**Satellite to Map Image Conversion**

FINAL PROJECT REPORT

***Submitted by:***

**Saloni Sharma -181B179**

**Stuti Verma-181B220**

**Trijal Singh-181B230**

***Under the guidance of :* Dr Partha Sarathy Banerjee**

****

May-2021

***Submitted in partial fulfilment for the award of the degree of***

**BACHELOR OF TECHNOLOGY**

**IN**

**COMPUTER SCIENCE AND ENGINEERING**

**Department of Computer Science & Engineering**

JAYPEE UNIVERSITY OF ENGINEERING & TECHNOLOGY, AB ROAD, RAGHOGARH, DT. GUNA-473226 MP, INDIA



**DECLARATION BY THE STUDENT**

I hereby declare that the work reported in the B. Tech 5th semester project entitled as “Satellite to Map Image Conversion”, in partial fulfilment for the award of degree of B. Tech submitted at Jaypee University of Engineering and Technology, Guna, as per best of my Knowledge and belief there is no infringement of intellectual property right and copyright. In case of any violation, I will solely be responsible.

**CERTIFICATE**

This is to certify that the work titled “***Satellite To Map Image Conversion***” submitted by ***Saloni Sharma (181B179), Stuti Verma (181B220), Trijal Singh (181B230)*** in partial fulfilment for the award of degree of B tech of Jaypee University of Engineering & Technology, Guna has been carried out under my supervision. As per best of my knowledge and belief there is no infringement of intellectual property right and copyright. Also, this work has not been submitted partially or wholly to any other Institute for the award of this or any other degree or diploma.

In case of any violation concern student will solely be responsible.

**Dr. Partha Sarathy Banerjee**

Professor

Date:



**ACKNOWLEDGEMENT**

We are highly indebted to our mentor ‘***Dr. Partha Sarathy Banerjee***’ for his guidance and constant supervision as well as for providing necessary information regarding the project & also for his support in completing the project.

We would like to express our special gratitude and thanks to University persons for giving us such attention and time.

Saloni Sharma – 181B179

Stuti Verma – 181B220

Trijal Singh – 181B230

**EXECUTIVE SUMMARY**

The main objective of the project was to create a framework for converting Satellite to Map Image Conversion. We emphasize the importance of human-readability of a map and aim to construct accurate human-readable maps directly from a satellite/aerial image of the location. The satellite/aerial image specifies the zoom level and resolution of the required map.

**TABLE OF CONTENTS**

|  |  |
| --- | --- |
| 1 | Introduction |
| 2 | Objective and Goal statement |
| 3 | Feasibility Study |
| 4 | Requirement Analysis |
| 5 | Proposal |
| 6 | Tools and Framework used |
| 7 | Model |
| 8 | Detailed description of the code and functions used |
| 9 | Student Details |

**INTRODUCTION**

Every year students in the 3rd and 4th year of engineering are required to make projects using the technical and practical knowledge they have gotten to know about in the past semesters of their degree. The projects usually carry a lot of importance for the students and are a part of both their required criteria to complete degree and also are very important in the placement aspect as they give the prospective employers a chance to see what the student actually knows instead of just his grade and helps them judge on the basis of that.

Our project is to convert Satellite Images to Map Images. An accurate map must reflect all changes on the ground in a timely manner. Up-to-date geospatial data is continuously collected from flyover imaging air-crafts or satellites. It is important to emphasize that in addition to the map being accurate, most of the apps created by the companies have a major human-interaction component associated with their success.

# **OBJECTIVE**

# ***“****Satellite-To-map Image Conversion****”***

# 

# **GOAL STATEMENT**

“Using Pix2Pix GAN for the translation of Satellite Images to Map Images”

# 

# **FEASIBILITY STUDY**

# Technical Feasibility

# The main technologies and tools required for the project are:

# • Keras

# • Matplotlib

# • Numpy

# Each of the above-mentioned technologies are freely available and the technical skills required for them are manage-able. Hence the project is technically feasible.

Operational Feasibility

# The users are expected to be comfortable working with the working of the code since all the outputs are direct, concise and visualized for a better understanding.

Economic Feasibility

# All the technologies being used are free to use keeping in mind the scale of our project. Therefore, the project is financially feasible.

Schedule Feasibility

# With the technologies mentioned previously and the right amount of technical knowledge about them the project can be completed before the deadline.

# **REQUIREMENT ANALYSIS**

# **•** Hardware Requirements – A computer system that can run Python3 would be sufficient for our program.

# • Data– We need to have a dataset with satellite images for training plus another dataset with satellite images and .

# **OUR PROPOSAL**

1. We propose a project that will generate a Map Image from Satellite Image.

2. The project will be capable of showing the expected map image and also the generated image from our model.

3. Determining number of relations for a pair of entities.

4. Visualizing the source, generated and expected image for input set.

5. Work on 256x256x3 dimensional satellite images to generate

256x256x3 maps.

**Goal**

GeoGAN is a model that takes as its input a satellite image at a specified zoom level and resolution and produces the corresponding human-readable map for that location. The model is trained to emulate the style of the maps available from the Google Maps API for the generated map. The model is trained as a conditional GAN with the input satellite image as the condition for the generation of the map. While it is relatively straightforward to condition the output of a generator, there is no clear way to condition the output of the discriminator. We perform several experiments on the methods to condition the discriminator.

**About**

The GAN architecture consists of a generator model for outputting new plausible synthetic images, and a discriminator model that classifies images as real (from the dataset) or fake (generated). The discriminator model is updated directly, whereas the generator model is updated via the discriminator model. As such, the two models are trained simultaneously in an adversarial process where the generator seeks to better fool the discriminator and the discriminator seeks to better identify the counterfeit images.

The Pix2Pix model is a type of conditional GAN, or cGAN, where the generation of the output image is conditional on an input, in this case, a source image. The discriminator is provided both with a source image and the target image and must determine whether the target is a plausible transformation of the source image.

**Data Resources**

The dataset has been picked from the official Pix2Pix site and has satellite images of New York and their corresponding Google maps images. The train folder contains 1,097 images, whereas the validation dataset contains 1,099 images. Images have a digit filename and are in JPEG format. Each image is 1,200 pixels wide and 600 pixels tall and contains both the satellite image on the left and the Google maps image on the right.

This dataset was then prepared for training a Pix2Pix GAN model with Keras. We had to work with the images in the training dataset. Each image should be loaded, rescaled, and split into the satellite and Google map elements. The result will be 1,097 color image pairs with the width and height of 256×256 pixels.

**TOOLS AND FRAMEWORKS**

**1. PYTHON**

Here is the list of features of Python which makes it more suitable for web scraping.

● **Ease of Use:** Python is simple to code. You do not have to add semicolons “;” or curly-braces “{}” anywhere. This makes it less messy and easy to use.

● **Large Collection of Libraries:** Python has a huge collection of libraries such as Numpy, Matplotlib, Pandas, Tensorflow etc., which provides methods and services for various purposes. Hence, it is suitable for web scraping and for further manipulation of extracted data.

● **Dynamically typed:** In Python, you don’t have to define data types for variables, you can directly use the variables wherever required. This saves time and makes your job faster.

● **Easily Understandable Syntax:** Python syntax is easily understandable mainly because reading a Python code is very similar to reading a statement in English. It is expressive and easily readable, and the indentation used in Python also helps the user to differentiate between different scope/blocks in the code.

● **Small code, large task**: Web scraping is used to save time. But what’s the use if you spend more time writing the code? Well, you don’t have to. In Python, you can write small codes to do large tasks. Hence, you save time even while writing the code.

● **Community**: What if you get stuck while writing the code? You don’t have to worry. The Python community has one of the biggest and most active communities, where you can seek help from.

**2. NUMPY**

NumPy is an open-source numerical Python library. NumPy contains a multi-dimensional array and matrix data structures. It can be utilised to perform a number of mathematical operations on arrays such as trigonometric, statistical, and algebraic routines.

1. **Asarray** - The asarray() function is used to convert an given input to an array. Input data, in any form that can be converted to an array. This includes lists, lists of tuples, tuples, tuples of tuples, tuples of lists and ndarrays. By default, the data-type is inferred from the input data.
2. **Vstack** - The vstack() function is used to stack arrays in sequence vertically (row wise). This is equivalent to concatenation along the first axis after 1-D arrays of shape (N,) have been reshaped to (1,N). This function makes most sense for arrays with up to 3 dimensions. For instance, for pixel-data with a height, width, and r/g/b channels (third axis).
3. **Precision\_recall\_fscore\_support** - Load arrays or pickled objects from .npy, .npz or pickled files.If the file contains pickle data, then whatever object is stored in the pickle is returned. If the file is a .npy file, then a single array is returned. If the file is a .npz file, then a dictionary-like object is returned, containing key-value pairs, one for each file in the archive. If the file is a .npz file, the returned value supports the context manager protocol in a similar fashion to the open function.

**3. KERAS**

Keras is a powerful and easy-to-use free open source Python library for developing and evaluating deep learning models. It wraps the efficient numerical computation libraries Theano and TensorFlow and allows you to define and train neural network models in just a few lines of code.

1. **keras.initializers** - Initializer that generates tensors with constant values. The resulting tensor is populated with values of type dtype, as specified by arguments value following the desired shape of the new tensor (see examples below).The argument value can be a constant value, or a list of values of type dtype . If the value is a list, then the length of the list must be less than or equal to the number of elements implied by the desired shape of the tensor.
2. **keras.models** - Model groups layers into an object with training and inference features. A model can be instantiated either with the "Functional API", where you start from Input, you chain layer calls to specify the model's forward pass, and finally you create your model from inputs and outputs or by subclassing the Model class: in that case, you should define your layers in \_\_init\_\_ and you should implement the model's forward pass in call.
3. **keras.layers** - Layers are the basic building blocks of neural networks in Keras. A layer consists of a tensor-in tensor-out computation function (the layer's call method) and some state, held in TensorFlow variables (the layer's weights).

**THE MODEL**

## *CNN*

A CNN is an artificial neural network with some specialization for being able to detect patterns and make sense of them. This pattern detection is what makes CNNs so useful for image analysis. CNNs preserve the spatial aspect of the dataset.

## *Image to Image Translation*

Image to image translation is the task of taking images from one domain and transforming them so they have the style (or characteristics) of images from another domain.

## *Generative Adversarial Nets*

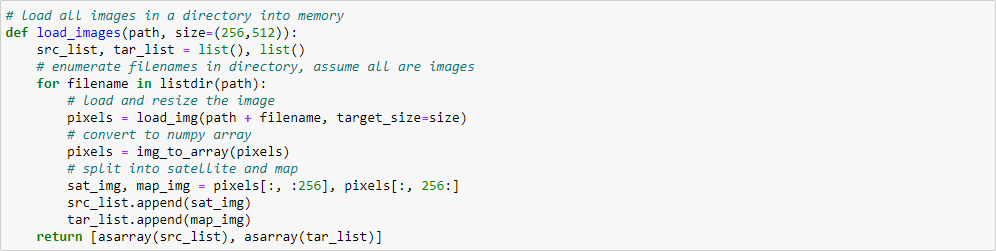
Generative Adversarial nets are an approach to generative modelling using deep learning methods, such as CNNs. GAN is an architecture for automatically training a generative model by treating the unsupervised problem as supervised and using both a generative and discriminative model.

## *Pix2Pix*

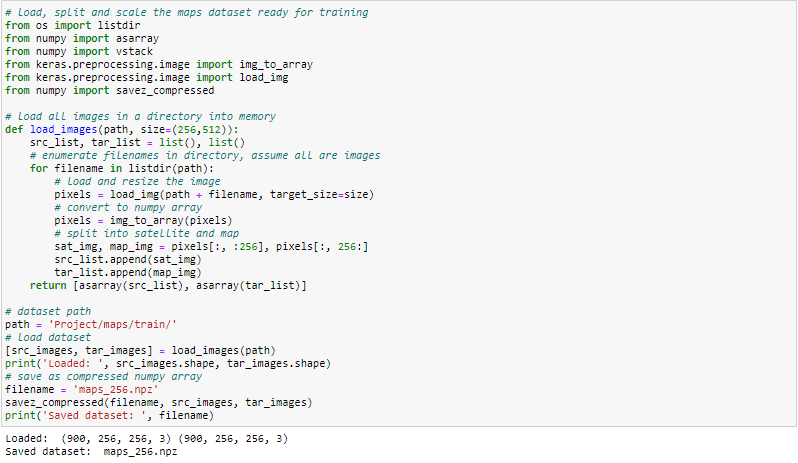
The Pix2PIx model is a type of conditional GAN, or cGAN, where the generation of the output image is conditional on an input, in this case, a source image.

**IMAGE TRANSLATION**

The *load\_images()* function below implements this. It enumerates the list of images in a given directory, loads each with the target size of 256×512 pixels, splits each image into satellite and map elements and returns an array of each.



This loads all images in the training dataset, summarizes their shape to ensure the images were loaded correctly, then saves the arrays to a new file called maps\_256.npz in compressed NumPy array format.

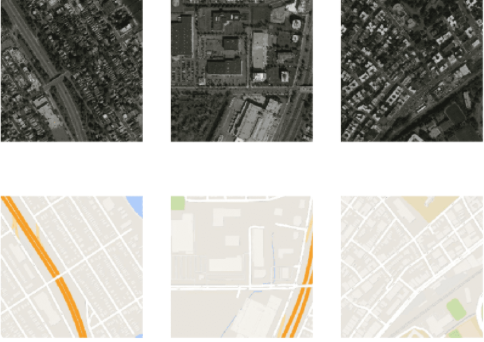


A plot of three image pairs is also created showing the satellite images on the top and Google map images on the bottom.

We can see that satellite images are quite complex and that although the Google map images are much simpler, they have color codings for things like major roads, water, and parks



**OUTPUT:**



**Developing and Training**

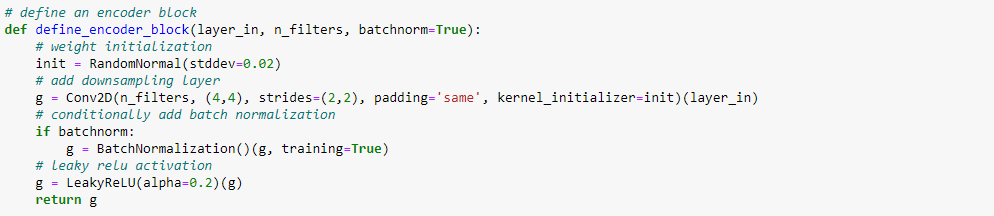
1. Discriminator

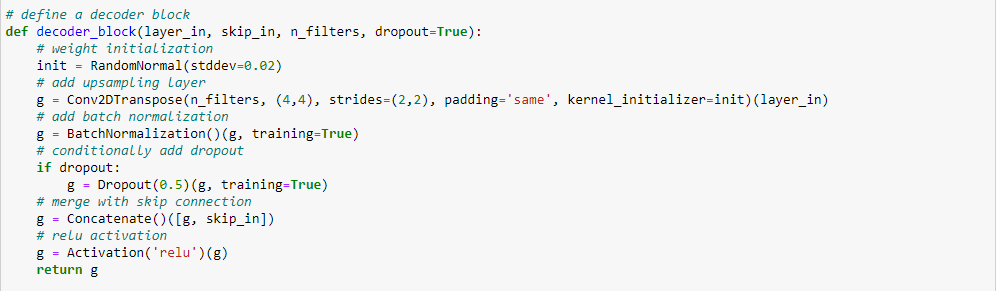
The discriminator is a deep convolutional neural network that performs image classification, based on the effective receptive field of the model, which defines the relationship between one output of the model to the number of pixels in the input image. This is called a PatchGAN model.The *define\_discriminator()* function below implements the 70×70 PatchGAN discriminator model as per the design of the model in the paper. The model takes two input images that are concatenated together and predicts a patch output of predictions.



1. Generator

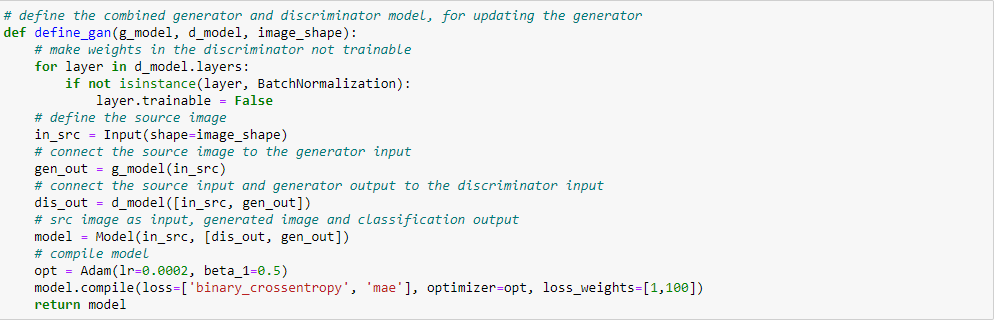
The generator is an encoder-decoder model using a U-Net architecture. The model takes a source image (e.g. satellite photo) and generates a target image (e.g. Google maps image).The encoder and decoder of the generator are standardized blocks of convolutional,batch normalization, dropout, and activation layers.The *define\_generator()* function below implements the U-Net encoder-decoder generator model. It uses the *define\_encoder\_block()* helper function to create blocks of layers for the encoder and the *decoder\_block()* function to create blocks of layers for the decoder.



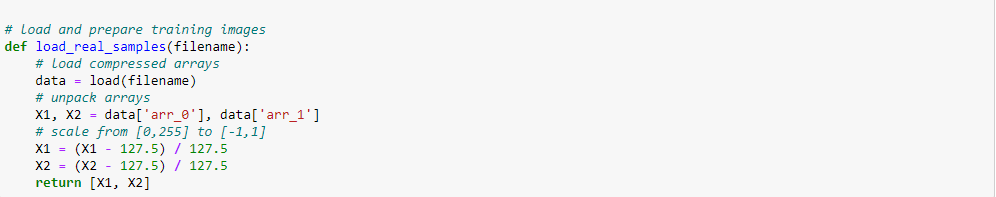


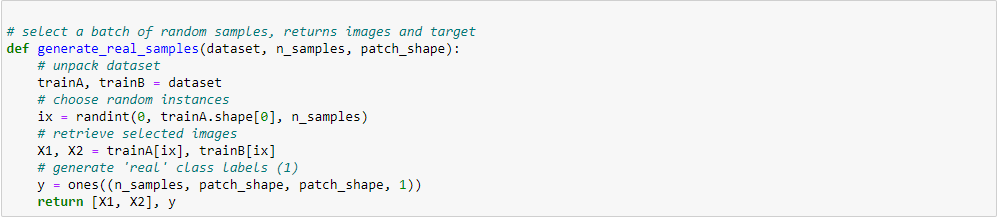


The *define\_gan()* function below implements this, taking the already-defined generator and discriminator models as arguments and using the Keras functional API to connect them together into a composite model. Both loss functions are specified for the two outputs of the model and the weights used for each are specified in the *loss\_weights* argument to the *compile()* function.

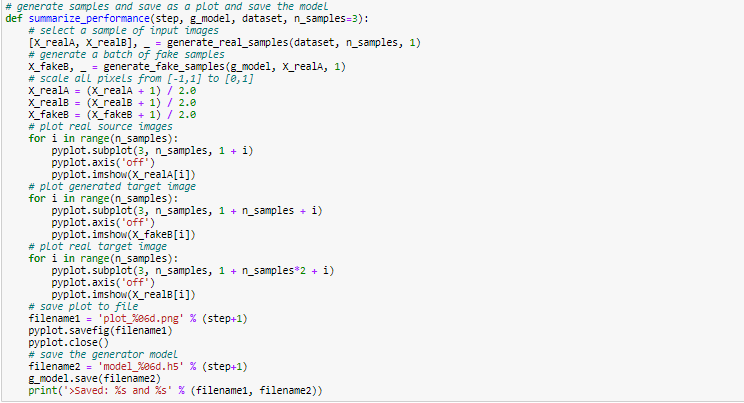


We can load our paired images dataset in compressed NumPy array format.This will return a list of two NumPy arrays: the first for source images and the second for corresponding target images The *generate\_real\_samples()* function below will prepare a batch of random pairs of images from the training dataset, and the corresponding discriminator label of *class=1* to indicate they are real.



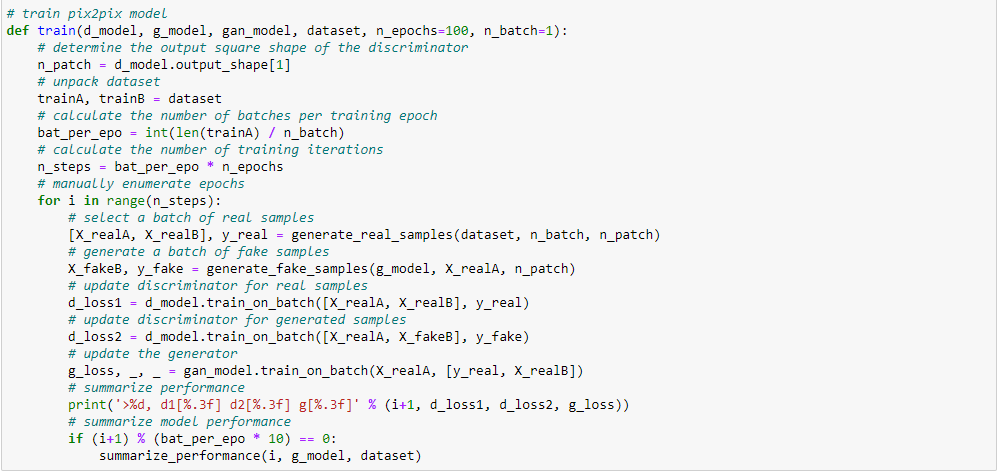


We can then review the generated images at the end of training and use the image quality to choose a final model.The *summarize\_performance()* function implements this, taking the generator model at a point during training and using it to generate a number, in this case three, of translations of randomly selected images in the dataset. The source, generated image, and expected target are then plotted as three rows of images and the plot saved to file. Both the image and model filenames include the training iteration number, allowing us to easily tell them apart at the end of training.



Finally, we can train the generator and discriminator models.The *train()* function below implements this, taking the defined generator, discriminator, composite model, and loaded dataset as input. The number of epochs is set at 100 to keep training times down, although 200 was used in the paper. A batch size of 1 is used as is recommended in the paper.

Training involves a fixed number of training iterations. There are 1,097 images in the training dataset. One epoch is one iteration through this number of examples, with a batch size of one means 1,097 training steps. The generator is saved and evaluated every 10 epochs or every 10,970 training steps, and the model will run for 90 epochs, or a total of 109,700 training steps. Finally, the loss for each update is reported to the console each training iteration and model performance is evaluated every 10 training epochs.



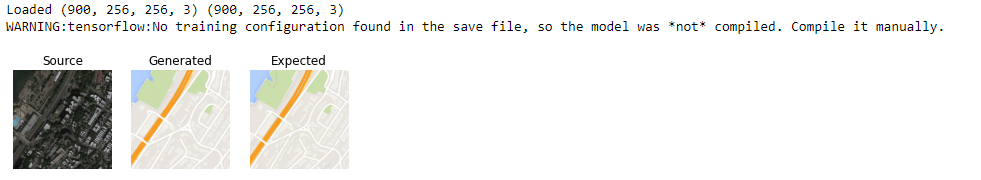
After the first 10 epochs, map images are generated that look plausible, although the lines for streets are not entirely straight and images contain some blurring. Nevertheless, large structures are in the right places with mostly the right colors.Generated images after about 50 training epochs begin to look very realistic, at least to mean, and quality appears to remain good for the remainder of the training process.



**Translate Images**

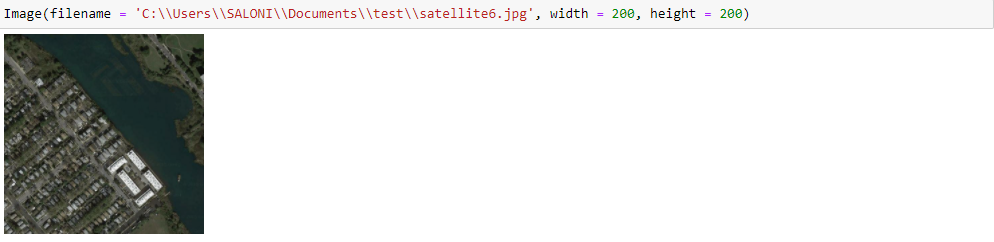
We will load the training dataset. We use the same function named *load\_real\_samples()* for loading the dataset as was used when training the model.Next, we can load the saved Keras model. We can choose a random image pair from the training dataset to use as an example.We can provide the source satellite image as input to the model and use it to predict a Google map image.Finally, we can plot the source, generated image, and the expected target image.The *plot\_images()* function below implements this, providing a nice title above each image.This function can be called with each of our source, generated, and target images.

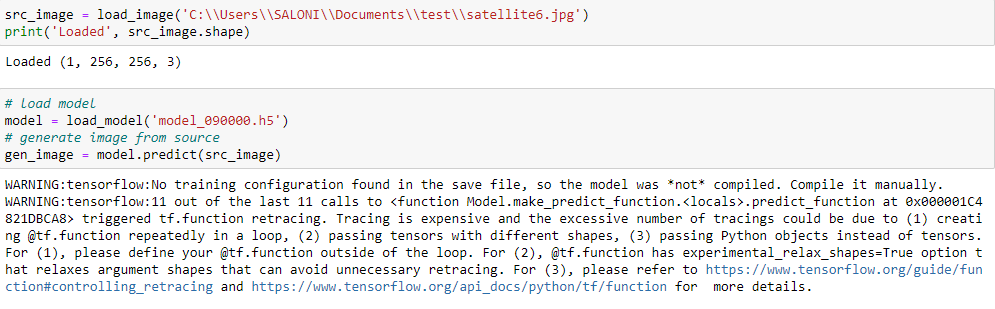


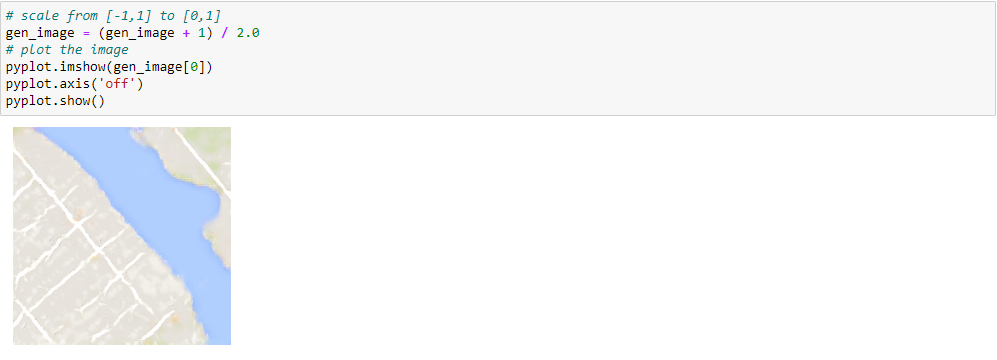


**Image Conversion**

1. **Satellite Image as Input**

**2. Image Translation**

** 3. Generated Map Image**

****

**Student Details**

Name: Saloni Sharma

Enrollment no. : 181B179

E-mail id: 181B179@juetguna.in

Name: Stuti Verma

Enrollment no. : 181B220

E-mail id: 181B220@juetguna.in

Name: Trijal Singh

Enrollment no. : 181B230

E-mail id: 181B230@juetguna.in