

*Image To Image Translation Using GAN* After TimeLapse

*Anirima Ghosh*

***Dissertation submitted in partial fulfilment for the degree of Master of Science in Artificial Intelligence***

***September 2022***

# Abstract:

With the progress of astronomy and space technology, the number of satellite launched into space have grown drastically since the satellite was first launched in 1957. At present, there are more than 2000 active satellites capturing various types of images from flood plains, vege- tation, and mines to the urban landscape.

These satellite images are hugely important in managing and creating effective solutions for natural disaster management, crop yield monitoring in agriculture, infrastructure and con- struction monitoring, mining, oil pipeline monitoring and so on[1]. In the commercial space of using satellite imagery, Bird. I is one such startup which provides unique access to the latest satellite images capturing real estate and construction developments, infrastructure develop- ments, mining work and pipeline images in oil industries. These images are used by various clients of Bird. I to monitor progress in development activities and survey in real-time with- out spending a lot of resources on physical surveying.

This project is conceived by Bird. I[2] and tries to solve the following research question: Can Computer Vision and Deep Learning be used to perform an image-to-image translation of a certain landscape given a time-lapse like 3 months or 6 months.

A novel dataset is sourced from the satellite images of active constructions from Bird. I data- base and GAN-based approaches are implemented to find out a solution. Images are broken down into tiles level and Pix2Pix and CycleGAN methods are applied and performance on the test set is evaluated. These approaches clearly demonstrate a possible way for further re- search and development in future.

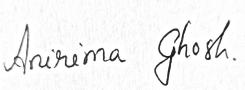
# Attestation

I understand the nature of plagiarism, and I am aware of the University!s policy on this.

I certify that this dissertation reports original work by me during my University project except for the following (adjust the list below according to the circumstances):

The data is collected by Bird. I and all the images are ©Copyright by Bird.I

## Signature: Date:23.09.2022



### Acknowledgements:

I would like to thank Dr Keiller Nogueira for the opportunity to work on this project.

I would like to thank my family for supporting me throughout my Master's Programme.

[Abstract… 2](#_TOC_250048)

[Attestation 3](#_TOC_250047)

[Acknowledgements 4](#_TOC_250046)

1. [Introduction 8](#_TOC_250045)
   1. [Motivation 8](#_TOC_250044)
   2. [Scope and Objectives 9](#_TOC_250043)
   3. [Achievements 9](#_TOC_250042)
   4. [Overview of Dissertation 10](#_TOC_250041)
2. [Background 11](#_TOC_250040)
   1. [Generative Adversarial Network 12](#_TOC_250039)
   2. Conditional GANs CGANs 13
   3. [Encoder- Decoder 15](#_TOC_250038)
   4. [Fully Connected Convolutional Network 16](#_TOC_250037)
   5. U-Net Architecture 19
      1. [U-Net with Skip Connections 20](#_TOC_250036)
      2. [PatchGAN 21](#_TOC_250035)
   6. [Pix2Pix Image Translation 22](#_TOC_250034)
      1. [Pix2Pix Generator Architecture 23](#_TOC_250033)
      2. Pix2Pix Discriminator Architecture 23
   7. [Cycle GAN - Unpaired Image to Image Translation 25](#_TOC_250032)
      1. [Cycle GAN Architecture 27](#_TOC_250031)
   8. [State Of The Art - Previous Works And MedGAN 29](#_TOC_250030)
3. [Python Libraries Used For Image To Image Translation And Hardware 32](#_TOC_250029)
   1. [Python DataLoader Class 32](#_TOC_250028)
   2. [Torchvision Transform Module 33](#_TOC_250027)
   3. [Torch.nn Module 33](#_TOC_250026)
   4. [PyTorch TorchMetrics 33](#_TOC_250025)
   5. [Environment… 33](#_TOC_250024)
4. **[5.1 Nvidia and CUDA 33](#_TOC_250023)**
5. [6 PyTorch and Hardware 34](#_TOC_250022)
6. [Dataset Preparation, PreProcessing and Challenges 34](#_TOC_250021)
   1. [The Image Source 36](#_TOC_250020)
   2. [The Image Preprocessing… 38](#_TOC_250019)
   3. [Image Augmentation 38](#_TOC_250018)
   4. [Challenges of The Data Set Preparation 39](#_TOC_250017)
      1. [Inadequate Number of Images 40](#_TOC_250016)
      2. [Location Contamination 40](#_TOC_250015)
7. [Experiment Set Up & Evaluation Metrics 42](#_TOC_250014)
   1. [Dataset Split… 42](#_TOC_250013)
   2. [Methodology… 42](#_TOC_250012)

**[5. 2.1 Pix2Pix… 42](#_TOC_250011)**

[5. 2. 2 CycleGAN 52](#_TOC_250010)

[5. 3 Experiment… 52](#_TOC_250009)

1. [4 Evaluation Metric 54](#_TOC_250008)

[5. 4 .1 Clean FID 56](#_TOC_250007)

[5. 4. 2 SSIM 56](#_TOC_250006)

5. 34 3 PSNR 58

1. [Results 59](#_TOC_250005)
   1. [Results - Comparison Of Different Models Using FID 62](#_TOC_250004)
   2. [Evaluation 64](#_TOC_250003)
2. [Conclusion 65](#_TOC_250002)
   1. [Summary… 66](#_TOC_250001)
   2. Limitations 66
   3. Future work 66
3. [Bibliography 68](#_TOC_250000)

## Introduction

## Motivation

Satellite Imagery is a very valuable dataset for various domains like Agriculture, Natural Dis- aster Management, construction, Oil & Gas Sector. The accurate, real time images help to monitor and understand the real time changes occurring day to day or month to month ba- sis[1] on earth like Disaster Management. Satellite images have been successfully used to di- agnose Thunderstorms Tournedos[3]. Images available with Bird. I actually cover in- frastructure, oil and gas monitoring, construction and mining. The images help monitor the real-time progress of construction in housing[2] or the images can provide a key benefit to the Infrastructure Sector. Large areas of land captured in one image give the user up-to-date views of lands ranging across several kilometres[2]. The images capture roads, bridges, malls, railway tracks and other active construction sites on a periodical basis frequently by satellites.

Though these images are immensely helpful for progress monitoring and other applications, currently there is no application available which can predict the progress and changes of lands throughout the lifetime of construction for infrastructure or oil and gas monitoring or any other sector. An active development site goes through many changes like paving the road, foundations being laid, houses or bridges or factories or oil pipelines being constructed etc.

So an Image captured in a particular location will show some changes in the constructions compared to a previous image captured in the same location.

The goal of this project is to find out a possible approach to generate images similar to the changed landscape given input images of the same location with a time-lapse basically trans- lating to predict an image from a given image after a certain time lapse. Image to Image translation techniques available within the GAN family is chosen specifically as GAN has been successfully used to generate images for generating maps[5] or rivers or different land- scapes from satellite images[4].

## Scope and Objectives

The aim of this project is to propose GAN-based models to perform Image to Image Trans- lation on Satellite Images.

The main tasks are listed below:

* Review the literature on GAN-based Models.
* Learn the PyTorch framework and Error Metric in order to be able to build and train GAN models.
* Data Preprocessing, extracting tile levels data from satellite images sourced from Bird. I.
* Propose two GAN-based models pix2pix and cycleGAN for the Image to Image Translation task.
* Build models, hyper tuning of models to increase the quality of the images and make performance comparisons between them.
* Report the findings and evaluation of the GAN-based approach for this project.

## Achievements

This project is unique and complex in terms of the project requirements and dataset. So far there are no published works available to solve similar problems. The use of satellite images using computer vision for commercial building architecture is itself a new and novel concept that very few startups are working on. Keeping that in mind, the feasibility of the project is proved to the point that it opens up interesting possible future work. This project achieves preparing a dataset that itself was challenging and scarce, to begin with. Also, this project is able to compare the different image-to-image translation approaches and compare the results which open up possible opportunities to explore.

# Overview of Dissertation

**Chapter 1 -** Introduction of the dissertation, motivation, background of the topic, scope and objectives of the project and achievements.

**Chapter 2:** State of the Art deep learning GAN techniques are introduced in this chapter.

**Chapter 3:** The libraries of PyTorch used in the project are discussed in detail.

**Chapter 4:** The dataset preparation, preprocessing and challenges are discussed here.

**Chapter 5:** The experiment set-up and methodologies and evaluation metrics are explained in this chapter.

**Chapter 6:** This chapter discusses the results and comparison of different models in this project.

**Chapter 7:** The summary, conclusion and future work are discussed in this chapter.

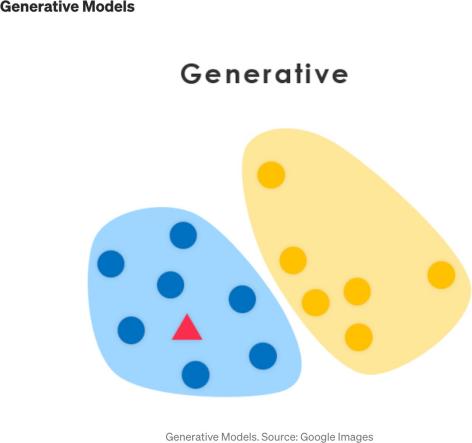
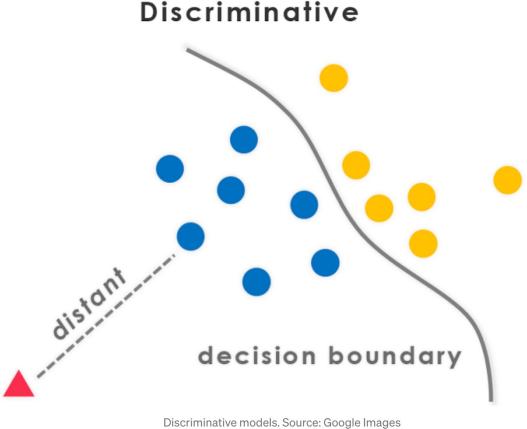
**Chapter 8:** Bibliography

## Background

### Generative Adversarial Network

Image to Image Translation technique belongs to a class of computer vision problems where the goal is to learn the mapping between the input image and the output image using a pair of images[6]. Image to Image Translation has taken off in the last few years with the advent of GAN; GAN is a special kind of deep learning framework first introduced in 2014[7]. Ian

J. Goodfellow et al[7] in their paper “Generative Adversarial Nets” or GAN proposed a new architecture of a generative model through the process of adversarial networks. A generative model is distinctive from a discriminating model. Discriminating models come under the cat- egory of Statistical Classification where the model separates the classes of data and learns the boundaries between different classes of dataset. But Generative models belong to a different class where they can generate new data points and are based on Bayes Theorem[8].



**Figure1: discriminative model. Figure2: a generative model**

Image Source: Google Images

The GAN framework is very unique in the world of deep learning. In the original paper[7] Ian J. Goodfellow et al proposed a framework which consisted of two architectures. A Gener- ator (G) and a Discriminator (D). It is assumed that the reader is familiar with Multilayer Perception and gradient descent. Both the Generator and the Discriminator are Multilayer Perceptron-based models that were given an input image a Generator tries to generate a new image similar to the output image and the Discriminator tries to classify the generated images as “real” or “fake”. The generator’s job is to generate images as realistic as possible and fool the discriminator to classify the image as “real”.

Ian.J.Goodfellow has given an analogous comparison in his paper as “the generative model tries to create a counterfeit note and the discriminator tries to detect counterfeit note as po- lice”[7].

Here the Generator is based on generative modelling and random noise is passed through the multilayer perceptron and the discriminator is nothing but a classifier. The errors in the outer layer are passed through backpropagation and dropout as same in the neural network. Figure 3 below describes the architecture of a GAN. Generator G is fed with random noise and gen- erates fake samples. The discriminator D classify the image as “real” or “fake”. If the image is classified as “fake”, the errors are passed through backpropagation and the generator learns to generate a better image.

**Figure3: GAN**

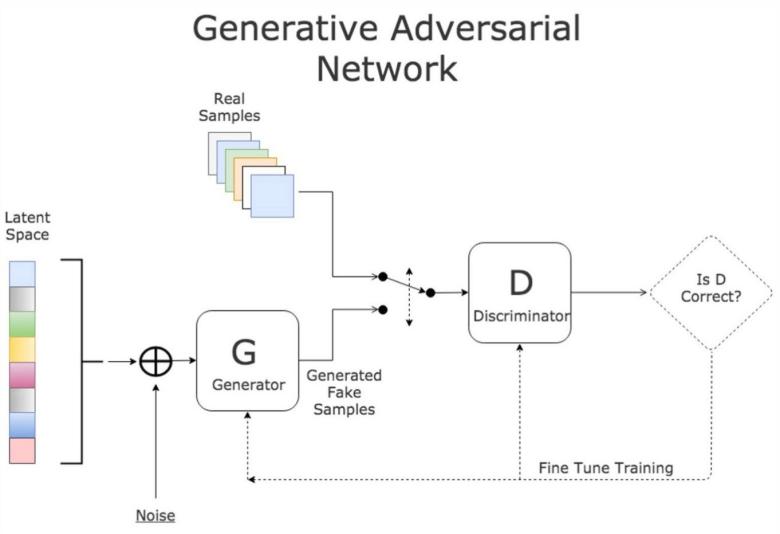


Image Source: https://medium.com/ai-society/gans-from-scratch-1-a-deep-introduction-with-code-in-pytorch-and- tensorflow-cb03cdcdba0f

The loss function of GAN is referred as the “Min-Max Game”[7] and is as follows:

*min G max D V (D, G) = Ex*"*pdata(x) [log D(x)] + Ez*"*pz(z) [log(1 − D(G(z)))]* [7] Where x represents a real image, D(x) is the output of the discriminator.

The first part of the equation (*Ex*"*pdata(x) [log D(x)] )* wants to maximise to identify real im- ages.

*min V (G) = Ez*"*pz(z) [log(1 − D(G(z)))]* this part minimises V so that it can fool the discrimi- nator[7].

The training of both the generator and discriminators is also very novel. First, the generator’s parameters are defined, and then the discriminator is trained using one iteration of gradient descent.

In the next step, keeping the discriminator fixed the generator is trained for one iteration.

Both the models are trained in alternate steps until the generator starts generating images which are classified by the discriminator as “real”. As per the paper, the discriminator is updated by its ascending stochastic gradient descent value:

#*θd 1 m Xm i=1 h log D x (i) + log 1 − D G z (i) i* [7]

And the generator is updated by its descending stochastic gradient descent value:

#*θg 1 m Xm i=1 log 1 − D G z (i)* [7]

A few of the generated images are referenced below:

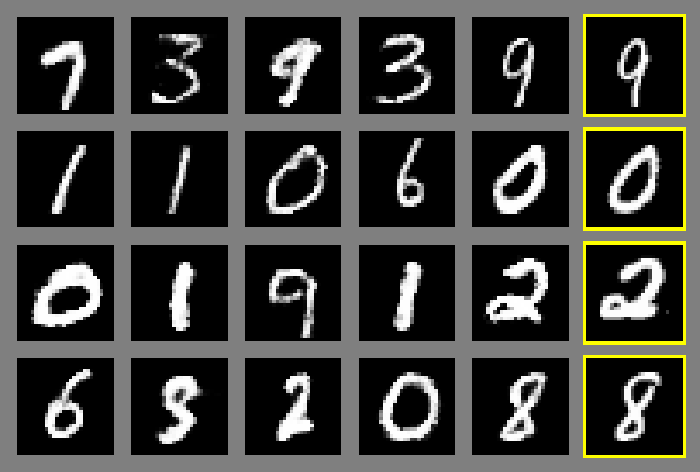


Figure4: MNIST (Rightmost column shows the nearest training example of the neighbouring sample[7])

### Conditional GANs or CGANs

A GAN framework is good at generating “fake” images which look “real” but no control can be applied to the modes of the generated image but to translate one image to another, GANs are trained on the conditional setting. cGAN(as it is commonly referred to) uses class labels or data from other modes to train both the generator and discriminator. Consequently, cGAN learns multiple modes of mapping from input images to output images.

As the paper proposes in 2018[9], cGANS learn a mapping between an input image (x), ran- dom noise vector(z) and output image(y):

*G:{x,z} ——> y* [9]

Whereas the mapping between the input and output image for a GAN is:

13

*G:{z} ——> y* [9]

The learning objective of a conditional GAN is as follows:

*LcGAN (G, D) =Ex,y[log D(x, y)]+ Ex,z[log(1 − D(x, G(x, z))]*[9]

Where G tries to minimise the objective and D tries to maximise it following the same principles of adversarial nets. There are two crucial traits of cGAN which make them dif- ferent from GANs are: unlike GAN where random noise fed into the generator is a gaussian noise, cGAN provides noise by applying dropout to the several layers of the gen- erator at both training and testing[9].

Also to minimise the learning objective, L1 distance is considered rather than L2 as L1 distance encourages the model to generate output images that are less blurry[9].

The final learning objective as proposed in the paper as follows:

*G* $ *= arg min G max D LcGAN (G, D) + λLL1(G) [9]*

*Where LcGAN (G, D) is Ex,y[log D(x, y)]+ Ex,z[log(1 − D(x, G(x, z))] [9].*

Phillip Isola, 2018 et al describe in the paper the architecture of the Generator(G) as the U-Net[9]. U-Net is an upgraded version of Encoder-Decoder with stop connections.

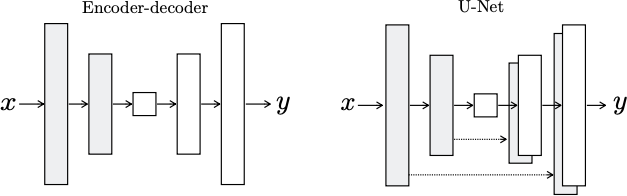


Figure5: generator architecture[9]

### Encoder- Decoder

Encoder-decoder is a different version of a feedforward neural network where there is no dif- ference between an input image and output image but encoder-decoder compresses the input images into lower dimensions also called “Latent - Space Representation”[10]. When En- coder-decoder reproduces the output image, it basically reconstructs the image from latent space. In that sense, they are also unsupervised machine learning techniques because there are no x and corresponding y’s in the encoder-decoder. They can just be fed with an input image and an output image will be produced. An encoder-Decoder framework is also “data- specific”[10] which means this framework is trained on specific features of the image and compresses the image so the output image will be always the same as the input image. hand- written digits are fed into the encoder-decoder, it will never output cat images. This frame- work consists of two separate architectures Encoder and decoder connected by code. Both the encoder and decoders are Fully connected Artificial Neural Network(ANN). The number of layers and nodes inside the encoder-decoder can vary depending on the task. The number of nodes in the encoder decreases with subsequent layers and in the decoder part, the num- ber of nodes increases with layers as displayed below[10].

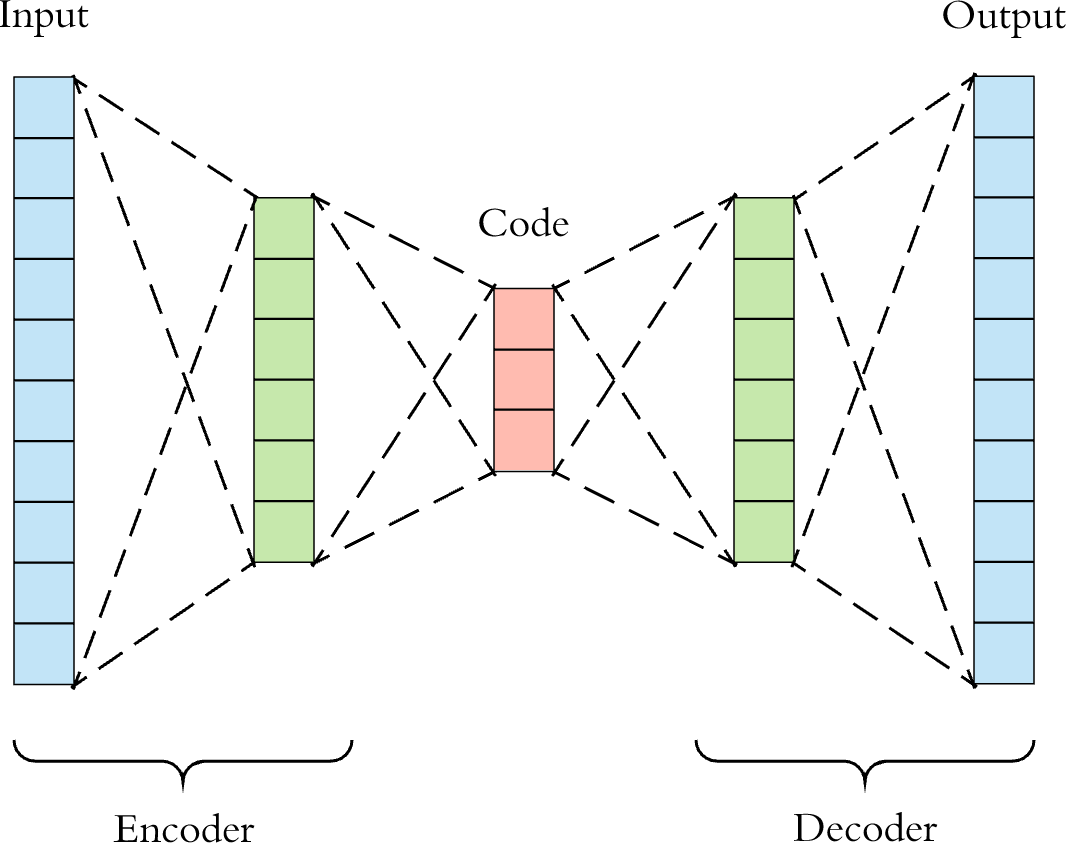


Figure6: encoder-decoder (image taken from[10])

### What is Fully connected ANN

Artificial Neural Network are the backbone of deep learning consisting of an input layer, multiple hidden layers and an output layer. An input layer consists of the input features (X). The hidden layer consists of nodes. Each node has two parts: one, a weighted sum of the nodes of the previous layer and an activation function. The activation function in the hidden layers is non-linear and useful for finding the hidden information of input features as dis- played in the image below. The input layer consists of 3 nodes which represent the input fea- tures(X). Each of the two hidden layers consists of four nodes. The output layer consists of one representing the output(Y).

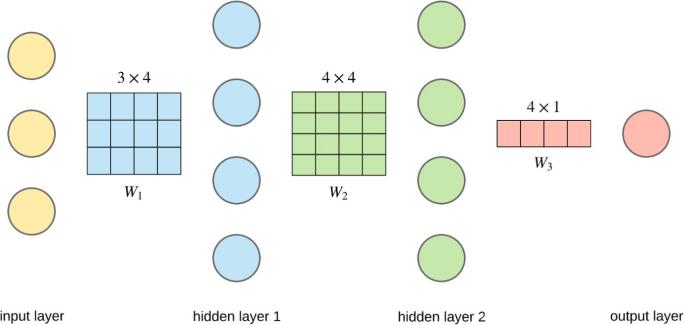


Figure7: Artificial Neural Network taken from https://towardsdatascience.com/applied-deep-learning-part-1-artifi- cial-neural-networks-d7834f67a4f6

Each node in the hidden layers (a1-hidden layer 1 and a2- hidden layer 2) and output layer(y) has the following activation functions:

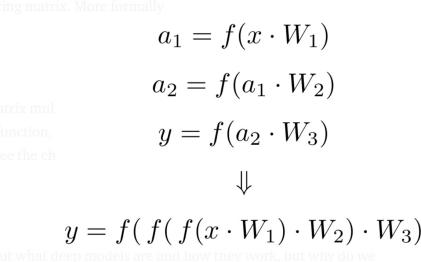


Figure8: activation function(image taken from (https://towardsdatascience.com/applied-deep-learning-part-1- artificial-neural-networks-d7834f67a4f6)

The ANN becomes fully connected when each node in the current layer is connected with each node in the previous layer, the network becomes a fully connected ANN model as dis- played in the image below: Here the input and the hidden layers are fully connected.

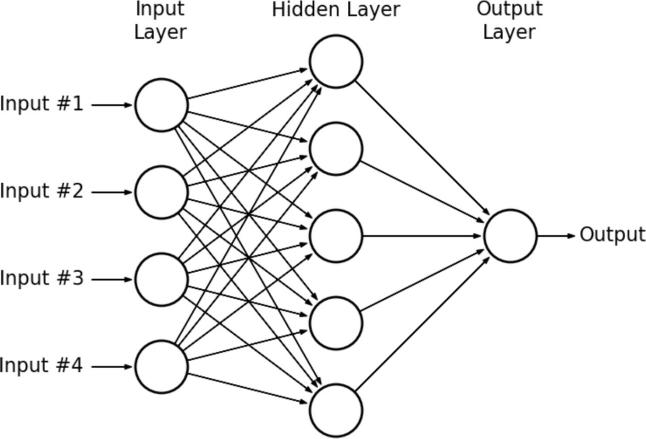


Figure9:Fully Connected ANN (image source: https://[www.kdnuggets.com/2019/07/convolutional-neural-networks-](http://www.kdnuggets.com/2019/07/convolutional-neural-networks-) python-tutorial-tensorflow-keras.html)

### Fully Connected Convolutional Network

Convolutional Network or CNN is the backbone of any image processing task today and this architecture is also used in image translation. CNN is basically a network Input layer, Convo- lution layer, Pooling Layer, and Fully connected layer.

The input layer is where the image is fed into the architecture. The convolution layer consists of convolution filters to extract the feature of the input image which is followed by the pool- ing layer which helps reduce the dimensions of the features extracted and is usually followed by a fully connected layer(FCN). FCN as explained in 2.1.1 connects the features extracted from the previous layer to the output layer and classifies the image.

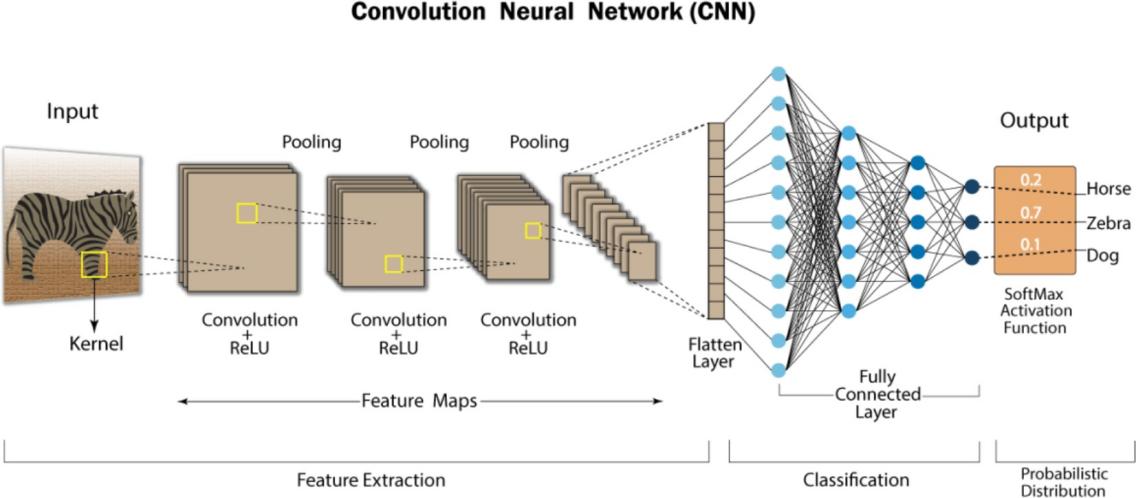


Figure10: CNN architecture(image taken from https://developersbreach.com/convolution-neural- network-deep-learning)

* 1. **U-Net Architecture**

U-Net is an upgradation of a Fully Connected Convolutional Network. It was developed by Olaf Ronneberger et al[11] for biomedical segmentation in 2015. The paper proposes a framework for deep convolution that relies on data augmentation strategies rather than a high volume of data[11].

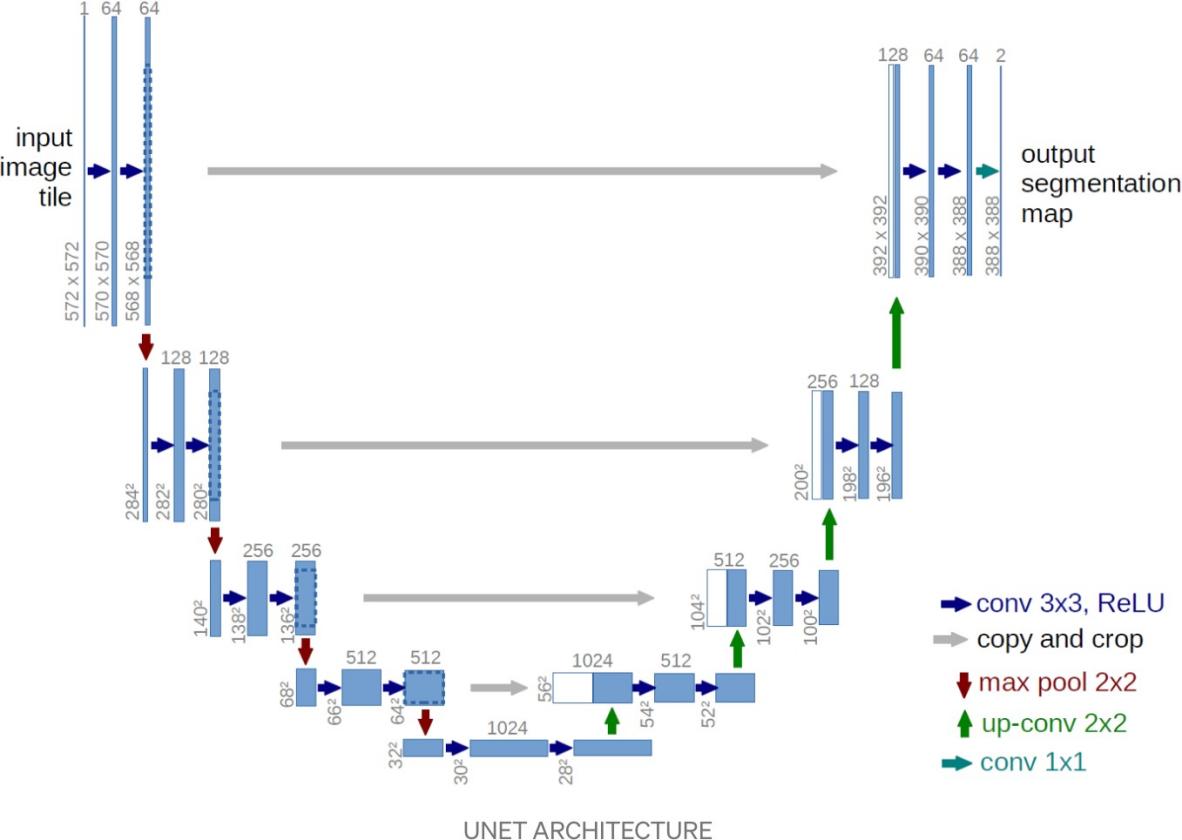


Figure11: U-Net Architecture (image source: https://medium.com/analytics-vidhya/what-is- unet-157314c87634)

The U-net as displayed in the image above has two parts. A contracting path on the left side and an expanding path on the right side[11]. As the image shows, the contracting path of U- net has a repeated 3x3 convolution layer followed by ReLU and 2x2 max pool with stride 2 used in downsampling with a number of feature channels doubling at each step[11]. The ex- pansive path consists of an upsampling of the features by 2x2 convolution which halves the number of feature channels as shown in figure 9[11]. U-net can help increase the resolution of the output images and can give more precise localised information in the output image.

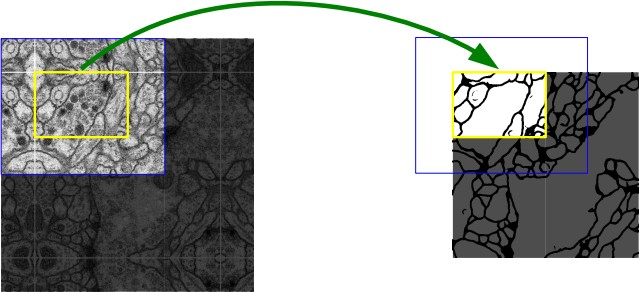


Figure 12: example of seamless segmentation of large images using U-Net[11]

### U-Net with Skip Connections

Skip connections in the U-net architecture mean missing or skipping a few of the neural net layers and feeding the output to the subsequent layers. The generator of image-to-image translation uses U-net with skip connections as proposed by Phillip Isola, 2018 et al[9]. For many previous studies, the input features are passed through the downsampled layers in the encoder-decoder until the bottleneck layer. For Image to Image translation problems, there is a lot of low-level information that can be directly transferred circumventing the bottleneck using the skip connection following the shape of U-Net[9].

### PatchGAN

PatchGAN is a special type of convolutional neural net only where the image patch is pro- cessed independently and identically. PatchGAN is usually used in the discriminator part of the GAN. The difference between patchGAN and a regular discriminator is the output of the discriminator. The output of a regular from 256x256 image is a single scalar output which is classified as “real” or “fake” but out of patch GAN from 256x256 image is an NxN array of the output x where each element in signifies of the patch is “real” or” fake”.

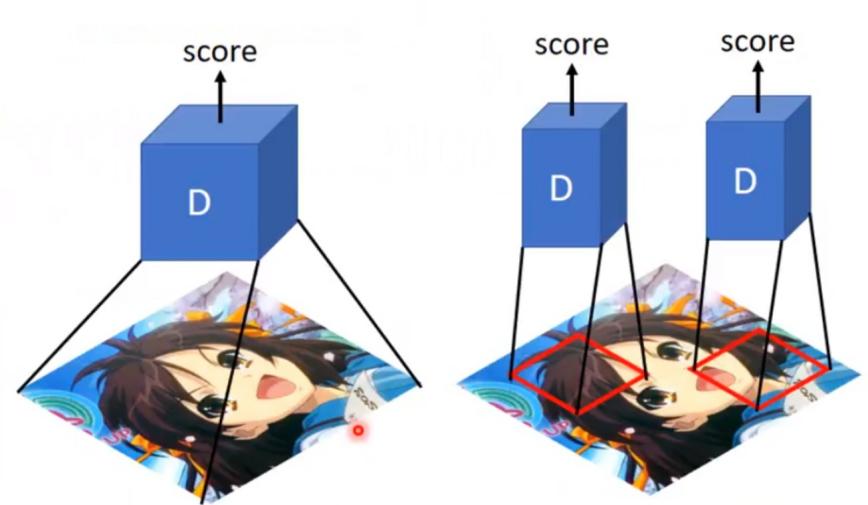


Figure 13: How PatchGAN works (image taken from https://blog.paperspace.com/unpaired-image-to-image-trans- lation-with-cyclegan/)

### Pix2Pix Image Translation



Figure 14: application of pix2pix (image taken from [9])

As shown in figure 13, pix2pix architecture is used to generate maps from satellite im- ages and satellite images from maps[9].

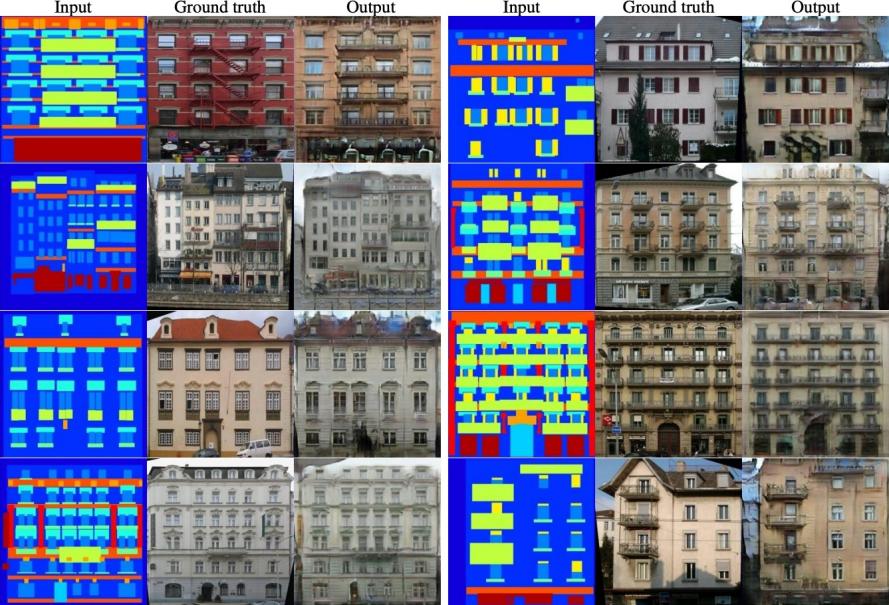


Figure 15:facades labels —-> photos (image taken from [9])

Another significant application of pix2pix is creating facades from labels[9]. Here The ground truth image is paired with the mapping image of the input image. The output image is generated from the input where the ground truth image is the expected image.

Another noteworthy application of pix2pix has been generating images from sketches as shown in figure 15 below[9].



Figure 16:automatically detected edges to shoes (image taken from [9])

### Pix2Pix Generator Architecture

The generator architecture consists of an encoder-decoder as discussed in the earlier chapter. Encoder : C64-C128-C256-C512-C512-C512-C512-C512 [9]

Decoder: CD512-CD512-CD512-C512-C256-C128-C64 [9]

The U-net architecture has skip connections between each layer(i) in the encoder with the previous layer(n-i) in the decoder where the total number of layers are n [9].

U-net decoder: CD512-CD1024-CD1024-C1024-C1024-C512 -C256-C128 [9].

* + 1. **Pix2Pix Discriminator Architecture** The 70x70 discriminator :C64-C128-C256-C512 [9]. The 1x1 discriminator : C64-C128 [9].

The 16x16 discriminator : C64-C128 [9].

The 286x286 discriminator : C64-C128-C256-C512-C512-C512 [9].

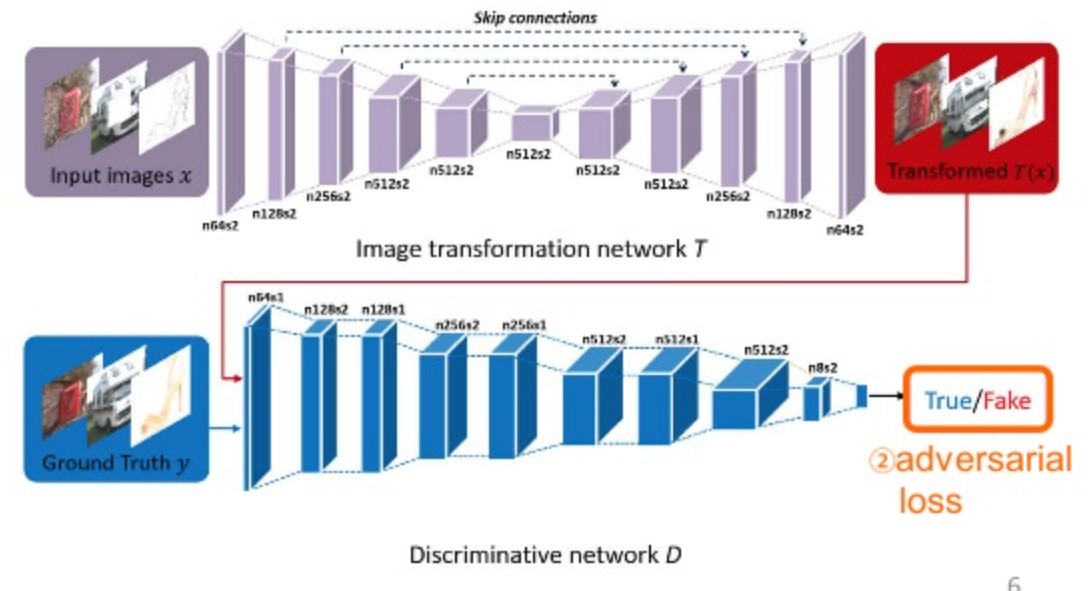
blog.paperspace.com/unpaired-image-to-image-translation-with-cyclegan/)

Figure 17:pix2pix architecture (https://

As discussed above, figure 16 shows the architecture of the generator and discriminator. The input images are fed through the downsampling layers and reduce the dimensions of the in- put image and again moved through the upsampling layers to increase the dimensions. The discriminator uses patchGAN to assign a patch of generated images as “real” or “fake” and then classifies the generated images.

### Cycle GAN - Unpaired Image to Image Translation

cycleGAN is a different way of translating images where there are no paired images and one style of image is generated from another. The technique introduced in the paper by Jun-Yan Zhu, 2020 et al[12] has no paired training input images. The model captures specific charac- teristics of an image set and translates that into another[12].

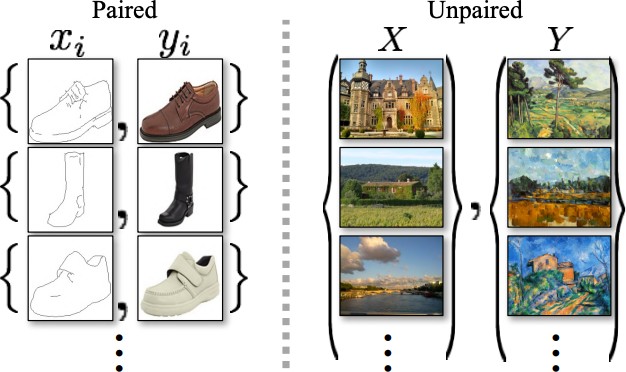


Figure 18: paired (left) and unpaired (right) image translation(image source [12])

Cycle GAN trains images in a cycle-consistent way. As the paper describes, there are two generators, G which maps images from domain X to Y and F which maps images from Y to X[12]. The adversarial losses generate images that match the mapping from Y and X respec- tively but are not visually similar to each other. For example, cycle GAN is trained from a domain containing summer images(X) to a domain containing winter images(Y). The adver- sarial losses during training produce images which match the feature distribution of Y and can be any combination of the feature distributions of Y and would look different than im- ages from X[12]. The authors in this paper have described a “cycle-consistency loss” and a cycle-constrained mapping to augment the training process[12]. Each image from domain X, translating to an image from domain Y and being back to domain X should bring back the original image. that is x % G(x) % y % F(y) ≈ x which can be written as x%G(x)%F(G(x)) ≈ x.

This is named as “forward cycle consistency”, Similarly, for each image y from domains Y,

G and F the following equation is derived y % F(y) % G(F(y)) ≈ y which is named as “back- ward cycle consistency” [12].

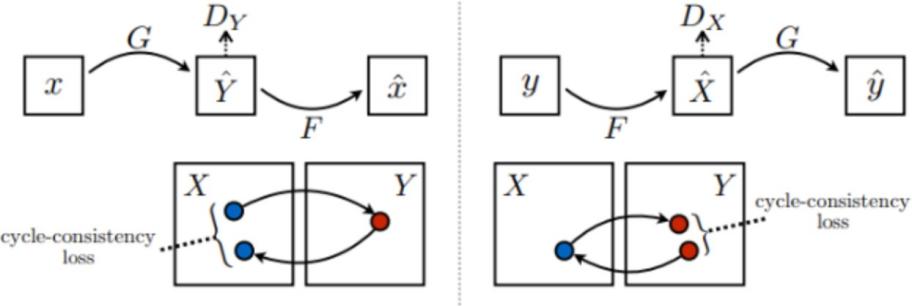


Figure 19: forward and backward cycle consistency[12])

The paper also introduces a different adversarial loss than in pix2pix. For the mapping func- tion G: X ——> Y the objective is as follows:

*LGAN(G, DY , X, Y ) = Ey*"*pdata(y) [log DY (y)] + Ex*"*pdata(x) [log(1 − DY (G(x))]* [12]

Where G generates images that look similar to images from domain Y; G tries to minimise the objective as D tries to maximise it.

The cycle consistency loss objective is:

*Lcyc(G, F) = Ex*"*pdata(x) [kF(G(x)) − xk1] + Ey*"*pdata(y) [kG(F(y)) − yk1] [12]*

The full loss objective combining together is as follows:

L(G, F, DX, DY ) =LGAN(G, DY , X, Y ) + LGAN(F, DX, Y, X) + λLcyc(G, F)[12].

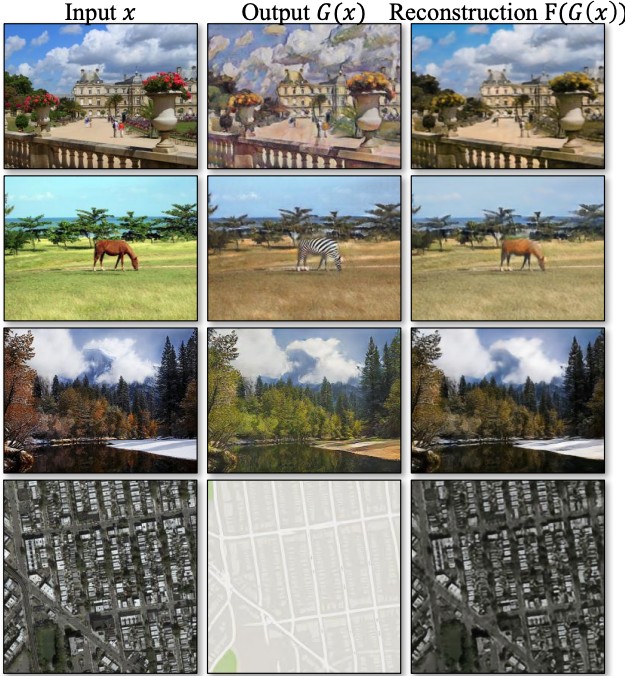


Figure20: examples of output images by cycle GAN[12])

### Cycle GAN Architecture

There are 6 residual blocks for 128 × 128 training images, and 9 residual blocks for 256 ×

256 images.

The network with 6 residual blocks consists of:

c7s1-64,d128,d256,R256,R256,R256, R256,R256,R256,u128,u64,c7s1-3 [12].

The network with 9 residual blocks consists of:

c7s1-64,d128, d256, R256, R256,R256, R256, R256, R256, R256, R256, R256, u128 u64,c7s1-3 [12].

The discriminator architecture consists of 70 × 70 PatchGAN [12]. The discriminator archi- tecture is: C64-C128-C256-C51[12].

The cycleGAN has been used to generate Maps to Aerial Images, Summer to Winter Images, Horses to Zebra images, facades label to building images etc [12].

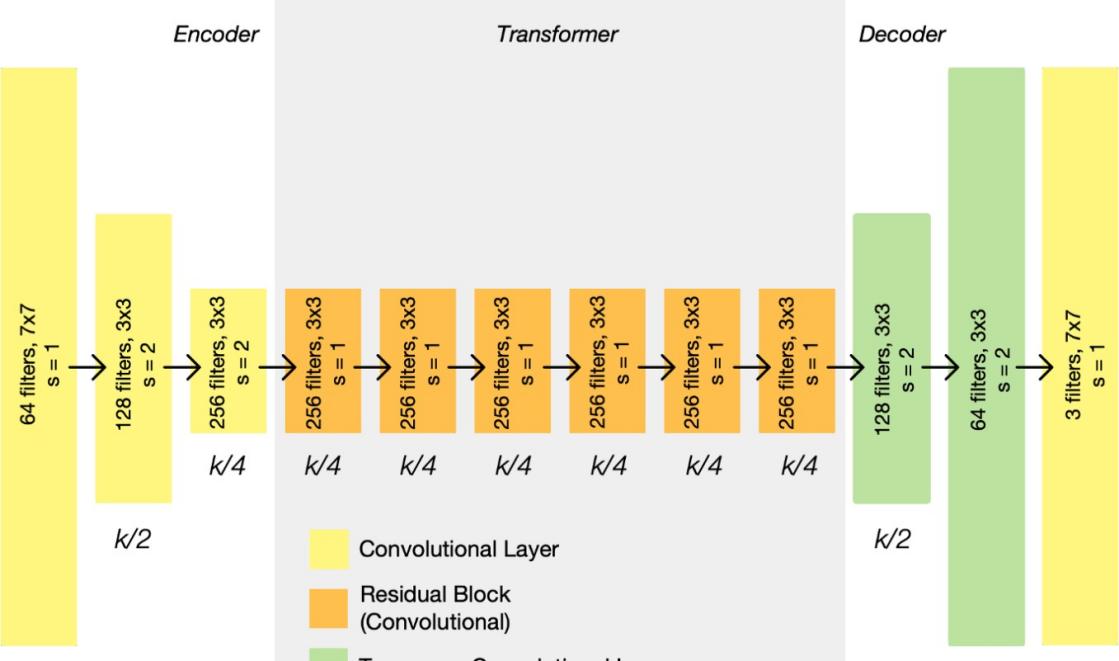


Figure21: Architecture of cycle GAN(image taken from https://towardsdatascience.com/cyclegan-learning-to-translate-images-without- paired-training-data-5b4e93862c8d)

## State Of The Art - Previous Works And MedGAN

Since the introduction of GAN in 2014[7], there have been many studies on different ap- plications of image-to-image translation techniques. Much of the noteworthy work has been done in medical imaging. Usually, medical imaging means magnetic resonance imaging (MRI), positron emission tomography (PET) Scan, and Computed Tomography or CT scan. Translating images is important in the medical sector as often images need to enhance, in resolution or feature-wise. There have been many different ways to do that like pseudo-CT images are generated from MRI scan K-nearest neighbours[15]. There is one work to en- hance the resolution of OF MRI scan and is based on a sparse representation framework with multi-scale edge analysis and a dimensionality reduction technique[16]. Recently with the advances in CNN, there have been many works on lesson detection and classification[17], semantic segmentation [18], and image enhancement[19].

There has been one more work of MRI image reconstruction using generator architecture

[20] but it was more MRI scan application specific.

But these previous works have been very application specific and one architecture can not be applied to another problem. To tackle this a more general architecture was introduced in 2019[14]. Which Proposes a new architecture called “MedGAN”[14]. MedGAN is a new generator architecture with the adversarial framework of non-adversarial losses[14]. A new architecture named “CasNet” was introduced which chained together with several fully con- ventional encoder-decoder network with skip connections[14]. CasNet is unique because it is not specific to any applications. Contemporary to this work, a similar work named as “stacked U-net” is developed for natural image segmentation[21].

29

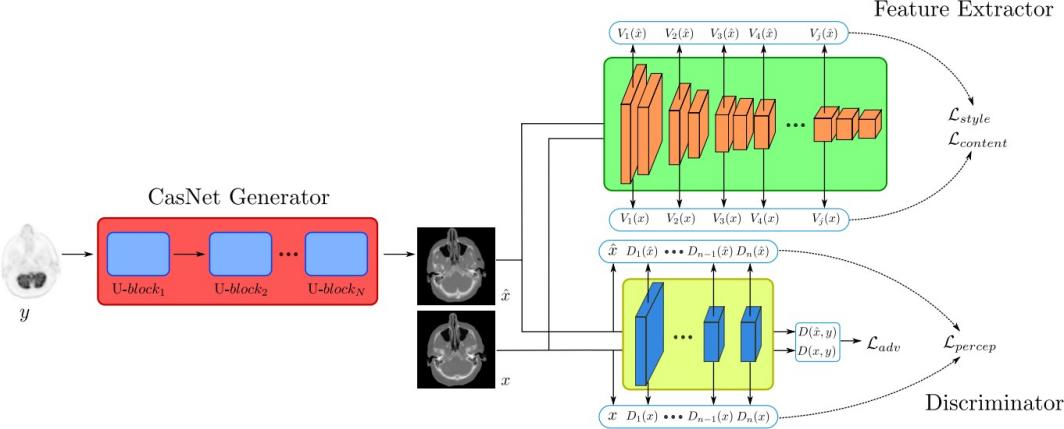


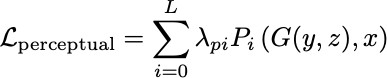
Figure22: the architecture of med-GAN[14])

Figure 19 shows the architecture of a med-GAN which consists of a generator which is a block of U-nets, a pre-trained feature extractor and a discriminator. The feature extractor is used to extract deep rich features[14] to calculate style transfer so that the generated image is similar to the style and content of the Ground Truth image. The paper introduces a new “Perceptual Loss”[14]. Often generated images suffer from pixel losses which can not capture the ability of human perception images often appear to human eyes as blurry images. To capture the pixel losses MAE between the feature of the target and the translated image is calculated.

 *[14]*

Here, D denotes the features from the discriminator network and h,w and d represent the height, width and depth of the feature spaces respectively[14].

The perceptual loss is calculated as follows[14]:

[14]

Another type of loss which is taken into consideration is style transfer losses. These losses transfer the style of an input image to the generated image, preserving the texture and details of an input image in the process. That is why the pre-trained feature extractor is used to cap- ture the features from the hidden layers of CNN[14].

MedGAN generator architecture consists of CasNets which is a collection of U-net blocks as shown in figure 29. The output of the first U-block is passed to the input of the second U- block till the end of the architecture[14]. The number of convolutional filters is 64,128,256,512,512,512 and 512. The paper shows that the medGAN outperformed the previous approaches in translating three different kinds of images. MedGAN is able to pro- duce more homogenous and sharper images which appear clearer visually to the human eyes[14].

The works presented so far involves a different kind of architectures in generator and losses showing a possible approach for the problem at hand. GAN has been successfully used in the last couple of years in many cases where it has produced images which have never existed before with newly added features in the generated images. This is similar to the problem this work is based upon. For the problem at hand GAN should generate an image similar to the ground truth where input and ground truth have the underlying same structure but different features.

# Python Libraries Used For Image To Image Translation And Hardware

One of the most widely used libraries in python used for deep learning is PyTorch. PyTorch is object-oriented based programming which has many specific libraries and associated class- es used extensively for Image Classification, Segmentation, Image To Image Translation, etc. A few of the important libraries are discussed below.

### Python DataLoader Class

torch.utils.data.DataLoader is the core of the data loading process in PyTorch[13].

DataLoader(dataset, batch\_size=1, shuffle=False, sampler=None, batch\_sampler=None, num\_workers=0, collate\_fn=None, pin\_memory=False, drop\_last=False, timeout=0, worker\_init\_fn=None, \*, prefetch\_factor=2, persistent\_workers=False)

The data loader class loads the data into the PyTorch framework and transforms it into ten- sors. There are many hyperparameters, important ones needed for this project are described below.

batch\_size: how many training samples per batch to load. The default value is 1. num\_workers: how many subprocesses are to be used for data loading.

Shuffle: if data needs to be reshuffled at every epoch.

### Torchvision Transform Module

Torchvision. transforms is one of the most important modules as the module helps transform an image into a tensor image. A tensor image is a tensor with shapes C, H, and W where C is the number of channels, H is the height of the image and W is the width of the image. A batch of the tensor image is of shapes B, C, H, and W where B is the number of images in a batch. The different image transformation techniques can be strung together with composing function. Below are the functions which are important for preprocessing images.

transforms.Resize() - resize the input image as per the requirements. transforms.CenterCrop()- crops an image at the centre. transforms.RandomVerticalFlip()- vertically flips an image randomly with a probability.

transforms.ToTensor()- transforms an image to a tensor.

transforms.Normalize()-normalizes a tensor with the mean and standard deviation.

### Torch.nn Module

torch.nn.The module is a base class for all neural network-related modules in PyTorch.

torch.nn.Conv2d(in\_channels,

out\_channels, kernel\_size, stride=1, padding=0, dilation=1, groups=1, bias=True, padding

\_mode=‘zeros', device=None, dtype=None)

This class applies a 2d convolution in the network. in\_channels and out\_channels correspond to the shape of an image. kernel\_seize determines the size of the filter. The device should be “Coda” if the network is being trained on GPU.

### PyTorch TorchMetrics

PyTorch TorchMetrics is a collection of Error Metric implementations. Error metrics will be discussed in detail in chapter 6.

### Environment

Convolutional Neural Network requires a huge amount of computation which would be time-consuming if performed on a multi-core processing unit (CPU). For this kind of com- puting heavy tasks, a graphical processing unit or GPU is really good as it can perform paral- lel computations. Neural Networks are also run parallelly with thousands of small computa- tions run simultaneously. That is why it is advisable to train CNN on GPU.

### 5.1 Nvidia and CUDA

Nvidia designs CUDA which is software based on their GPU hardware and literally cuts the computation time in halves. All Model training is developed and performed on GPU and im- age preprocessing, and most of the error metric evaluation is done on CPU.

### 6 PyTorch and Hardware

The PyTorch framework gives the users an open-source ML platform that specifically works well with deep learning and computer vision-related tasks. A beneficial trait of PyTorch is that it has CUDA as an underlying software This enables training models without the user knowing much about the hardware and only knowing python programming and being famil- iar with PyTorch Library would suffice. While writing this dissertation the author assumes the reader to be well versed in python and PyTorch to understand the nitty-gritty of the project.

## Dataset Preparation, PreProcessing and Challenges

The goal of the project as stated in Chapter 1 is to generate a satellite image which would show the transition of building development from the previous image after a certain time lapse (for this project time-lapse is 3 months). This requires a pair of images where one image is the “Input Image” and the other image is the “Expected image” or “Ground Truth”. The Expected Image should show some transition of developments in terms of building, road construction, laying and paving the road etc. when compared to the Input Image. Both the images have a time difference of 3 months.



Figure23: Input Image(left) & Ground Truth(right)(©Copyright byBird.I)

Figure 21 & 22 show the input image on the left and the expected image or ground truth on the right. Both images show a difference in the construction in roads and buildings within a time difference of 3 months.



Figure24: Input Image(left) & Ground Truth(right)(©Copyright byBird.I)

### The Image Source

As stated in chapter 1 BirdI collects satellite images spanning across several countries like USA and Australia. The images are collected on a periodical basis and fed into their data- base. For this project, the images are downloaded from the AWS S3 bucket. AWS provides Amazon Simple Storage Service or AWS S3, a storage service for easy, scalable uploading and downloading of images. The uploading and downloading of data becomes more conve- nient because Python has a library “Boot3” which directly calls AWS API and makes the in- teractions easy.

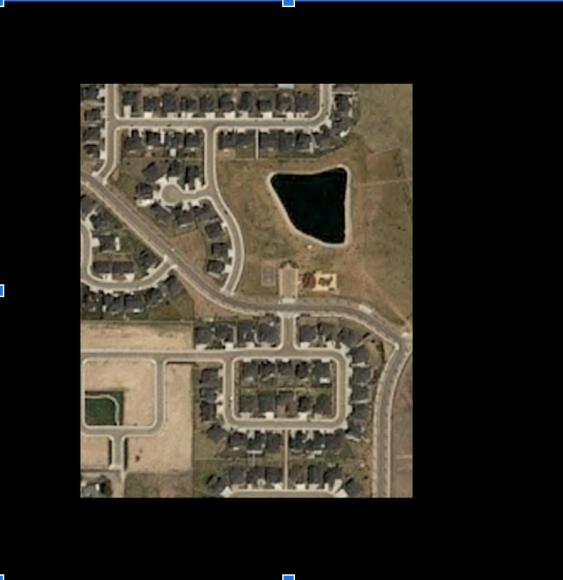
 

Figure25: Input Tile(left) & Ground Truth Tile(right)(©Copyright byBird.I)

### The Image Preprocessing

The Images downloaded from the AWS S3 bucket are of sizes 1792x1792 and they some- times contain background black space which masks the entire images. Each Image of 1792x1792 is called a tile and contains a block of 7 to 9 small images.

First Images are downloaded from the AWS S3 bucket as stated in 4.1 with the help of image keys provided by BirdI in JSON format. The Images are paired mapping where both the in- put and ground truth images correspond to the same location. Locations are embedded with the image as metadata as latitude and longitude. The latitudes and longitudes are masked and not shared here due to data protection and confidentiality issues. The paired mapping of the images is important for this project as Image to Image Translation will take the Input Tile and generate an image similar to the image on the right side (figure 23). A function used to download the images from AWS S3 using python boto3.

For the requirement of the task, the image tiles (1792x1792) are broken down into smaller tiles(256x256) and the background black masks are removed. For this task 2 key functions are used which are displayed below.

A function is written to create a matrix of tiles coordinates given tile size and image input resolution. Tile size is 256 and image input resolution is 1792 (image input resolution corre- sponds to the entire image block as shown in figure 23).

The second most important script for data preprocessing is the tile\_matrix\_from\_image func- tion which creates a matrix of 0s and 1s where 1 means the corresponding image is white padded and 0 means the corresponding tile is black padded. For example, a 7x7 tile matrix means there are 7x7 images of which the 0s are black padded and can be discarded while processing and 1s are white padded and kept during the processing of the images.

The next preprocessing step involves extracting smaller tiles of size 256x256 from the 1792x1792 images.

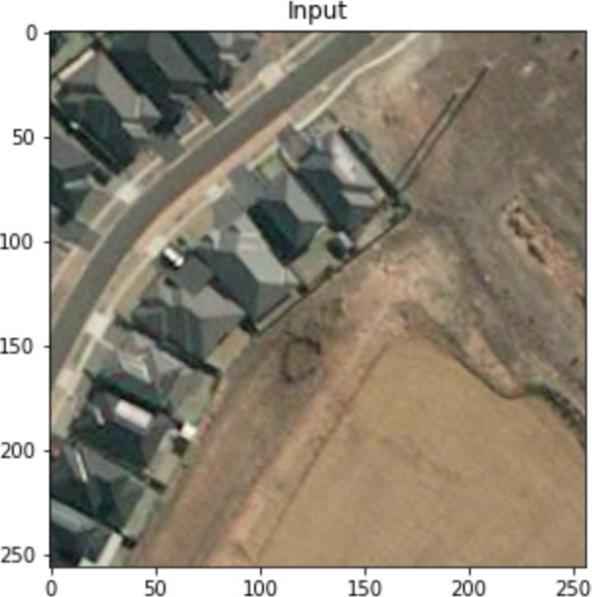


Figure26: a processed 256x256 image (©Copyright byBird.I)

The step which further complicates the process is that each image of size 1792x1792 gener- ates 20-40 images of size 256x256. That means each input image generates 20-40 images and the corresponding ground truth image also generates 20-40 images and while creating the dataset, each of 256x256 tile from the input image should correspond to the similar ground truth tile of 256x256 from the ground truth image.

If the location of the extracted tiles gets mixed up, the mapping of the input and ground truth image will get mixed up. A simple way to tackle this problem is to name the images as per the indexes of the list where the image names were stored in while extracting and saving the images.

For this project many times the input and ground truth images have to be concatenated and deconcatenated and both the images are named as per the indexes and it becomes easy to track the corresponding images. One example of the input image and ground truth image is given below:



Figure27: the example of a concatenated image (©Copyright byBird.I)

Figure 27 shows a concatenated image where the left part is the input image and the right part is the ground truth image and the shape of the image is 256x512. This kind of concate- nated images can be easily cropped to input and ground truth images of size 256x256 if re- quired.

### Image Augmentation

The images in the dataset were augmented and transformed using the PyTorch transform module before the training process. The images were cropped and resized to 256x512 (input image and ground truth image are concatenated and fed into the model. RandomVerticalFlip was used in the script to randomly flip the images as suggested in the paper[9]. Additionally, the images were normalised with a mean of 0.5, and variance of 0.5 and transformed to Py- Torch Tensors.

### Challenges of The Data Set Preparation

Dataset preparation is one of the most challenging aspects of this project. The steps involved manually inspecting the images, and preprocessing images from 192x1792 sizes to 256x256. Few of the challenges are mentioned below.

### Inadequate Number of Images

One of the major challenges of this project includes gathering enough number of images re- quired for deep learning tasks. The pair of input and ground truth images are required to have some transition as discussed above. But to collect images which show some transitions in terms of development has been difficult as most of the images were downloaded from bird.I repository showed no transitions.

The first set of images which are collected and preprocessed contained 3744 images but upon inspection most of the images were found to not have any transition. There were only 30 im- ages from the entire image set which showed some transitions.



Figure28: the input(left) and ground truth(right) shows no transition (©Copyright byBird.I)

Figure 28 shows an example of a concatenated image which shows no transition or difference between the input image and the ground truth image. The inadequate dataset can not be really solved by upsampling as duplicating the same images will have an adverse impact on the training.

39

Another set of data has been downloaded from which 38 images of 1792x1792 are selected as they show some transition. Each Image yields 10-20 images when pre-processed and 1000 tiles of 256x256 are derived. Now, these 1000 images have to be split into train, test and val- idation datasets in 0.8, 0.1,0.1 ratio so which creates around 800 images for training. Another thing to be noted here is that not all the tiles of 256x256 show transitions simply because not every part of the 1792x1792 image has transitioned in terms of building or land develop- ment. As an example, the image in figure 20 only shows some building development at the

left-hand bottom corner part of the image so when preprocessed not all 256x256 input and ground truth pair images will have any transition, only the tiles corresponding to that position will show the transition from input to ground truth image. This really brings down the num- ber of actual pairs of images which show some difference between input and ground truth images and would certainly impact the training as the minimum number of images required for pix2pix satellite to maps is 2000 as suggested in the paper[9].

In fact, pix2pix satellite-to-map image translation relatively simple maps as ground truth have less complexity compared to the satellite images used in this project. The images in this project are complex and have lots of different features and layouts leading to high variance of the training set which might require more than 2000 images.

### Location Contamination

The important part of preprocessing step is when to break down the image (1792x1792) to tile level. If this is done after splitting the images in training, testing and validation, that leads to the overfitting of the model and generating unusually good images for the same locations where some of the training images came from. This problem will be discussed in detail in chapter 5.

To prevent this location contamination, the images are split into train, test and validation even before the preprocessing based on their location data.

A function is scripted for extracting the latitude and longitudes of the images which are em- bedded in their metadata and splitting the images into train, test and validation sets before preprocessing.



Figure29: the split of images into the train, test and val based on geo locations

Figure 29 shows the visualisations of the locations of the images from the train, test and vali- dation folders making sure that all the images from the train, test and validation set are geo- graphically well separated.

Another issue to be noted here for Both the datasets is that the number of actual transitions from the input image to the ground truth image is actually low where around 70-80% of im- ages of both the training sets show no difference in building or architecture.

Hence, there are two datasets used in training, the difference between the two sets is geo- graphical and the breaking down of the tiles at the preprocessing step as discussed above.

|  |  |  |
| --- | --- | --- |
|  | Data set 1 | Data set 2 |
| Train | 1800 | 800 |
| Test | 100 | 50 |
| Validation | 100 | 50 |

## Experiment Set Up & Evaluation Metrics:

#### Dataset Split

As discussed in the earlier chapter there are two sets of data models are trained. They will be referred to as ‘dataset one’ and ‘dataset two’ in this chapter. Both the datasets are split into train, test and validation sets using the small script below:

Dataset one consists of 1800 training images, 100 images each for both validation and test set. Dataset two consists of 800 training images with 100 images for each test and validation set.

#### Methodology

As discussed in the previous chapter, the pix2pix and cycle GAN-based architectures are suit- able for image-to-image translation tasks. The pix2pix model is developed using the genera- tor and discriminator architecture based on the paper[9]. Another pix2pix and cycleGAN model was trained using the existing pertained model[24].

#### 2.1 Pix2Pix

As described earlier, the pix2pix model consists of both a generator and a discriminator[9].

. The structure of 1 encoder block of U-net is as follows:

* 1. LeakyReLU (the previous layer)
  2. Conv2D
  3. BatchNorm

### LeakyReLU

LeakyReLU is a non-linear activation function which is used more often than ReLU. ReLU or Rectified Linear Unit is nothing but an activation function which brings some non-lineari- ty to otherwise linear function in a neural network. ReLU is defined as y = max(0,x) which simply means all negative values are considered as 0 in the function.

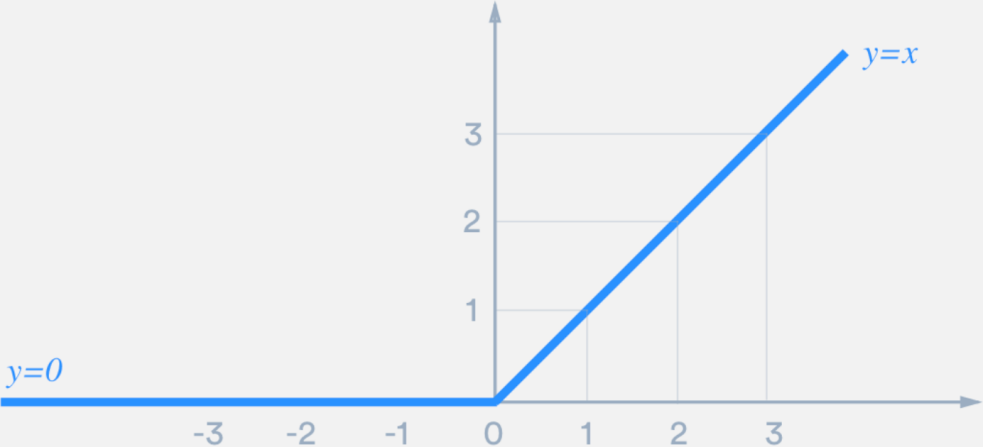


Figure30: ReLU(image taken from [https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7)](https://medium.com/%40danqing/a-practical-guide-to-relu-b83ca804f1f7))

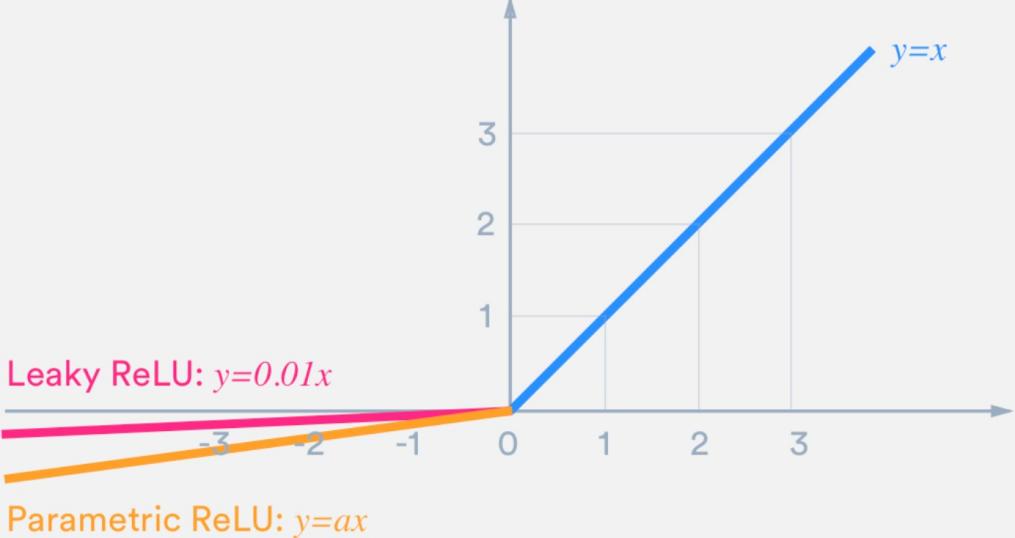
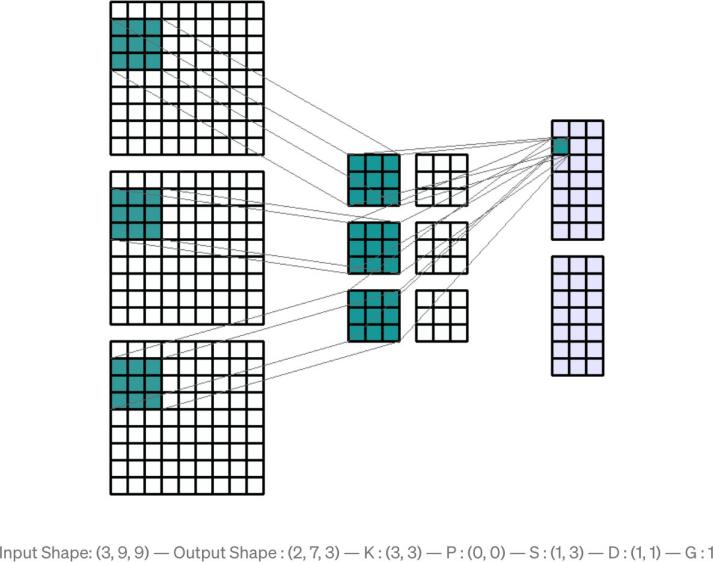


Figure31: Leaky ReLU(image taken from [https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7)](https://medium.com/%40danqing/a-practical-guide-to-relu-b83ca804f1f7))

Sometimes, while learning ReLU can be less effective because it can turn all the negative val- ues into zero and that can interfere with the learning process of a neural net if the learning rate is too high and the network gets stuck in a negative zone and the output is always zero.

To mitigate this problem, in computer vision often the leaky ReLU is used. Leaky ReLU as- signs a small value for all negative numbers instead of zero. Leaky ReLU is not always effec- tive but for image-to-image translation, Leaky ReLU is the preferred choice in the papers dis- cussed in previous chapters. Leaky ReLU is used here to build the pix2ix model in PyTorch.

Figure32: Convolution Process(image taken from https://towardsdatascience.- com/conv2d-to-finally-understand-what- happens-in-the-forward-pass-1bbaaf- b0b148)

### Conv2D

Conv2d is the module of PyTorch which works as a convolution layer in an encoder or de- coder block. Conv2d has many hyper parameters. The most important of them are as fol- lows:

in\_channels - the number of channels in an input Image. out\_channels - the number of channels produced by the convolution. kernel\_size - the size of convolving kernel or filter

Stride - the stride of the convolution and the default value is 1. Padding - zero padding added to both sides of the input.

A kernel or convolution matrix scans through an input image and creates a convolution product. The output image is basically a very compressed image. Usually, there are 3x3, 5x5, and 7x7 convolution filters. The odd numbers of kernels are the usual choice in any comput- er vision task including image-to-image translation because using an odd number of channels creates a symmetrical shape around the central pixel but for this architecture 4x4 size is cho- sen.

Padding is basically wrapping the matrix with 0s or 1s to create a layer around the input im- age so that while convolving the information on the edges of the input images does not get lost. Padding would be used or not varies from task to task.

Stride is the number of steps a convolution filter takes to scan from left to right and top to bottom across the input image. The default value in Conv2D is 1 and it can be changed ac- cordingly depending upon the requirement.

For the generator architecture, kernel size is 4x4, stride 2 and padding 1.

### Batchnorm:

Batch Normalization or BatchNorm as it is referred to in PyTorch makes the training of the neural network of any computer vision task more stable and also speeds up the learning process. It is basically a computing process which normalizes the vectors in the hidden layer using the mean and variance of the current batch[25]. It is usually implemented either before or after the Convolution layer or in this case Conv2D in PyTorch.

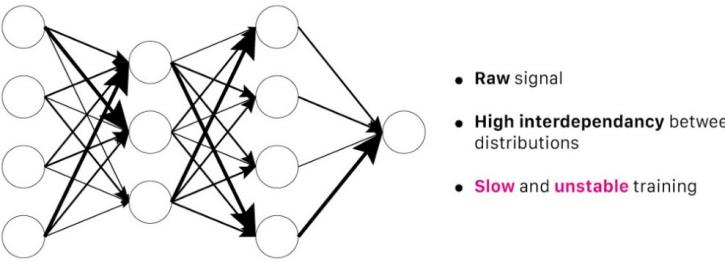


Figure33: A Neural Net With Batch-norm(image taken from https://towardsdatascience.com/batch-normalization- in-3-levels-of-understanding-14c2da90a338)

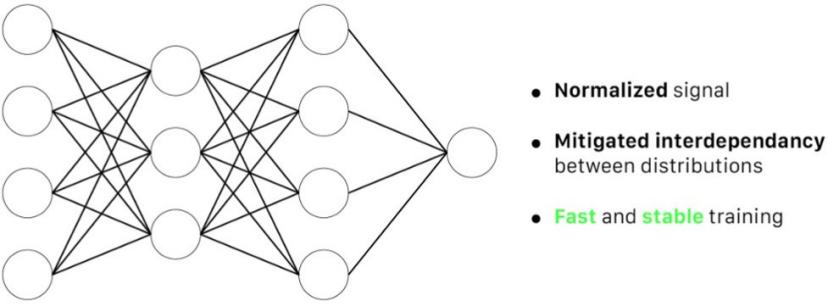


Figure34: A Neural Net With Batch-norm(image taken from https://towardsdatascience.com/batch-normalization- in-3-levels-of-understanding-14c2da90a338)

The figure 34 shows the training of a multilayer perceptron without batch normalization.

The signals are erratic and unstable as shown in the figure. But figure 32 shows the training of neural net with batch normalization which makes the training stable and fast[25]. As in- troduced by Sergey Ioffe, and Christian Szegedy et al in 2015[25] batch norm reduces the “Internal Covariant Shift” which is defined as “the change in the network parameters during training”[25].

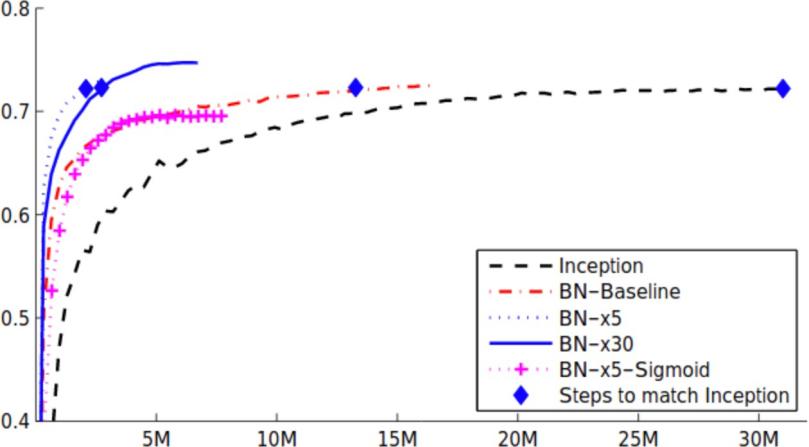


Figure35: Batch Normalization effect on ImageNet training( image taken from [25])

The figure 35 shows the effect of batch normalization on ImageNet Training[25]. Batch- Norm leads to better and faster convergence and higher accuracy.

The generator architecture of the pix2pix model developed consists of the following blocks of encoders and decoders:

Generator(

(encoder1): Conv2d(3, 64, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False) (encoder2): Sequential(

* + 1. : LeakyReLU(negative\_slope=0.2, inplace=True)
    2. : Conv2d(64, 128, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
    3. : BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

)

(encoder3): Sequential(

1. : LeakyReLU(negative\_slope=0.2, inplace=True)
2. : Conv2d(128, 256, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
3. : BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

)

(encoder4): Sequential(

1. : LeakyReLU(negative\_slope=0.2, inplace=True)
2. : Conv2d(256, 512, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
3. : BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

)

(encoder5): Sequential(

1. : LeakyReLU(negative\_slope=0.2, inplace=True)
2. : Conv2d(512, 512, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
3. : BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

)

(encoder6): Sequential(

1. : LeakyReLU(negative\_slope=0.2, inplace=True)
2. : Conv2d(512, 512, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
3. : BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

)

(encoder7): Sequential(

1. : LeakyReLU(negative\_slope=0.2, inplace=True)
2. : Conv2d(512, 512, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)

)

(decoder1): Sequential( (0): ReLU(inplace=True)

1. : ConvTranspose2d(512, 512, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
2. : BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)
3. : Dropout(p=0.5, inplace=False)

)

(decoder2): Sequential( (0): ReLU(inplace=True)

1. : ConvTranspose2d(1024, 512, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
2. : BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)
3. : Dropout(p=0.5, inplace=False) ) (decoder3): Sequential(
4. : ReLU(inplace=True)
5. : ConvTranspose2d(1024, 512, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
6. : BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)
7. : Dropout(p=0.5, inplace=False)

)

(decoder4): Sequential( (0): ReLU(inplace=True)

1. : ConvTranspose2d(1024, 256, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
2. : BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

)

(decoder5): Sequential( (0): ReLU(inplace=True)

1. : ConvTranspose2d(512, 128, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
2. : BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

)

(decoder6): Sequential( (0): ReLU(inplace=True)

1. : ConvTranspose2d(256, 64, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
2. : BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)

)

(decoder7): Sequential( (0): ReLU(inplace=True)

1. : ConvTranspose2d(128, 3, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
2. : Tanh()

)

)

The discriminator architecture is as follows:

Discriminator( (structure): Sequential(

1. : Conv2d(6, 64, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
2. : LeakyReLU(negative\_slope=0.2, inplace=True)
3. : Conv2d(64, 128, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
4. : BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)
5. : LeakyReLU(negative\_slope=0.2, inplace=True)
6. : Conv2d(128, 256, kernel\_size=(4, 4), stride=(2, 2), padding=(1, 1), bias=False)
7. : BatchNorm2d(256, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)
8. : LeakyReLU(negative\_slope=0.2, inplace=True)
9. : Conv2d(256, 512, kernel\_size=(4, 4), stride=(1, 1), padding=(1, 1), bias=False)
10. : BatchNorm2d(512, eps=1e-05, momentum=0.1, affine=True, track\_running\_stats=True)
11. : LeakyReLU(negative\_slope=0.2, inplace=True)
12. : Conv2d(512, 1, kernel\_size=(4, 4), stride=(1, 1), padding=(1, 1), bias=False)
13. : Sigmoid()

)

)

### Dropout

The generator uses dropout in its training network as shown in the generator architecture. The hidden layers in a neural network are very good at extracting features from input images but because of their high computational ability but they are also prone to overfitting the training model. Overfitting happens when a model learns way too much of the training data and performs poorly on the unseen data or validation data. Overfitting is a major problem in any neural network training process. Additionally, neural networks have lots of hyperparame- ters and take a lot of time to train. The ‘Dropout’ technique helps to tackle the problems by making a few units of hidden layers inactive during the training process[26]. Dropout is computed as a probability in PyTorch and usually, the value is 0.5.

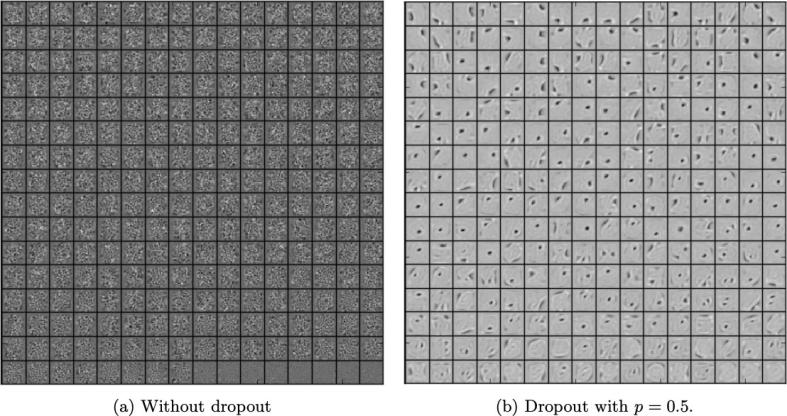


Figure36: Effect of dropout on the features learned in the training of MNIST with one hidden layer autoencoder( image taken from [26])

### Weight Initialisation

Effective initialisation of weights is very important for the training process of a neural net. When starting to train a convolution model, the weight matrix as discussed in chapter 2.2.1 needs to be initialised with different values, these can be assigned as 0 or random values or these values also can be assigned from other pertained model which is already trained in the similar computer vision tasks. The weight initialisation with 0s is not effective mostly because the network can not learn much. Randomly initialising the weights helps the model learn more effectively and also for sometimes initialising weights with values from the pre-trained model is a better option comparatively as it will be discussed in the experiment section in de- tail.

The pix2pix model which was built for this project has a random weight initialisation.

### Learning Rate

As it is explained in chapter 2.2.2, the weights are updated during the training process con- tinuously so that the model can learn better. The hyperparameter which controls the upda- tion of weights is the learning rate. Choosing an appropriate learning rate is critical for any computer vision tasks. The value of the learning rate can vary in the range of 0.0 and 1.0. If the learning rate is too large, it can cause the model to converge too quickly giving a subop- timal solution or if the learning rate is too small, the network might not learn well. One of the main challenges of any computer vision task is to select fine-tuning learning rate during the training process.

### Adam Optimizer

Optimization in the neural network training refers to finding the optimal values for the weights that minimise the loss function. Adam is a kind of optimising algorithm that is sto- chastic gradient-based, computationally efficient and does not need much memory while training[27]. It is assumed that the reader is familiar with the concept of gradient descent. The pseudo-code for Adam is as follows.

*Require: α: Stepsize*

*Require: β1, β2 ∈ [0, 1): Exponential decay rates for the moment estimates Require: f(θ): Stochastic objective function with parameters θ*

*Require: θ0: Initial parameter vector m0 ← 0 (Initialize 1st moment vector) v*

*0 ← 0 (Initialize 2nd moment vector) t ← 0 (Initialize timestep)*

*while θt not converged do t ← t + 1*

*gt ← ∇θft(θt−1) (Get gradients w.r.t. stochastic objective at timestep t) mt ← β1 · mt−1 + (1 − β1) · gt (Update biased first moment estimate)*

*vt ← β2 · vt−1 + (1 − β2) · g 2 t (Update biased second raw moment estimate) mb t ← mt/(1 − β t 1 ) (Compute bias-corrected first moment estimate)*

*vbt ← vt/(1 − β t 2 ) (Compute bias-corrected second raw moment estimate)*

*θt ← θt−1 − α · mb t/( √ vbt + ) (Update parameters) end while return θt (Resulting parame- ters) [27]*

49

### Epoch

An epoch represents a complete training cycle of a neural net. An epoch consists of one or multiple batches of training. The number of epochs a model should run to get an optimal solution is a critical decision. If the model is trained for a small number of epochs, the model will not learn enough to generate a good output but if the model is trained for more number of epochs the model will learn more than it should and overfit.

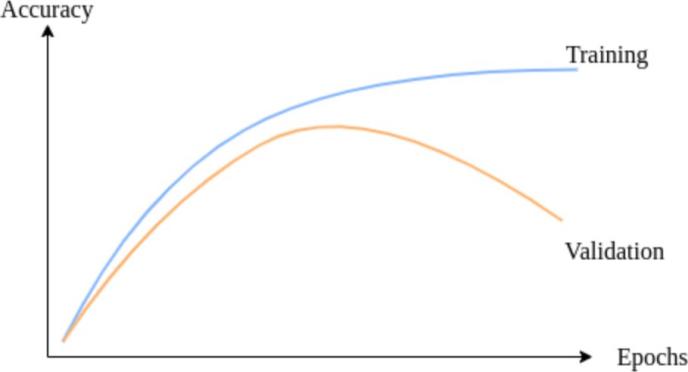


Figure37: epoch vs accuracy(image taken from https[://w](http://www.baeldung.com/cs/epoch-neural-net-)ww[.ba](http://www.baeldung.com/cs/epoch-neural-net-)e[ldung.com/cs/epoch-neural-net-](http://www.baeldung.com/cs/epoch-neural-net-)

works)

As figure 37 shows that though the training accuracy keeps increasing with the increase in the number of epochs, validation accuracy starts going down after a certain number of epochs indicating the model overfilling the training data.

For any GAN model, the usual approach is to train the model for a certain number of epochs but at every 5th epoch, the model is saved and evaluated. As the generator and discriminator do not converge, hence, it is not necessary that if the model is run for 100 epochs, the 100th epoch will give the best results. A better image can be generated before the 100th epoch as well. That is why it is necessary to save and evaluate the model every 5th epoch.

### Model Summary

The pix2pix model is developed using the generator and the discriminator architecture as discussed in the previous paragraphs. The model is initialised using random weights and the model summary is as follows:

Layer (type) Output Shape Param #

================================================================

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Conv2d-1 | [-1, 64, 128, 128] | | | | | 3,072 |
| LeakyReLU-2 | [-1, 64, 128, 128] | | | | | 0 |
| Conv2d-3 | [-1, 128, 64, | | | | 64] | 131,072 |
| BatchNorm2d-4 | [-1, 128, 64, | | | | 64] | 256 |
| LeakyReLU-5 | [-1, 128, 64, | | | | 64] | 0 |
| Conv2d-6 | [-1, 256, 32, | | | | 32] | 524,288 |
| BatchNorm2d-7 | [-1, 256, 32, | | | | 32] | 512 |
| LeakyReLU-8 | [-1, 256, 32, | | | | 32] | 0 |
| Conv2d-9 | [-1, 512, 16, | | | | 16] | 2,097,152 |
| BatchNorm2d-10 | [-1, 512, 16, | | | | 16] | 1,024 |
| LeakyReLU-11 | [-1, 512, 16, | | | | 16] | 0 |
| Conv2d-12 | [-1, | 512, | | 8, | 8] | 4,194,304 |
| BatchNorm2d-13 | [-1, | 512, | | 8, | 8] | 1,024 |
| LeakyReLU-14 | [-1, | 512, | | 8, | 8] | 0 |
| Conv2d-15 | [-1, | 512, | | 4, | 4] | 4,194,304 |
| BatchNorm2d-16 | [-1, | 512, | | 4, | 4] | 1,024 |
| LeakyReLU-17 | [-1, | 512, | | 4, | 4] | 0 |
| Conv2d-18 | [-1, | 512, | | 2, | 2] | 4,194,304 |
| ReLU-19 | [-1, | 512, | | 2, | 2] | 0 |
| ConvTranspose2d-20 | [-1, | 512, | | 4, | 4] | 4,194,304 |
| BatchNorm2d-21 | [-1, | 512, | | 4, | 4] | 1,024 |
| Dropout-22 | [-1, | 512, | | 4, | 4] | 0 |
| ReLU-23 | [-1, 1024, | | | 4, | 4] | 0 |
| ConvTranspose2d-24 | [-1, 512, | | | 8, | 8] | 8,388,608 |
| BatchNorm2d-25 | [-1, 512, | | | 8, | 8] | 1,024 |
| Dropout-26 | [-1, 512, | | | 8, | 8] | 0 |
| ReLU-27 | [-1, 1024, | | | 8, | 8] | 0 |
| ConvTranspose2d-28 | [-1, 512, | | 16, | | 16] | 8,388,608 |
| BatchNorm2d-29 | [-1, 512, | | 16, | | 16] | 1,024 |
| Dropout-30 | [-1, 512, | | 16, | | 16] | 0 |
| ReLU-31 | [-1, 1024, | | 16, | | 16] | 0 |
| ConvTranspose2d-32 | [-1, 256, | | 32, | | 32] | 4,194,304 |
| BatchNorm2d-33 | [-1, 256, | | 32, | | 32] | 512 |
| ReLU-34 | [-1, 512, | | 32, | | 32] | 0 |
| ConvTranspose2d-35 | [-1, 128, | | 64, | | 64] | 1,048,576 |
| BatchNorm2d-36 | [-1, 128, | | 64, | | 64] | 256 |
| ReLU-37 | [-1, 256, | | 64, | | 64] | 0 |
| ConvTranspose2d-38 | [-1, 64, 128, 128] | | | | | 262,144 |
| BatchNorm2d-39 | [-1, 64, 128, 128] | | | | | 128 |
| ReLU-40 | [-1, 128, 128, 128] | | | | | 0 |
| ConvTranspose2d-41 | [-1, 3, 256, 256] | | | | | 6,144 |
| Tanh-42 | [-1, 3, 256, 256] | | | | | 0 |

================================================================

Total params: 41,828,992

Trainable params: 41,828,992

Non-trainable params: 0

Input size (MB): 0.75

Forward/backward pass size (MB): 103.53 Params size (MB): 159.56

Estimated Total Size (MB): 263.85

### 5. 2. 2 CycleGAN

CycleGAN as discussed in the State of The Art is a good candidate for this task. For this project, A pre-trained model is trained with the training set and tested on the validation data set.

The generator of the pre-trained model consists of 3 parts. An encoder, a transformer, a de- coder. It contains a U-net 128 for 128x128 input images and a U-net 256 for 26x25 input images. The generator has the following hyperparameters. The transformer contains 6 resid- ual blocks and 9 residual blocks. The output from the transformer passed through the de- coder.

The architecture of the generator is:

c7s1-64, d128, d256, R256, R256, R256

,R256, R256, R256, u128, u64, c7s1-3

where c7s1-k represents a 7×7 Convolution-InstanceNorm-ReLU layer with k filters and stride 1.

dk signifies a 3 × 3 Convolution-InstanceNorm-ReLU layer with k filters and stride 2.

Rk signifies a residual block that contains two 3 × 3 convolution layers with the same number of filters on both layers.

uk represents a 3 × 3 fractional-strides-Convolution-InstanceNorm-ReLU layer with k filters and stride 1/2.

The discriminator architecture is a PatchGAN as discussed in the state of the art which converts a 256x256 array to 70x70 array of output x matrix where every element of x represents where that patch of x is a “real image” or “ fake image”[9].

#### 5. 3 Experiment

This section demonstrates the training process including both cycleGAN and Pix2Pix on the datasets prepared. The models were trained on 2 sets of data as described in chapter 4. They were trained on AWS-based Deep Learning PyTorch VM instance. The training was done several times using a different number of epochs and learning rates. The models were saved at every 5th epoch to be evaluated later.

Initially, the first pix2pix model was trained after preparing the first dataset which consisted of 1800 images. The model was then evaluated on the validation data set and in the next steps a second dataset was prepared that was trained using pix2pix with different learning rates and run multiple times for different epochs. Again the models were saved for every 5th epoch so that the model could be loaded later and evaluated. As stated in the

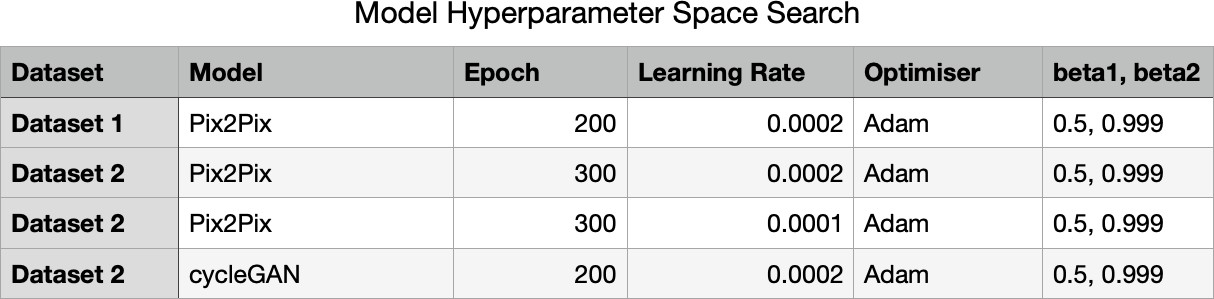
previous chapters, the generator and discriminator of GAN never converge so GAN may not give the best results at the end of the training. For example, if a GAN model is run for 200 epochs, the model saved at the 180th epoch might give a better result than the model saved at the 200th epoch.

For the next set of experiments, a pre-existing code base was used to train for cycle GAN. cy- cleGAN was also run using different learning rates for different numbers of epochs and mod- els were saved at every 5th epoch for each run.

After the training processes were completed, the images were generated and evaluated using evaluation metrics.

### Hyperparameter Search Space

As discussed above the models are experimented with different hyper-parameters. The search space is documented below:



The optimiser and values for beta1 and beta2 have not been changed throughout the exper- iment and kept the same for both models as suggested by the papers[9].

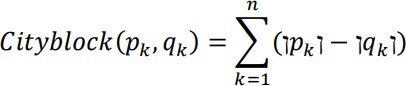
#### 5. 4 Evaluation Metric

For any computer vision task, it is important to assess qualitatively and quantitatively how the model is performing. For any image classification task evaluation is done by comparing the predicted output image to the input image. But for GAN the generated images are actually fake and are trained to look as real as possible to the ground truth images. So assessing a GAN-based model is different from any other computer vision task. The evaluation metric should have two traits basically. One is Fidelity and the second is Diversity. Fidelity represents the quality or resolution of images and Diversity signify that how the generated images are inherent with the input images. Any evaluation metric for assessing GAN has to take these two traits into consideration. These can be achieved by actually the input image and the gen- erated image pixel by pixel distance or feature by feature distance. The pixel distance repre- sents the distance between pixels of two images and the feature distance signifies comparing feature-based distance using a pre-trained image classification model using the intermediate layer of a model.

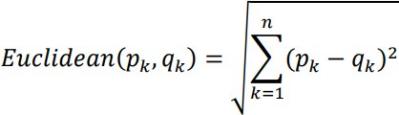
Every Image basically has some unique and distinct features. The similarity of the feature between the ground truth and generated images can be computed by using different distance metrics[28].

Some examples of distance measurements are as follows:

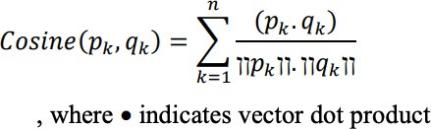
Cityblock Distance: computes the path between pixels of the two images[28].

 Figure38: cityblock distance[28]

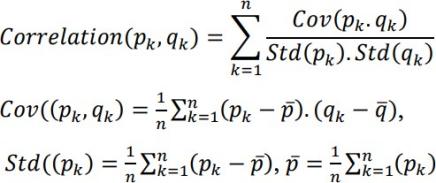
Euclidean Distance: this is the most common distance measure which calculates the sum of the square root of the difference between features of two images.[28]

Figure39: Euclidean distance[28]

Cosine Distance: computes the normalised dot products of two feature points between two images[28].

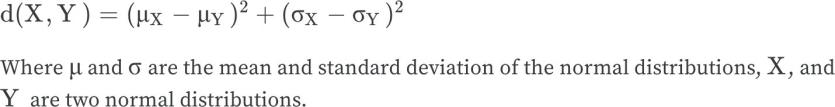
Figure40: Cosine distance computation[28]

Correlation Distance: computes the correlation of distance between two feature points be- tween two images[28].

Figure41: Correlation distance computation[28]

Fréchet distance: it is a measure of similarity of the distance between curves that considers the location and points along the curve. Fréchet distance is mostly used to measure the dis- tance between two different distributions.

Fréchet distance is used to calculate FID or Fréchet Inception Distance the ground truth im- age and the generated image by GAN. Two other metrics PSNR and SSIM used as well for this project to evaluate the quality of images generated by GAN.

Figure42: Fréchet distance computation[29]

### 5. 4 .1 Clean FID

Clean FID is based on Fréchet distance and created as a Fréchet Inception Distance metric to calculate the FID score of the GAN-generated images. The inception v3 model is used as a pre-trained model on the imagined dataset[30]. ”The use of activations from the Inception V3 model to summarize each image gives the score its name of &Fréchet Inception Distance”

or FID[29]. The clean FID actually computes the same FID whether it is in TensorFlow or

PyTorch[30].FID scoring is consistent and very sensitive to image blurriness. One drawback of clean FID is that it requires a huge number of sample images[31]. FID is tested on the CIFAR10 dataset and sometime FID does not compute properly if the number of images per class is low[31]. The lower the value of the FID, the better the quality of the generated im- ages.[32]. To compute FID for the images in the test set, a clean FID library is used on the

test set folder in VM.

Figure43: Average FID score of different datasets[32]

### 5. 4. 2 SSIM

SSIM or Structural Similarity Index is a qualitative measurement of images. Instead of mea- suring the “ visibility of errors” between the ground truth and generated image, SSIM at- tempts to assess images through human perception. Introduced in the paper by Zhou Wang, Member, IEEE, Alan Conrad Bovik et al in 2004[33], SSIM is one of the widespread metrics used in computer vision tasks. SSIM attempts to measure the luminosity, contrast and sym- metry of the two images.

56

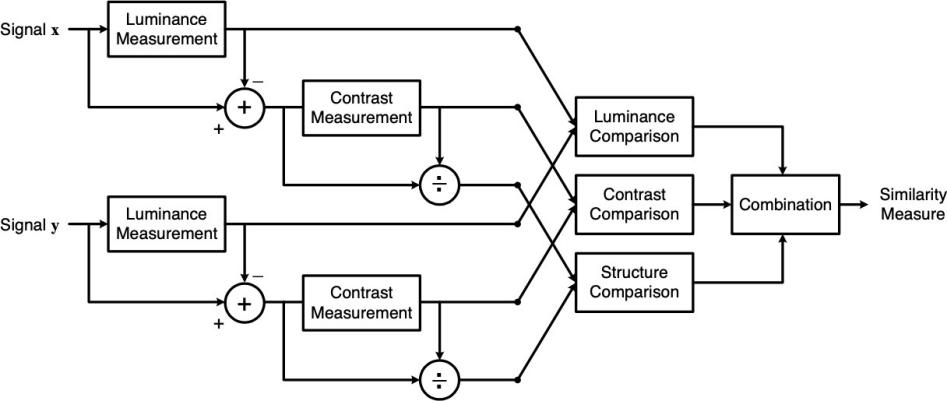


Figure44: SSIM measurement system(image taken from[33])

The Function of SSIM is computed from the function of luminosity, function of contrast and function of symmetry. SSIM value ranges between 0 to 1, 1 representing the highest quality images[34]. For this project, StructuralSimilarityIndexMeasure is used PyTorch torchmetrics library[38].

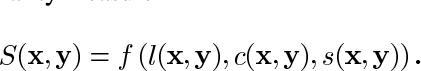


Figure45: SSIM function(image taken from [33])

### 4. 3 PSNR

PSNR or Peak Signal Noise Ratio is strictly a quantitative measurement of images online SSIM which takes human visual perception into consideration. PSNR is based on MSE or mean squared error between pixel values of two images. MSE signifies the average of the square of the errors between the actual and the noisy image. The higher the error, the more degraded and different the noisy image is from the input image.

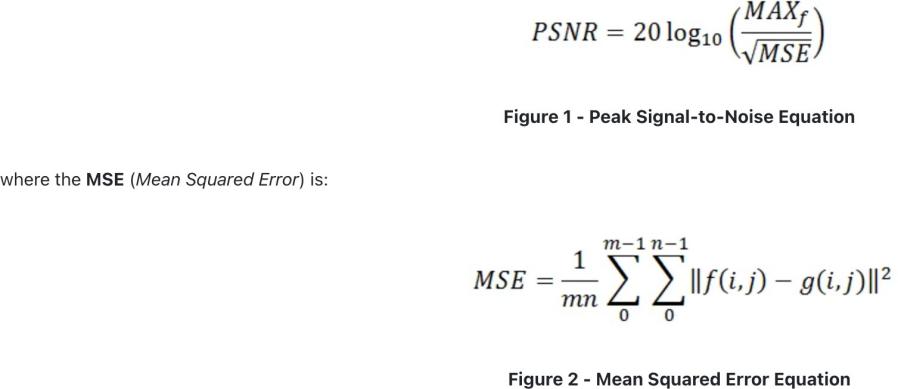


Figure46: PSNR computation(image taken from [35])

Figure 46 shows PSNR computation where f signifies the original image,

g signifies the degraded output image and MAXf is the maximum signal value that exists in the original image[35]. In PyTorch, PSNR is computed using the PeakSignalNoiseRatio function from the torchmetrics library[38].

## Results

The first pix2pix model trained on dataset 1 (training dataset 1800 images) is run for 200 epochs. FID, PSNR and SSIM are computed on the test set. One set of examples is given below.



Figure47: the input image(left) predicted image(middle) ground truth(right) by pix2pix(©Copyright byBird.I)



Figure48: the input image(left) predicted image(middle) ground truth(right) by pix2pix(©Copyright byBird.I)

The second model is trained on dataset 2 (training set of -800 images) for 300 epochs with a learning rate of 0.0001. FID, PSNR and SSIM are computed on the test set as shown below.

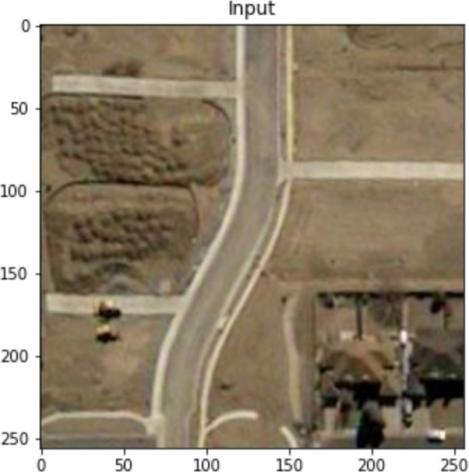
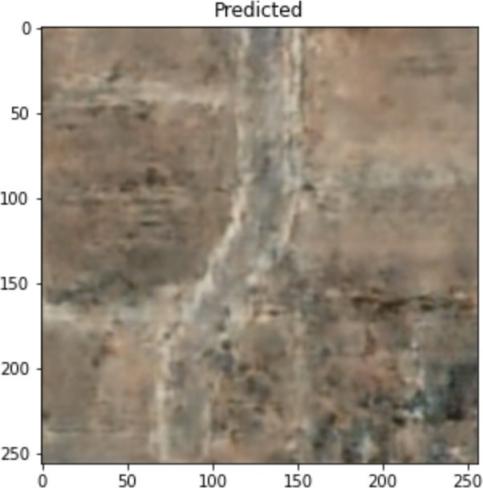
  

Figure49: the input(left) predicted(middle) ground truth(right) by pix2pix(©Copyright byBird.I)

The third model is trained on dataset 2 for 300 epochs with a learning rate of 0.0002 and the evaluation metrics are computed. One example of the generated image is shown below.

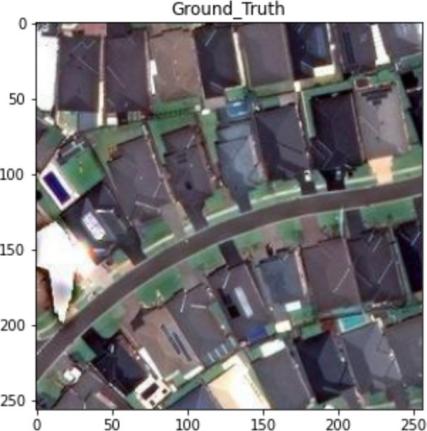
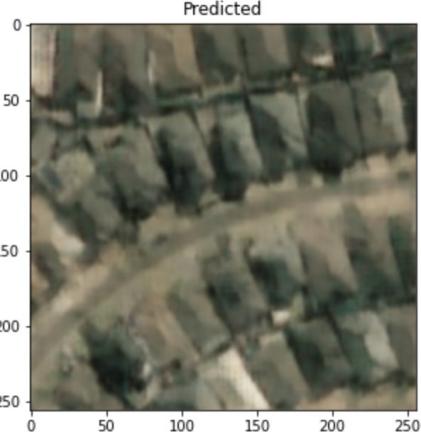
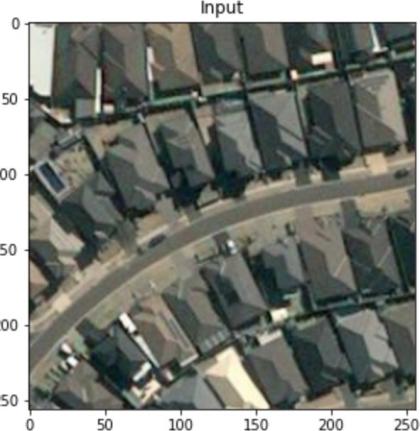


Figure50: the input(left) predicted(middle) ground truth(right) by pix2pix(©Copyright byBird.I)

Figure 47,48,49,50 and 51 show the comparisons of the predicted images and ground truth im- ages. The next model is trained on dataset 2 for 300 epochs with a learning rate of 0.0001 and the evaluation metrics are computed. One example of the generated image is shown below.

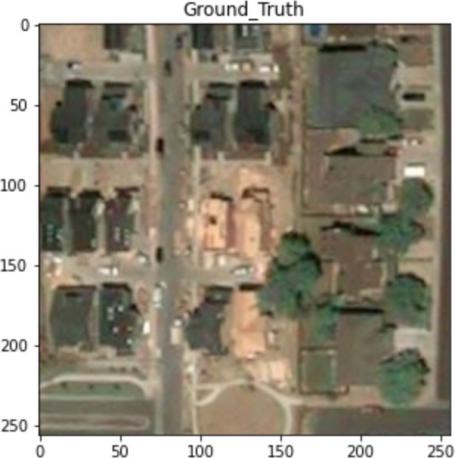
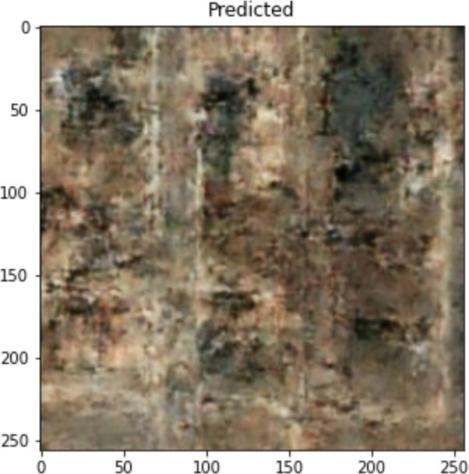
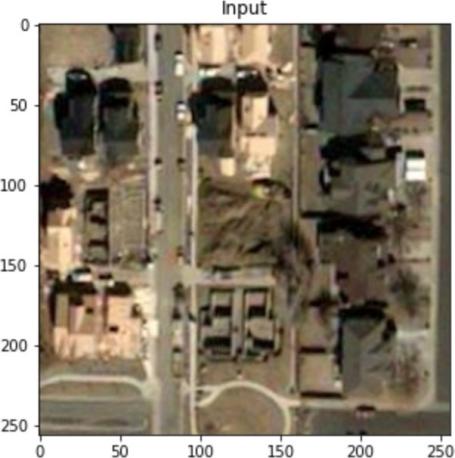


Figure51: the input(left) predicted(middle) ground truth(right) by pix2pix(©Copyright byBird.I)

The next training is performed using cycle GAN using the pre-existing code as discussed in the methodology. One example of the image set is shown below.

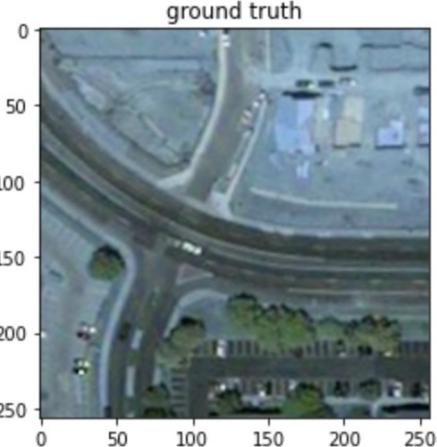
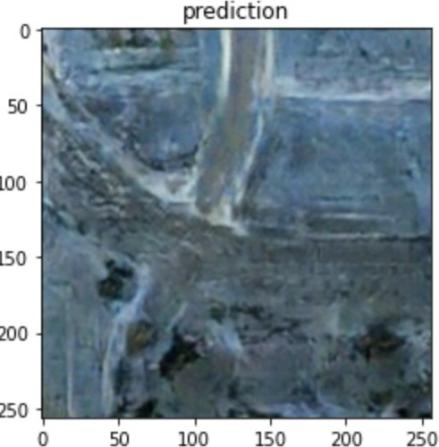
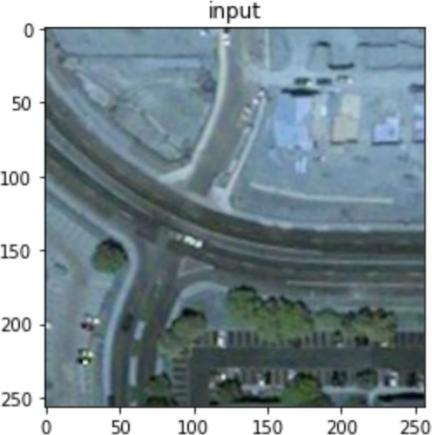


Figure52: the input(left) predicted(middle) ground truth(right) by cycleGAN(©Copyright byBird.I)

### Results - Comparison Of Different Models Using FID

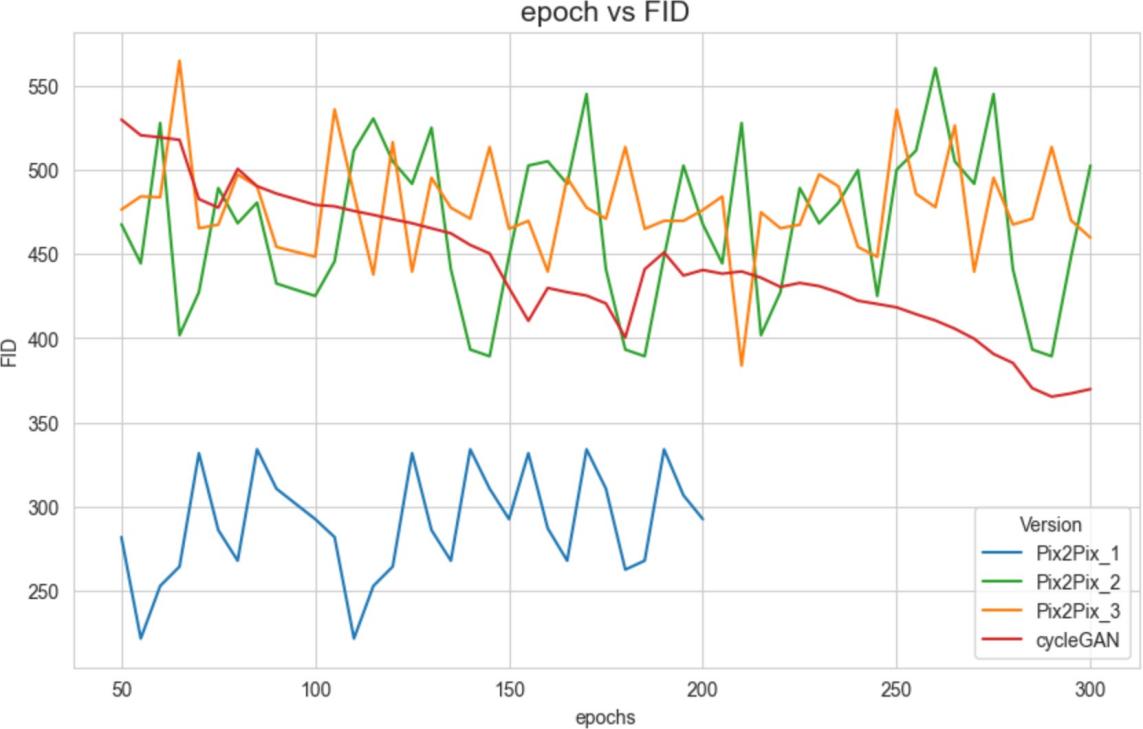
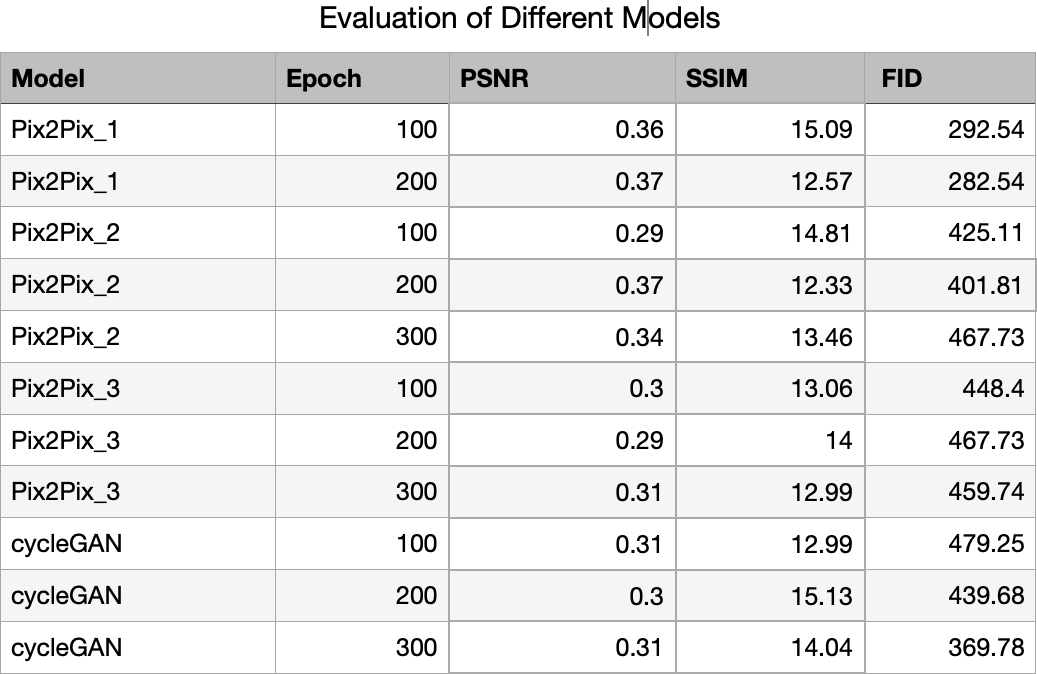


Figure53: epochs vs FID(pix2pix & cycle GAN)

Even though the models do not generate expected results, comparing the FIDs for different models gives a clear idea of how the models have performed. As stated earlier FID is calcu- lated using the clean FID PyTorch library in the VM server. This particular library can not be used on the CPU. A classic way of calculating FID for GANs is to compute and print FID for every 5th epoch and save the model. Then choose the model with the lowest FID to gen- erate the best possible image and compare it with the ground truth. But for the computational time, that approach is not followed here. Rather after the training is completed, the models are loaded in VM and FID is computed on the test set. One more thing to consider is that Clean FID calculates FID on the test set folder which contains all test images. So average FID can be calculated by dividing the total FID by the number of images in the test set.

Though Figure 50 shows the total FID computed on the test set containing all images.

It is evident for Figure 50 that the first model(pix2pix\_1) which was trained on a larger num- ber of training sets performs better because the FID of the first model is much lower than all the other models. Pix2pix\_2 with a learning rate of 0.0002 has a lot more fluctuating scoring than Pix2pix\_3 with a learning rate of 0.001. Surprisingly cycle GAN has a higher FID score signifying that cycleGAN has not performed better than other pix2pix models. The below table shows the PSNR, SSIM, and FID scores of different models at the 100th, 200th and 300th epochs.



## Evaluation

Figures in the results section show the comparison between the input, predicted images gen- erated images by the pix2pix model and cycleGAN. The hue of the ground truth image in figure 50 is different from the input image but there is no transition from input to ground truth. The generated image looks identical to the ground truth image. In figures 49 & 51 there are some transitions as new buildings can be seen in the ground truth. But the generated im- age looks similar to the input image.

Figure 46 shows the predicted image blurry and not similar to the ground truth image.

The figure 51 shows a blurry generated image. This image is produced by the model at the 250th epoch. The model is able to create the road but the buildings are blurry and not clear. Figure 49,50 & 51 show the generated images from the next set of models with different learning rate and that has not helped the model training as the generated image looks very hazy and buildings and roads are not completely formed. Figure 52 shows the image generat- ed by cycleGAN, the hue of the images are different as they come from different code bases. Here also the predicted image is not clear.

Comparing the generated images from different models, one fact comes out as obvious and that is the size of the training dataset. As introduced in the paper[9] the minimum number of training images for pix2pix satellite images to maps is 2000. This matches with the findings from the first training where the training size was 1800. The image quality is comparable to the original image. But the image quality degrades when the number of training images comes down to 800. Another thing to consider is that the variance of the images and the actu- al numbers of images which have any transition is very low even in the first dataset. The fea- tures in the images vary greatly in terms of the layout of the land, building architecture etc.

That increases the variance of the dataset but the sample size is less and the models find it difficult to capture the variance and difference of input and ground truth image as the number of images with transitions are very low. It will be interesting to see how the pix2pix performs when the dataset size increases and all the input and ground truth images have some actual difference.

## Conclusion

### Summary

This project has demonstrated a few possible approaches for the image-to-image translation task of satellite images. Though it has not produced viable results, the project has opened up some interesting possibilities to work further.

One more thing has certainly been proved that pix2pix and cycle GAN can indeed be used to generate satellite images. All the projects accomplished in the papers[9,11, 12] produce im- ages of maps from satellite images or bags, shoes from sketches to paintings etc.

Satellite images are complex and have lots of features and high variance, pix2pix was able to generate “fake images” that look identical to the ground truth images with good resolution seemed quite promising.

Both Pix2pix and cycleGAN were successfully built and two hyper-parameters i.e learning rate and different epochs were tested to see if the models perform better. The pix2pix model which shows a clear image in figure 47 also gives an inference on a random image collected and preprocessed from BirdI.

Though the results of the project are encouraging, the model generates images which do not have great FID or PSNR and SSIM values which clearly do not give us a model to deploy.

One of the major limitations of this project was the non-availability of the input image and ground truth images with some difference in architectures between them. Also, due to scarce resource and availability, the number of training samples were not adequate enough. Many hyperparameter searches can further be tweaked to generate better images.

But due to the lack of enough training sizes, only the learning rate was changed and model performance was experimented with running different epochs. Time and resources were def- initely a constraint for this project. Apart from pix2pix and cycleGAN, there are a few archi- tectures like medGAN[11] which could be developed with more complex generators to test if they can generate better images. But before that, it is important to gather enough training samples for further experiments.

### Evaluation

Before building the architecture, it was necessary to understand the different classes of Py- Torch. For dataset loading, it was important to understand how to download data from the AWS S3 bucket programmatically. Also as a part of dataset preprocessing it was necessary to understand the geographical location of the images and segregate them based on their loca- tions. A pix2pix model was successfully built and an Inference image was provided to Bird. I. Though the project remains inconclusive, pix2pix and cycleGAN models were explored and experimented with. Data availability of course hindered trying different approaches or mod- els but overall a thorough effort was given to explore the models.

### Limitations

One of the limitations of this project is definitely the size of training data which has hindered the project in a few ways. Another limitation that can be dealt with is the kind of images col- lected. Rather than the satellite images covering a variety of different landscapes, if only the data is collected for a certain kind of landscape for example a training data set which consists of either paving roads or laying foundations or similar looking buildings, that would bring down the variance of the training set which may help in improving the performance. Anoth- er limitation is the scarce time for this project. Considering the complexity of this project, it requires more time and effort in terms of collecting and preparing the data and experiment- ing with GAN architectures.

### Future work

As stated before, this project is quite unique in terms of the goals it wants to accomplish and its commercial viability. Though the project is not able to produce conclusive results, it cer- tainly opens up further opportunities to explore. This project demonstrates that pix2pix can create realistic-looking satellite images. For further exploration, a data set is prepared with sample images, there are many different architectures of GAN like medGAN[14], Style- GAN[36], and PoseGAN[37]. The authors of the medGAN show that the quality of gener- ated images increases with the complexity of generator architecture[14]. The performance of this project may improve by experimenting with the different generator architecture as well.

### Bibliography

1. Satellite Data Services: Satellite Imaging Applications and Benefits for Land and Sea. https[://w](http://www.researchdive.com/blog/the-most-important-applications-and-benefits-of-)ww[.r](http://www.researchdive.com/blog/the-most-important-applications-and-benefits-of-)e[searchdive.com/blog/the-most-important-applications-and-benefits-of-](http://www.researchdive.com/blog/the-most-important-applications-and-benefits-of-) satellite-data-services#:~:text=The%20most%20important%20applications%20are,In- frastructure%20%26%20construction%20monitoring. Accessed 23 Sept. 2022.
2. ‘Up to Date Satellite Images | Recent Satellite Images’. Bird.i, https://hibirdi.com/prod- ucts/satellite-images/up-to-date-satellite-images/. Accessed 23 Sept. 2022.
3. Use of satellite imagery to diagnose events leading to frontal thunderstorms: Part II of a case study, [online], https://rmets.onlinelibrary.wiley.com/doi/abs/10.1002/ met.5060020102, accessed 2022-09-07.
4. Gautam, Akshat, et al. Realistic River Image Synthesis Using Deep Generative Adversari- al Networks. arXiv, 27 July 2021. arXiv.org, https://doi.org/10.48550/arXiv.2003.00826.
5. Isola, Phillip, et al. ‘Image-to-Image Translation with Conditional Adversarial Networks’. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), IEEE, 2017, pp. 5967–76. DOI.org (Crossref), https://doi.org/10.1109/CVPR.2017.632.
6. Zhu, Jun-Yan, et al. ‘Unpaired Image-to-Image Translation Using Cycle-Consistent Ad- versarial Networks’. 2017 IEEE International Conference on Computer Vision (ICCV), IEEE, 2017, pp. 2242–51. DOI.org (Crossref), https://doi.org/10.1109/ICCV.2017.244.
7. Goodfellow, Ian J., et al. Generative Adversarial Networks. 2014. DOI.org (Datacite), https://doi.org/10.48550/ARXIV.1406.2661
8. ‘LII. An essay towards solving a problem in the doctrine of chances. By the late Rev. Mr. Bayes, F. R. S. communicated by Mr. Price, in a letter to John Canton, A. M. F. R. S’. Philosophi- cal Transactions of the Royal Society of London, vol. 53, Dec. 1763, pp. 370–418. DOI.org (Crossref), https://doi.org/10.1098/rstl.1763.0053.
9. Isola, Phillip, et al. Image-to-Image Translation with Conditional Adversarial Networks. arXiv, 26 Nov. 2018. arXiv.org, [http://arxiv.org/abs/1611.07004.](http://arxiv.org/abs/1611.07004)
10. Dertat, Arden. ‘Applied Deep Learning - Part 3: Autoencoders’. Medium, 8 Oct. 2017, https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798.
11. Ronneberger, Olaf, et al. U-Net: Convolutional Networks for Biomedical Image Segmenta- tion. arXiv, 18 May 2015. arXiv.org, [http://arxiv.org/abs/1505.04597.](http://arxiv.org/abs/1505.04597)
12. Zhu, Jun-Yan, et al. Unpaired Image-to-Image Translation Using Cycle-Consistent Ad- versarial Networks. arXiv, 24 Aug. 2020. arXiv.org, [http://arxiv.org/abs/1703.10593.](http://arxiv.org/abs/1703.10593)
13. PyTorch. https[://w](http://www.pytorch.org/)ww[.p](http://www.pytorch.org/)y[torch.org.](http://www.pytorch.org/) Accessed 23 Sept. 2022.
14. Armanious, Karim, et al. ‘MedGAN: Medical Image Translation Using GANs’. Comput- erized Medical Imaging and Graphics, vol. 79, Jan. 2020, p. 101684. arXiv.org, https:// doi.org/10.1016/j.compmedimag.2019.101684.
15. Yang, Wei, et al. ‘Predicting CT Image From MRI Data Through Feature Matching With Learned Nonlinear Local Descriptors’. IEEE Transactions on Medical Imaging, vol. 37, no. 4, Apr. 2018, pp. 977–87. IEEE Xplore, https://doi.org/10.1109/ TMI.2018.2790962.
16. Rueda, Andrea, et al. ‘Single-Image Super-Resolution of Brain MR Images Using Over- complete Dictionaries’. Medical Image Analysis, vol. 17, no. 1, Jan. 2013, pp. 113–

32. DOI.org (Crossref), https://doi.org/10.1016/j.media.2012.09.003.

1. Dou, Q., Chen, H., Yu, L., Qin, J., Heng, P. A., 2017. Multi-level contextual 3-D CNNs for false positive reduction in pulmonary nodule detection. volume 64, pages 1558– 1567.
2. Havaei, M., Davy, A., Warde-Farley, D., Biard, A., Courville, A., Bengio, Y., Pal, C., Jodoin, P.-M., Larochelle, H., 2015. Brain tumor segmentation with deep neural net- works. In Medical Image Analysis, volume 35, pages 18 – 31. He, K., Zhang, X., Ren, S., Sun, J., 2011.

19 Chen, H., Zhang, Y., K. Kalra, M., Lin, F., Chen, Y., Liao, P., Zhou, J., Wang, G., 2017. Low-Dose CT with a residual encoder-decoder convolutional neural network. In IEEE Transactions on Medical Imaging, volume 36, pages 2524–2535.

1. Quan, T. M., Nguyen-Duc, T., Jeong, W. K., 2018. Compressed sensing MRI reconstruc- tion using a generative adversarial network with a cyclic loss. In IEEE Transactions on Medical Imaging, volume 37, pages 1488–1497. Radford, A., Metz, L., Chintala, S.
2. Shah, Sohil, et al. Stacked U-Nets: A No-Frills Approach to Natural Image Segmenta- tion. arXiv, 27 Apr. 2018. arXiv.org, [http://arxiv.org/abs/1804.10343.](http://arxiv.org/abs/1804.10343)
3. Gatys, L. A., Ecker, A. S., Bethge, M., 2016. Image style transfer using convolutional neural networks. In IEEE Conference on Computer Vision and Pattern Recognition, pages 2414–2423.
4. Karras, Tero, et al. A Style-Based Generator Architecture for Generative Adversarial Net- works. arXiv, 29 Mar. 2019. arXiv.org, [http://arxiv.org/abs/1812.04948.](http://arxiv.org/abs/1812.04948)
5. Zhu, Jun-Yan, et al. Unpaired Image-to-Image Translation Using Cycle-Consistent Ad- versarial Networks. arXiv, 24 Aug. 2020. arXiv.org, [http://arxiv.org/abs/1703.10593.](http://arxiv.org/abs/1703.10593)
6. Ioffe, Sergey, and Christian Szegedy. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. arXiv, 2 Mar. 2015. arXiv.org, https:// doi.org/10.48550/arXiv.1502.03167.
7. Nitish Srivastava, Improving Neural Networks with Dropout, 2013, available at: [http://www.cs.toronto.edu/~nitish/msc\_thesis.pdf.](http://www.cs.toronto.edu/~nitish/msc_thesis.pdf)
8. Kingma, Diederik P., and Jimmy Ba. Adam: A Method for Stochastic Optimization. arX- iv, 29 Jan. 2017. arXiv.org, [http://arxiv.org/abs/1412.6980.](http://arxiv.org/abs/1412.6980)
9. Kavitha, K., and B. Thirumala Rao. ‘Evaluation of Distance Measures for Feature Based Image Registration Using AlexNet’. *International Journal of Advanced Computer Science and Ap- plications*, vol. 9, no. 10, 2018. *arXiv.org*, https://doi.org/10.14569/IJACSA.2018.091034.
10. ‘Weights & Biases’. W&B, https://wandb.ai/ayush-thakur/gan-evaluation/reports/How- to-Evaluate-GANs-using-Frechet-Inception-Distance-FID---Vmlldzo0MTAxOTI. Accessed 23 Sept. 2022.
11. Parmar, Gaurav, et al. On Aliased Resizing and Surprising Subtleties in GAN Evaluation. arXiv, 20 Jan. 2022. arXiv.org, [http://arxiv.org/abs/2104.11222.](http://arxiv.org/abs/2104.11222)
12. Shmelkov, Konstantin, et al. How Good Is My GAN? arXiv, 25 July 2018. arXiv.org, [http://arxiv.org/abs/1807.09499.](http://arxiv.org/abs/1807.09499)
13. Lucic, Mario, et al. Are GANs Created Equal? A Large-Scale Study. arXiv, 29 Oct. 2018. arXiv.org, <http://arxiv.org/abs/1711.10337>
14. Zhou Wang, A. C. Bovik, H. R. Sheikh and E. P. Simoncelli, "Image quality assessment: from error visibility to structural similarity," in IEEE Transactions on Image Processing, vol. 13, no. 4, pp. 600-612, April 2004, doi: 10.1109/TIP.2003.819861.
15. Nilsson, Jim, and Tomas Akenine-Möller. Understanding SSIM. arXiv, 29 June 2020. arXiv.org, [http://arxiv.org/abs/2006.13846.](http://arxiv.org/abs/2006.13846)
16. PA. Horé and D. Ziou, "Image Quality Metrics: PSNR vs. SSIM," 2010 20th In- ternational Conference on Pattern Recognition, 2010, pp. 2366-2369, doi: 10.1109/ ICPR.2010.579.
17. Karras, Tero, et al. A Style-Based Generator Architecture for Generative Adversarial Networks. arXiv, 29 Mar. 2019. arXiv.org, [http://arxiv.org/abs/1812.04948.](http://arxiv.org/abs/1812.04948)
18. Liu, Kanglin, et al. PoseGAN: A Pose-to-Image Translation Framework for Camera Lo- calization. arXiv, 22 June 2020. arXiv.org, [http://arxiv.org/abs/2006.12712.](http://arxiv.org/abs/2006.12712)
19. Welcome to TorchMetrics — PyTorch-Metrics 0.9.3 Documentation. https://torch- metrics.readthedocs.io/en/stable/. Accessed 23 Sept. 2022.

### The List Of Figures

Figure1: discriminative model Figure2: a generative model Figure3: GAN

Figure4: MNIST

Figure5: generator architecture Figure6: encoder-decoder Figure7: Artificial Neural Network Figure8: activation function Figure9: Fully Connected ANN Figure10: CNN architecture Figure11: U-Net Architecture

Figure 12: Example of seamless segmentation of large images using U-Net Figure 13: How PatchGAN works

Figure 14: application of pix2pix Figure 15:facades labels —-> photos

Figure 16:automatically detected edges to shoes Figure 17:pix2pix architecture

Figure 18: paired (left) and unpaired (right) image translation Figure 19: forward and backward cycle consistency

Figure 20: examples of output images by cycle GAN Figure21: Architecture of cycle GAN

Figure22: the architecture of med-GAN

Figure23: Input Image(left) & Ground Truth(right) Figure24: Input Image(left) & Ground Truth(right) Figure25: Input Tile(left) & Ground Truth Tile Figure26: a processed 256x256 image

Figure27: the example of a concatenated image

Figure28: the input(left) and ground truth(right) show no transition

Figure29: the split of images into the train, test and val based on geo locations Figure30: ReLU

Figure31: Leaky ReLU Figure32: Convolution Process

Figure33: A Neural Net With Batch-norm Figure34: A Neural Net With Batch-norm

Figure35: Batch Normalization effect on ImageNet training

Figure36: Effect of dropout on the features learned in the training of MNIST with one hid- den layer autoencoder

Figure37: epoch vs accuracy Figure38: city-block distance Figure39: Euclidean distance

Figure40: Cosine distance computation Figure41: Correlation distance computation Figure42: Fréchet distance computation[29]

Figure43: Average FID score of different datasets[32] Figure44: SSIM measurement system(image taken from[33]) Figure45: SSIM function(image taken from [33])

Figure46: PSNR computation(image taken from [35])

Figure47: the input image(left) predicted image(middle) ground truth(right) by pix2pix Figure48: the input image(left) predicted image(middle) ground truth(right) by pix2pix Figure49: the input(left) predicted(middle) ground truth(right) by pix2pix

Figure50: the input(left) predicted(middle) ground truth(right) by pix2pix Figure51: the input(left) predicted(middle) ground truth(right) by pix2pix Figure52: the input(left) predicted(middle) ground truth(right) by cycleGAN Figure53: epochs vs FID(pix2pix & cycle GAN)