Change Detection Using Deep Learning

PROJECT SUBMITTED TO

Gurugram Metropolitan Development Authority

BY

Chetan Yadav

BTech in Artificial Intelligence 4th semester

(SAP ID: 70122100090)

Mukesh Patel School of Technology Management and Engineering

Under the Esteemed guidance of

Sh. NAVEEN KUMAR YADAV

Geospatial Officer (Software)

Mukesh Patel School of Technology Management and Engineering

Mumbai, Maharashtra, 400056

SVKM'S NMIMS Deemed to be UNIVERSITY

MUKESH PATEL SCHOOL OF TECHNOLOGY MANAGEMENT & ENGINEERING

Date: 28th April, 2023

TO WHOM SO EVER IT MAY CONCERN

This is to certify that **Mr. Chetan Yadav (70122100090)** is a bonafide student of SVKM's NMIMS, MUKESH PATEL SCHOOL OF TECHNOLOGY MANAGEMENT & ENGINEERING studying in 2nd year B Tech (Artificial Intelligence) for the academic year 2022-2023.

This certificate is issued on his request for the purpose of internship with Gurugram Metropolitan Development Authority from 7th May 2023 to 23rd June 2023.

Dr. Vaishali Kulkarni
Program Head of Artificial Intelligence
MPSTME

Mukesh Patel School of Tech. Mangt & Engg. SVKM'S NMIMS Mumbai BG

SVKM'S
Narsee Monjee Institute of Management Studies
Deemed to be UNIVERSITY
Bhakti Vedant Swami Marg, Vile Parle (West), Mumbai - 400 056, India.
Tel: (91-22) 42334000
Email: enquiry.mpstme@nmims.edu | Web: **www.engineering.nmims.edu**

CATEGORY-I UNIVERSITY

F.No – G-40001/1/2018-Head GIS
GURUGRAM METROPOLITAN DEVELOPMENT AUTHORITY
Plot No – 03, Sector -44, GURUGRAM-122001

Memo No. Dated.

CERTIFICATE

TO WHOM IT MAY CONCERN

This to certify that project report compiled by **Mr Chetan Yadav** (70122100090) student of 2nd year of BTech (Artificial Intelligence) for the academic year 2022-2023, **Mukesh Patel School of Technology Management and Engineering,** Mumbai Maharashtra have completed their summer internship at GMDA. To the best of our knowledge this is an original and bonafide work done by him. He has worked on Finding Change in Drone Imagery using Deep learning techniques project titled **"Change Detection using Deep Learning"** for **Gurugram Metropolitan Development Authority (GMDA)**. Starting from 7th May to 22rd June 2023.

During his tenure at this organization, he was found to be sincere and meticulous in his work. We appreciated their enthusiasm & dedication towards the work assigned to them.

We wish them every success.

Geospatial Officer (Software Application),
Gurugram Metropolitan Development Authority,
Gurugram

Gurugram Metropolitan development authority

A Statutory Authority under GMDA Act 2017

UNDERTAKING

I declare and undertake that the project title **'Change Detection using Deep Learning'** for Gurugram Metropolitan Development Authority submitted by **CHETAN YADAV,** SAP Id **70122100090** from Mukesh Patel School of Technology and Management is my original work. I have not plagiarized this project or submitted it anywhere else.

Date: Name: Chetan Yadav

Place: Gurugram Signature of the student

ACKNOWLEDGEMENT

The accomplishment of a project gives a great sense of satisfaction, but it is never complete without acknowledging those people who made it possible.

Here, I take this opportunity to extend my sincere thanks to all of them who have directly or indirectly helped me in these endeavors. I take this occasion to thank God, almighty for blessing me with his grace and taking my endeavor to a successful culmination.

I would like to express special thanks to **Dr. Sultan Singh, Chief Geospatial Officer (CGO**) & **Head GIS-GMDA (Govt. Of Haryana)** for giving me an opportunity to work in such a reputed organization. Besides, I would like to thank the **Gurugram Metropolitan Development Authority (GMDA)**, for providing me with a congenial environment and extending necessary facilities to complete this project. I am also thankful to all the staff members of **GIS Division-GMDA** for the valuable suggestions, encouragement, support and the facilities provided to complete the project work.

I am highly grateful to **Sh. Naveen Kumar Yadav, Geospatial Officer- GMDA** for his content guidance and supervision in condition of this project and providing needed support which enabled me in presenting internship project report in this shape.

Contents

1) INTRODUCTION

1.1 About the Project:

Change detection through Deep Learning uses deep learning architecture called Unet to find buildings in given image. Then we take same area images from different time then compare them to find differences in the image.

1.2 Purpose of the Project:

The purpose of the project is to detect illegal or new constructions in an area

2) DEEP AND LEARNING AND ITS TERMINOLOGIES

2.1 Overview of Deep Learning:

Deep learning is a subset of machine learning that focuses on training artificial neural networks with multiple layers, allowing them to learn hierarchical representations of data. It is inspired by the structure and functioning of the human brain.

Key points about deep learning:

1. Neural Networks: Deep learning models are built using neural networks, which consist of interconnected artificial neurons (also known as nodes or units). Each neuron receives inputs, applies an activation function, and produces an output.

2. Deep Architectures: Deep learning models have multiple layers of neurons, enabling them to learn complex representations of data. These layers are typically composed of an input layer, one or more hidden layers, and an output layer.

3. Learning from Data: Deep learning models learn from labeled examples by iteratively adjusting their internal parameters (weights and biases) to minimize the difference between predicted and true values. This process is called training and is often done using optimization algorithms like gradient descent.

4. Automatic Feature Extraction: Deep learning models can automatically learn useful features from raw data, reducing the need for manual feature engineering. The hidden layers of the network extract progressively higher-level features, allowing the model to understand complex patterns and relationships.

5. Unsupervised and Semi-Supervised Learning: Deep learning can be applied to unsupervised learning tasks, where the model learns patterns and structures in unlabeled data. It can also leverage both labeled and unlabeled data for semi-supervised learning, making it useful when labeled data is scarce.

6. Popular Architectures: Deep learning encompasses various neural network architectures, including Convolutional Neural Networks (CNNs) for image-related tasks, Recurrent Neural

Networks (RNNs) for sequential data, and Generative Adversarial Networks (GANs) for generating new data.

7. Large-scale Computing: Deep learning often requires significant computational resources, especially for training complex models on large datasets. Graphics Processing Units (GPUs) and specialized hardware like Tensor Processing Units (TPUs) are commonly used to accelerate deep learning computations.

8. Diverse Applications: Deep learning has shown remarkable performance in various domains, including computer vision, natural language processing, speech recognition, recommendation systems, and healthcare. It has achieved state-of-the-art results in tasks such as image classification, object detection, machine translation, and voice synthesis.

Deep learning has revolutionized the field of artificial intelligence and has become a powerful tool for solving complex problems in different industries. Its ability to learn from data, automatically extract features, and handle large-scale tasks makes it a driving force behind many advancements in machine learning.

2.2 Components in Deep Learning:

In deep learning, several key components work together to build and train effective models. Here are the main components:

1. **Neural Networks:** Neural networks are the foundation of deep learning. They consist of interconnected artificial neurons (nodes) organized into layers. The nodes in each layer receive inputs, apply activation functions, and produce outputs that serve as inputs to the next layer.

2. **Layers:** Deep learning models typically have multiple layers, including an input layer, one or more hidden layers, and an output layer. Each layer performs specific operations on the data, such as computing weighted sums, applying activation functions, or generating predictions.

3. **Weights and Biases:** Neural networks use weights and biases to adjust the influence of each input and neuron on the model's output. During training, these parameters are iteratively updated to minimize the difference between predicted and true values, optimizing the model's performance.

4**. Activation Functions:** Activation functions introduce non-linearity into the neural network, enabling it to learn complex patterns and relationships in the data. Common activation functions include sigmoid, tanh, ReLU (Rectified Linear Unit), and softmax (used for multi-class classification)

5. **Loss Functions:** Loss functions quantify the difference between predicted and true values and are used to evaluate the model's performance. The choice of the loss function depends on the task at hand, such as mean squared error (MSE) for regression or categorical cross-entropy for classification

6. **Optimization Algorithms:** Optimization algorithms, such as stochastic gradient descent (SGD) or Adam, adjust the model's weights and biases during training to minimize the loss function. They determine how the model learns from the data and update the parameters in an efficient manner.

7. **Backpropagation:** Backpropagation is a crucial algorithm for training deep learning models. It calculates the gradients of the loss function with respect to the model's weights and biases, enabling the optimization algorithm to update these parameters correctly.

8. **Regularization Techniques:** Regularization techniques, such as L1 and L2 regularization or dropout, are employed to prevent overfitting. They introduce penalties or modifications to the model's structure to encourage generalization and improve performance on unseen data.

9. **Validation and Testing:** Validation and testing are crucial steps in deep learning. The model's performance is evaluated on a separate validation set during training to monitor its generalization ability. The final evaluation is performed on a separate test set to assess the model's performance on unseen data.

10. **Preprocessing and Data Augmentation:** Data preprocessing involves transforming and normalizing the input data to facilitate effective learning. Data augmentation techniques, such as rotation, scaling, or flipping, are applied to artificially increase the size and diversity of the training data.

These components work together to build deep learning models that can learn complex representations from data and make accurate predictions. Proper configuration, parameter tuning, and understanding of these components are crucial for achieving optimal model performance.

2.3 How is Deep Learning useful for us:

Deep learning has proven to be incredibly useful across a wide range of domains, including those involving spatial data analysis and geospatial applications.

1. **Image Recognition:** Deep learning excels in image recognition tasks, allowing it to automatically identify and classify objects, features, or patterns within images. This capability has applications in fields like remote sensing, where deep learning models can analyze satellite imagery to detect land cover changes, monitor vegetation health, or identify specific features like roads or buildings.

2. **Object Detection and Segmentation:** Deep learning models can accurately detect and segment objects within images or point clouds, even in complex or cluttered scenes. This capability is valuable in applications like autonomous vehicles, where deep learning algorithms can identify and track other vehicles, pedestrians, traffic signs, and obstacles.

3. **Natural Language Processing:** Deep learning enables powerful natural language processing, allowing machines to understand, generate, and respond to human language. In geospatial applications, this can be leveraged for tasks like sentiment analysis of location-based social media data, automatic geotagging of textual data, or extracting geospatial information from documents or reports.

4. **Time Series Analysis:** Deep learning models, such as recurrent neural networks (RNNs) and Long Short-Term Memory (LSTM) networks, are effective in analyzing temporal data. This is valuable for tasks like predicting environmental phenomena (e.g., weather forecasting), analyzing temporal patterns in geospatial data (e.g., urban mobility patterns), or forecasting time-dependent events (e.g., predicting disease outbreaks).

5. **Anomaly Detection:** Deep learning models can learn patterns from large datasets and identify anomalies or outliers. In geospatial applications, this can be used to detect unusual behavior, such as detecting anomalies in satellite imagery for environmental monitoring or identifying unusual patterns in sensor data for infrastructure monitoring.

6. **Spatial Data Fusion:** Deep learning can facilitate the fusion of heterogeneous data sources, such as satellite imagery, sensor data, and textual information. By integrating and processing diverse data types, deep learning models can generate comprehensive and enriched geospatial information, providing valuable insights for decision-making processes.

7. **Data Compression and Reconstruction:** Deep learning techniques, such as auto encoders, can be utilized for data compression and reconstruction. This is useful in scenarios where storage or bandwidth is limited, allowing efficient representation and reconstruction of geospatial data.

These are just a few examples of how deep learning is beneficial without explicitly mentioning GIS. Deep learning's ability to learn from complex data, extract meaningful features, and make accurate predictions has far-reaching applications across numerous domains, including those involving spatial data analysis and geospatial applications.

2.4 How does Deep Learning work:

Deep learning is a subfield of machine learning that focuses on training artificial neural networks with multiple layers to learn and extract meaningful representations from data. The mathematics involved in deep learning encompasses linear algebra, calculus, and optimization techniques. Here's a detailed explanation of deep learning, along with the associated math:

1. **Neural Network Architecture:** A deep learning model is constructed using interconnected artificial neurons organized in layers. Each neuron receives inputs, applies a linear transformation, adds a bias term, and applies a non-linear activation function to produce an output.

2. **Feedforward Pass:** During the feedforward pass, the input data is propagated through the layers of the neural network. The output of a neuron is calculated as the weighted sum of its inputs, represented by the equation $z = Wx + b$, where W is the weight matrix, x is the input vector, and b is the bias vector. The output is then passed through an activation function to produce the neuron's activation value, denoted as $a = \sigma(z)$, where $\sigma()$ is the activation function.

3. **Activation Functions:** Activation functions introduce non-linearities to the neural network, enabling it to model complex relationships. Common activation functions include sigmoid (σ), tanh, and Rectified Linear Unit (ReLU). They transform the output of a neuron to a specific range, allowing for non-linear behavior.

4. **Loss Function:** The loss function quantifies the discrepancy between the predicted output of the model and the true output or labels. The choice of the loss function depends on the task at hand. For example, mean squared error (MSE) is commonly used for regression tasks, while categorical cross-entropy is used for multi-class classification tasks.

5. **Backpropagation and Gradient Descent:** Backpropagation is an algorithm used to compute the gradients of the loss function with respect to the model's parameters. The chain rule from calculus is applied to efficiently calculate these gradients layer-by-layer, starting from the output layer to the input layer. Gradient descent algorithms, such as stochastic gradient descent (SGD), utilize these gradients to update the model's parameters and minimize the loss function iteratively.

6. **Gradient Descent Optimization:** Gradient descent algorithms update the model's parameters by iteratively moving in the opposite direction of the gradients to minimize the loss function. The learning

rate determines the step size for each update. Other optimization techniques, such as momentum, adaptive learning rates (e.g., Adam), or regularization methods (e.g., L1 or L2 regularization), can be employed to improve convergence and prevent overfitting.

7. **Matrix Operations:** Deep learning extensively employs linear algebra operations, such as matrix multiplication and element-wise operations. These operations enable efficient computation across multiple inputs and neurons simultaneously, speeding up training and inference processes using parallel processing.

8. **Batch Processing:** To improve training efficiency, deep learning models often process data in batches rather than individually. Batch processing allows for vectorized computations and takes advantage of parallel processing, significantly accelerating training using GPUs or TPUs.

9. **Initialization and Regularization:** The initial values of the model's parameters, such as weights and biases, can influence the learning process. Proper initialization methods, such as Xavier or He initialization, help ensure stable and efficient learning. Regularization techniques, like L1 or L2 regularization, introduce penalty terms to the loss function to prevent overfitting and encourage the learning of simpler representations.

10. **Optimization Metrics:** In addition to the loss function, various evaluation metrics are used to assess the performance of deep learning models during training and testing. These metrics depend on the specific task and can include accuracy, precision, recall, F1-score for classification tasks, or mean squared error (MSE), R-squared for regression tasks.

Understanding the mathematical foundations of deep learning, including linear algebra, calculus, and optimization techniques, is crucial for developing, training, and optimizing deep learning models. By leveraging these mathematical concepts, deep learning models can effectively learn intricate patterns and relationships in data and make accurate predictions across a wide range of applications.

2.5 Important Terms used in Deep Learning:

Terms	Explanation
Deep Learning	A subfield of machine learning that focuses on training artificial neural networks with multiple layers.
Neural Network	A computational model inspired by the human brain, consisting of interconnected artificial neurons organized in layers.
Neuron	An individual node in a neural network that receives inputs, applies a transformation, and produces an output
Input Layer	The first layer of a neural network that receives input data and passes it to the subsequent layers.
Hidden Layer	One or more layers between the input and output layers that perform computations and extract features from the input.
Output Layer	The final layer of a neural network that produces the desired output or prediction.
Activation Function	A mathematical function applied to the output of a neuron, introducing non-linearities and enabling complex modeling.

Weight	A parameter of a neuron that determines the strength of influence one neuron has on another in the neural network.
Bias	An additional input to a neuron that helps adjust its behaviour and allows flexibility in the learned representation.
Feedforward Pass	The process of passing input data through the neural network to obtain an output by sequentially applying computations.
Loss Function	A function that measures the discrepancy between the predicted output and the true output, used to optimize the model.
Backpropagation	An algorithm that calculates the gradients of the loss function with respect to the model's parameters for optimization.
Gradient Descent	An optimization algorithm that iteratively updates the model's parameters by moving in the opposite direction of gradients.
Learning Rate	A hyperparameter that determines the step size of parameter updates during optimization.
Activation	The output value of a neuron after applying the activation function to the weighted sum of its inputs.
Matrix Multiplication	An operation that combines the elements of two matrices to produce a new matrix, used for efficient computations.
Batch Processing	Training the model using multiple samples simultaneously, improving efficiency through vectorized computations.
Initialization	Setting initial values for the model's parameters, such as weights and biases, to facilitate stable and efficient learning.
Regularization	Techniques used to prevent overfitting by introducing additional terms or constraints to the loss function.
Optimization Metrics	Metrics used to evaluate the performance of a deep learning model during training and testing.

3) TOOLS AND TECHNOLOGIES

3.1 ArcGIS Pro:

ArcGIS Pro is a professional desktop Geographic Information System (GIS) software developed by Esri. It provides a comprehensive set of tools and capabilities for working with spatial data, performing geospatial analysis, creating maps and visualizations, and sharing and managing geospatial information. Here's a detailed explanation of ArcGIS Pro:

1. **User Interface:** ArcGIS Pro features a modern and intuitive user interface that allows users to access tools, menus, and functionalities through a ribbon-style interface. It provides a consistent and streamlined experience for performing GIS tasks

2. **Data Management:** ArcGIS Pro enables users to manage various types of spatial data, including vector data, raster data, and tabular data. It supports a wide range of data formats, such as shapefiles, geodatabases, CAD files, imagery, and more. Users can import, create, edit, and analyze spatial data within the software.

3. **Map Authoring:** Users can create visually appealing and informative maps using ArcGIS Pro. It offers advanced cartographic capabilities, allowing users to customize symbology, labels, and layout elements. Maps can be created by adding layers, configuring their properties, and arranging them in a desired layout.

4. **Geospatial Analysis:** ArcGIS Pro provides a comprehensive suite of geospatial analysis tools for performing spatial queries, overlay analysis, proximity analysis, terrain analysis, statistical analysis, and more. These tools allow users to extract valuable insights from spatial data and make informed decisions.

5. **3D Visualization:** ArcGIS Pro offers robust capabilities for creating and visualizing 3D data. Users can import and visualize 3D data, such as terrain models, buildings, and 3D point clouds. It supports interactive exploration of 3D scenes, including flythrough animations, dynamic symbology, and realistic rendering effects.

6. **Geoprocessing:** Geoprocessing tools in ArcGIS Pro allow users to automate GIS workflows and perform advanced spatial analysis tasks. Users can chain together multiple geoprocessing tools to create models and automate complex processes. Python scripting is also integrated, allowing for custom scripting and automation.

7. **Data Integration:** ArcGIS Pro facilitates the integration of various data sources and systems. It provides tools for data transformation, data loading, and data integration. Users can combine data from different sources, perform data cleansing and validation, and create relationships between datasets.

8. **Collaboration and Sharing:** ArcGIS Pro enables users to share their GIS work with others. Users can publish maps, scenes, and geoprocessing services to ArcGIS Online or ArcGIS Enterprise for sharing with colleagues or the public. Collaboration tools allow multiple users to work together on the same project and share data and maps.

9. **Spatial Analysis Workflows:** ArcGIS Pro supports the creation and execution of complex spatial analysis workflows. Users can design and automate spatial analysis processes by chaining together a series of geoprocessing tools, models, and scripts. This allows for repeatable and scalable analysis workflows.

10. **Data Visualization and Exploration:** ArcGIS Pro provides powerful visualization and exploration tools. Users can interactively explore spatial data, perform on-the-fly analysis, create charts and graphs, and dynamically update visualizations as data changes. It supports the creation of interactive dashboards for data exploration and communication.

ArcGIS Pro is a versatile and powerful GIS software that offers a wide range of tools and capabilities for working with spatial data. It empowers users to analyze, visualize, and share geospatial information, making it an essential tool for professionals in various industries, including environmental management, urban planning, natural resource analysis, transportation, and many more.

3.2 Python (3.11.3):

Python is a high-level programming language renowned for its simplicity and readability. Created by Guido van Rossum, Python has become widely used in various domains such as web development, data analysis, scientific computing, machine learning, and artificial intelligence. Its versatility makes it suitable for different programming paradigms, including procedural, object-oriented, and functional programming. Python's clean and readable syntax, with its emphasis on indentation, enhances code clarity and maintainability. As an interpreted language, Python eliminates the need for compilation, enabling developers to write and test code quickly. Python's extensive standard library provides numerous modules and functions for common tasks, while its large ecosystem of third-party packages extends its capabilities further. It supports cross-platform compatibility, allowing code to be written on one operating system and run on others without significant modifications. Python excels in data analysis and scientific computing, thanks to libraries like NumPy and Pandas, which offer powerful tools for numerical computations, data manipulation, and statistical analysis. Moreover, Python has gained prominence in machine learning and artificial intelligence with libraries like scikit-learn, TensorFlow, and PyTorch. In web development, frameworks such as Django and Flask provide robust solutions for building scalable web applications and APIs. Python's thriving community offers extensive documentation, resources, and support, making it an accessible and popular choice for developers of all levels of experience.

3.2.1 Libraries Used:

Libraries in Python are pre-written collections of code that provide additional functionality and features beyond what is available in the Python standard library. They save time and effort by offering ready-made solutions for common programming tasks. Python libraries cover various domains such as data analysis, scientific computing, web development, machine learning, image processing, and more. Here are the few libraries I have used

3.2.1.1. Tensorflow (2.12.0):

TensorFlow is an open-source library for machine learning and deep learning developed by Google. It provides a flexible and comprehensive platform for building and deploying machine learning models. TensorFlow uses a computation graph to represent operations and data flow, with tensors as the central data structure. It includes a high-level API called Keras for building deep learning models

easily. TensorFlow supports automatic differentiation for efficient model training, GPU acceleration for faster computations, and distributed computing for scaling up workloads. It offers various deployment options and includes TensorBoard for visualization and monitoring. TensorFlow has a large community and ecosystem, making it a popular choice for machine learning tasks.

3.2.1.2. Glob:

glob is a Python library module that provides a simple and efficient way to retrieve file pathnames matching a specified pattern. It allows you to search for files and directories in a directory hierarchy based on wildcard patterns.

3.2.1.3. CV2 (4.7.0):

cv2 is a Python library for computer vision and image processing. It offers functions and algorithms for tasks such as image and video I/O, image manipulation, filtering and enhancement, object detection and recognition, feature extraction and matching, and more. cv2 integrates well with NumPy and has a large community and documentation resources available.

3.2.1.4. Matplolib (3.7.1):

Matplotlib is a Python library for creating visualizations such as line plots, scatter plots, bar plots, and more. It offers customization options, supports various data formats, integrates with NumPy and Pandas, and has interactive features for exploring plots. Matplotlib is widely used, has extensive documentation, ansked a strong community.

3.2.1.5. Keras (2.12.0):

Keras is a high-level deep learning library in Python that simplifies the process of building and training neural network models. It provides a user-friendly and intuitive interface, allowing developers to quickly prototype and implement deep learning architectures. Keras is built on top of other powerful libraries, such as TensorFlow or Theano, which handle the backend computations. With Keras, users can easily define layers, activation functions, optimization algorithms, and loss functions to create complex neural networks. It also offers convenient tools for data preprocessing, model evaluation, and visualization. Keras's simplicity, versatility, and extensive documentation have made it a popular choice for deep learning practitioners.

3.2.1.6. Sklearn (1.2.2):

Scikit-learn, also known as sklearn, is a popular Python library for machine learning. It provides a wide range of algorithms and tools for various tasks in machine learning, including classification, regression, clustering, dimensionality reduction, and model selection. Sklearn offers a consistent and user-friendly API, making it easy to implement and evaluate machine learning models. It includes functions for data preprocessing, feature extraction, and feature selection. Sklearn also provides utilities for model evaluation, cross-validation, and hyperparameter tuning. With its comprehensive documentation and vast collection of algorithms, sklearn is widely used by data scientists and machine learning practitioners to develop and deploy machine learning models efficiently.

3.2.1.7. Numpy (1.23.5):

NumPy is a fundamental Python library for numerical computing. It provides a powerful N-dimensional array object, along with a collection of functions and tools for working with arrays. NumPy enables efficient and fast operations on arrays, making it a cornerstone for scientific and data-

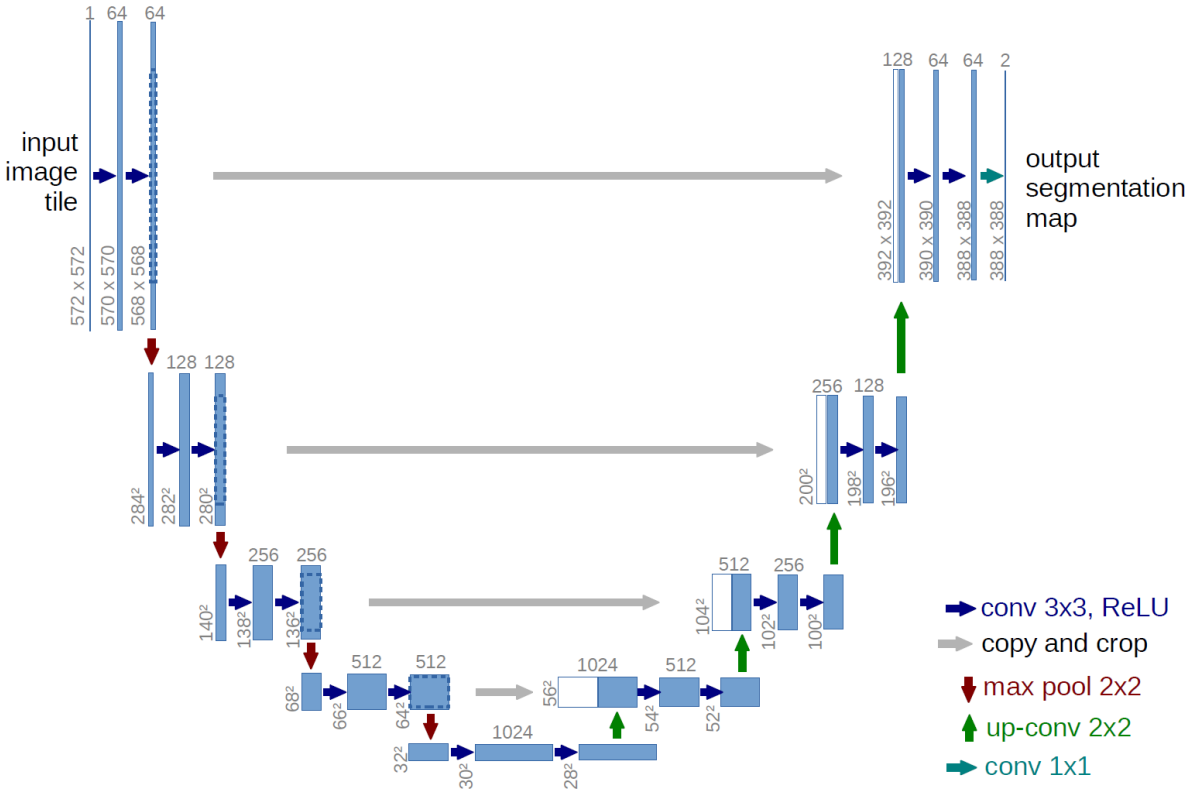
related tasks. With NumPy, you can perform mathematical and logical operations on arrays, manipulate array shapes, slice and index arrays, and perform linear algebra and statistical computations. It also integrates seamlessly with other Python libraries and tools, such as Matplotlib for visualization and Pandas for data analysis. NumPy's efficient array processing capabilities, extensive functionality, and broad community support have made it an essential library for numerical computing in Python.

3.3 Visual Studio code:

Visual Studio Code (VS Code) is a lightweight, cross-platform source code editor with a rich ecosystem of extensions. It provides intelligent code editing features, integrated terminal and debugging support, version control integration, task automation, and a vibrant community.

4) DEEP LEARNING ARCITECHTURE AND TECHNIQUES USED

4.1 Unet Architecture:



The U-Net architecture is a popular deep learning architecture commonly used for image segmentation tasks. It was introduced by Olaf Ronneberger, Philipp Fischer, and Thomas Brox in their 2015 paper titled "U-Net: Convolutional Networks for Biomedical Image Segmentation."

The U-Net architecture is designed to address the challenge of pixel-level segmentation, where the goal is to classify each pixel in an image into specific classes or regions. It is especially effective in scenarios where the target objects or regions are relatively small compared to the overall image size.

The key characteristic of the U-Net architecture is its U-shaped or symmetric structure. It consists of an encoder pathway and a decoder pathway. The encoder pathway captures the context and extracts high-level features from the input image, while the decoder pathway recovers the spatial information and generates the segmentation map.

Here's a high-level overview of the U-Net architecture:

1. **Encoder Pathway:** The encoder part of the U-Net architecture resembles a typical convolutional neural network (CNN). It consists of a series of convolutional and pooling layers. The convolutional layers apply filters to capture features at different spatial scales, while the pooling layers downsample the spatial dimensions to reduce the computational load.

2. **Bridge:** At the bottom of the U-Net architecture, there is a bridge that connects the encoder and decoder pathways. It typically consists of multiple convolutional layers without any pooling operations. This bridge helps to retain spatial information and establish better connectivity between the encoder and decoder pathways

3. **Decoder Pathway:** The decoder part of the U-Net architecture is symmetric to the encoder pathway. It consists of upsampling and convolutional layers. The upsampling layers increase the spatial dimensions to match the original image size, and the convolutional layers refine the features and produce the final segmentation map. In some variations of U-Net, skip connections are also used to concatenate feature maps from the encoder pathway with corresponding layers in the decoder pathway. These skip connections help to preserve fine-grained details and improve the localization of segmentation boundaries.

4. **Output:** The output of the U-Net architecture is a segmentation map, which has the same spatial dimensions as the input image. Each pixel in the segmentation map corresponds to a specific class or region, indicating the predicted label for that pixel.

The U-Net architecture has been widely adopted in various image segmentation tasks, particularly in the field of biomedical imaging. It has demonstrated excellent performance in segmenting organs, tumors, cells, and other structures of interest. The U-Net's ability to capture both local and global context, combined with skip connections, makes it effective in preserving fine details and accurately delineating object boundaries.

4.2 Digital negatives of images:

It is the process in which we inverse the pixel values of images this is done so that model learns to find patterns in the images and not only just see color in the images and classify the pixels

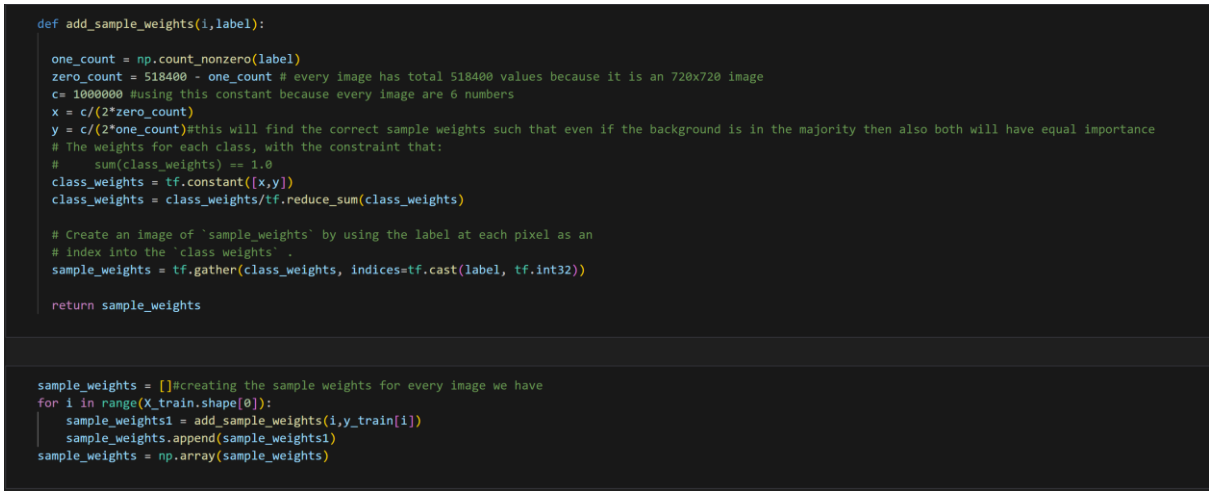
```
#we will create digital negative of every train image so model does not train only on color and learns to find more parameters
digital_negative = []
for i in range(len(X_train)):
    dig_neg = 1 - X_train[i]
    digital_negative.append(dig_neg)
digital_negative = np.array(digital_negative)
X_train = np.concatenate((X_train,digital_negative))
print(len(X_train))
✓ 2.0s
```

An example of digital negative of an image will be:

4.3 Assigning class weights:

Class weights in deep learning are used to address class imbalance in a dataset during training. Class imbalance refers to an unequal distribution of samples across different classes in a classification problem. Class weights assign higher weights to the minority class and lower weights to the majority

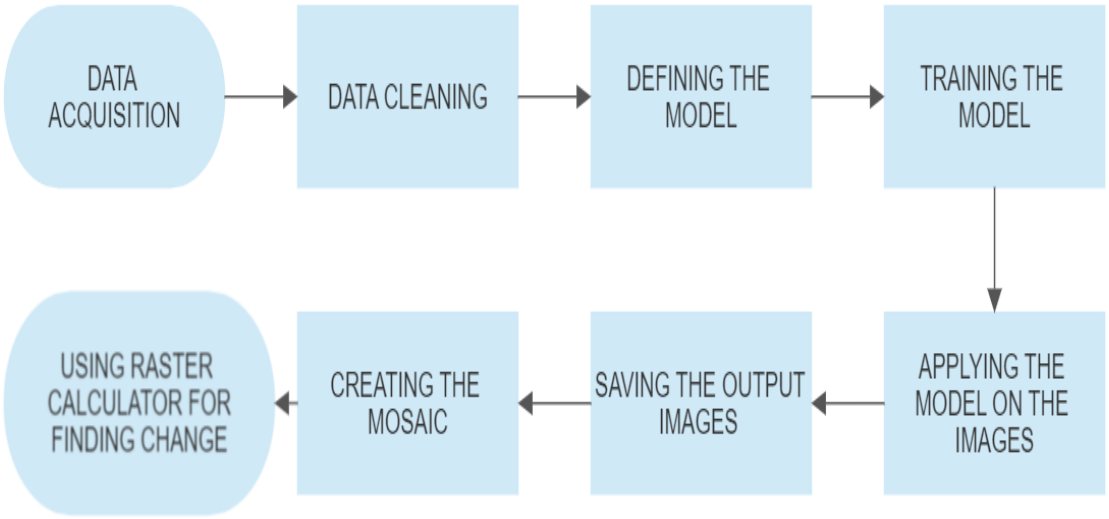
class, adjusting the contribution of each class to the overall loss function. By doing so, class weights help to mitigate the bias towards the majority class and improve the model's ability to learn from the minority class. This can lead to better performance on imbalanced datasets and improve the model's generalization. Class weights are computed based on the class distribution in the dataset and are incorporated into the loss function during training. They are multiplied with the per-sample loss to adjust the contribution of each sample based on its class. The weighted loss is then minimized during training, allowing the model to focus on learning from all classes equally. Class weights can be set manually or calculated using methods like inverse class frequency. The implementation of class weights depends on the deep learning framework being used. As the building pixels are less than the corresponding background pixels we need to use this techniques and this done through the following snippet of code



```
def add_sample_weights(i,label):

  one_count = np.count_nonzero(label)
  zero_count = 518400 - one_count # every image has total 518400 values because it is an 720x720 image
  c= 1000000 #using this constant because every image are 6 numbers
  x = c/(2*zero_count)
  y = c/(2*one_count)#this will find the correct sample weights such that even if the background is in the majority then also both will have equal importance
  # The weights for each class, with the constraint that:
  #     sum(class_weights) == 1.0
  class_weights = tf.constant([x,y])
  class_weights = class_weights/tf.reduce_sum(class_weights)

  # Create an image of `sample_weights` by using the label at each pixel as an
  # index into the `class weights` .
  sample_weights = tf.gather(class_weights, indices=tf.cast(label, tf.int32))

  return sample_weights
```

```
sample_weights = []#creating the sample weights for every image we have
for i in range(X_train.shape[0]):
    sample_weights1 = add_sample_weights(i,y_train[i])
    sample_weights.append(sample_weights1)
sample_weights = np.array(sample_weights)
```

5) FLOWCHART


DATA ACQUISITION
DATA CLEANING
DEFINING THE MODEL
TRAINING THE MODEL
APPLYING THE MODEL ON THE IMAGES
SAVING THE OUTPUT IMAGES
CREATING THE MOSAIC
USING RASTER CALCULATOR FOR FINDING CHANGE

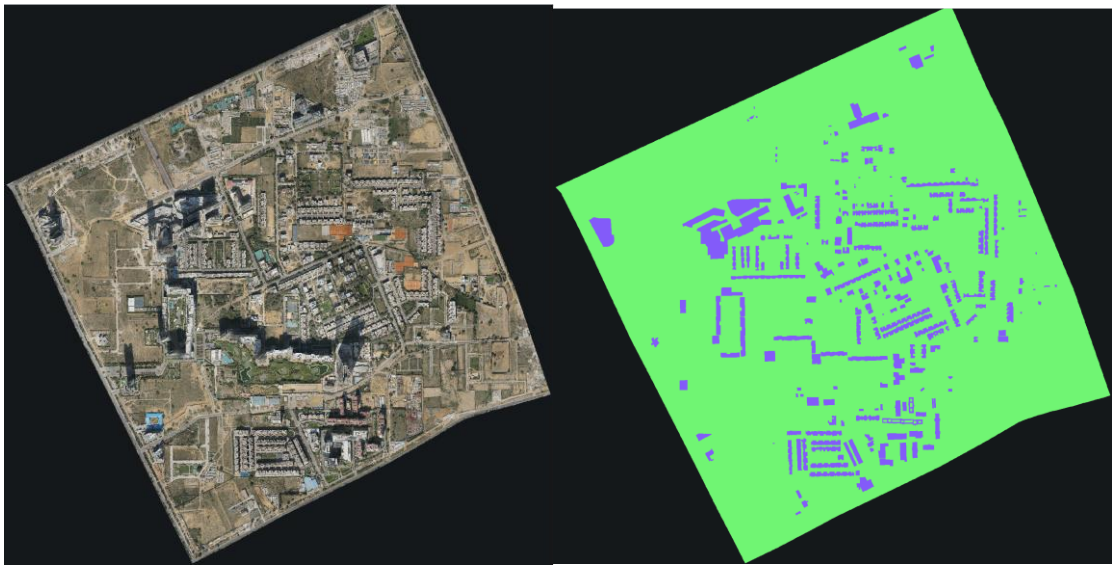
6) DATA ACQUISITION AND CLEANING

6.1 Data acquisition

Data acquisition in deep learning involves collecting and preparing data for training. This includes defining data requirements, collecting data from various sources, preprocessing the data, annotating it if necessary, splitting it into training/validation/test sets, augmenting the training data, and addressing class imbalance if present. Effective data acquisition is crucial for training accurate and robust deep learning models.

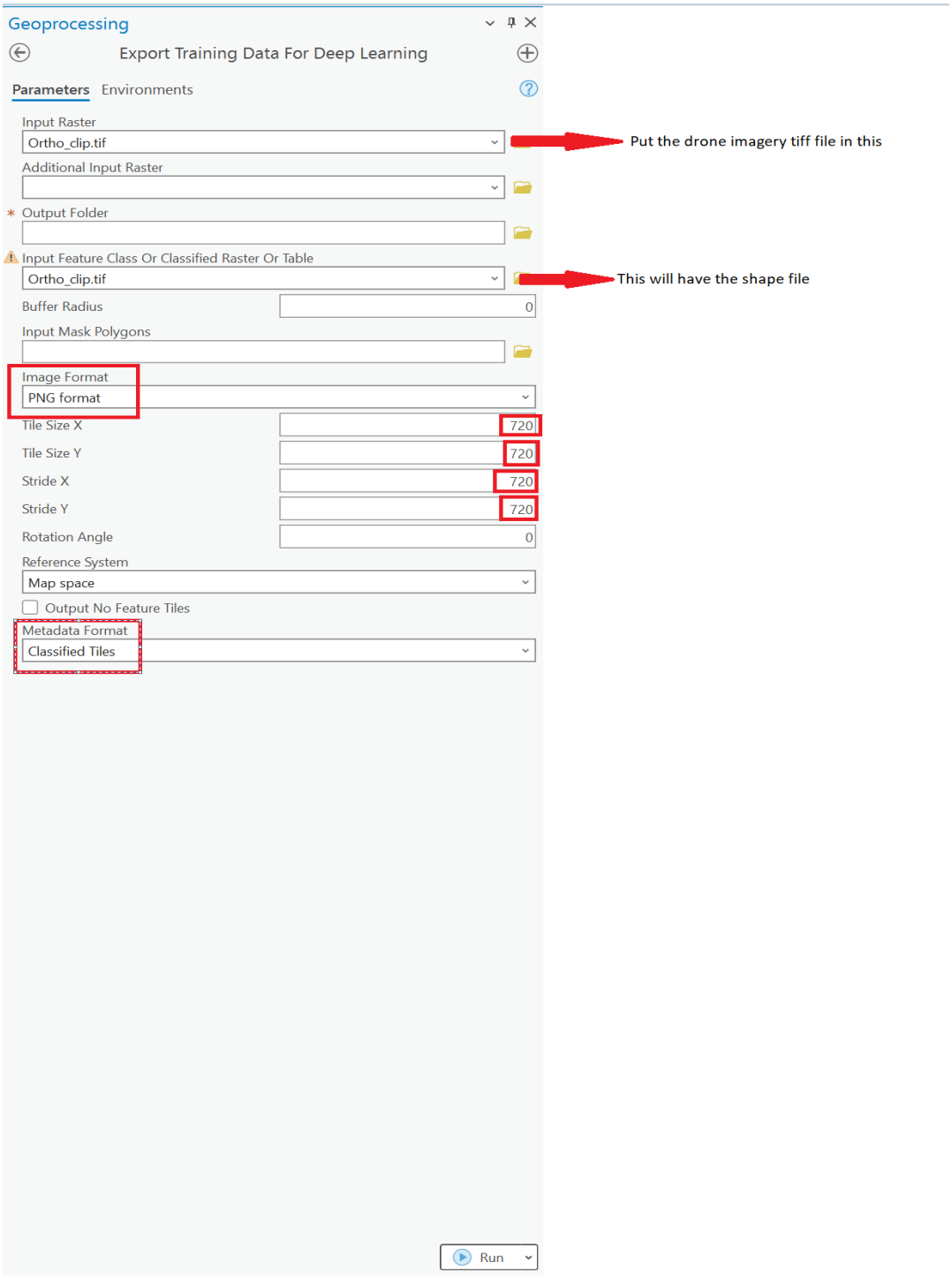
We have taken the drone imagery of Sector 65 Gurugram and its representative shape file that has all the buildings marked on it the following are the snippets of the images

Drone imagery of Sector 65: Shape file of sector 65:



Purple represents buildings, green represents the background

Next step is taking these two tiff files and putting it into ArcGIS pro tool named Export Training data for Deep Learning the parameters of the tools will be as follows



This will create png of 720x720 of the entire drone imagery along with the corresponding masks and make corresponding files that contain the geographical information.

6.2 Data Cleaning

Data cleaning is a critical step in the data preprocessing pipeline that focuses on improving the quality and reliability of a dataset. It involves identifying and addressing various issues such as missing values, outliers, inconsistencies, and duplicates. By performing data cleaning, we ensure that the dataset is accurate, complete, and consistent, which helps in obtaining reliable and meaningful insights from the data analysis or training accurate machine learning models.

For data cleaning we will only take images that have both building and background in the images and no images with null values in them these is possible through the following code snippet

Removing the images with Null values

```
test = []
bin = []
counter3 = 0
for i in range(len(labels)):
    x = [2] in labels[i] #this will filter the images with 0 in them and remove them
    if x is False:
        test.append(labels[i].astype('float32'))
    else:
        bin.append(i) #making a list of all the images with null values
        counter3 += 1
labels1 = np.array(test)
```

```
test2 = []
counter5 = 0
for i in range(len(train_images)):
#using the list above made to remove the images with null values
    if i in bin:
        counter5 += 1
    else:
        test2.append(train_images[i])
train_images1 = np.array(test2)
```

Removing the images with only background

```
#Only use this if you want pictures with both buildings and background
bin = []
test = []
for i in range(len(labels1)):
    x = [1] in labels1[i]
    if x is True:
        test.append(labels1[i].astype('float32'))
    else:
        bin.append(i)
        counter3 += 1
labels2 = np.array(test)
```

```
print(len(test))
print(len(bin))
ar2 = np.unique(labels2)
print(ar2)#doing some sanity checks to make sure all the data is getting used
```

```
368
511
[0. 1.]
```

```
test2 = []
counter5 = 0
for i in range(len(train_images1)):
#using the list above made to remove the images with null values
    if i in bin:
        counter5 += 1
    else:
        test2.append(train_images1[i])
train_images2 = np.array(test2)
```

7) DEFINING MODEL AND TRAINING THE MODEL

```python
def down_block(x, filters, kernel_size=(3, 3), padding="same", strides=1):
    c = keras.layers.Conv2D(filters, kernel_size, padding=padding, strides=strides, activation="relu")(x)
    c = keras.layers.Conv2D(filters, kernel_size, padding=padding, strides=strides, activation="relu")(c)
    p = keras.layers.MaxPool2D((2, 2), (2, 2))(c)
    return c, p

def up_block(x, skip, filters, kernel_size=(3, 3), padding="same", strides=1):
    us = keras.layers.UpSampling2D((2, 2))(x)
    concat = keras.layers.Concatenate()([us, skip])
    c = keras.layers.Conv2D(filters, kernel_size, padding=padding, strides=strides, activation="relu")(concat)
    c = keras.layers.Conv2D(filters, kernel_size, padding=padding, strides=strides, activation="relu")(c)
    return c

def bottleneck(x, filters, kernel_size=(3, 3), padding="same", strides=1):
    c = keras.layers.Conv2D(filters, kernel_size, padding=padding, strides=strides, activation="relu")(x)
    c = keras.layers.Conv2D(filters, kernel_size, padding=padding, strides=strides, activation="relu")(c)
    return c
```

```python
def UNet():
    f = [16, 32, 64, 128, 256]
    inputs = keras.layers.Input((patch_size, patch_size, 3))

    p0 = inputs
    c1, p1 = down_block(p0, f[0]) #128 -> 64
    c2, p2 = down_block(p1, f[1]) #64 -> 32
    c3, p3 = down_block(p2, f[2]) #32 -> 16
    c4, p4 = down_block(p3, f[3]) #16->8

    bn = bottleneck(p4, f[4])

    u1 = up_block(bn, c4, f[3]) #8 -> 16
    u2 = up_block(u1, c3, f[2]) #16 -> 32
    u3 = up_block(u2, c2, f[1]) #32 -> 64
    u4 = up_block(u3, c1, f[0]) #64 -> 128

    outputs = keras.layers.Conv2D(1, (1, 1), padding="same", activation="sigmoid")(u4)
    model = keras.models.Model(inputs, outputs)
    return model
```

```python
model = UNet()
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-5), loss= "binary_focal_crossentropy", metrics=["acc"])
model.summary()
```

```
history1 = model.fit(X_train, y_train,
                     batch_size = 16,
                     verbose=1,
                     epochs= 30,
                     validation_data=(X_test, y_test),
                     shuffle=False,
                     sample_weight=sample_weights)
```

The provided code defines a U-Net model for image segmentation using the Keras library. Here's a basic explanation of the code:

1. **down_block:** This function performs a down-sampling operation by applying two convolutional layers with ReLU activation and then a max-pooling layer.

2. **up_block:** This function performs an up-sampling operation by applying an upsampling layer, concatenating it with the corresponding skip connection from the down-sampling path, and then applying two convolutional layers with ReLU activation.

3. **bottleneck:** This function represents the bottleneck layer of the U-Net, where two convolutional layers with ReLU activation are applied.

4. **UNet:** This function defines the entire U-Net architecture. It specifies the input shape and defines the down-sampling and up-sampling paths using the previously defined functions. The final output is obtained by applying a convolutional layer with sigmoid activation.

5. **Model:** An instance of the U-Net model is created using the defined architecture.

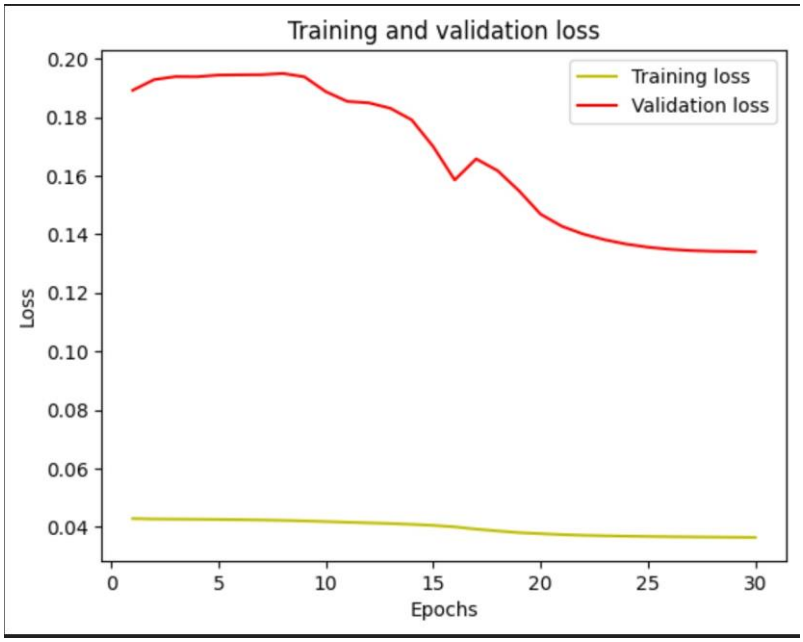
6. **model.compile:** The model is compiled with the Adam optimizer and the loss function is set to "binary_focal_crossentropy". The metric used for evaluation is accuracy.

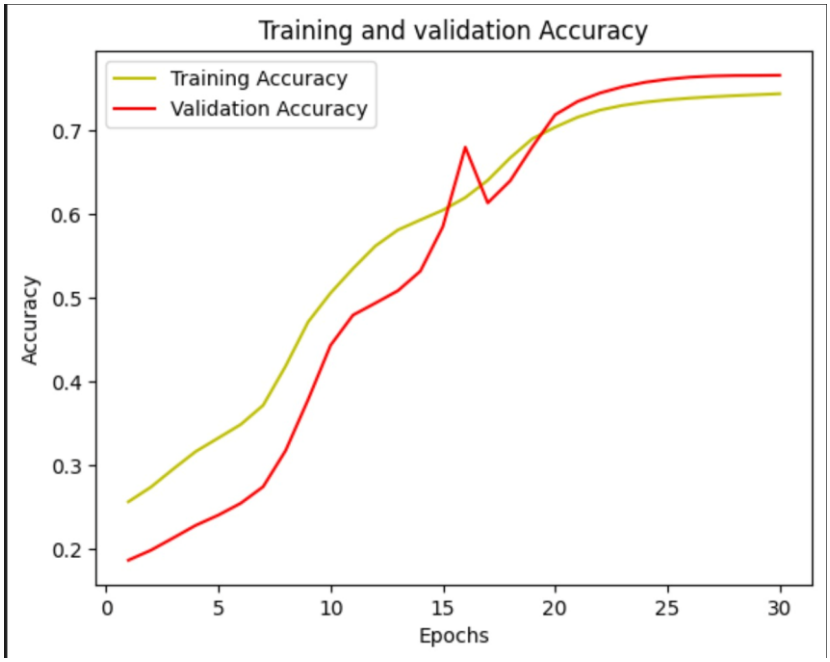
7. **model.summary:** This prints a summary of the model architecture, displaying the layers and the number of parameters.

8. **model.fit:** The model is trained using the fit function. It takes the training data, batch size, number of epochs, and validation data as input. The training process is executed and the history of training is stored in the `history1` variable.

In summary, the code defines a U-Net model, compiles it, and trains it on a given dataset for image segmentation. The U-Net architecture consists of down-sampling and up-sampling paths with skip connections to capture both high-level and low-level features. The model is trained using the Adam optimizer and the binary focal cross-entropy loss function, which is commonly used for imbalanced binary segmentation tasks.

The model Accuracy and loss:





8) APPLYING THE MODEL

Using keras tool load model we are going to call the model

```
model = load_model(os.path.join('models','inverse_model_unet.h5'))
```

Then using the predict function in keras library we are going to predict the output for each image

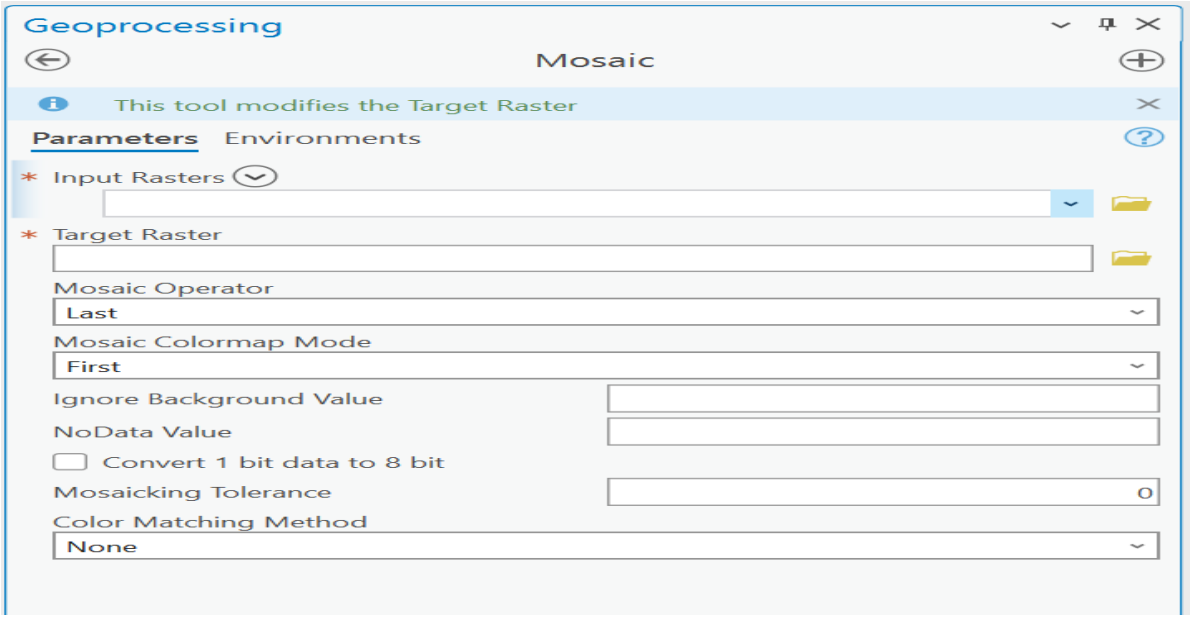
```
result = model.predict(train_images)
```

Now once the predictions are done we are going to export these predictions into a folder using the following snippet of code.

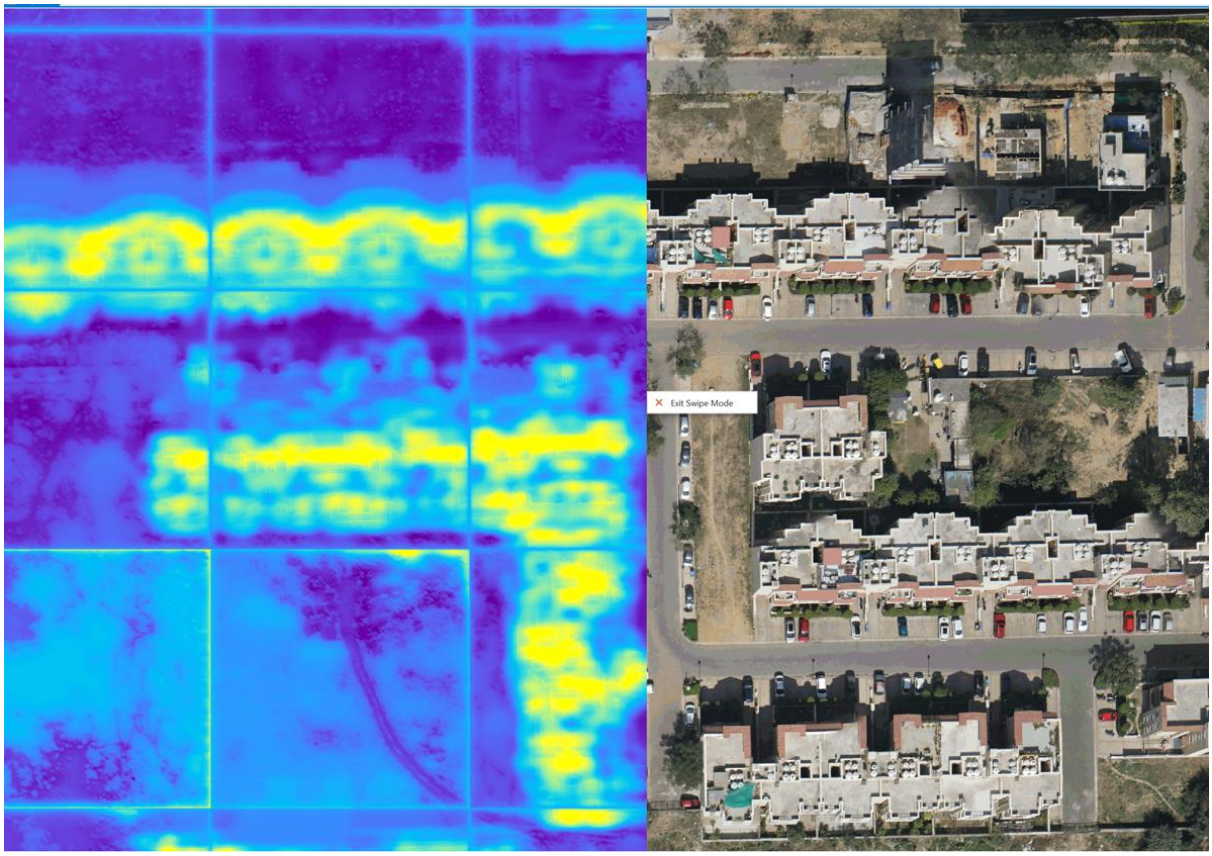
```
my_dpi = 144
for i in range(result.shape[0]):
    gun = (result[i]*255).astype(int)
    fig = plt.figure(figsize=(936/my_dpi, 936/my_dpi), dpi=my_dpi)
    fig = plt.imshow(gun)
    plt.axis('off')
    a = modified_list[i]
    plt.savefig(f'{a}.png', bbox_inches='tight', pad_inches = 0)
```

9) CHANGE DETECTION

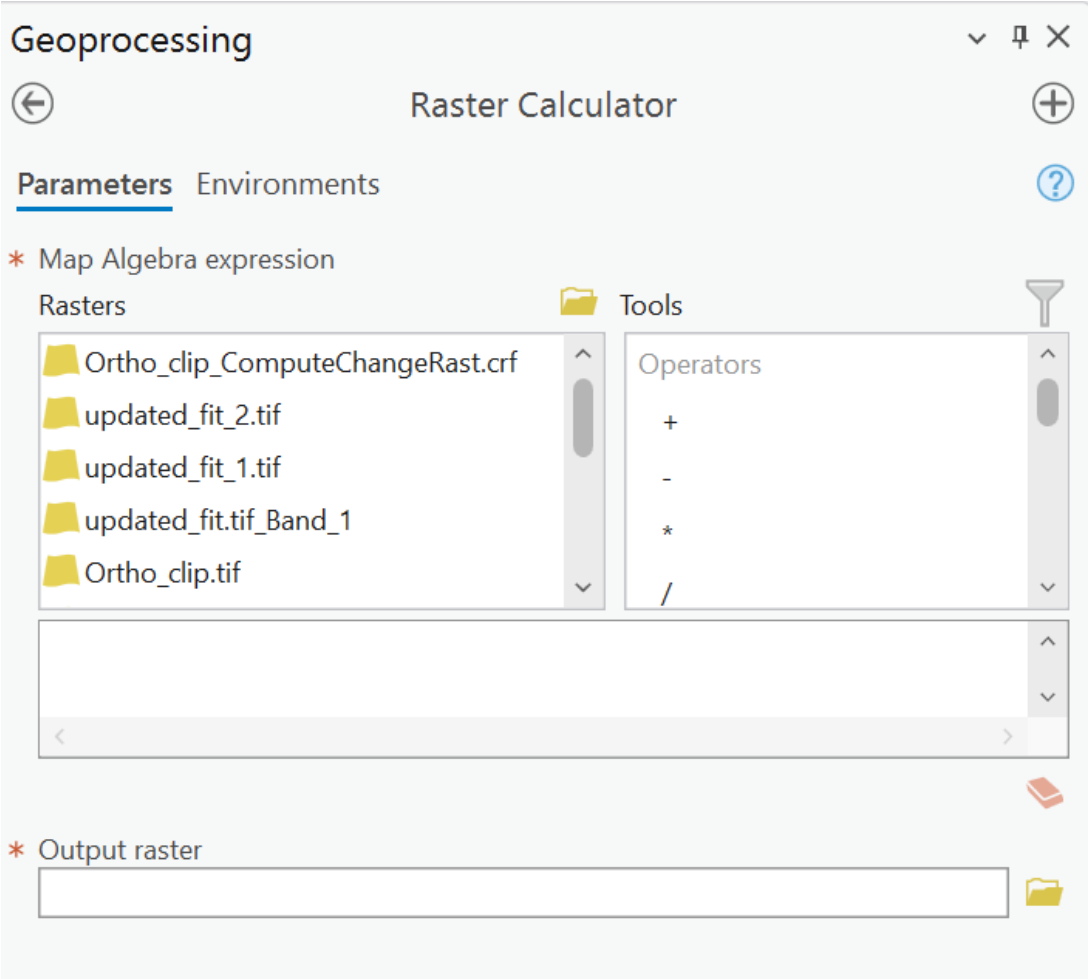
Now the next step is to take the output taken by the model and convert them into mosaic using the mosaic tool



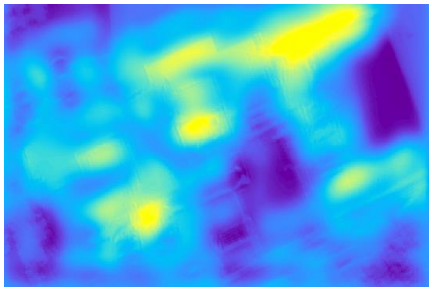
Once the mosaic is made the output will look like the below picture now you repeat this process for both the different imagery from different time.



Now once the mosaic for both the outputs is created, Using raster calculator we will subtract these outputs and we will be left only the buildings that weren't in the old image the below is a good representation.



For example the above building did not exist in 2018 drone imagery the model made this observation



10) REFERENCES

https://onemapggm.gmda.gov.in/

https://www.harsac.org/

https://www.gmda.gov.in/

Olaf Ronneberger, Philipp Fischer, Thomas Brox U-Net: Convolutional Networks for Biomedical Image Segmentation

Dosovitskiy, A., Springenberg, J.T., Riedmiller, M., Brox, T.: Discriminative unsupervised feature learning with convolutional neural networks. In: NIPS (2014)

Girshick, R., Donahue, J., Darrell, T., Malik, J.: Rich feature hierarchies for accurate object detection and semantic segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2014)