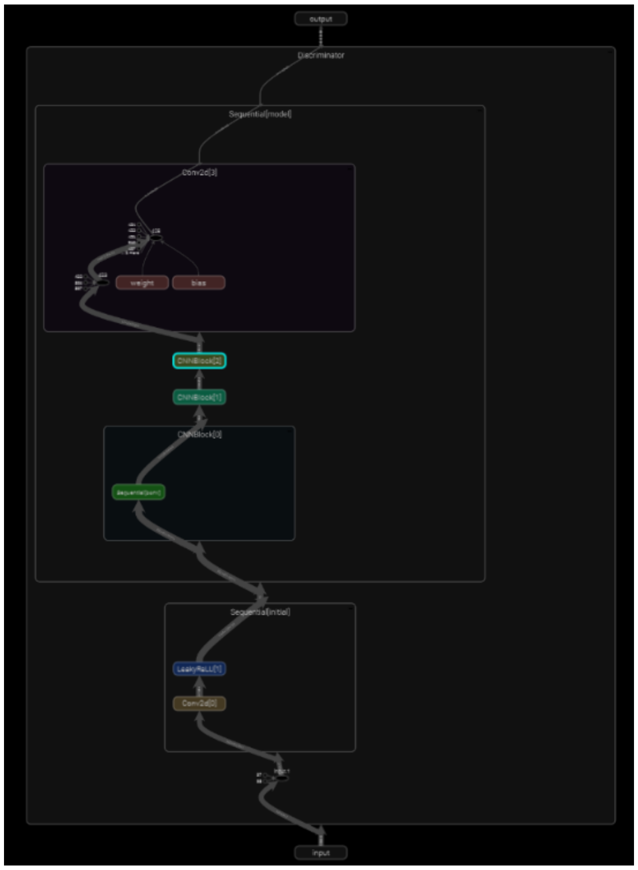
```
            else:
                input_image = image[:, :256, :]
                target_image = image[:, 256:, :]
        else:
            output_dir = os.path.join(self.root_dir, self.list_files[0])
            input_dir = os.path.join(self.root_dir, self.list_files[1])
            output_img_file = os.listdir(output_dir)[index]
            input_img_file = os.listdir(input_dir)[index]
            output_img_path = os.path.join(output_dir, output_img_file)
            input_img_path = os.path.join(input_dir, input_img_file)
            # print(output_img_path, input_img_path)
            target_image = np.array(Image.open(output_img_path))
            input_image = np.array(Image.open(input_img_path))

        # Apply data augmentations to input and target images
        augmentations = both_transform(image=input_image, image0=target_image)
        input_image, target_image = augmentations["image"], augmentations["image0"]
        input_image = transform_only_input(image=input_image)['image']
        target_image = transform_only_mask(image=target_image)['image']
        return input_image, target_image
```

4. Modelling

We will now model the Generator and Discriminator and the training environment to train our data. Both the models need their weights to be initialized with mean 0.0 and standard deviation 0.02.

a. Discriminator

As Discussed earlier, Discriminator is a PatchGAN of architecture C64-C128-C256-C512, where CX represents a convolution followed by a BatchNorm followed by ReLU. Here is the architecture.



b. *Generator*

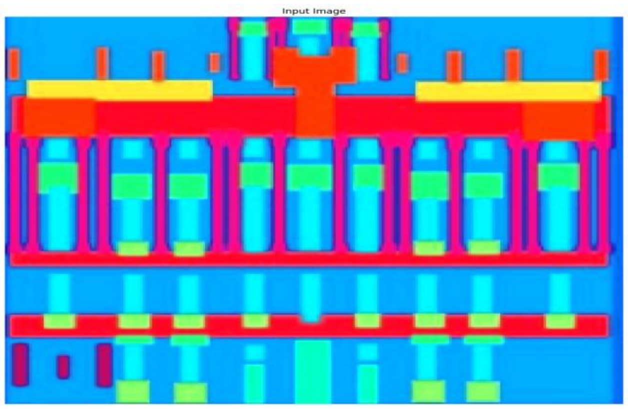
Generator is a modified U-Net with skip connections. Here is the architecture:

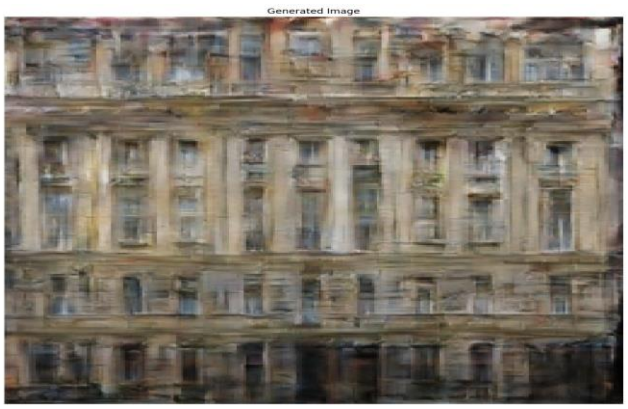
```
Generator(
  (initial_down): Sequential(
    (0): Conv2d(3, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), padding_mode=reflect)
    (1): LeakyReLU(negative_slope=0.2)
  )
  (down1): Block(
    (conv): Sequential(
      (0): Conv2d(64, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False,
padding_mode=reflect)
      (1): InstanceNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
      (2): LeakyReLU(negative_slope=0.2)
    )
    (dropout): Dropout(p=0.5, inplace=False)
  )
  (down2): Block(
    (conv): Sequential(
      (0): Conv2d(128, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False,
padding_mode=reflect)
      (1): InstanceNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
      (2): LeakyReLU(negative_slope=0.2)
    )
    (dropout): Dropout(p=0.5, inplace=False)
  )
  (down3): Block(
    (conv): Sequential(
      (0): Conv2d(256, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False,
padding_mode=reflect)
      (1): InstanceNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
      (2): LeakyReLU(negative_slope=0.2)
    )
    (dropout): Dropout(p=0.5, inplace=False)
  )
  (down4): Block(
    (conv): Sequential(
      (0): Conv2d(512, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False,
padding_mode=reflect)
      (1): InstanceNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
      (2): LeakyReLU(negative_slope=0.2)
    )
    (dropout): Dropout(p=0.5, inplace=False)
  )
  (down5): Block(
    (conv): Sequential(
      (0): Conv2d(512, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False,
padding_mode=reflect)
      (1): InstanceNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
      (2): LeakyReLU(negative_slope=0.2)
    )
    (dropout): Dropout(p=0.5, inplace=False)
  )
  (down6): Block(
    (conv): Sequential(
      (0): Conv2d(512, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False,
padding_mode=reflect)
      (1): InstanceNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
      (2): LeakyReLU(negative_slope=0.2)
    )
    (dropout): Dropout(p=0.5, inplace=False)
  )
  (bottle_neck): Sequential(
    (0): Conv2d(512, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), padding_mode=reflect)
    (1): ReLU()
  )
  (up1): Block(
    (conv): Sequential(
      (0): ConvTranspose2d(512, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
      (1): InstanceNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
      (2): ReLU()
    )
    (dropout): Dropout(p=0.5, inplace=False)
  )
  (up2): Block(
    (conv): Sequential(
      (0): ConvTranspose2d(1024, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
      (1): InstanceNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
      (2): ReLU()
    )
    (dropout): Dropout(p=0.5, inplace=False)
  )
  (up3): Block(
    (conv): Sequential(
      (0): ConvTranspose2d(1024, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
      (1): InstanceNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
      (2): ReLU()
    )
    (dropout): Dropout(p=0.5, inplace=False)
  )
  (up4): Block(
    (conv): Sequential(
      (0): ConvTranspose2d(1024, 512, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
      (1): InstanceNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
      (2): ReLU()
    )
    (dropout): Dropout(p=0.5, inplace=False)
  )
  (up5): Block(
    (conv): Sequential(
      (0): ConvTranspose2d(1024, 256, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
      (1): InstanceNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
      (2): ReLU()
    )
    (dropout): Dropout(p=0.5, inplace=False)
  )
  (up6): Block(
    (conv): Sequential(
      (0): ConvTranspose2d(512, 128, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
      (1): InstanceNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
      (2): ReLU()
    )
    (dropout): Dropout(p=0.5, inplace=False)
  )
  (up7): Block(
    (conv): Sequential(
      (0): ConvTranspose2d(256, 64, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
      (1): InstanceNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=False)
      (2): ReLU()
    )
    (dropout): Dropout(p=0.5, inplace=False)
  )
  (final_up): Sequential(
    (0): ConvTranspose2d(128, 3, kernel_size=(4, 4), stride=(2, 2), padding=(1, 1))
    (1): Tanh()
  )
)
```

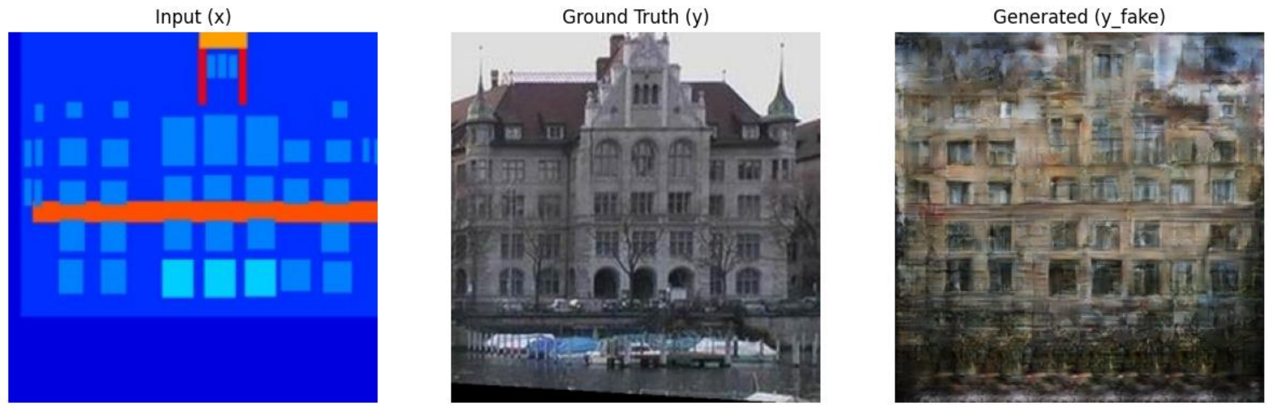
c. Utilities

We set Batch size to 1, learning rate to 2e-4, lambda to 100, and epochs to 400.

d. Results



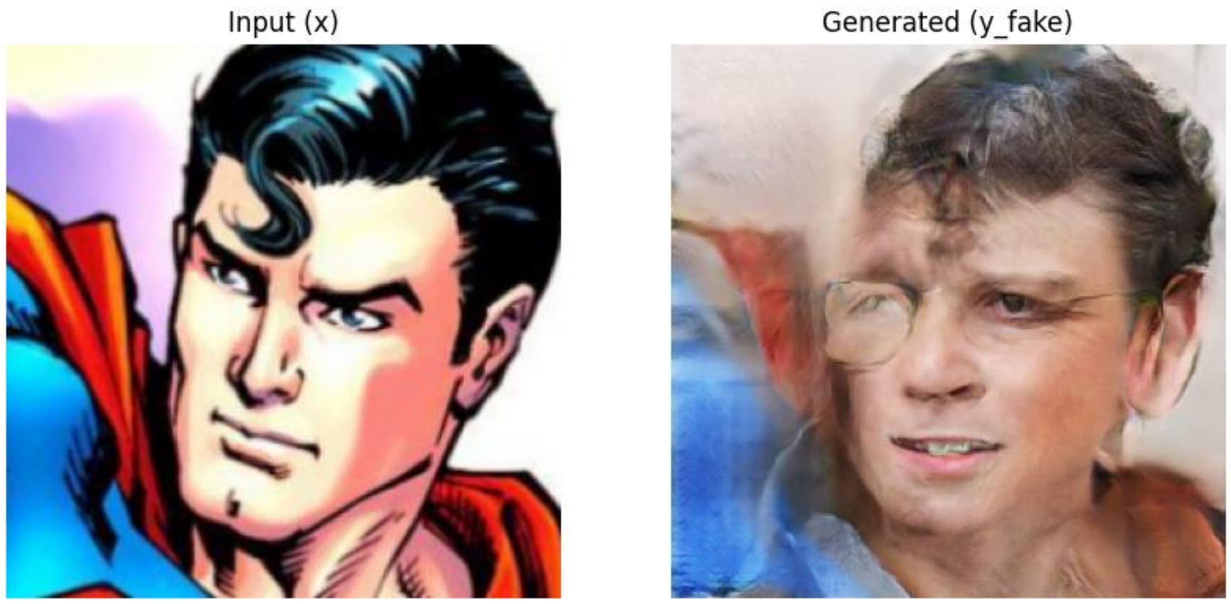


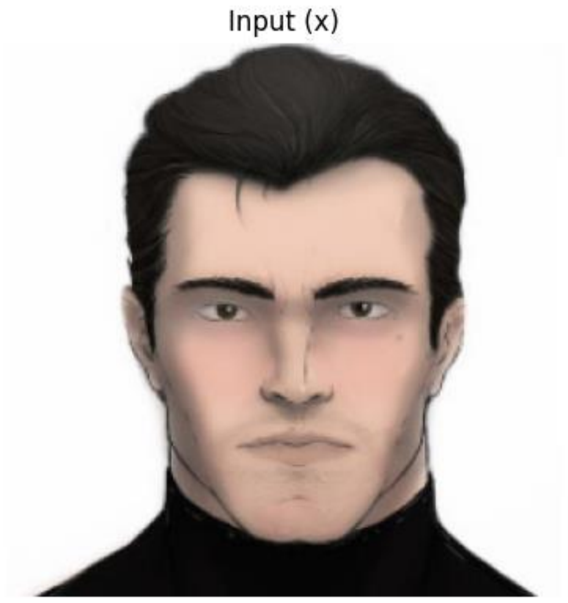


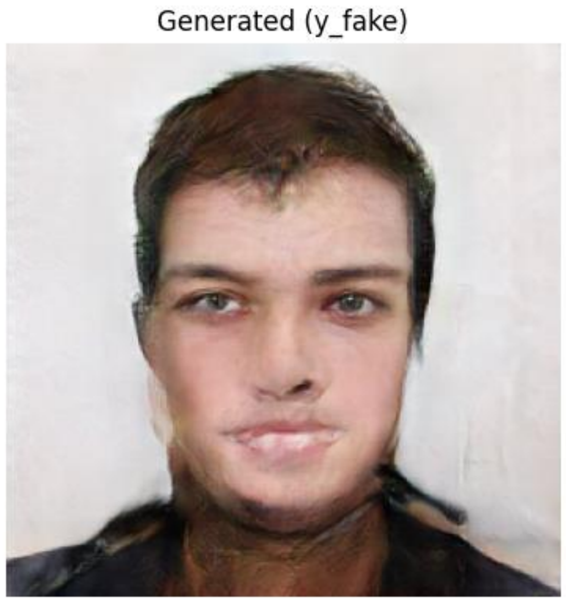
The results are subpar, but this may be due to less no. of epochs, wrong weight initialization and many other reasons. Although, we can clearly observe that the model follows closely to all the expected shapes in the input.

e. Improving the model

We will now replace BatchNorm with InstanceNorm, and increase batch size to 4, and train the comic to face dataset. The results we obtained are passable:







These images capture the essence of face, and have been a bit skewed by our input data, but still give Somewhat a human like face for superman and batman.

5. Evaluation

For evaluation, we used the same AMT evaluation method, to distinguish real from fake, and found that the model isn’t that amazing, and requires further improvement. Some cherry-picked data do show exceptional result, but in general, this is not a very successful model. Further improvement is needed.

This might be due to the fact that I’ve just been introduced to the idea of GANs a month ago. But this has obviously forged a path for me to improve towards better results.

6. Deployment

For now, in terms of deployment, we will simply provide the weights and architecture, allowing people to deploy directly. Later on, we can shift to StreamLit hosted webpage, which loads the model weights in background and generates images for people. This is omitted for now due to lack of time, but will be deployed later on.

Conclusion

GANs is now a lot popular with a lot of different possibilities in terms of generative models, and Pix2Pix plays an important role in allowing us to convert images from one format to another. Although it is a bit difficult to train GANs, and they're sensitive to changes in hyperparameter, WGAN type method can be implemented to overcome this. At the end, I would like to conclude by saying that this has been a great learning experience and I have learned a lot about training models in Artificial Intelligence, the GANs, as well as preprocessing of data, for which I express immense gratitude to our professor.

References

- P. Isola, J. -Y. Zhu, T. Zhou and A. A. Efros, "Image-to-Image Translation with Conditional Adversarial Networks," 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Honolulu, HI, USA, 2017, pp. 5967-5976, doi: 10.1109/CVPR.2017.632.
- A. Creswell, T. White, V. Dumoulin, K. Arulkumaran, B. Sengupta and A. A. Bharath, "Generative Adversarial Networks: An Overview," in IEEE Signal Processing Magazine, vol. 35, no. 1, pp. 53-65, Jan. 2018, doi: 10.1109/MSP.2017.2765202.
- M. Mirza and O. Simon, "Conditional generative adversarial nets", 2014.
- M. Mehdi and O. Simon, "Conditional Generative Adversarial Nets", no. arXiv:1411.1784v1 [cs.LG], Nov 2014.
- www.kaggle.com
- Façade Dataset: https://www.kaggle.com/datasets/vikramtiwari/pix2pix-dataset
- Comic to Face dataset: https://www.kaggle.com/datasets/defileroff/comic-faces-paired-synthetic
- Trained Weights : Link
- Github : https://github.com/Scarnhost/AI