Satellite Map to Road View Map Conversion using Pix2Pix GAN

A PROJECT REPORT
SUBMITTED BY:

Amogh Prabhu (2014160)

Manjusha Kolli (2014051)

in partial fulfillment for the award of the degree

of

BACHELOR OF TECHONOLOGY

IN

ELECTRONICS AND COMMUNICATION ENGINEERING

at

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

NATIONAL INSTITUTE OF TECHNOLOGY SILCHAR

SILCHAR, ASSAM (INDIA)-788010

MAY 2024

DECLARATION

I hereby declare that the project entitled "Satellite Map to Road view Map conversion using Pix2Pix GAN." submitted for the B. Tech. (ECE) degree is my original work and the project has not formed the basis for the award of any other degree, diploma, fellowship, or any other similar titles.

Amogh Prabhu

Registration No : 20-1-4-160

Manjusha Kolli

Registration No : 20-1-4-051

Place: Silchar,Assam

Date: 17/05/2024

CERTIFICATE FROM SUPERVISOR

It is certified that the work contained in this End–Semester report entitled “**Satellite Map to Road View Map conversion using Pix2Pix GAN.**”submitted by Amogh Prabhu(Reg No:20-1-4-160) and Manjusha Kolli(Reg No:20-1-4-051) for the partial fulfilment of the award of Bachelor of Technology,Electronics and Communication Engineering,is absolutely based on their own work carried out under my supervision and their work thesis has not been submitted elsewhere for any degree.

Dr. GAURAV SINGH BAGHEL

Assistant Professor

National Institute of Technology (NIT) Silchar

Department of Electronics and Communications Engineering

Silchar, Assam, 788010, India

Place: Silchar,Assam

Date: 17/05/2024

ACKNOWLEDGEMENT

We would like to express our heartfelt gratitude and special thanks to Dr. Gaurav Singh Baghel, Assistant Professor in the Electronics and Communication Engineering Department at NIT Silchar and Dr. Gaurav Singh Baghel, Assistant Professor in the Electronics and Communication Department at NIT Silchar, for their invaluable mentorship during our project. Their guidance, encouragement, and support have been instrumental in our success.

Throughout the duration of our project, They provided us with their expert advice and direction that greatly influenced our research. Their insightful suggestions and constructive feedback helped shape the trajectory of our work, ensuring its quality and depth. We are truly indebted to them for imparting their wisdom and knowledge, which have been pivotal in our intellectual growth. Their guidance and co-operation are sincerely acknowledged.

ABSTRACT

The proposed project utilizes the pix2pix GAN architecture to tackle the task of satellite-to-map image translation, leveraging its capacity for conditional generation to achieve context-aware transformations. Central to the architecture are the Generator and Discriminator components, which undergo adversarial training to generate realistic map images from satellite inputs. While the Generator learns the intricate mapping between the satellite and map domains, the Discriminator serves as a classifier, discerning between real and generated map images. Through the strategic combination of Binary Cross-Entropy Loss and L1 Loss functions, the training process guides the model towards producing high-quality and precise map representations that closely align with the input satellite images. Rigorous evaluation using metrics like SSIM, PSNR, MSE, and MAE provides quantitative validation of the model's performance and fidelity to the original data. By advancing image translation techniques through deep learning, the project aims to contribute to fields such as urban planning, navigation, and environmental monitoring. The ability to accurately and efficiently translate satellite imagery into map representations holds significant promise for various real-world applications, offering valuable insights and facilitating decision-making processes in diverse domains. As such, the project underscores the transformative potential of the pix2pix GAN architecture in addressing complex image translation tasks with practical implications across multiple industries and sectors.

TABLE OF CONTENTS

1. INTRODUCTION

This project embarks on an extensive exploration of image-to-image translation, harnessing the power of a Generative Adversarial Network (GAN) to specifically convert satellite images into corresponding map representations. The dataset employed is a meticulously curated collection of paired satellite and map images.The GAN architecture is meticulously crafted, featuring both a generator and discriminator, with convolutional layers seamlessly integrated to facilitate the intricate process of translation.

The central goal of the U-Net-based generator is to proficiently generate realistic map images from satellite inputs, while the PatchGAN discriminator is meticulously designed to play a pivotal role in distinguishing between authentic and generated map images at the patch level. The training paradigm is thoughtfully designed, involving the simultaneous optimization of both the U-Net generator and PatchGAN discriminator through a combination of adversarial loss and L1 loss functions. This dual-objective optimization framework ensures that the generator produces map images that not only captivate visually but also align closely with their authentic counterparts. The training process unfolds over multiple epochs, strategically leveraging a batched dataset for computational efficiency.

Post-training, a meticulous evaluation is conducted by generating predictions for a random subset of the dataset. This evaluation vividly illustrates the U-Net generator's proficiency in transforming satellite images into visually compelling and contextually meaningful map representations.

The culmination of the project involves the preservation of the trained U-Net generator model, opening avenues for potential future applications and advancements. This work not only significantly contributes to the broader field of image-to-image translation but also highlights the promising potential of GANs, U-Net generators, and PatchGAN discriminators in unlocking meaningful insights from satellite imagery, thereby enriching the domain of spatial data analysis and visualization.

1.1 Problem Identification :

Manual Efforts and Infrequent Updates: Manual processes lead to labor-intensive efforts and infrequent map updates, particularly in less populated areas.

Complex Road Structures and Irregular Features: Conventional techniques struggle to accurately represent complex road structures and irregular features, resulting in inaccuracies in map generation.

Coarse Resolution Data and Pixel-wise Translation Requirements: Conventional methods struggle with generating high-resolution maps and lack pixel-wise translation capabilities, limiting accuracy.

Costly and Time-Consuming Process: Manual efforts are time-consuming and expensive, leading to high costs and limited update frequency.

Dependency on External Data Sources: Conventional techniques rely heavily on external data sources, which may have limitations in coverage and availability.

Limited Automation and Difficulty in Scaling: Traditional approaches lack automation and struggle to scale effectively to handle large volumes of satellite imagery.

Visual Obstructions: Factors like shadows and clouds present challenges, impacting the quality of map-style images.

2. LITERATURE REVIEW

The review outlines the foundational concept of generative models, as described in papers [1], [2], and [3], which assume data creation based on specific distribution parameters. These papers establish the theoretical framework of GANs, elucidating how algorithms approximate underlying distributions to generate data, extending beyond traditional classification tasks.

Furthermore, the review mentions the practical applications of GANs discussed in papers [4], [5], and [6], emphasizing their role in generating diverse datasets and producing unlabeled or uncategorized data. These papers likely present case studies or experiments demonstrating the effectiveness of GANs across different domains, such as image synthesis and data generation.

Another method combines cVAE-GAN [7] and cLR-GAN model [8] to generate diverse and realistic outputs. In detail, they utilize cVAE-GAN [7] to encode ground truth target image x_B into a latent space Z and the generator uses input source image x_A conditioned with randomly sampled latent code z to produce translated image x_{AB}. The process can be denoted as $x_B \rightarrow z \rightarrow x_{AB}$. Then, they exploit cLR-GAN model [8] which generates translated output image x_{AB} with input x_A and random latent code z and then tries to reconstruct the latent code from x_{AB}. Similarly, the training process can be denoted as $z \rightarrow x_B \rightarrow z$. By combining the two objectives into a hybrid model, BicycleGAN can generate diverse and realistic outputs.

In [9] Variational Autoencoders (VAEs) present several challenges in their operation. Firstly, they often struggle to achieve a uniform distribution of latent points, potentially hindering diversity in generated outputs. Directly mapping input images to points in the latent space poses difficulties, limiting the model's ability to capture the full variability of the data. Additionally, while VAEs aim to reduce noise in reconstructed images, effectively generating noise remains a challenge in certain cases. The training process for VAEs is complex, requiring careful optimization of reconstruction and divergence losses, along with meticulous tuning of hyperparameters. Finally, like Generative Adversarial Networks (GANs), VAEs are prone to mode collapse, where the model fails to capture the full diversity of the data distribution, leading to limited variation or repetitive patterns in synthesized outputs.

3. TECHNICAL NOVELITY

1. **Customized Architecture**: The project utilizes a custom architecture for both the generator and discriminator networks tailored specifically for the task of image-to-image translation from satellite images to maps. The generator employs a U-net-like architecture with skip connections to preserve spatial information during upsampling, while the discriminator network is designed to effectively differentiate between real and generated images.

2. **Conditional GAN Training**: The project employs conditional adversarial training, where both the generator and discriminator networks are conditioned on input satellite images. This conditioning enhances the contextual awareness of the model, allowing it to generate maps that correspond accurately to the input satellite images.

3. **Advanced Loss Functions**: The project utilizes a combination of adversarial loss and L1 loss for training the generator network. This hybrid loss function encourages the generator to produce realistic maps while also minimizing pixel-wise differences between the generated and ground truth maps, resulting in visually compelling outputs.

4. **Comparision** :Here in this project we compare the results from training the generator with various Loss functions.The comparision is done between the models with generator loss function as L1 ,GAN loss and L1 + GAN loss and we saw significant improvement in L1+GAN loss compared to other models.

5. **Efficient Training Pipeline**: The training pipeline is optimized for efficiency, utilizing TensorFlow's GradientTape for automatic differentiation and Adam optimizer with custom learning rates. Additionally, the training loop is implemented to efficiently process batches of data, speeding up the overall training process.

6. **Evaluation and Visualization**: The project includes mechanisms for evaluating and visualizing the performance of the trained model. This includes randomly sampling input satellite images, generating corresponding maps, and comparing them against ground truth maps to assess the quality of the generated outputs.

4. METHODOLOGY

4.1 Data Collection:

The data used in the research conducted by the Berkeley AI Research (BAIR) Laboratory at UC Berkeley, led by Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, and Alexei A. Efros, was collected by scraping information from Google Maps. This dataset served as the foundation for training the image-to-image translation model, enabling the synthesis of maps and aerial photos[12].

4.2. Data Preprocessing:

Enhancing image quality, correcting distortions, and removing artifacts.Techniques include geometric correction, color balancing, and noise reduction.

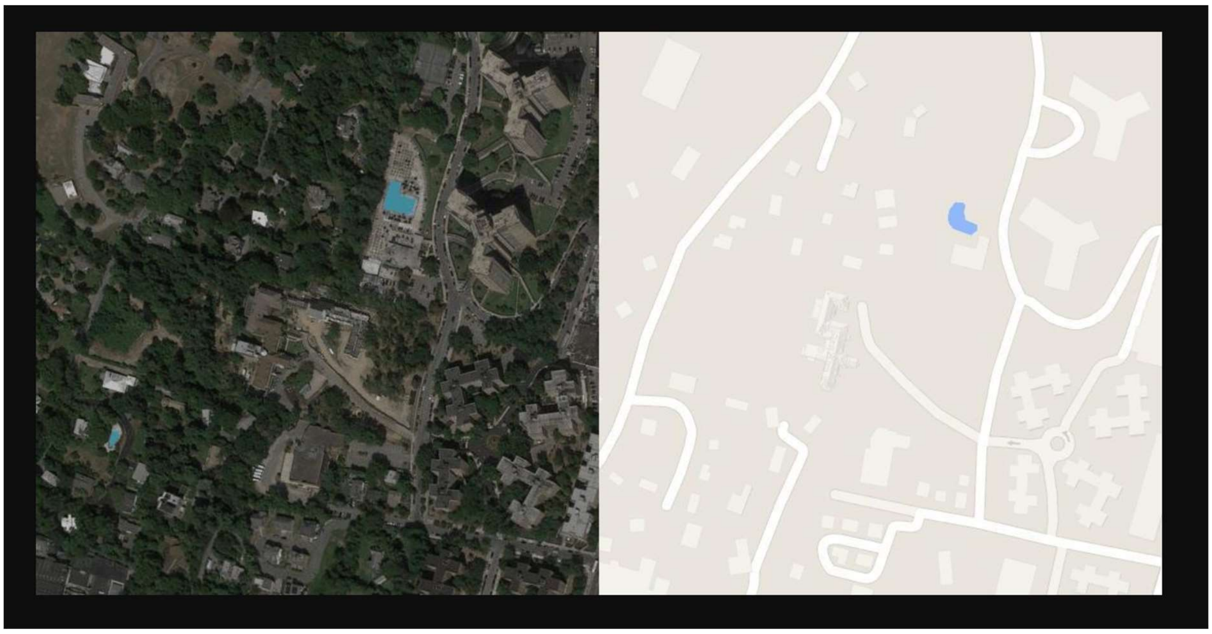


Figure 4.1 A datapoint consisting of Satellite and Road View map.

4.3 Model Architecture :

4.3.1 Generative Adversial Network (GAN) :

Generative Adversarial Networks (GANs)[11] represent a revolutionary framework in artificial intelligence, introduced by Ian Goodfellow and his colleagues in 2014. GANs consist of two neural networks, the generator and the discriminator, engaged in a competitive game.

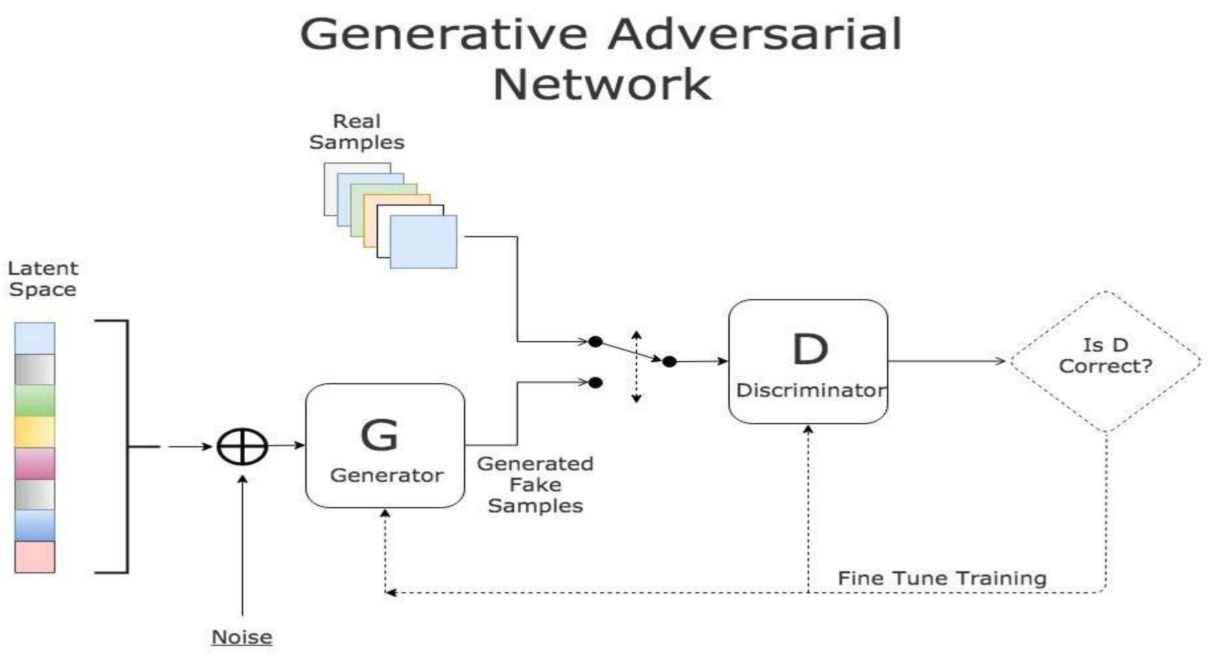


Figure 4.2 Archiecture of GAN [10]

Generator: The generator network takes random noise as input and attempts to generate synthetic data samples that resemble real data from a given distribution.

Discriminator: The discriminator network, on the other hand, learns to distinguish between real data samples and those generated by the generator. It receives both real and fake data samples as input and produces a binary classification indicating whether each sample is real or fake.

Adversarial Training: GANs are trained through an adversarial process where the generator and discriminator networks compete against each other. The generator aims to produce increasingly realistic samples to fool the discriminator, while the discriminator strives to correctly classify real and fake samples.

Loss Function: The standard GAN loss function, also known as the **min-max loss,** was first described in a 2014 paper by Ian Goodfellow et al., titled "Generative Adversarial Networks"[5].

$\min_G \max_D V(D,G) = E_{x\sim pdata(x)}[\log D(x)] + E_{z\sim pdata(z)}[\log(1- D(G(z)))]$

D(x) represents the output of the discriminator when input is real data.

D(G(z)) represents the output of the discriminator when input is generated image of generator.

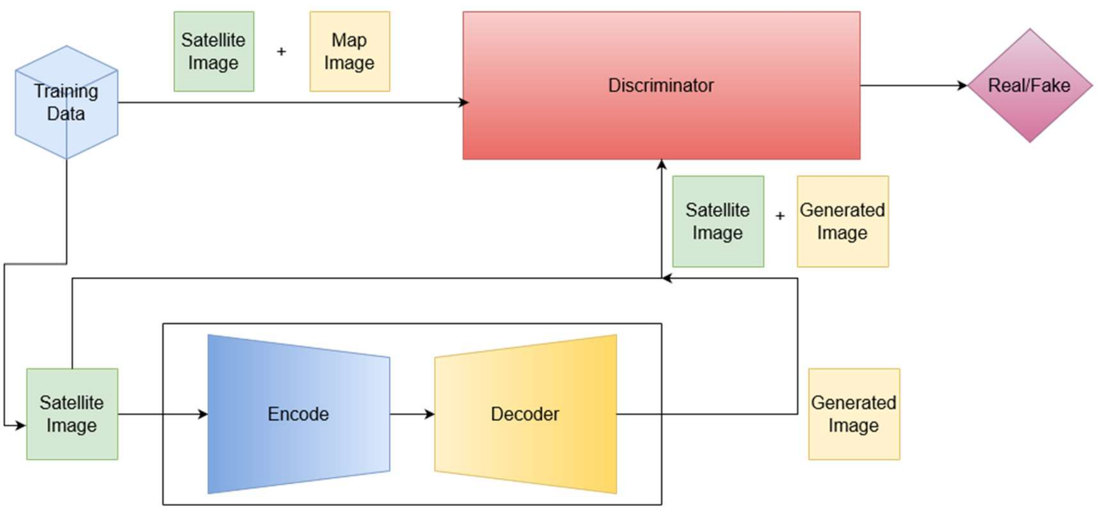


Fig 4.3 complete architecture of pix2pix GAN.

4.3.2 Generator Architecture

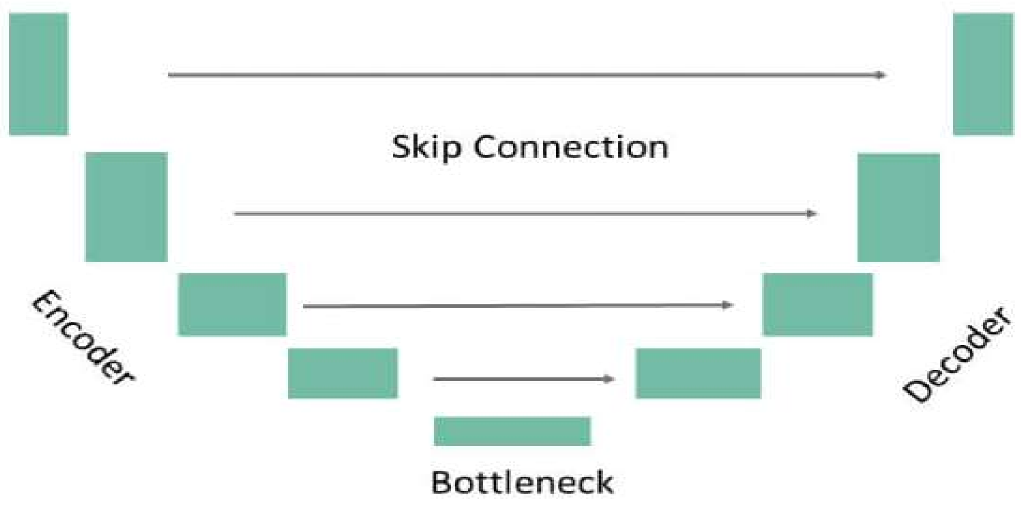


Figure 4.4 A sample U- NET Architecture used as Generator in GAN [11]

1. **Input Layer:** The Generator takes an input image tensor of shape (256, 256, 3), representing a satellite image. This input tensor serves as the starting point for the generative process.
2. **Encoder:** The encoder part of the Generator consists of several downscaling blocks, each reducing the spatial dimensions of the input tensor while increasing the number of filters (or channels). These downscaling blocks use convolutional layers with Leaky ReLU activation, batch normalization, and no bias terms.

3. **Latent Space:** After passing through the encoder blocks, the input tensor reaches the latent space representation. In this architecture, the latent space is further downsampled using another downscaling block, reducing the spatial dimensions while preserving the depth.

4. **Decoder:** The decoder section of the Generator comprises several upscaling blocks, each aimed at increasing the spatial dimensions of the latent space representation. These upscaling blocks use transposed convolutional layers (Conv2DTranspose) to perform the upsampling operation. Additionally, Leaky ReLU activation, batch normalization, and rectified linear unit (ReLU) activation are applied in each block.

5. **Skip Connections:** Throughout the encoding process, intermediate feature maps are stored in a list called 'skips.' These skip connections capture high-level spatial information from earlier layers, which is then utilized during the decoding phase.

6. **Concatenation:** In the decoding phase, each upscaling block receives both the upsampled latent space representation and the corresponding skip connection from the encoder. These skip connections are concatenated with the upsampled features to provide spatial context and detail.

7. **Output Layer:** Finally, the output layer of the Generator consists of a Conv2DTranspose layer with tanh activation, producing a three-channel tensor representing the synthesized map image. This output tensor represents the generator's prediction of the road view map corresponding to the input satellite image.

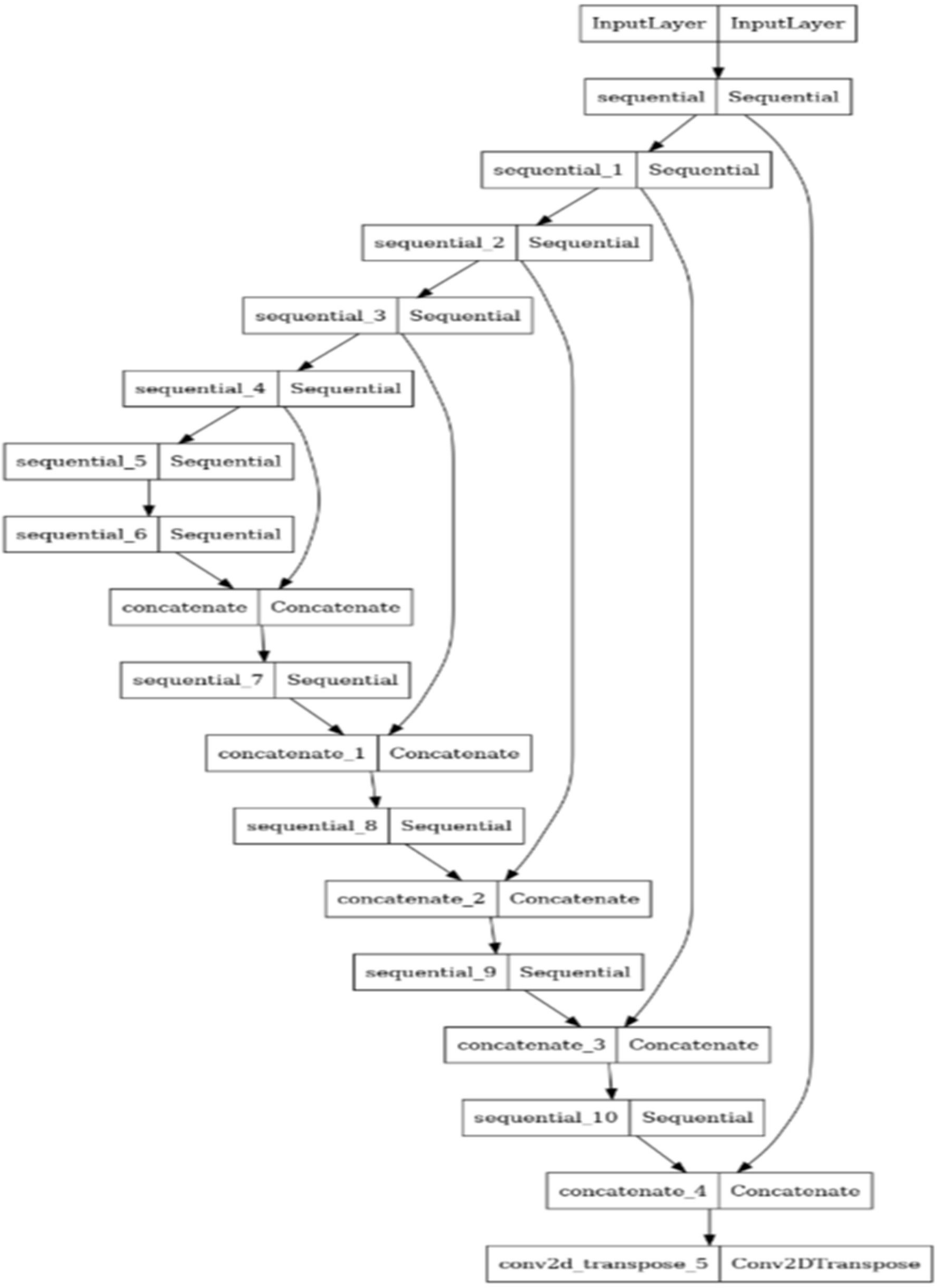

InputLayer | InputLayer
sequential | Sequential
sequential_1 | Sequential
sequential_2 | Sequential
sequential_3 | Sequential
sequential_4 | Sequential
sequential_5 | Sequential
sequential_6 | Sequential
concatenate | Concatenate
sequential_7 | Sequential
concatenate_1 | Concatenate
sequential_8 | Sequential
concatenate_2 | Concatenate
sequential_9 | Sequential
concatenate_3 | Concatenate
sequential_10 | Sequential
concatenate_4 | Concatenate
conv2d_transpose_5 | Conv2DTranspose

Figure 4.5 Actual Architecture U-NET Generator Implemented.

4.3.3 Discriminator Architecture (Patch GAN)

The "PatchGAN" architecture, at its core, is essentially a convolutional neural network (convnet). The fundamental principle underlying convnets is the processing of each image patch independently and identically, which offers computational efficiency in terms of parameters, time, and memory utilization. Remarkably, this approach has proven to be highly effective in various image processing tasks.

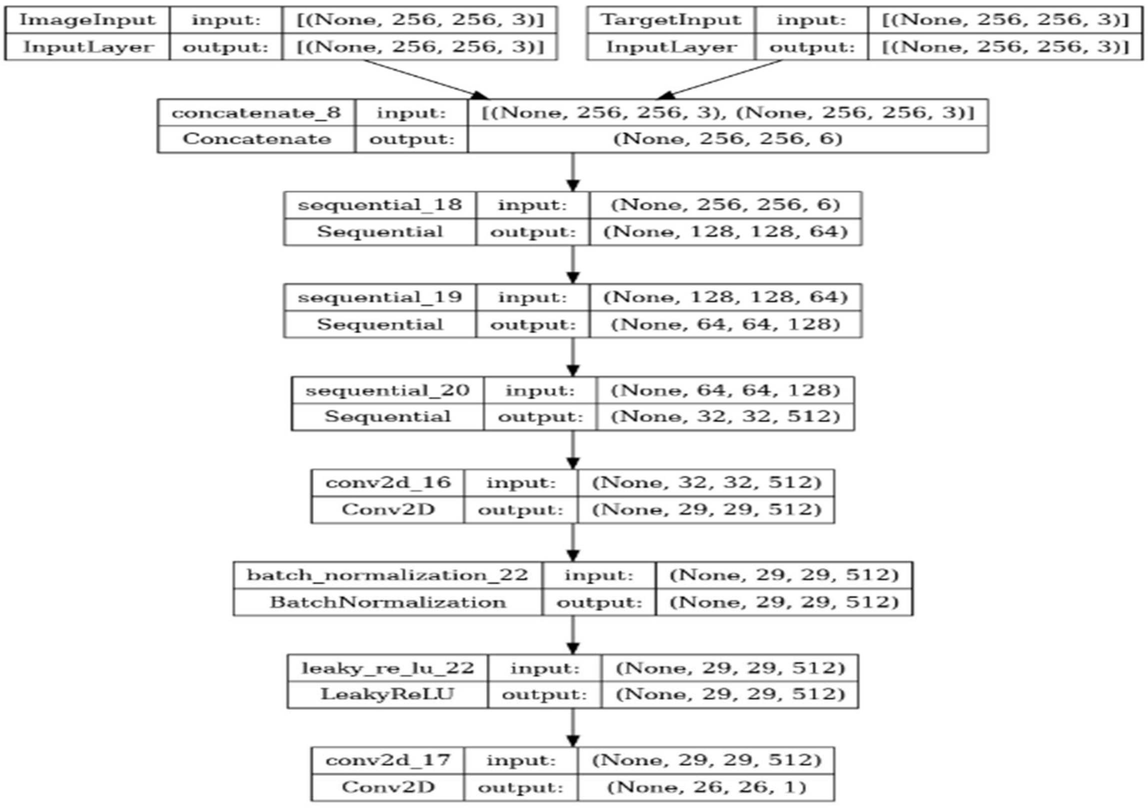
In the context of GANs, the key distinction between a PatchGAN and a regular GAN discriminator lies in their output representation. While a regular GAN discriminator maps a full 256x256 image to a single scalar output, indicating whether the input image is real or fake, a PatchGAN produces an NxN array of outputs, denoted as X. Here, each element X_{ij} in the output array signifies whether the corresponding patch ij in the input image is real or fake.

1. **Input Layers:** The Discriminator takes two input images: the satellite image (image) and its corresponding road view map (target), both of size (256, 256, 3). These images are concatenated along the channel axis to form a single input tensor.
2. **Concatenation:** The concatenated input tensor combines the satellite image and the target road view map, providing the Discriminator with information about both the input image and its ground truth.
3. **Downscaling Layers:** The concatenated input tensor undergoes a series of downscaling operations using convolutional layers with Leaky ReLU . These layers progressively reduce the spatial dimensions of the input tensor while increasing the number of filters, extracting hierarchical features from the input images.
4. **Convolutional Layers:** After downscaling, the feature representation is further processed by a couple of convolutional layers. These layers use a kernel size of 4x4 and apply a stride of 1 to maintain the spatial dimensions while increasing the depth of the feature maps.
5. **Batch Normalization:** Batch normalization is applied after each convolutional layer to stabilize the training and improve the generalization of the model.

6. **Leaky ReLU Activation:** The Leaky ReLU activation function is used to introduce non-linearity into the network, allowing the Discriminator to capture complex patterns and features in the input images.

7. **Output Layer:** The final output layer consists of a convolutional layer with a kernel size of 4x4 and a single filter, producing a single-channel output tensor. This output represents the Discriminator's prediction of whether the input image and its corresponding target map are real or fake.

8. **Patch GAN:** The Discriminator architecture follows the Patch GAN approach, where instead of producing a single output for the entire input image, it generates multiple outputs (patches) by applying convolutional filters across the spatial dimensions. This allows the Discriminator to make fine-grained predictions at the patch level, enabling more detailed and localized discrimination between real and fake images.



ImageInput | input: | [(None, 256, 256, 3)]
InputLayer | output: | [(None, 256, 256, 3)]
TargetInput | input: | [(None, 256, 256, 3)]
InputLayer | output: | [(None, 256, 256, 3)]
concatenate_8 | input: | [(None, 256, 256, 3), (None, 256, 256, 3)]
Concatenate | output: | (None, 256, 256, 6)
sequential_18 | input: | (None, 256, 256, 6)
Sequential | output: | (None, 128, 128, 64)
sequential_19 | input: | (None, 128, 128, 64)
Sequential | output: | (None, 64, 64, 128)
sequential_20 | input: | (None, 64, 64, 128)
Sequential | output: | (None, 32, 32, 512)
conv2d_16 | input: | (None, 32, 32, 512)
Conv2D | output: | (None, 29, 29, 512)
batch_normalization_22 | input: | (None, 29, 29, 512)
BatchNormalization | output: | (None, 29, 29, 512)
leaky_re_lu_22 | input: | (None, 29, 29, 512)
LeakyReLU | output: | (None, 29, 29, 512)
conv2d_17 | input: | (None, 29, 29, 512)
Conv2D | output: | (None, 26, 26, 1)

Figure 4.6 Actual Architecture Patch GAN Discriminator Implemented

4.4 Loss Functions and Optimizers

4.4.1 Generator Loss

1. Adversarial Loss (GAN Loss) using BCE:

   - Let $D(G(z|y))$ denote the discriminator's output for generated images, where $G(z|y)$ represents the generated output by the generator.
   - The binary cross-entropy loss for the adversarial loss (GAN loss) is given by

     GAN Loss = BinaryCrossEntropy(1 , $D(G(z|y))$)

     GAN Loss = $\min_G V(D,G) = E_{Z\sim pdata(z)}[-\log(D(G(z|y)))]$

   - Here, we use the binary cross-entropy function directly, passing in the target label 1 (indicating "real").

2. L1 Loss:

   - Let I denote the target (ground truth) image.
   - The L1 loss measures the mean absolute difference between the generated output $G(z|y)$ and the target image I, as described previously.

3. Total Loss:

   - The total generator loss combines the adversarial loss and the scaled L1 loss: **Total Loss= 100×L1 Loss + GAN Loss**
   - The L1 loss is scaled by a factor of 100 to balance its contribution to the total loss.

4.4.2 Discriminator Loss

1. Real Loss:

   - Let $D(x|y)$ denote the discriminator's output for real images x.
   - The binary cross-entropy loss for the real images is calculated as

     Real Loss = BinaryCrossEntropy(1 , $D(x|y)$)

     Real Loss = $E_{x\sim pdata(x)}[-\log D(x|y)]$

Here, we compare the discriminator's output for real images with the target label 1 (indicating "real").

2. Fake Loss:

   - Let $D(G(z|y))$ denote the discriminator's output for generated images $G(z|y)$, where $G(z|y)$ represents the generated output by the generator.
   - The binary cross-entropy loss for the generated images is calculated as:

     Fake Loss = BinaryCrossEntropy(0 , $D(G(z|y))$)

     Fake Loss = $E_{z\sim pdata(z)}[-\log(1- D(G(z|y)))]$

   - Here, we compare the discriminator's output for generated images with the target label 0 (indicating "fake").

3. Total Loss:

   - The total discriminator loss is the sum of the real loss and the fake loss:

     Total Loss = Real Loss + Fake Loss

     Total Loss = $E_{x\sim pdata(x)}[-\log D(x|y)] + E_{z\sim pdata(z)}[-\log(1- D(G(z|y)))]$

4.4.3 Optimizers

- Adam optimizer initialized for updating generator and discriminator parameters.
- Adaptive learning rate method, popular for neural network training.
- Learning rate set to 2e-4 (0.0002) and beta_1 parameter to 0.5.

4.5 Training

1. **Initialization:**

   - Create instances of the Generator and Discriminator classes.
   - Initialize the Adam optimizer for both the Generator and Discriminator.
   - Set up the Binary Cross-Entropy Loss function for the Discriminator.

2. **Training Loop:**

   - Iterate over the dataset containing pairs of satellite images and their corresponding map images.

3. **Discriminator Training:**

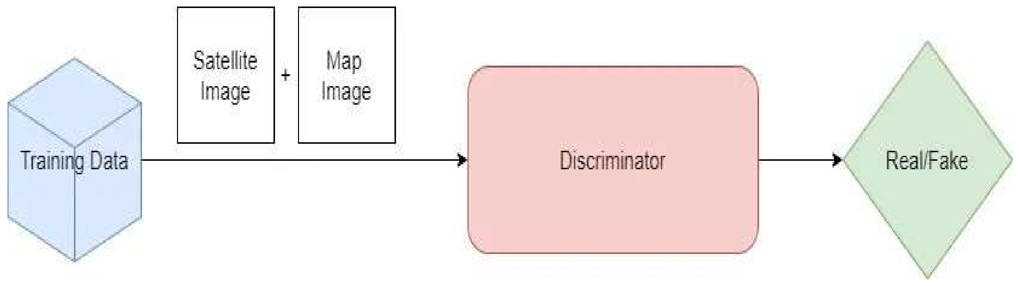


Figure 4.7 Discriminator Training with Real Data

   - Pass real concatenated images (satellite images and their corresponding real map images) through the Discriminator and calculate the Binary Cross-Entropy Loss.
   - Pass fake concatenated images (satellite images and generated map images) through the Discriminator and calculate the Binary Cross-Entropy Loss.
   - Compute the total loss for the Discriminator, which is the sum of the losses calculated for real and fake data.
   - Backpropagate this total loss through the Discriminator and update its weights using the optimizer.

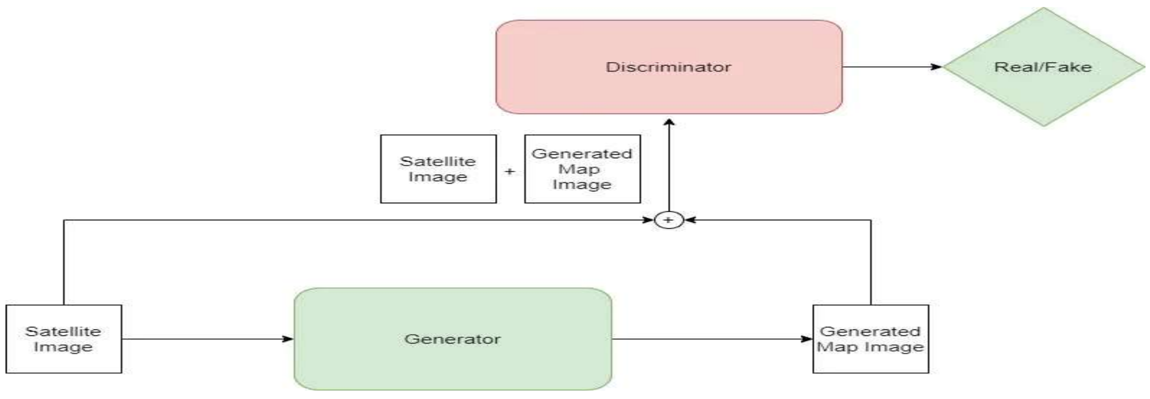


Figure 4.8 Discriminator Training with Fake Data

4. Generator Training:

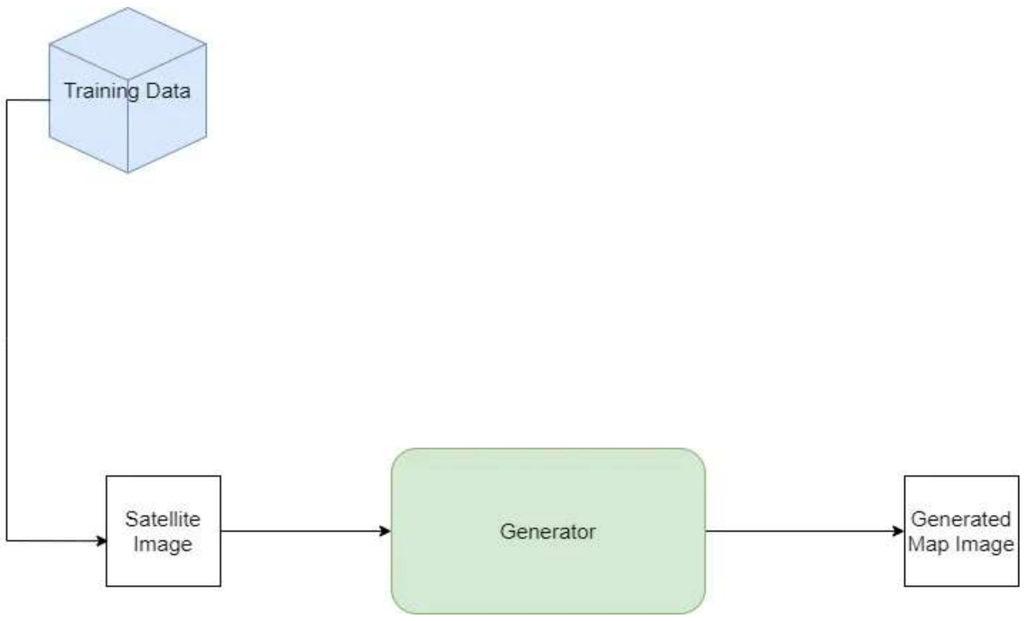


Figure 4.9 Generator Training to generate and deceive Discriminator.

- Pass a satellite image through the Generator to produce a generated map image.
- Calculate the Binary Cross-Entropy Loss for the Discriminator using the generated map image (considered as real) and a concatenated input of the real satellite image and its corresponding real map image.
- Compute the L1 Loss between the generated map image and the actual map image.
- Calculate the total loss for the Generator, which is the sum of the adversarial loss and the L1 loss.
- Backpropagate this total loss through the Generator and update its weights using the optimizer.

5. Repeat:

- Repeat steps 2-4 for a specified number of training epochs or until convergence.

5 RESULTS AND DISCUSSION

1.Visual Inspection: Display the generated images alongside the corresponding ground truth images (satellite images and map images). Used the show_predictions function to visualize the generated images.

2.Comparison Metrics: Utilized quantitative metrics to compare the generated images with the ground truth images. Common metrics for image generation tasks include:

- Structural Similarity Index (SSIM)
- Peak Signal-to-Noise Ratio (PSNR)
- Mean Squared Error (MSE)
- Root Mean Squared Error(RMSE)
- Mean Absolute Error (MAE)

3.User Feedback: Gathered feedback from users or domain experts who can provide insights into the quality and relevance of the generated images for the intended application.

HYPERPARAMETERS	**ALGORITHM/VALUE**
L1 lambda	0,100
Optimizer	Adam
Epochs	100,150,200,300
Dropout rates	0.15,0.25,0.35
Batch sizes	8,16,32,64

Table 5.1 Table showing Hyperparameters.

We Trained our generator with just L1 loss , just GAN loss and with both L1 and GAN loss.We have seen significant improvement while using L1 + GAN loss for generator as compared with just one of them.And then used generator as U-NET with 4,5,6,7,8 layers of Encoder – Decoder Architecture.And then we trained out best performing model with 200,300 Epocs to again increase the performance of that model.

Variation of Performance Metrics with different Loss functions

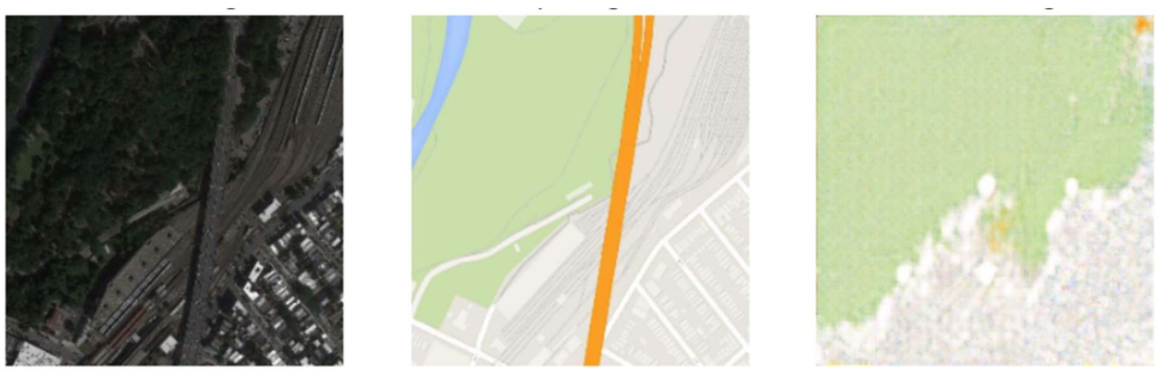


Fig 5.1 Generator having only GAN loss

Average MSE * 100	Average MAE * 100	Average RMSE * 100
1.145	6.2525	10.7007

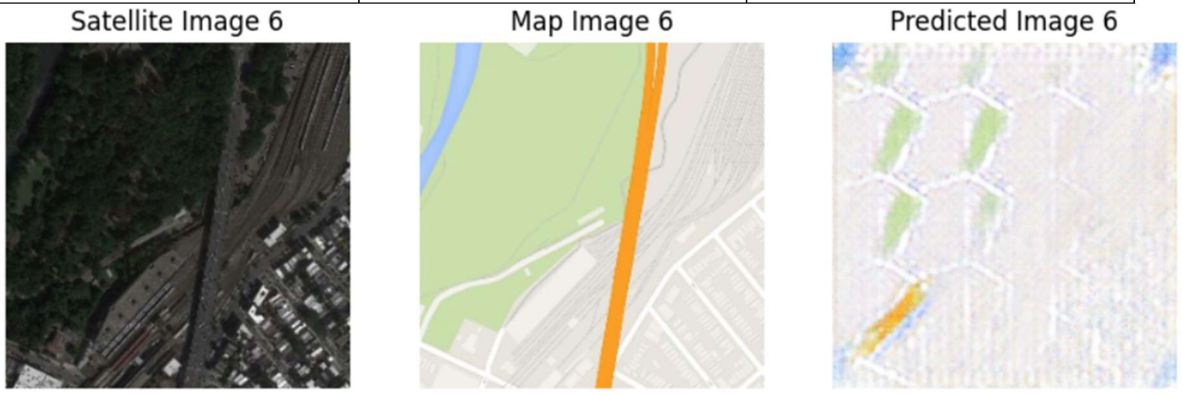


Fig 5.2 Generator having only L1 loss

Average MSE * 100	Average MAE * 100	Average RMSE * 100
1.6054	8.0799	12.6707

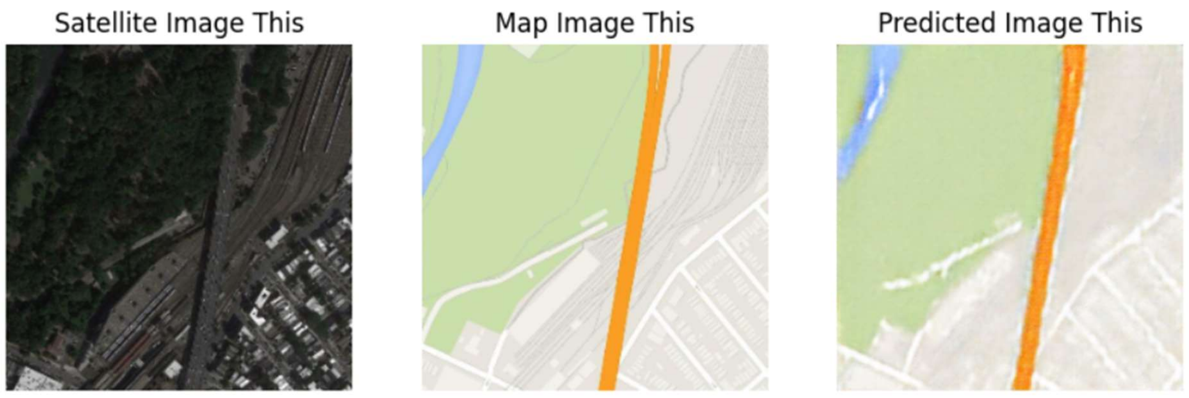


Fig 5.3 Generator having L1 + GAN loss

Average MSE * 100	Average MAE * 100	Average RMSE * 100
0.1687	2.5187	4.1084

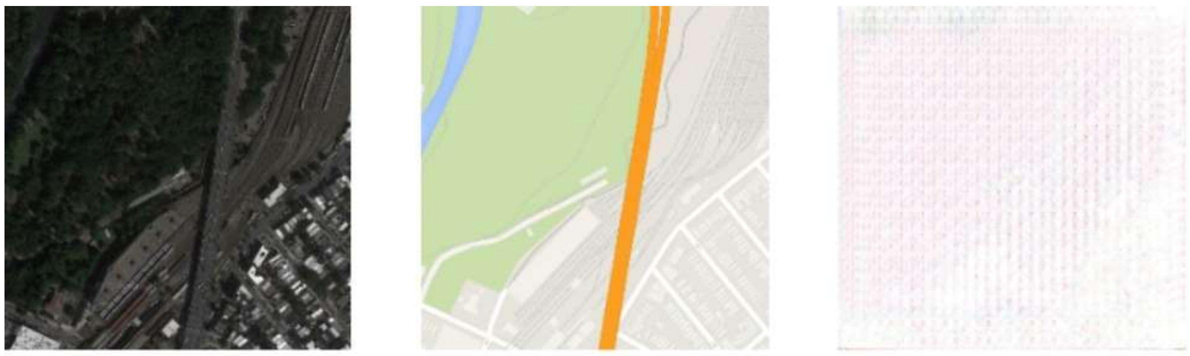


Fig 5.4 Prediction with Dice loss + GAN Loss in Generator

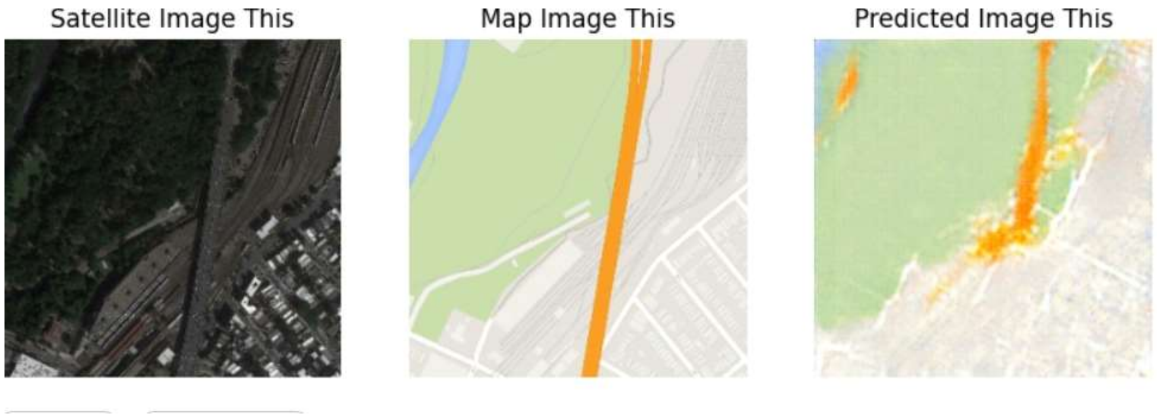


Fig 5.5 Prediction with Cross Entropy loss + GAN Loss in Generator.

Loss Function	**100 * MSE**	**100 * MAE**	**100 * RMSE**
Dice Loss	1.8603	9.9385	13.63
Cross Entropy Loss	0.7480	5.3358	8.6492

Table 5.5 Comparison metrics for Generator with different Loss Function.

Variation of Performance Metrics with Number of Layers in U-NET

Number of Layers in Generator (U-NET)	**100* MSE**	**100 * MAE**	**100 * RMSE**
4	0.7067	4.7260	8.4071
5	0.1841	2.6367	4.2913
6	0.1687	2.5187	4.1084
7	0.6690	5.1514	8.1797
8	0.1942	2.6870	4.4076

Table 5.6 variation of comparion metrics with number of layers in generator(U-NET)

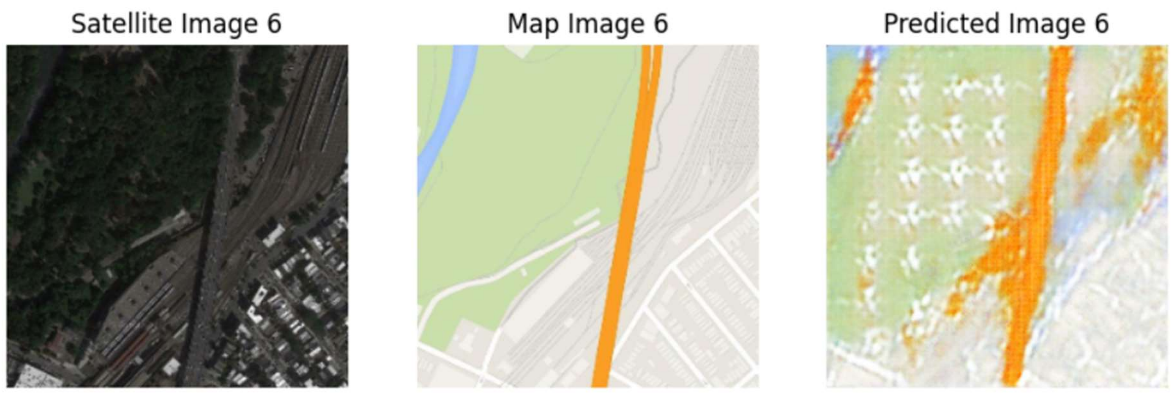


Fig 5.6 Generator(U-NET) with 4 layers.

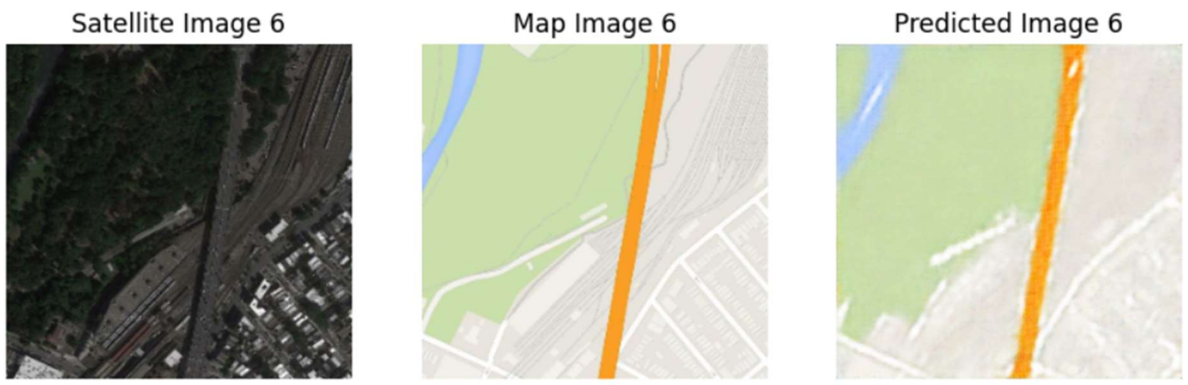


Fig 5.7 Generator(U-NET) with 5 layers.

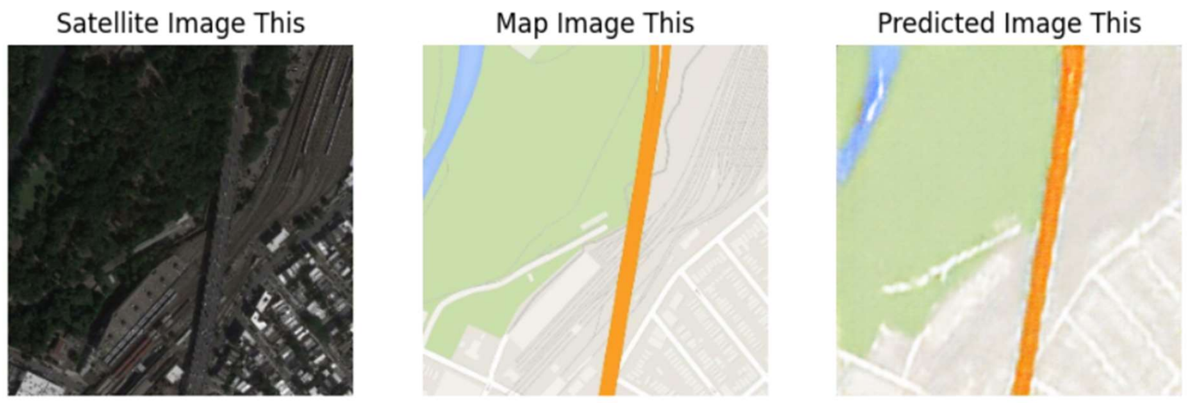


Fig 5.8 Generator(U-NET) with 6 layers.(100 Epochs)

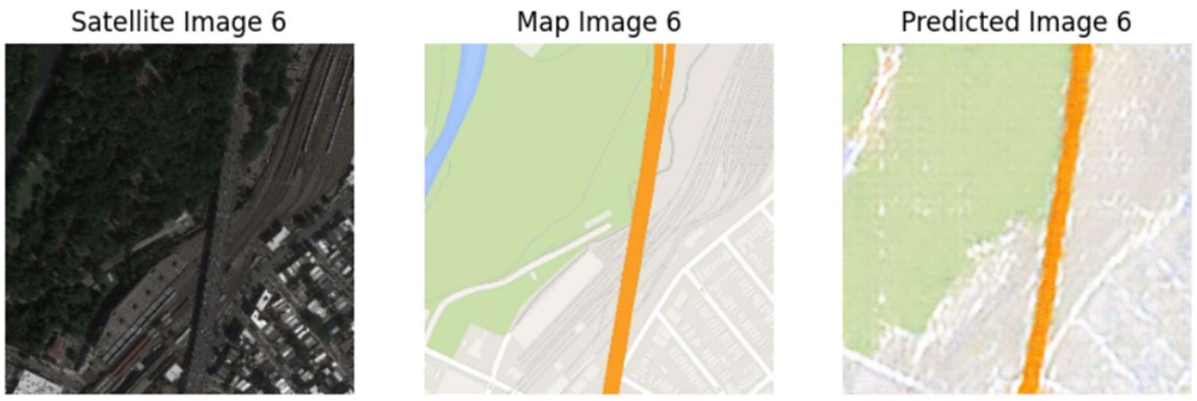


Fig 5.9 Generator(U-NET) with 7 layers.

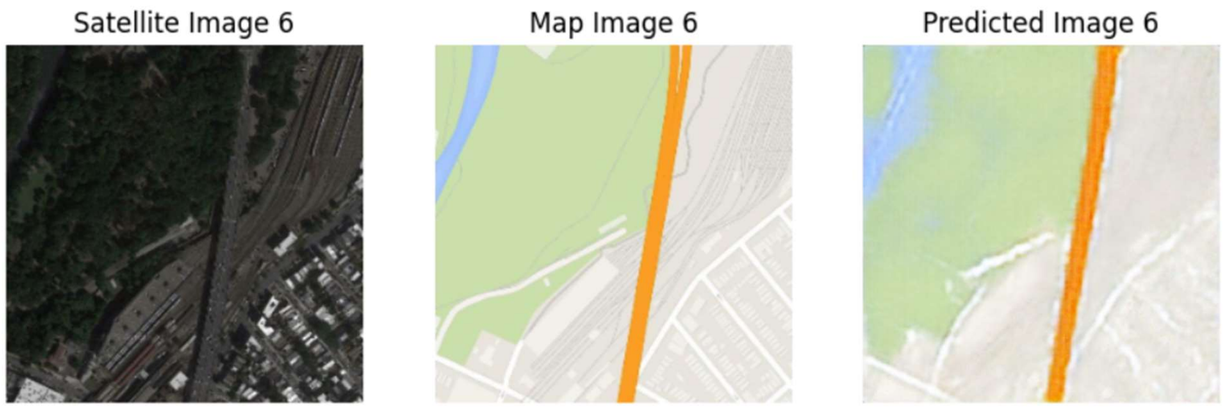


Fig 5.10 Generator(U-NET) with 8 layers.

By above results we conclude that the generator with 5 layers in its U – NET architecture outperforms all other models,

Variation of Performance Metrics with Number of Epochs in U-NET

we train best performing model for different epochs to obtain more promising results.

Number of Epochs	100* MSE	100 * MAE	100 * RMSE
100	0.1687	2.5187	4.1084
150	0.1956	2.6935	4.4228
200	0.1732	2.6235	4.1620
300	0.1937	2.7489	4.4014

Table 5.7 Variation of Comparison metrics Number of Epochs trained.

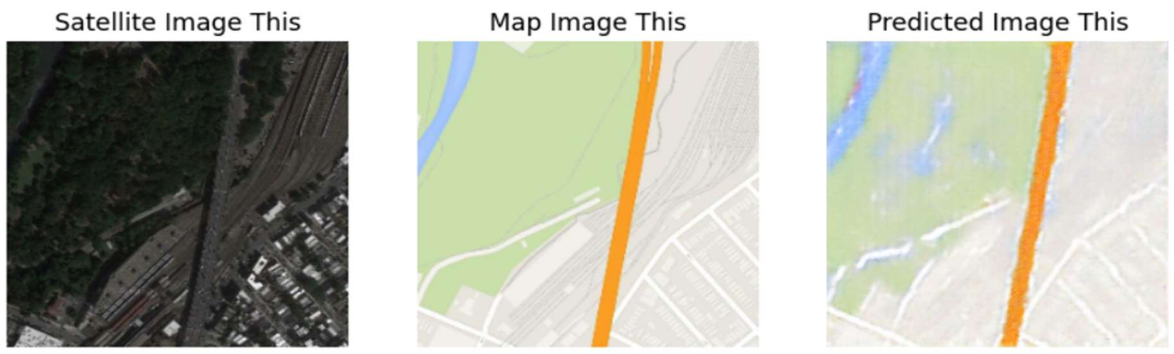


Fig 5.11 Generator with 6 layers trained for 150 Epochs.

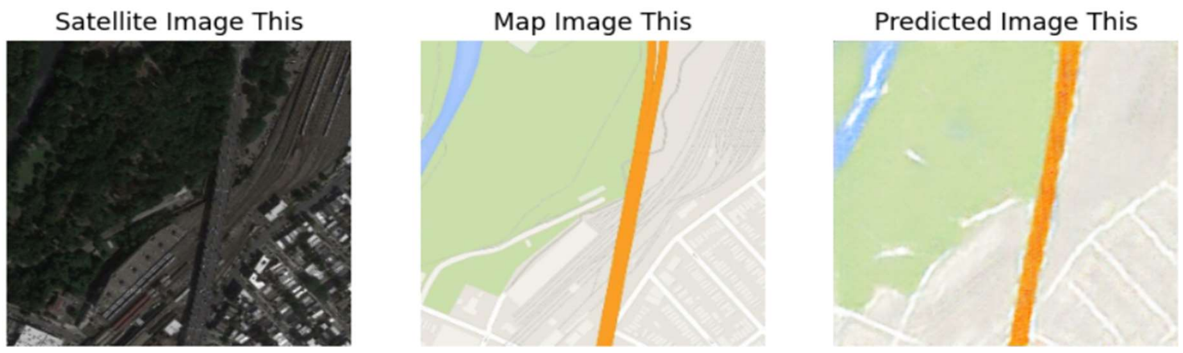


Fig 5.12 Generator with 6 layers trained for 200 Epochs.

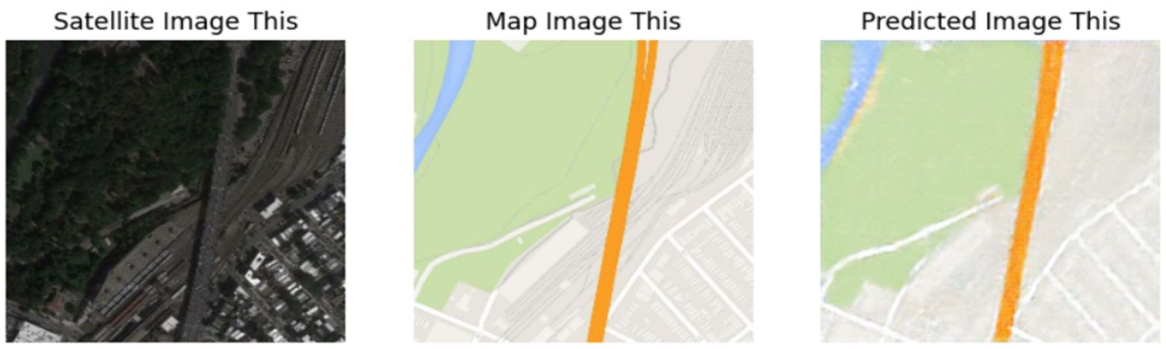


Fig 5.13 Generator with 6 layers trained for 300 Epochs.

Variation of Performance Metrics with Dropout rates in U-NET

Dropout Rate	**100 * MSE**	**100 * MAE**	**100 * RMSE**
0	0.1687	2.5187	4.1084
0.15	0.2490	2.8465	4.9901
0.25	0.3825	3.7288	6.1852
0.35	0.4415	3.7942	6.6448

Table 5.8 Comparison metrics for different dropout rates.

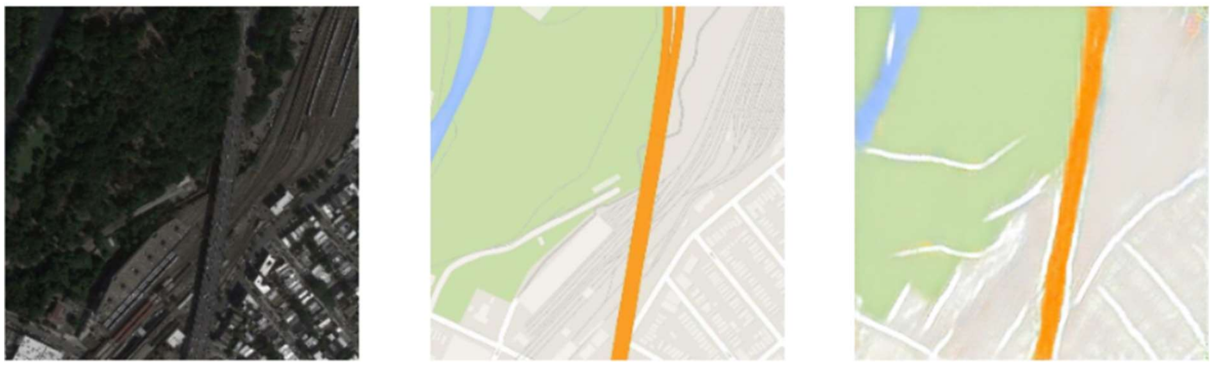


Fig 5.14 Prediction for dropout rate of 0.15.

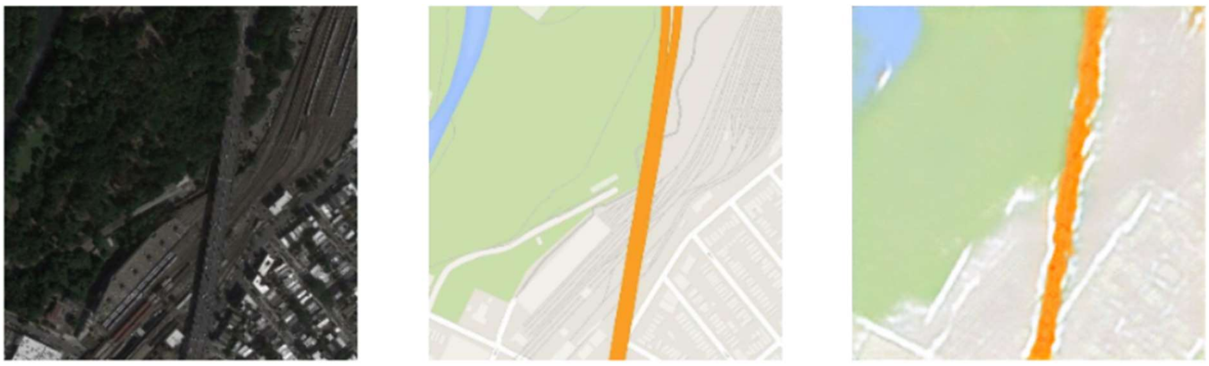


Fig 5.15 Prediction for dropout rate of 0.25.

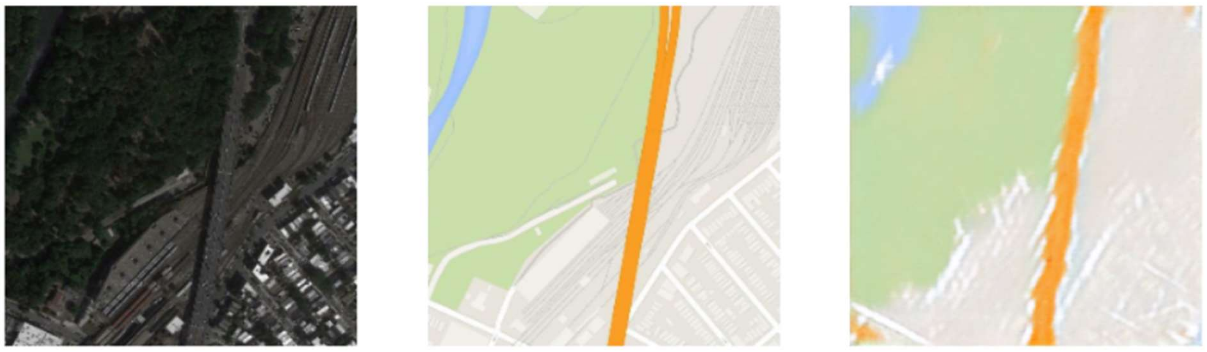


Fig 5.16 Prediction for dropout rate of 0.35.

Variation of Performance Metrics with Different Batch sizes

Batch size	**100 * MSE**	**100 * MAE**	**100 * RMSE**
8	0.3047	3.0956	5.5204
16	0.2638	5.2138	20.3140
32	0.3825	3.7288	6.1852
64	0.7081	4.6555	8.4152

Table 5.9 Comparison metrics for different Batch sizes.

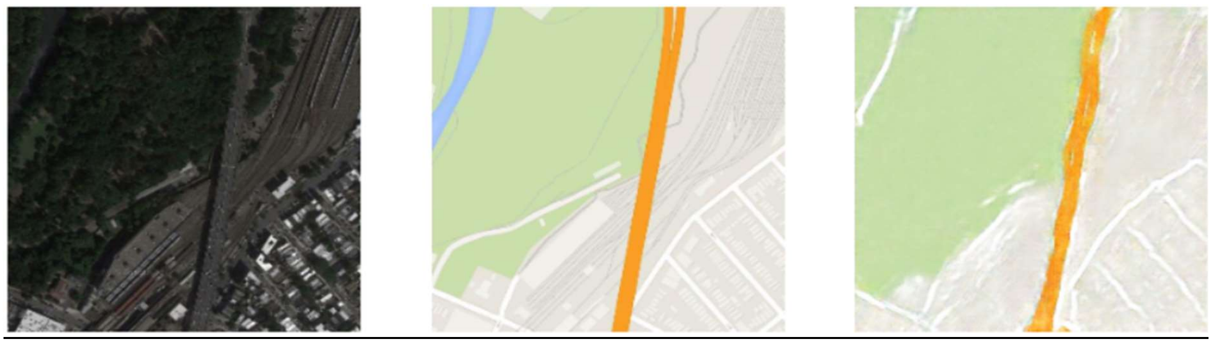


Fig 5.17 Prediction for Batch size of 8.

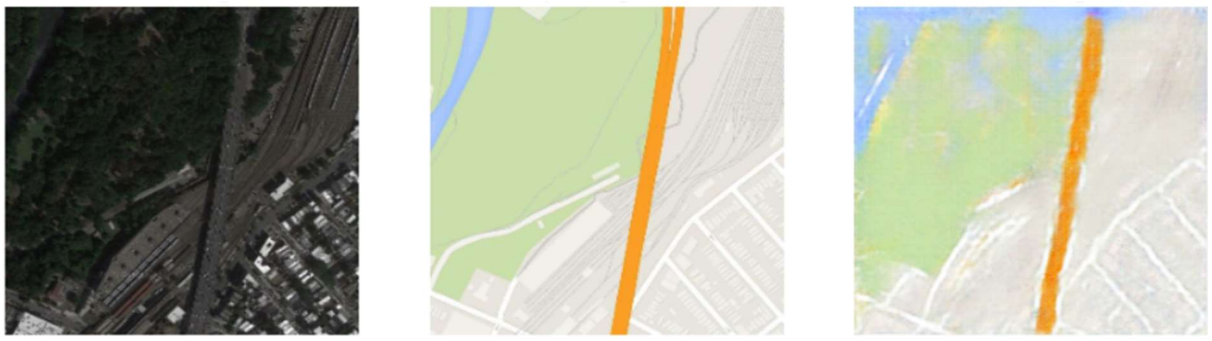


Fig 5.18 Prediction for Batch size of 16

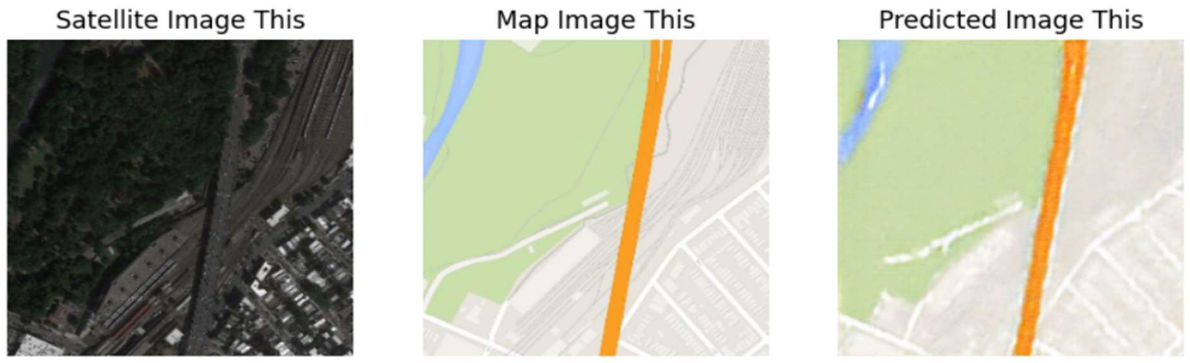


Fig 5.19 Prediction for Batch size of 32.

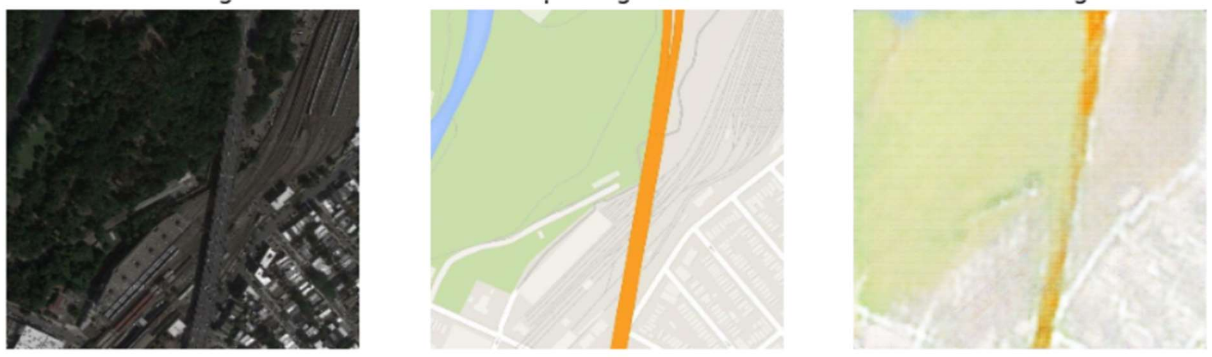


Fig 5.20 Prediction for Batch size of 64.

Variation of Performance Metrics with Residuals connections

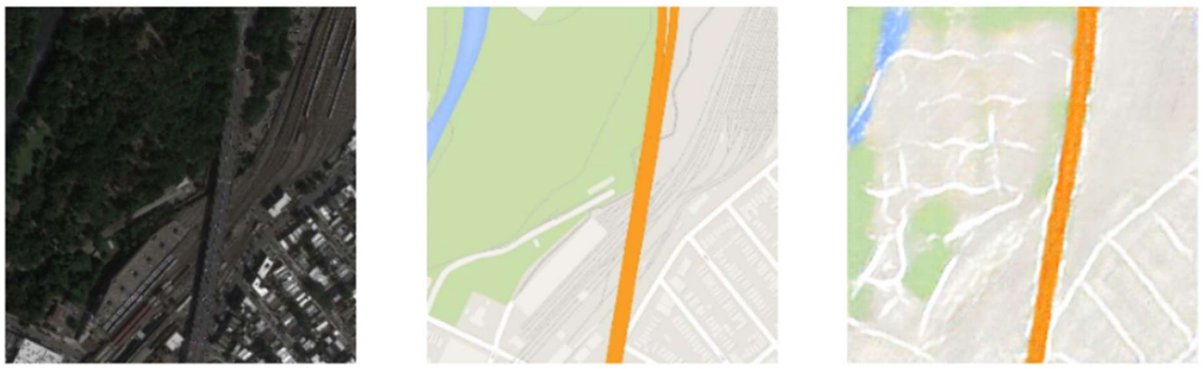


Fig 5.21 Prediction with 2 Residual connections and 16 Batch size.

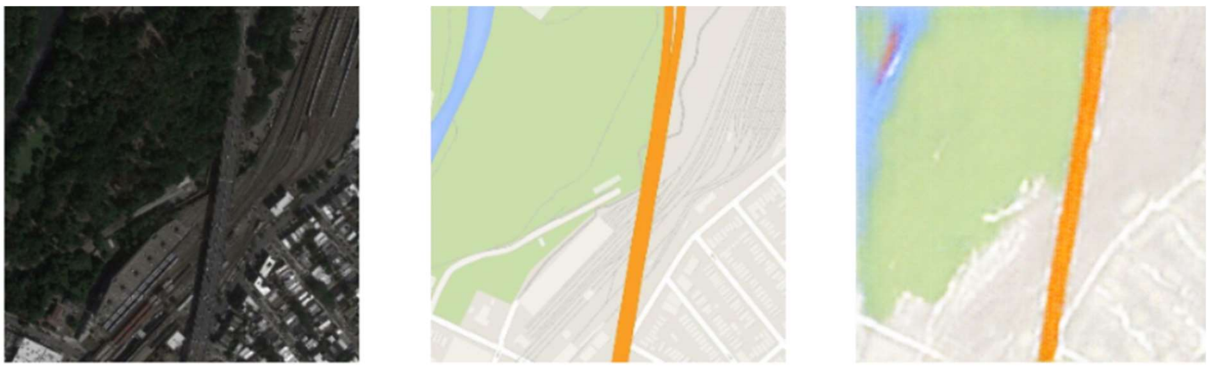


Fig 5.22 Prediction with 2 Residual connections and 32 Batch size.

Batch size	100 * MSE	100 * MAE	100 * RMSE
16	0.2000	2.6820	4.4726
32	0.2067	2.7672	4.5464

Table 5.10 Comparison metrics for Generator with 2 Residual connections.

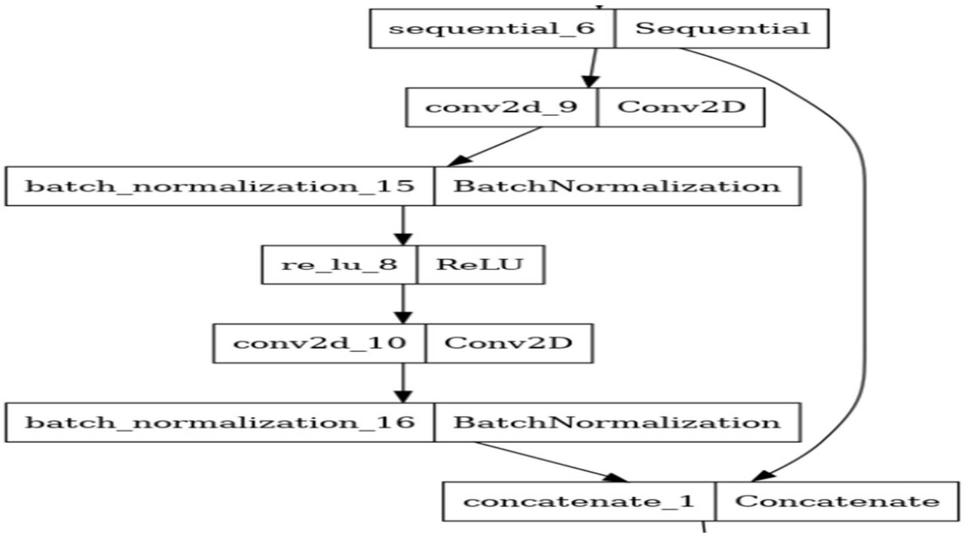

sequential_6 | Sequential
conv2d_9 | Conv2D
batch_normalization_15 | BatchNormalization
re_lu_8 | ReLU
conv2d_10 | Conv2D
batch_normalization_16 | BatchNormalization
concatenate_1 | Concatenate

Fig 5.23 Residual connection added between U – NET encoder and decoder.

Using stacked Unets connected in series as a Generator.

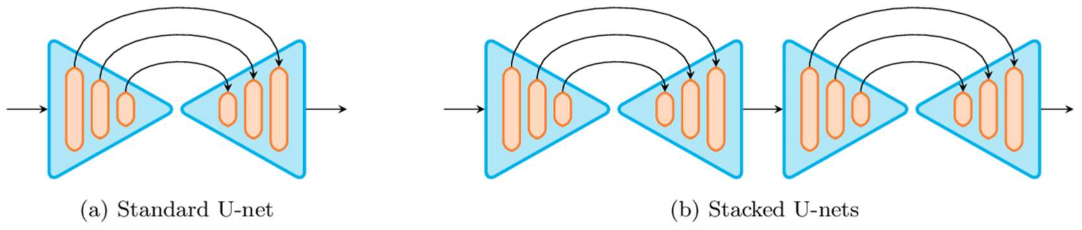


Fig 5.24 schematic representation of stacked u-net generator

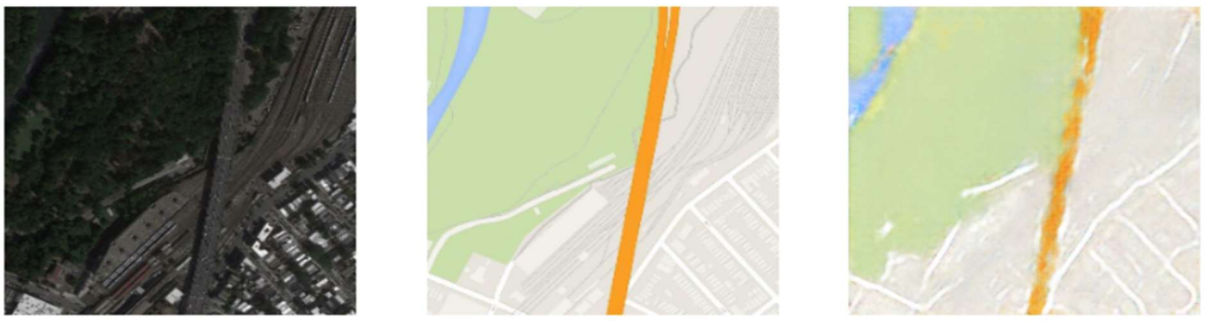


Fig 5.25 output when 2 u-nets is used as generator.

Average MSE * 100	Average MAE * 100	Average RMSE * 100
0.1687	2.5187	4.1084

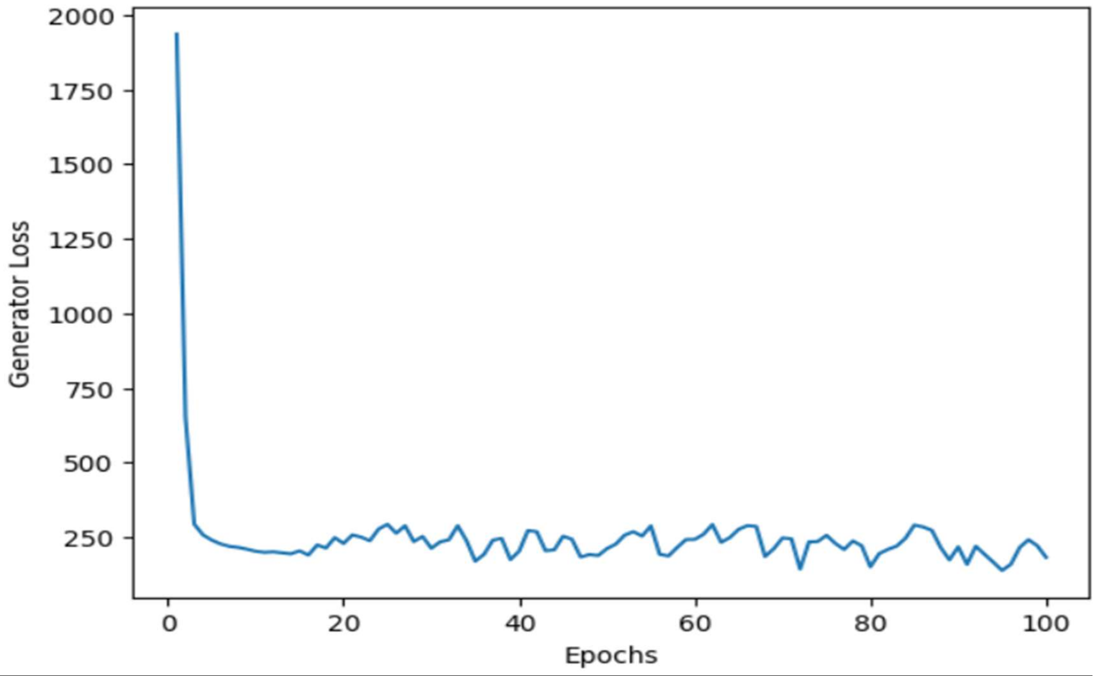


Fig 5.26 Variation of Generator Loss with Number of Epochs Trained.

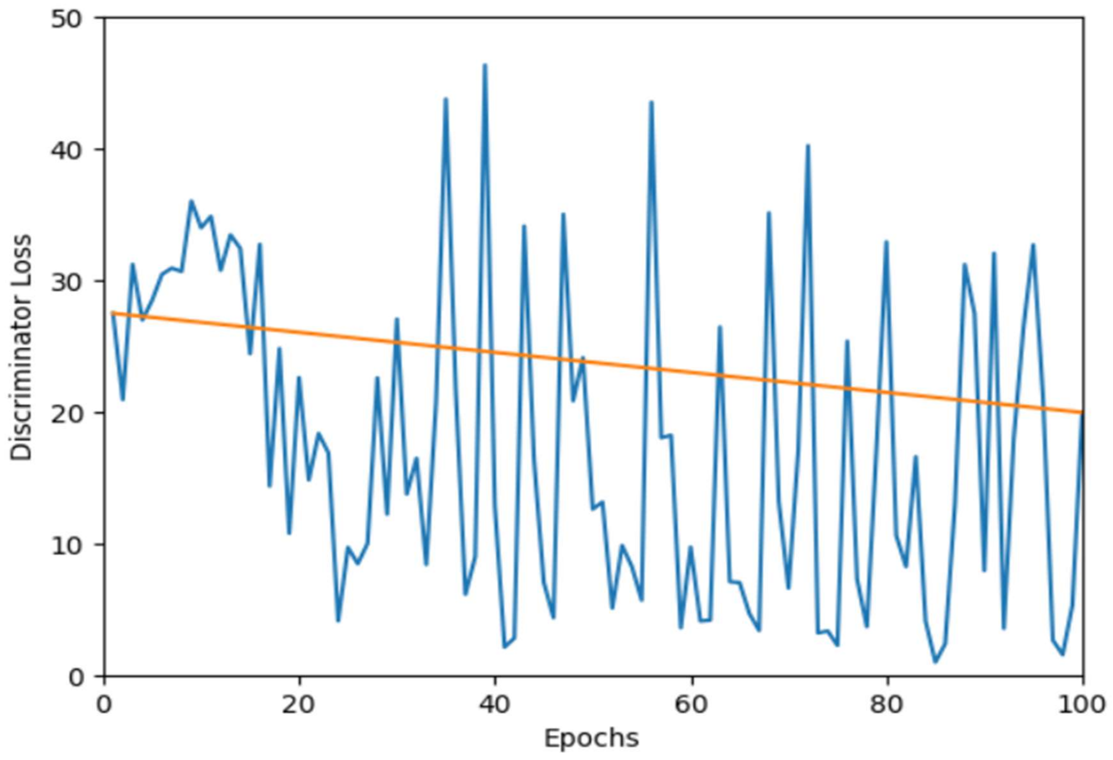


Fig 5.26 Variation of Discriminator Loss with Number of Epochs Trained.

<u>Road Extraction using Satellite maps</u>

UNet outperforms GANs in generating road views from satellite maps due to its direct supervision, simplicity, and effectiveness in semantic segmentation tasks. The supervised learning approach of UNet, where each input image is paired with a corresponding ground truth label, provides clear guidance during training. Additionally, UNet's architecture is tailored for pixel-wise labeling, allowing it to learn intricate details in road extraction more effectively. Conversely, GANs require unpaired data and more complex training regimes, making them less suitable for specific characteristic generation tasks like road extraction. While GANs offer powerful capabilities for image translation tasks, such as style transfer or image-to-image translation, UNet's straightforward approach makes it the preferred choice for tasks demanding precise pixel-wise predictions, like road segmentation from satellite imagery.

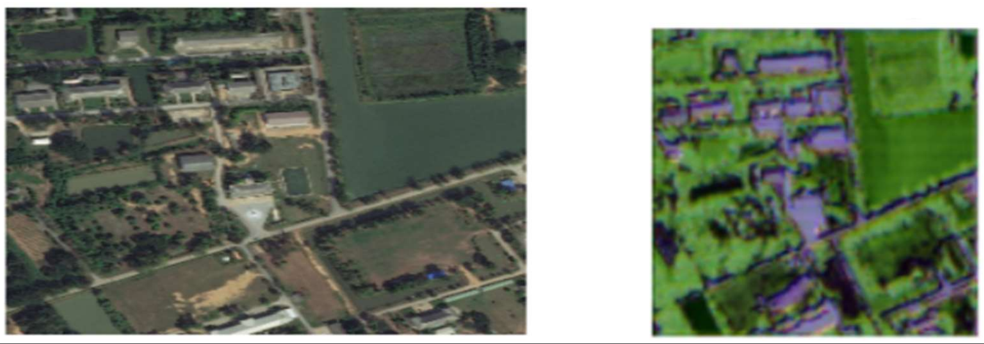


Fig 5.15 Random sample Prediction of Routes using GAN.

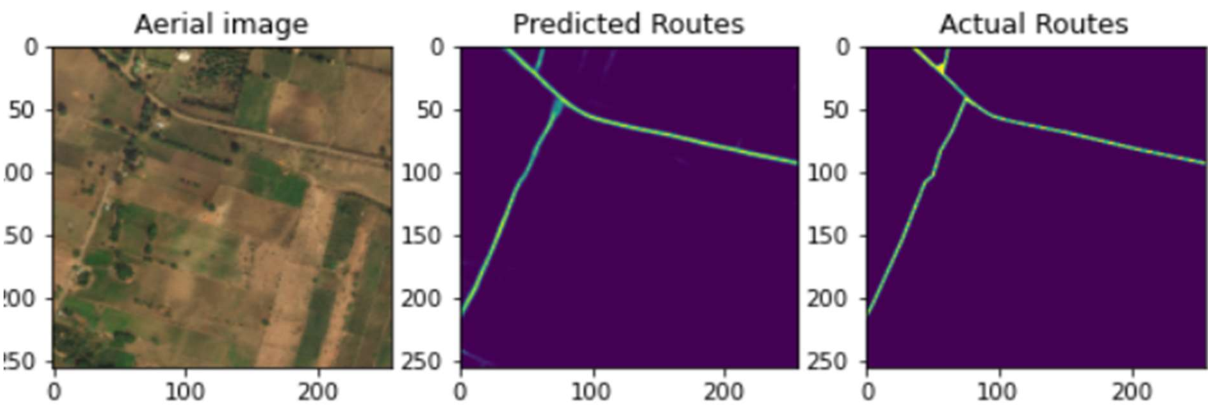


Fig 5.16 Random sample Prediction of Routes using U-NET.

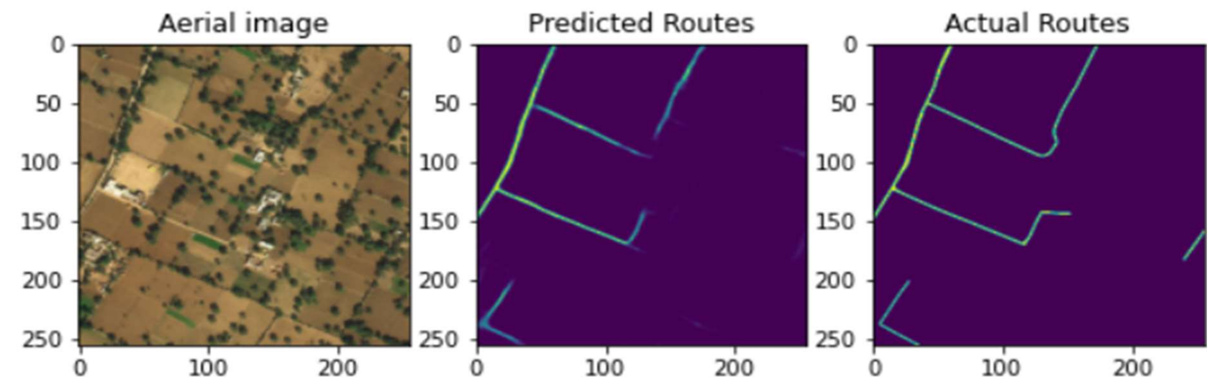


Fig 5.17 Random sample Prediction of Routes using U-NET.

6 CONCLUSION

1.Experimented with Different Loss Functions: Explored various ways of measuring the model's errors, including L1 Loss, GAN Loss, and their combination, to find which worked best for our task.

2.Tuned Dropout Rates: Adjusted the dropout rates during training to control how much the model ignores certain information, testing rates from 0 to 0.35.

3.Varied Batch Sizes: Tried different batch sizes, which are how many pictures the model sees at once during training, ranging from 8 to 64.

4.Explored Different Numbers of Layers: Tested models with different numbers of layers to see how it affected performance, trying 4, 5, 6, 7, and 8 layers.

5.Trained for Different Epochs: Trained the model for different numbers of epochs, or training cycles, including 100, 150, and 200 epochs, to see how much training time affected results.

6.Utilized Residual Connections: Incorporated residual connections between layers to improve information flow and gradient propagation.

7.Implemented Stacked UNETs: Experimented with using multiple UNETs stacked together to enhance the model's capacity to capture features at different scales.

8.Selected Optimal Configuration: After comprehensive experimentation, identified the best-performing model configuration: 6 layers, no dropout, a batch size of 32, trained for 100 epochs, using a combination of L1 Loss and GAN Loss.

Comparision of Results :

MSE comparison for models of different Layes

7 – layered(suggested by [12]	6 – layered
0.6690	0.1687

MAE comparison for models of different Layes

7 – layered(suggested by [12]	6 – layered
5.1514	2.5187

RMSE comparison for models of different Layes

7 – layered(suggested by [12]	6 – layered
8.1797	4.1084

MSE comparison with architectural modifications

Without residual connections	With residual connections
0.1687	0.2067

MAE comparison with architectural modifications

Without residual connections	With residual connections
2.5187	2.7672

RMSE comparison with architectural modifications

Without residual connections	With residual connections
4.1084	4.5464

7 FUTURE PLAN OF WORK

7.1 Experiment with Architectural Modifications:

•Explore architectural modifications or variations of the pix2pix GAN, such as U-Net variants, attention mechanisms, or residual connections, to enhance the model's capacity to capture spatial dependencies and long-range correlations.

7.2 Hyperparameter Tuning:

•Learning Rate: Experiment with different learning rates to find the optimal value that accelerates convergence without causing instability.

•Beta1: Adjust the momentum term beta1 in the Adam optimizer to control the exponential decay rates of the first moment estimates.

•Batch Size: Explore the impact of batch size on training dynamics and model performance.

•Image Size: Investigate the effect of image size on the quality of generated images and training stability.

•L1 Lambda: Fine-tune the weight of the L1 loss term in the total loss function to balance between adversarial and reconstruction losses.

•Lambda_GP: If applicable, experiment with the gradient penalty coefficient for improving the stability of Wasserstein GANs.

•Epochs: Determine the optimal number of training epochs to achieve convergence while avoiding overfitting.

•Network Depth and Width: Investigate the impact of varying the depth and width of the Generator and Discriminator networks on model capacity, convergence speed, and generalization performance.

•Activation Functions: Explore alternative activation functions such as Swish, Mish, or GELU, and assess their effects on training stability and image quality.

•Weight Initialization: Experiment with different weight initialization schemes (e.g., He initialization, Xavier initialization) to ensure proper initialization of network parameters and mitigate vanishing or exploding gradients.

•Dropout and Regularization: Evaluate the effectiveness of dropout regularization and other regularization techniques (e.g., L2 regularization) to prevent overfitting and improve model generalization.

•Batch Normalization: Analyze the impact of batch normalization on training dynamics and model convergence.

•Optimizer Variants: Compare the performance of different optimizer variants and learning rate schedulers to find the most effective optimization strategy for training.

•Learning Rate Decay: Implement learning rate decay schedules to dynamically adjust the learning rate during training and improve convergence stability.

6.3 Implement More Comparison Metrics:

•Explore additional image quality metrics such as Frechet Inception Distance (FID), Inception Score (IS), and perceptual similarity metrics to provide a comprehensive evaluation of generated images.

•Investigate domain-specific metrics that capture the relevance and fidelity of generated map images to satellite images.

6.4 Incorporate Transfer Learning:

•Explore the feasibility of leveraging pre-trained models or transfer learning techniques to accelerate convergence and improve generalization performance, especially when dealing with limited training data.

REFERENCES

[1] Y. LeCun, S. Chopra, R. Hadsell, M. Ranzato, and F. Huang, “A tutorial on energy-based learning,” Predicting structured data, vol. 1, no. 0, 2006.

[2] J. Xu, H. Li, and S. Zhou, “An overview of deep generative models,” IETE Technical Review, vol. 32, no. 2, pp. 131–139, 2015.

[3] A. Oussidi and A. Elhassouny, “Deep generative models: Survey,” in 2018 International Conference on Intelligent Systems and Computer Vision (ISCV). IEEE, 2018, pp. 1–8.

[4] H.-M. Chu, C.-K. Yeh, and Y.-C. Frank Wang, “Deep generative models for weakly-supervised multi-label classification,” in Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 400–415.

[5] R. A. Yeh, C. Chen, T. Yian Lim, A. G. Schwing, M. Hasegawa-Johnson, and M. N. Do, “Semantic image inpainting with deep generative models,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 5485–5493.

[6] M. Tschannen, E. Agustsson, and M. Lucic, “Deep generative models for distribution-preserving lossy compression,” in Advances in Neural Information Processing Systems, 2018, pp. 5929–5940.

[7] G. E. Hinton and R. R. Salakhutdinov, “Reducing the dimensionality of data with neural networks,” science, vol. 313, no. 5786, pp. 504–507, 2006.

[8] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel, “Infogan: Interpretable representation learning by information maximizing generative adversarial nets,” in Advances in neural information processing systems, 2016, pp. 2172–2180.

[9] Panguluri, Koumudi & Kamarajugadda, Kishore. (2020). “Image Generation using Variational Autoencoders”. IJITEE (International Journal of Information Technology and Electrical Engineering). 9. 10.35940/ijitee.E2480.039520.

[10] Goodfellow, Ian & Pouget-Abadie, Jean & Mirza, Mehdi & Xu, Bing & Warde-Farley, David & Ozair, Sherjil & Courville, Aaron & Bengio, Y.. (2014). Generative Adversarial Networks. Advances in Neural Information Processing Systems.

[11] Zheng, Ziyang & Chen, Zhixiang & Wang, Shuqi & Wang, Wenpeng. (2023). Dual-attention U-Net and multi-convolution network for single-image rain removal. The Visual Computer.

[12] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, & Alexei A. Efros. (2018). Image-to-Image Translation with Conditional Adversarial Networks.