

Semantic Segmentation of land By Using U-NET

Submitted in partial fulfillment of the requirements for the award of Bachelor of Technology Degree in Information Technology

By

Leelavathi.V (Reg.No. 38120045)

Sabithra.D (Reg. No. 38120070)



**DEPARTMENT OF INFORMATION TECHNOLOGY
SCHOOL OF COMPUTING**

SATHYABAMA

**INSTITUTE OF SCIENCE AND TECHNOLOGY
(Deemed to be University)
Accredited with Grade "A" by NAAC**

JEPPIAAR NAGAR, RAJIV GANDHI SALAI, CHENNAI-600119, TAMILNADU

SATHYABAMA

INSTITUTE OF SCIENCE AND TECHNOLOGY

(DEEMED TO BE UNIVERSITY)

Accredited with Grade “A” by NAAC

(Established under section 3 of UGC Act ,1956)

JEPPIAAR NAGAR,RAJIV GANDHI SALAI,CHENNAI-

600119www.sathyabama.ac.in

DEPARTMENT OF INFORMATION TECHNOLOGY

BONAFIDE CERTIFICATE

This is to certify that this Project Report is the bonafide work of **Leelavathi.V(38120045),Sabithra.D (38120070)** who carried out the project entitled **“Semantic Segmentation of land By Using U-NET”** under our supervision from November 2021 to November 2022.

Internal Guide

Dr.Y.BevishJinila, M.E.,Ph.D,

Head of the Department

Dr.R.Subhashini, M.E., Ph.D.,

Submitted for Viva voce Examination held on_____

Internal Examiner

External Examiner

ACKNOWLEDGEMENT

I am pleased to acknowledge my sincere thanks to Board of Management **SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY** for their kind encouragement in doing this project and for completing it successfully. I am grateful to them.

I convey my thanks to **Dr. T. SASIKALA, M.E., Ph.D., Dean**, School of Computing and **Dr.R.SUBHASHINI M.E., Ph.D.**, Head of the Department, Dept.of Information Technology for providing me necessary support and details at the right time during the progressive reviews.

I would like to express my sincere and deep sense of gratitude to my Project Guide **DR.Y.BEVISHJINILA, M.E.,Ph.D**, for his valuable guidance, suggestions and constant encouragement paved way for the successful completion of my project work.

I wish to express my thanks to all Teaching and Non-teaching staff members of the **Department of Information Technology** who were helpful in many ways for the completion of the project.

DECLARATION

I, **Leelavathi.V (Reg.no:38120045)** and **Sabithra.D (Reg.No:38120070)** hereby declare that the Project Report entitled on “**SEMANTIC SEGMENTATION OF LAND BY USINGU-NET**” done by me under the guidance of **Dr.Y.BevisJinila, M.E.,Ph.D**, at Sathyabama university is submitted in partial fulfilment of the requirements for the award of Bachelor of Engineering degree in Information Technology.

DATE:

PLACE: CHENNAI

SIGNATURE OF THE CANDIDATE

ABSTRACT

Land-cover classification is one of the core applications in the field of remote sensing. It is a valuable resource for city planners to achieve sustainable development. Many metropolitan cities are experiencing disorganized growth with a high intensity of urban sprawl due to the economic pull and better standards of living offered in metropolitan cities when compared to the surrounding rural areas. If this pattern of growth continues, it will lead to unsustainable development. This leads to an increase in pressure on urban infrastructure and the ecosystem. The traditional methods which are used for urban mapping are time consuming. Land Cover Semantic Segmentation is a process of developing an automated system for land cover classification. This system takes a multiband satellite image of an area as input and outputs the land cover map of the area at the same resolution as the input. For this purpose UNET models were trained in the task of predicting the land cover semantic segmentation of satellite images.

TABLE OF CONTENTS

CHAPTER NO	TITLE NAME	PAGENO
	ABSTRACT	I
	TABLE OF CONTENTS	II
1	INTRODUCTION	
	1.1 General	1
	1.2 Motivation	1
	1.3 Outline of the project	2
	1.4 Objective of the project	3
2	DOMAIN	
	2.1 Domain description	4
3	AIM AND SCOPE	
	3.1 Aim	2
	3.2 Scope	3
4	ALGORITHMS AND DATASETS	
	4.1 Algorithm U-NET	5
	4.1.1 Application of U-Net	7
	4.1.2 Residual Network	7
	4.2 Data Sets	8
	4.2.1 Classes	10
5	SYSTEM REQUIREMENT	
	5.1 Methodology	

5.1.1 Hardware requirement	11
5.1.2 Software requirement	11
5.2 Software Description	
5.2.1 Google Colab	12
5.2.2 Libraries Used in Project	13

6 SYSTEM DESIGN

6.1 System architecture	16
6.2 Uml diagrams	16
6.3 Use case diagram	17
6.4 Sequence diagram	18
6.5 Activity diagram	19

7 RESULT AND PERFORMANCE ANALYSIS

7.1 Module implementation	20
---------------------------	----

8 CONCLUSION AND FUTURE WORK

8.1 Conclusion	22
8.2 Future Work	22

9 REFERENCES

10 APPENDIX

11 SOURCE CODE

12 OUTPUT

CHAPTER 1

INTRODUCTION

1.1 General

Land cover Segmentation play a major role in land management applications and it is a important role in image understanding and essential for image analysis. Also its Using earth observation satellites both facilitate and accelerate the task. Lately, deep learning methods have proven to be excellent at automating the mapping via semantic image segmentation. However, because deep neural networks require large amounts of labeled data, it is not easy to exploit the full potential of satellite imagery. Additionally, the land cover tends to differ in appearance from one region to another; therefore, having labeled data from one location does not necessarily help in mapping others. Furthermore, satellite images come in various multispectral bands Developing a model which accurately predicts the different types of land covering to find water areas, urban areas in earth surface and is of particular and important for monitoring environmental changes as well as the growth of human settlements in the environment Semantic segmentation is the process of inferring the progression from coarse to fine details in a scene. The process begins by performing classification which makes a prediction of the input.

1.2 MOTIVATION

This often requires experts to spend a lot of time watching satellite images and trying to label the land type.

Instead a model could be trained using labels from experts so that it can predict automatically the type of land presented in the satellite images.

1.3 OUTLINE OF THE PROJECT

The project is all about segmenting the land this project helps to classifying

the land areas, water areas, urban areas etc..we are using U-Net algorithm with backbone of resnet34 for land segmentation proposed framework are dataset collection, data preprocessing, feature extraction, training and testing the UNET and RESNET34 model and finally assessment of the accuracy

1.4 OBJECTIVE OF THE PROJECT

- The objective of the project is to build a automated system for land cover segmentation.
- This system takes multiband satellite image of an areas input and output the land cover map of the area at the same resolution as the input.
- Accurately predicts the different types of land covering is of the particular important for monitoring environmental changes as well as the growth of human settlements in the environment

CHAPTER 2

AIM AND SCOPE

AIM

The aim of our project is to change the representation of an image into something that is more meaningful and easier to analyze and Developing a model that accurately predicts the various types of land cover segmentation is critical for tracking environmental changes as well as the enhancement of human settlements in the environment.

SCOPE

- Extracts information from images and integrates it for several application
- Better for noisy images where edges are hard to identify

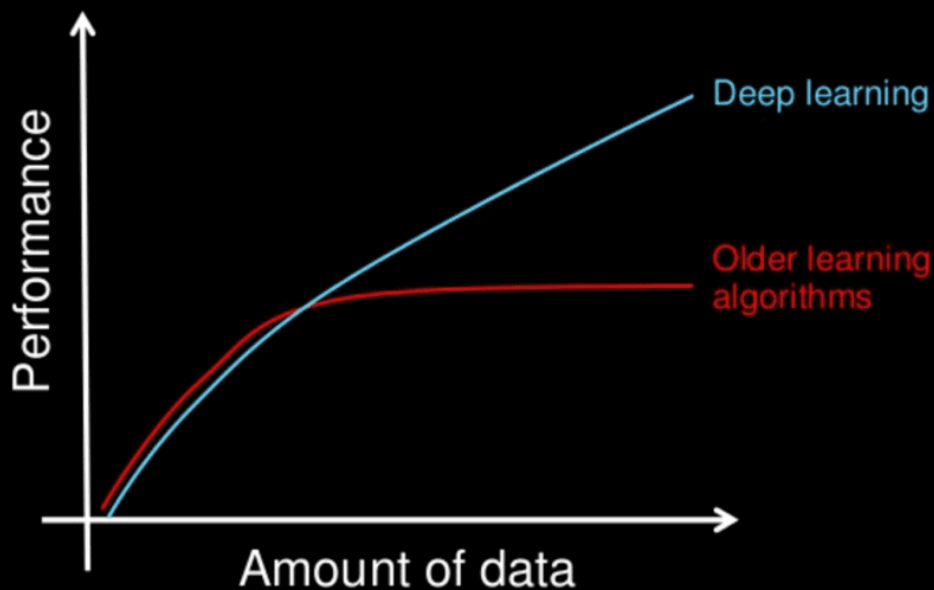
CHAPTER-3

3.1 DOMAIN DESCRIPTION

3.1.1 DEEP LEARNING

Deep Learning is a subfield of machine learning concerned with algorithms inspired by the structure and function of the brain called **artificial neural networks**. If you are just starting out in the field of deep learning or you had some experience with neural networks some time ago, you may be confused. I know I was confused initially and so were many of my colleagues and friends who learned and used neural networks in the 1990s and early 2000s. The leaders and experts in the field have ideas of what deep learning is and these specific and nuanced perspectives shed a lot of light on what deep learning is all about.

Why deep learning



How do data science techniques scale with amount of data?

Deep learning neural networks, or artificial neural networks, attempts to mimic the human brain through a combination of data inputs, weights, and bias. These elements work together to accurately recognize, classify, and describe objects within the data.

Deep neural networks consist of multiple layers of interconnected nodes, each building upon the previous layer to refine and optimize the prediction or categorization. This progression of computations through the network is called forward propagation. The input and output layers of a deep neural network are called *visible* layers. The input layer is where the deep learning model ingests the data for processing, and the output layer is where the final prediction or classification is made.

Another process called backpropagation uses algorithms, like gradient descent, to calculate errors in predictions and then adjusts the weights and biases of the function by moving backwards through the layers in an effort to train the model. Together, forward propagation and backpropagation allow a neural network to make predictions and correct for any errors accordingly. Over time, the algorithm becomes gradually more accurate.

The above describes the simplest type of deep neural network in the simplest terms. However, deep learning algorithms are incredibly complex, and there are different types of neural networks to address specific problems or datasets. For example,

- *Convolutional neural networks (CNNs)*, used primarily in computer vision and image classification applications, can detect features and patterns within an image, enabling tasks, like object detection or recognition. In 2015, a CNN bested a human in an object recognition challenge for the first time.
- *Recurrent neural network (RNNs)* are typically used in natural language and speech recognition applications as it leverages sequential or times series data.

CHAPTER 4

ALGORITHM AND METHODS

4.1 ALGORITHM

4.1.1 U-Net

U-Net is a [convolutional neural network](#) and This network is based on the fully convolutional network and its architecture was modified and extended to work with fewer training images and to yield more precise segmentations. Segmentation of a 512×512 image takes less than a second on a modern [GPU](#). The main idea is to supplement a usual contracting network by successive layers, where pooling operations are replaced by [upsampling](#) operators. Hence these layers increase the resolution of the output. A successive convolutional layer can then learn to assemble a precise output based on this information

One important modification in **U-Net** is that there are a large number of feature channels in the upsampling part, which allow the network to propagate context information to higher resolution layers. As a consequence, the expansive path is more or less symmetric to the contracting part, and yields a u-shaped architecture. The network only uses the valid part of each [convolution](#) without any fully connected layers. To predict the pixels in the border region of the image, the missing context is extrapolated by mirroring the input image. This tiling strategy is important to apply the network to large images, since otherwise the resolution would be limited by the [GPU](#) memory.

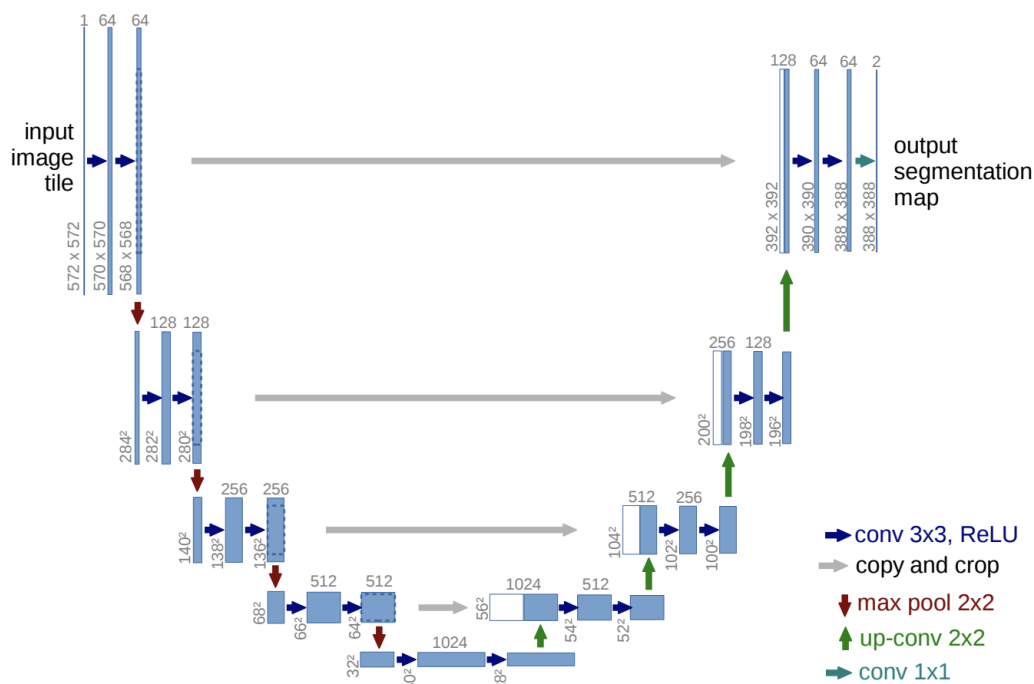


Fig. 1. U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

U-Net is an architecture for semantic segmentation. It consists of a contracting path and an expansive path. The contracting path follows the typical architecture of a convolutional network. It consists of the repeated application of two 3x3 convolutions (unpadded convolutions), each followed by a rectified linear unit ([ReLU](#)) and a 2x2 [max pooling](#) operation with stride 2 for downsampling.

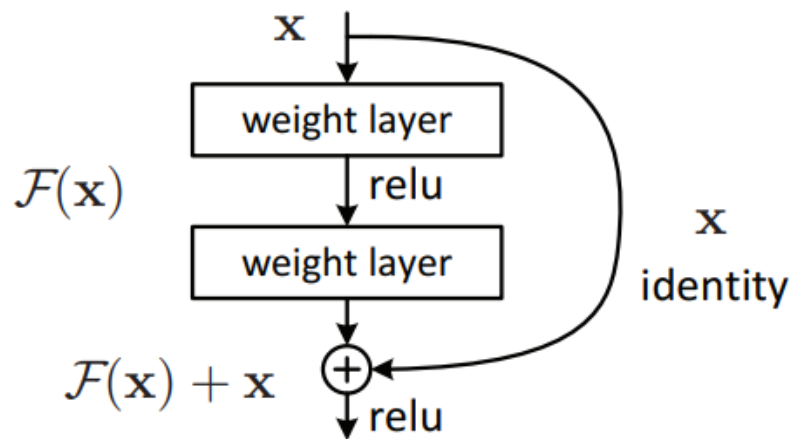
Every step in the expansive path consists of an upsampling of the feature map followed by a 2x2 [convolution](#) (“up-convolution”) that halves the number of feature channels, a concatenation with the correspondingly cropped feature map from the contracting path, and two 3x3 convolutions, each followed by a ReLU. The cropping is necessary due to the loss of border pixels in every convolution. At the final layer a [1x1 convolution](#) is used to map each 64-component feature vector to the desired number of classes. In total the network has 23 convolutional layers.

4.1.2 Applications of unet

1. Pixel-wise regression using U-Net and its application on pansharpening
2. 3D U-Net: Learning Dense Volumetric Segmentation from Sparse Annotation
3. TerausNet: U-Net with VGG11 Encoder Pre-Trained on ImageNet for Image Segmentation.
4. Image-to-image translation to estimate fluorescent stains
5. In binding site prediction of protein structure.

4.1.3 RESIDUAL NETWORKS

In order to solve the problem of the vanishing/exploding gradient, this architecture introduced the concept called Residual Network. In this network we use a technique called ***skip connections*** . The skip connection skips training from a few layers and connects directly to the output the approach behind this network is instead of layer learn the underlying mapping, we allow network fit the residual mapping. So, instead of say $H(x)$, initial mapping, let the network fit, $F(x) := H(x) - x$ which gives $H(x) := F(x) + x$.



The advantage of adding this type of skip connection is because if any layer hurt the performance of architecture then it will be skipped by regularization. So, this results in training very deep neural network without the problems caused by vanishing/exploding gradient.

4.2.DATA SET

The Segmentation A dataset is a collection of data used to automatically map buildings, woodlands, water, and roadways from aerial images. We chose to make a simple RGB-only dataset that was fully manually annotated and large enough to train a model. Exact segmentation this dataset collect from Kaggle

The classes are: 6 large tiles

1. Building: #3C3098
2. Land (unpaved area): #ff96ff
3. Road: #ffe65a
4. Vegetation: #00a000
5. Water: #0000ff
6. Unlabeled: #000000

This dataset consists of img- data select from satellite data used to developed the digital segmenting the In addition to NASA's Land-sat Mission, the(Esa) European Space Agency it is stepping up try to develop the Earth analysis through its Copernicus programme. The European Space Agency (ESA) performs a series of satellites is called as Sentinels under this programme. In this work, we use Sentinel-2 satellite multi - spectral data to address the challenge of cover Segmentation .Sentinel-2A is one of two satellites in the Sentinel-2A and Sentinel-2B land monitoring satellite constellation. The satellites were launched greatfully in June 2015 (Sentinel2A) and March 2017. (Sentinel-2B). Both sun-synchronous satellites use a Multispectral Imager (MSI) to capture the Earth's surface in the 13 spectral bands . The 3 bands B10, B01, and B09 are designed to be used for earth's atmosphere impact correction The residual bands are usually used to find out and monitor different types of land cover. The two satellites cover large islands, and also inland and coastal bodies of water, in addition to the mainland. Each satellite will provide imagery with such a spatial high- resolution of up to ten meter, per pixel for at least seven years. All satellites carry enough fuel for up to 12 years of operation, allowing the mission to be extended. The 2 cluster generates protection of almost the whole Earth's surface each 5 days, which means that the satellites capture every small point in the zone every five days. This quick routine process, combined with the Sentinel satellites' vision task, allows for continuously observing of the Land surface of the earth for the next 20 - 30 years. Most essientely, the data is open and freely available for use in any usage like the commercial and the non-commercial use

To create an image classification dataset, we went through the following two steps:

1) Aerial Image Gathering: we acquired aerial data images of cities in 34 European countries.

2) Dataset Generation: Using the aerial data images obtained, we generated a data of 27,000 geo-reference and labelled data-patches.The data patche are 64x64 pixels in size and have been checked by hand., Germany, Belarus, Netherlands, Spain, Bulgaria, Denmark, Cyprus, , Portuga Czech Republic (Czechia), Romania Estonia, Iceland, France, Austria Germany, ,Slovenia, Greece,

Hungary, Finland, Ireland, Belgium Latvia, Sweden, Lithuania, United Kingdom, Luxembourg, Ukraine, Macedonia, Italy / Holy See Malta, Republic of Moldova, Norway, Slovakia, Poland, and Switzerland are among the 34 European countries covered.

4.2.1 CLASSES:

The images are densely labeled and contain the following 6 classes:

- Building: A structure is an object that is permanently erected in one location. Greenhouses are not permitted. Because our images aren't real orthophotos, each structure is labelled as a roof.
- Woodland - An area covered in trees that are close together. Orchards and single trees are not included.
- Water includes both flowing and stagnant water, such as ponds and pools. Dry riverbeds and ditches are not included.
- Road - The infrastructure that supports road and rail transportation, including parking and dirt roads.
- Background - An area that is not classified into any of the classes. It can include things like fields, grass, and pavements, with all other objects being omitted.

CHAPTER -5

SYSTEM REQUIREMENT

5.1.1 HARDWARE REQUIREMENTS

The most common set of requirements defined by any operating system or software application is the physical computer resources, also known as hardware. A hardware requirements list is often accompanied by a hardware compatibility list, especially in case of operating systems. The minimal hardware requirements are as follows,

- 1. PROCESSOR : PENTIUM IV**
- 2. RAM : 8 GB**
- 3. PROCESSOR : 2.4 GHZ**
- 4. MAIN MEMORY : 8GB RAM**
- 5. PROCESSING SPEED : 600 MHZ**
- 6. HARD DISK DRIVE : 1TB**
- 7. KEYBOARD :104 KEYS**

5.1.2 SOFTWARE REQUIREMENTS

Software requirements deals with defining resource requirements and prerequisites that needs to be installed on a computer to provide functioning of an application. These requirements are need to be installed separately before the software is installed. The minimal software requirements are as follows,

- 1. OPERATING SYSTEM :WINDOWS 10**
- 2. GOOGLE COLAB**
- 3.PYTHON**
- 4.MATPLOTLIB**
- 5.NUMPY**
- 6.PILLOW**
- 7.TORCH**
- 8.TENSORFLOW**
- 9.KERA 11**

5.2 SOFTWARE DESCRIPTION

5.2.1 GOOGLE COLAB

Nowadays, machine learning and deep learning has become the hottest trend of the Computer Science industry. Many students are trying to learn and build amazing projects with it.

Google Colab was developed by Google to provide free access to GPU's and TPU's to anyone who needs them to build a machine learning or deep learning model. Google Colab can be defined as an improved version of Jupyter Notebook.

Google Colab Features

Google Colab provides tons of exciting features that any modern IDE offers, and much more. Some of the most exciting features are listed below.

- Interactive tutorials to learn machine learning and neural networks.
- Write and execute Python 3 code without having a local setup.
- Execute terminal commands from the Notebook.
- Import datasets from external sources such as Kaggle.
- Save your Notebooks to Google Drive.
- Import Notebooks from Google Drive.
- Free cloud service, GPUs and TPUs.
- Integrate with PyTorch, Tensor Flow, Open CV.
- Import or publish directly from/to GitHub.

- **Pre-installed libraries:**

Google Colab comes pre-installed with the most popular machine learning libraries. Colab comes pre-installed with Keras, PyTorch, TensorFlow, which saves you the time and hassle of setting up a local environment.

- **Saved on the cloud :**

Every Notebook you create in the Google GoogleColab is saved on the cloud. This lets you access and work with those Notebooks from any machine. All you need is a browser and a reliable network connection, and you can work from anywhere and anytime.

- **Collaboration :**

Collaboration is another amazing reason to choose Google GoogleColab when you are working on a project with a team of developers. You can share your Notebook with your teammates and assign them roles so that they can only perform operations that fit their roles. The various options available for each role is shown below:

– Editors can change permissions and share – Viewers and commenters can see the option to download, print, and copy

- **Free GPU and TPU use :**

Google Colab provides free access to GPUs and TPUs developed by Google Research. So you can work on your personal projects with powerful GPUs irrespective of your local machine.

5.2.2.Libraries used

PYTHON

Python is a widely used general-purpose, high level programming language. It was mainly developed for emphasis on code readability, and its syntax allows programmers to express concepts in fewer lines of code.

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

PYTHON FEATURES:

Python's features include –

- **Easy-to-learn** – Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- **Easy-to-read** – Python code is more clearly defined and visible to the eyes.
- **Easy-to-maintain** – Python's source code is fairly easy-to-maintain.
- **A broad standard library** – Python's bulk of the library is very portable and crossplatform compatible on UNIX, Windows, and Macintosh.
- **Interactive Mode** – Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- **Portable** – Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **GUI Programming** – Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- **Scalable** – Python provides a better structure and support for large programs than shell scripting.

MATPLOTLIB

Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. Matplotlib makes easy things easy and hard things possible.

Matplotlib is a [plotting library](#) for the [Python](#) programming language

- Create [publication quality plots](#).
- Make [interactive figures](#) that can zoom, pan, update.
- Customize [visual style](#) and [layout](#).
- Export to [many file formats](#) .
- Embed in [JupyterLab and Graphical User Interfaces](#).
- Use a rich array of [third-party packages](#) built on Matplotlib.

NUMPY

NumPy is the fundamental package for scientific computing in Python. It is a Python library that provides a multidimensional array object, various derived objects (such as masked arrays and matrices), and an assortment of routines for fast operations on arrays, including mathematical, logical, shape manipulation, sorting, selecting, I/O, discrete Fourier transforms, basic linear algebra, basic statistical operations, random simulation and much more. At the core of the NumPy package, is the *ndarray* object. This encapsulates *n*-dimensional arrays of homogeneous data types, with many operations being performed in compiled code for performance.

Pytorch

PyTorch is an open source machine learning (ML) framework based on the Python programming language and the Torch library. It is one of the preferred platforms for deep learning research. The framework is built to speed up the process between research prototyping and deployment. PyTorch is similar to NumPy and computes using tensors that are accelerated by graphics processing units (GPU). Tensors are arrays, a type of multidimensional data structure, that can be operated on and manipulated with APIs. The PyTorch framework supports over 200 different mathematical operations.

TENSORFLOW

TensorFlow is an open source library for fast numerical computing. It was created and is maintained by Google and released under the Apache 2.0 open source license. The API is nominally for the Python programming language, although there is access to the underlying C++ API. Unlike other numerical libraries intended for use in Deep Learning like Theano, TensorFlow was designed for use both in research and development and in production systems. It can run on single CPU systems, GPUs as well as mobile devices and large scale distributed systems of hundreds of machine.

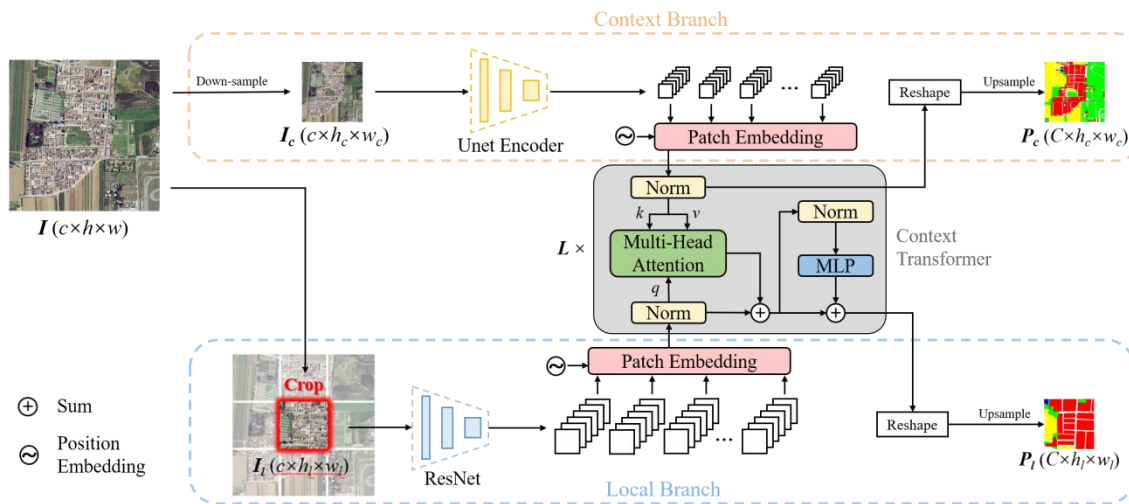
KERAS

Keras is a high-level, deep learning API developed by Google for implementing neural networks. It is written in Python and is used to make the implementation of neural networks easy. It also supports multiple backend neural network computation. Keras is relatively easy to learn and work with because it provides a python frontend with a high level of abstraction while having the option of multiple back-ends for computation purposes. This makes Keras slower than other deep learning frameworks, but extremely beginner-friendly.

CHAPTER 6

SYSTEM DESIGN

6.1 SYSTEM ARCHITECTURE



6.2 UML DIAGRAMS

UML stands for Unified Modeling Language. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group.

The goal is for UML to become a common language for creating models of object oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML.

The Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems.

The UML represents a collection of best engineering practices that have proven successful in the modeling of large and complex systems.

The UML is a very important part of developing objects oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

GOALS:

The Primary goals in the design of the UML are as follows:

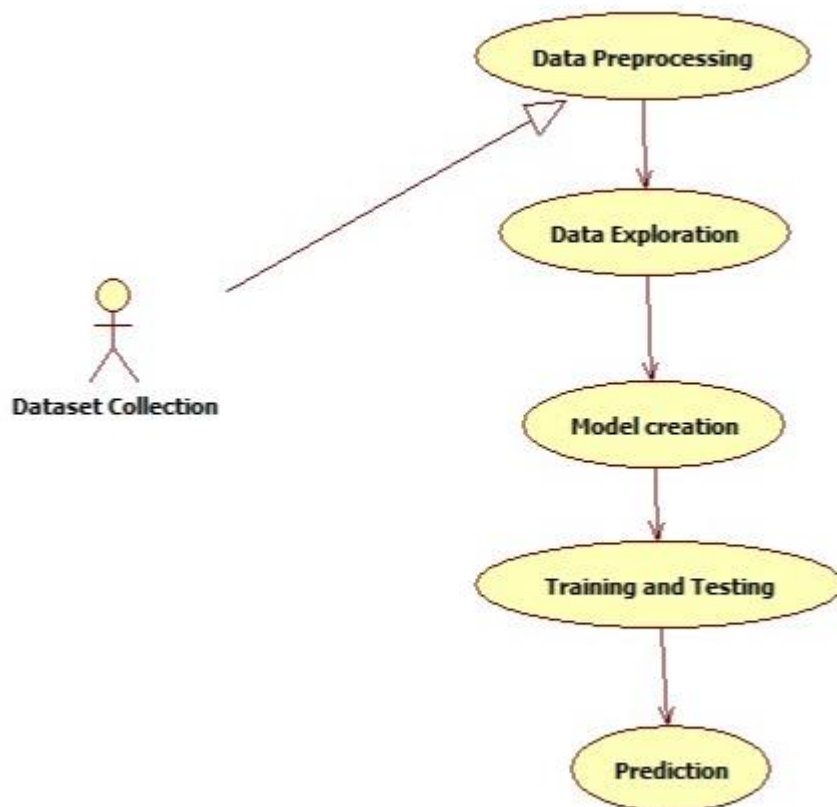
1. Provide users a ready-to-use, expressive visual modeling Language so that they can develop and exchange meaningful models.
2. Provide extendibility and specialization mechanisms to extend the core concepts.
3. Be independent of particular programming languages and development process.
4. Provide a formal basis for understanding the modeling language.
5. Encourage the growth of OO tools market.
6. Support higher level development concepts such as collaborations, frameworks, patterns and components.
7. Integrate best practices.

6.3 USE CASE DIAGRAM:

A use case diagram in the Unified Modeling Language (UML) is a type of behavioral diagram defined by and created from a Use-case analysis. Its purpose

is to present a graphical overview of the functionality provided by a system in

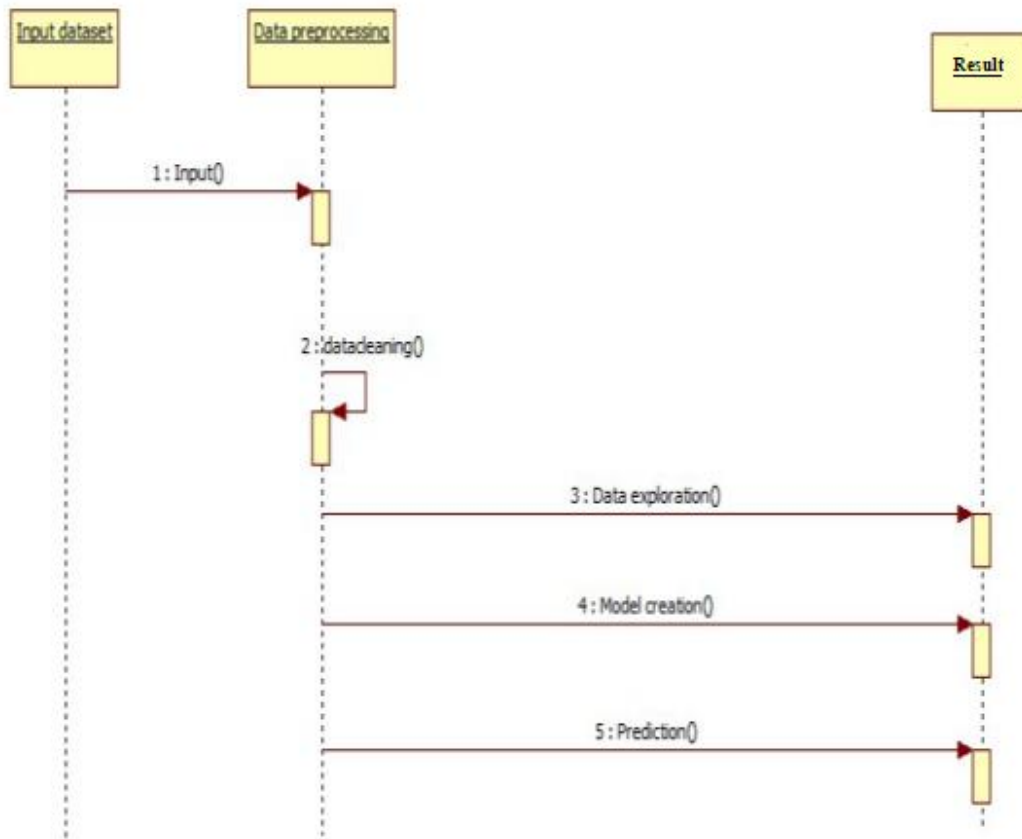
terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to show what system functions are performed for which actor. Roles of the actors in the system can be depicted.



6.4 SEQUENCE DIAGRAM:

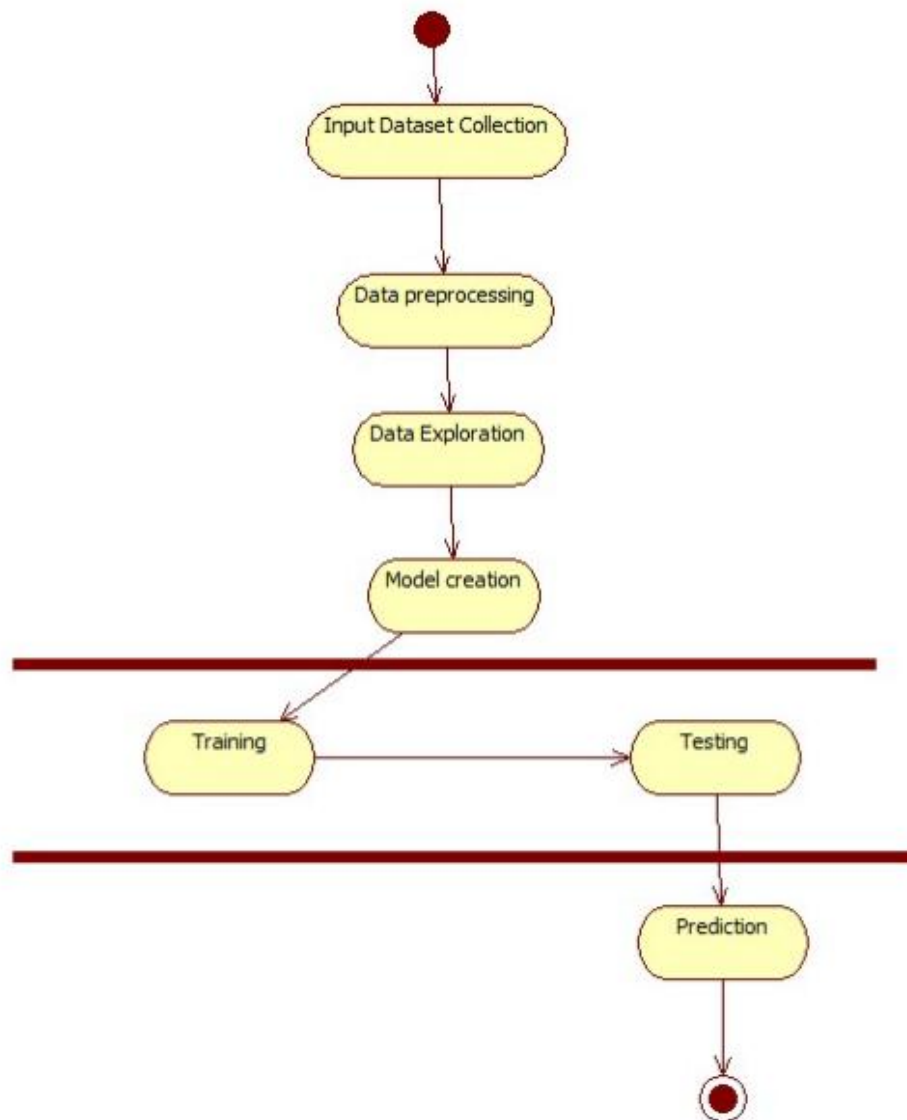
A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in

what order. It is a construct of a Message Sequence Chart. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams



6.5 ACTIVITY DIAGRAM

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modeling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.



CHAPTER 7

RESULTS AND PERFORMANCE ANALYSIS

7.1 MODULE IMPLEMENTATION

Pre-Processing

Preprocessing is a technique that is used to convert the raw data into a clean data set. In other words, whenever the data is gathered from different sources it is

20

collected in raw format which is not feasible for the analysis.

Segmentation

It involves dividing a visual input into segments to simplify image analysis. Segments represent objects or parts of objects, and comprise sets of pixels.

Image segmentation is typically used to locate objects and boundaries (lines, curves, etc.) in images. More precisely, image segmentation is the process of assigning a label to every pixel in an image such that pixels with the same label share certain characteristics.

Feature Extraction

Feature extraction is a process of dimensionality reduction by which an initial set of raw data is reduced to more manageable groups for processing. A characteristic of these large data sets is a large number of variables that require a lot of computing resources to process. Feature extraction is the name for methods that select and /or combine variables into features, effectively reducing the amount of data that must be processed, while still accurately and completely describing the original data set.

Building and Training Model:

After feature selection location and month attribute are used for training. The dataset is divided into pair of x_{train} , y_{train} and x_{test} , y_{test} . The algorithm's model is imported from sklearn. Building model is done using `model.fit(xtrain, ytrain)`. This phase would involve supervised classification methods like linear regression,

Ensemble classifiers (like Adaboost, Random Forest Classifiers), etc. We used a k-fold cross-validation method to test the network performance [24]. Two different

approaches were implemented, and both consisted of 10-fold cross-validation. The first approach was to randomly divide the data into 10 approximately equal portions so that each tumor category was equally present in each portion, referred to as

record-wise cross-validation. The second approach was to randomly divide the data into 10 approximately equal portions where the data from a single subject could only

21

be found in one of the sets. Each set, therefore, contained data from a couple of subjects regardless of the tumor class, referred to as subject-wise cross-validation. The second approach was implemented to test the generalization capability of the network in medical diagnostics. The generalization capability in clinical practice represents the ability to predict the diagnosis based on the data obtained from subjects from which there are no observations in the training process. Therefore, observations from individuals in the training set must not appear in the test set. If this is not the case, complex predictors can pick up a confounding relationship between identity and diagnostic status and so produce unrealistically high prediction accuracy. In order to compare the performance of our network with other state-of-the-art methods, we also tested our network without k-fold cross-validation (one test). In all the above-mentioned methods, two data portions were used for the test, two for validation, and six for training. Both datasets, normal and augmented, were tested using all the methods. The network was trained using an Adam optimizer, with a mini-batch size equal to 16 and data shuffling in every iteration.

CHAPTER 8

CONCLUSION AND FUTURE WORK

8.1 CONCLUSION

In this work we proposed a network for land cover segmentation from high-spatial resolution satellite image. The proposed network combines the benefits of deep learning models and has the following key features

The network learns low-level features and high-level contexts by replacing the contraction path of U-Net with a stack of dense blocks.

The skip connections of the network enable effective fusion, which aggregates the features map on multiple scales.

8.2 FUTURE WORK

REFERENCES

- [1]. Sebastian Wuttke-Wolfgang middleman-Uwe Stilla' Improving the efficiency of land cover classification by combining segmentation, Hirarchical clustering and Active learning",*IEEEJournalof selected topics in applied earth observations and remote sensing*,2018
- [2]. Pourya Shamsolmoali-masoumeh Zareapoor-Ruili Wang-Huiyu Zhou-Jie Yang,"A Novel Deep Structure U-NET For Sea-land Segmentation in remote sensing images",*IEEE Journal of selected topics in applied earth observations and remote sensing*,2019
- [3]. Zongolan Zhan-Xiaomeng Zhang-Yi Liu-Xiao Sun -Chao Pang-Chenbo Zhao," Vegetation land use/land cover extration from high-resolution satellite images based on Adaptive context Inference ",*IEEE Access*,2020.
- [4].Xin Luo-Xiaohua Tong-Haiyan Pan," Integrating multir solution and multitemporal sentinel-2 Imagenery for land -cover mapping in the Xiongan new area ,china",*IEEE Transactions on Geoscience and remote sensing*,2021
- [5].Chengmao Wu-Xlaokang Guo," A Novel single fuzzifier internel type-2 Fuzzy C-means clustering with local,information for land-cover segmentation ",*IEEE journal of selected topics in applied earth observations and remote sensing*,2021
- [6] A. Nivaggioli and H. Randrianarivo, "Weakly supervised semantic segmentation of satellite images," *CoRR*, vol. abs/1904.03983, 2019. [Online]. Available: <http://arxiv.org/abs/1904.03983>
- [7]S. Wang, W. Chen, S. M. Xie, G. Azzari, and D. B. Lobell, "Weakly supervised deep learning for segmentation of remote sensing imagery," *Remote Sensing*, vol. 12, no. 2, p. 207, 2020.
- [8] J. Ahn and S. Kwak, "Learning pixel-level semantic affinity with image-level supervision for weakly supervised semantic segmentation," *CoRR*, vol. abs/1803.10464, 2018. [Online]. Available: <http://arxiv.org/abs/1803.10464>

[9]. Scott, M. England, W. Starms, R. Marcum, and C. Davis, "Training deep convolutional neural networks for land-cover classification of high-resolution imagery," *IEEE Geoscience and Remote Sensing Letters*, vol. PP, pp. 1–5, 02 2017.

APPENDIX

CHAPTER-11

SOURCE CODE

Predict.py

```
import logging
import os

import numpy as np
import torch
import torch.nn.functional as F
from PIL import Image
from torchvision import transforms
from unet import UNet
from utils.data_vis import plot_img_and_mask
from utils.dataset import BasicDataset
import matplotlib.pyplot as plt
import glob
from torch.utils.data import DataLoader

def predict_img(net,
                full_img,
                device,
                scale_factor=1,
                out_threshold=0.5):
    net.eval()
```

```

img = torch.from_numpy(BasicDataset.preprocess(full_img, scale_factor))

img = img.unsqueeze(0)
img = img.to(device=device, dtype=torch.float32)

with torch.no_grad():
    output = net(img)
    # print(output)
    if net.n_classes > 1:
        probs = F.softmax(output, dim=1)
    else:
        probs = torch.sigmoid(output)

probs = probs.squeeze(0)

height, width = probs.shape[1], probs.shape[2]

mapping = {0: (0, 255, 255), #urban_land
           1: (255, 255, 0), #agriculture
           2: (255, 0, 255), #rangeland
           3: (0, 255, 0), #forest_land
           4: (0, 0, 255), #water
           5: (255, 255, 255), #barren_land
           6: (0, 0, 0)} #unknown

class_idx = torch.argmax(probs, dim=0)
image = torch.zeros(height, width, 3, dtype=torch.uint8)

for k in mapping:

    idx = (class_idx == torch.tensor(k, dtype=torch.uint8))
    validx = (idx == 1)
    image[validx,:] = torch.tensor(mapping[k], dtype=torch.uint8)

image = image.permute(2, 0, 1)

tf = transforms.Compose(
    [
        transforms.ToPILImage(),
        transforms.Resize(full_img.size[1]),
        transforms.ToTensor()
    ]
)

image = image.permute(1,2,0)
image = image.squeeze().cpu().numpy()

return image, class_idx

```

```

def preprocess_mask(pil_img, scale):
    w, h = pil_img.size
    newW, newH = int(scale * w), int(scale * h)
    assert newW > 0 and newH > 0, 'Scale is too small'
    pil_img = pil_img.resize((newW, newH))

    img_nd = np.array(pil_img)

    if len(img_nd.shape) == 2:
        img_nd = np.expand_dims(img_nd, axis=2)

    # HWC to CHW
    img_trans = img_nd.transpose((2, 0, 1))
    torch.set_printoptions(edgeitems=10)
    return img_trans

def get_args():
    parser = argparse.ArgumentParser(description='Predict masks from input images',
                                     formatter_class=argparse.ArgumentDefaultsHelpFo
rmatter)
    parser.add_argument('--model', '-m', default=r'/content/drive/MyDrive/land_cover_classification/checkpoints/CP_epoch30.pth',
                        metavar='FILE',
                        help="Specify the file in which the model is stored")
    parser.add_argument('--input', '-i', metavar='INPUT', nargs='+',
                        help='filenames of input images', required=True, default=
r'/content/drive/MyDrive/land_cover_classification/data/test_img_subset')

    parser.add_argument('--output', '-o', metavar='INPUT', nargs='+',
                        help='Filenames of ouput images')
    parser.add_argument('--viz', '-v', action='store_true',
                        help="Visualize the images as they are processed",
                        default=False)
    parser.add_argument('--no-save', '-n', action='store_true',
                        help="Do not save the output masks",
                        default=False)
    parser.add_argument('--mask-threshold', '-t', type=float,
                        help="Minimum probability value to consider a mask pixel whi
te",
                        default=0.5)
    parser.add_argument('--scale', '-s', type=float,
                        help="Scale factor for the input images",
                        default=1)

    return parser.parse_args()

def RGB_2_class_idx(mask_to_be_converted):

```

```

mapping = {(0 , 255, 255): 0,      #urban_land
           (255, 255, 0 ): 1,      #agriculture
           (255, 0 , 255): 2,      #rangeland
           (0 , 255, 0 ): 3,        #forest_land
           (0 , 0 , 255): 4,        #water
           (255, 255, 255):5,      #barren_land
           (0 , 0 , 0 ):6}          #unknown

temp = np.array(mask_to_be_converted)
temp = np.where(temp>=128, 255, 0)
class_mask=torch.from_numpy(temp)
h, w = mask_to_be_converted.shape[2], mask_to_be_converted.shape[3]
img_no=mask_to_be_converted.shape[0]
mask_out = torch.zeros(img_no,h, w, dtype=torch.long)

for j in range (img_no):
    class_index=0
    for k in mapping:
        idx = (class_mask[j,:,:,:] == torch.tensor(k, dtype=torch.uint8).unsqueeze(1)
).unsqueeze(2))
        validx = (idx.sum(0) == 3)
        temp=mask_out[j,:,:]
        temp[validx]=class_index
        mask_out[j,:,:]=temp
        class_index+=1

    return mask_out

def get_output_filenames(args):
    in_files = args.input
    out_files = []

    if not args.output:
        for f in in_files:
            pathsplit = os.path.splitext(f)
            out_files.append("{}_OUT{}".format(pathsplit[0], pathsplit[1]))
    elif len(in_files) != len(args.output):
        logging.error("Input files and output files are not of the same length")
        raise SystemExit()
    else:
        out_files = args.output

    return out_files

def mask_to_image(mask):

```

```

        return Image.fromarray((mask * 255).astype(np.uint8))

def compute_iou(predicted, actual, num_calsses):
    intersection = 0
    union = 0
    iou=np.zeros((num_calsses-1),dtype=float)
    for k in range(num_calsses-1):
        a = (predicted == k).int()
        b = (actual == k).int()
        intersection = torch.sum(torch.mul(a, b))
        union = torch.sum(((a + b) > 0).int())
        iou[k]=intersection/union
    mean_iou=(1/(num_calsses-1))*np.sum(iou)
    return mean_iou

if __name__ == "__main__":
    args = get_args()
    in_files = args.input

    img_scale=args.scale
    # dir_mask='data/test_set_full_set/masks_test/'
    dir_mask='data/masks_subset/'
    # dir_img='data/test_set_full_set/img_test/'
    gt_list=[]
    mask_dirs=sorted( filter( os.path.isfile, glob.glob(dir_mask+'*') ) )

    for filename in mask_dirs:
        mask = Image.open(filename)
        resized_mask=preprocess_mask(mask, img_scale)
        mask=np.asarray(resized_mask)
        gt_list.append(mask)
    gt_list = np.array(gt_list)
    gt_tensor=torch.Tensor(gt_list)
    net = UNet(n_channels=3, n_classes=7)

    logging.info("Loading model {}".format(args.model))

    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
    logging.info(f'Using device {device}')

    net.to(device=device)
    net.load_state_dict(torch.load(args.model, map_location=device))

    logging.info("Model loaded !")

    for i, fn in enumerate(in_files):

```

```

logging.info("\nPredicting image {} ...".format(fn))

img = Image.open(fn)

#Splitting input directory from the file name
name=fn.split('/')[ -1]
#Removing the file extension
name=name.split('.')[0]
seg, mask_indices = predict_img(net=net,
                                full_img=img,
                                scale_factor=args.scale,
                                out_threshold=args.mask_threshold,
                                device=device)

if i==0:
    seg_array=mask_indices.unsqueeze(0)
else:
    seg_array=torch.cat((seg_array,mask_indices.unsqueeze(0)),0)

if args.viz:
    im = Image.fromarray(seg)
    im.save(str(args.output[0])+'pred_'+name+'.jpeg')

#Metric Evaluation
gt_tensor=RGB_2_class_idx(gt_tensor)
num_calsses=7
total_IoU = compute_iou(seg_array, gt_tensor,num_calsses=7)

print("IoU Value:",total_IoU)

```

Unet

Unet_model.py

```

import torch.nn.functional as F

from .unet_parts import *

class UNet(nn.Module):
    def __init__(self, n_channels, n_classes, bilinear=True):
        super(UNet, self).__init__()
        self.n_channels = n_channels
        self.n_classes = n_classes

```

```

        self.bilinear = bilinear

        self.inc = DoubleConv(n_channels, 64)
        self.down1 = Down(64, 128)
        self.down2 = Down(128, 256)
        self.down3 = Down(256, 512)
        factor = 2 if bilinear else 1
        self.down4 = Down(512, 1024 // factor)
        self.up1 = Up(1024, 512 // factor, bilinear)
        self.up2 = Up(512, 256 // factor, bilinear)
        self.up3 = Up(256, 128 // factor, bilinear)
        self.up4 = Up(128, 64, bilinear)
        self.outc = OutConv(64, n_classes)

    def forward(self, x):
        x1 = self.inc(x)
        x2 = self.down1(x1)
        x3 = self.down2(x2)
        x4 = self.down3(x3)
        x5 = self.down4(x4)
        x = self.up1(x5, x4)
        x = self.up2(x, x3)
        x = self.up3(x, x2)
        x = self.up4(x, x1)
        logits = self.outc(x)
        return logits

```

Unet_model_half.py

```

import torch.nn.functional as F
import torch

from .unet_parts import *

torch.autograd.set_detect_anomaly(True)

class UNet(nn.Module):
    def __init__(self, n_channels, n_classes, bilinear=True):
        super(UNet, self).__init__()
        self.n_channels = n_channels
        self.n_classes = n_classes
        self.bilinear = bilinear

        ## Half Breadth Versions
        self.inc = DoubleConv(n_channels, 64//2)

```

```

self.down1 = Down(64//2, 128//2)
self.down2 = Down(128//2, 256//2)
self.down3 = Down(256//2, 512//2)
factor = 2 if bilinear else 1
self.down4 = Down(512//2, 1024//2 // factor)
self.up1 = Up(1024//2, 512 //2// factor, bilinear)
self.up2 = Up(512//2, 256//2 // factor, bilinear)
self.up3 = Up(256//2, 128//2 // factor, bilinear)
self.up4 = Up(128//2, 64//2, bilinear)
self.outc = OutConv(64//2, n_classes)

def forward(self, x):
    assert torch.isfinite(x).all()
    x1 = self.inc(x)
    assert torch.isfinite(x1).all()
    x2 = self.down1(x1)
    assert torch.isfinite(x2).all()
    x3 = self.down2(x2)
    assert torch.isfinite(x3).all()
    x4 = self.down3(x3)
    assert torch.isfinite(x4).all()
    x5 = self.down4(x4)
    assert torch.isfinite(x5).all()
    x = self.up1(x5, x4)
    assert torch.isfinite(x).all()
    x = self.up2(x, x3)
    assert torch.isfinite(x).all()

    x = self.up3(x, x2)
    assert torch.isfinite(x).all()

    x = self.up4(x, x1)
    assert torch.isfinite(x).all()

    logits = self.outc(x)
    assert torch.isfinite(logits).all()
    return logits

```

unet-parts.py

```

import torch
import torch.nn as nn
import torch.nn.functional as F

```

```

class DoubleConv(nn.Module):
    """(convolution => [BN] => ReLU) * 2"""

    def __init__(self, in_channels, out_channels, mid_channels=None):
        super().__init__()
        if not mid_channels:
            mid_channels = out_channels
        self.double_conv = nn.Sequential(
            nn.Conv2d(in_channels, mid_channels, kernel_size=3, padding=1),
            nn.BatchNorm2d(mid_channels),
            nn.ReLU(inplace=True),
            nn.Conv2d(mid_channels, out_channels, kernel_size=3, padding=1),
            nn.BatchNorm2d(out_channels),
            nn.ReLU(inplace=True)
        )

    def forward(self, x):
        return self.double_conv(x)

class Down(nn.Module):
    """Downscaling with maxpool then double conv"""

    def __init__(self, in_channels, out_channels):
        super().__init__()
        self.maxpool_conv = nn.Sequential(
            nn.MaxPool2d(2),
            DoubleConv(in_channels, out_channels)
        )

    def forward(self, x):
        return self.maxpool_conv(x)

class Up(nn.Module):
    """Upscaling then double conv"""

    def __init__(self, in_channels, out_channels, bilinear=True):
        super().__init__()

        if bilinear:
            self.up = nn.Upsample(scale_factor=2, mode='bilinear', align_corners=True)

            self.conv = DoubleConv(in_channels, out_channels, in_channels // 2)
        else:
            self.up = nn.ConvTranspose2d(in_channels, in_channels // 2, kernel_size=2, stride=2)
            self.conv = DoubleConv(in_channels, out_channels)

```

```

def forward(self, x1, x2):
    x1 = self.up(x1)
    # input is CHW
    diffY = x2.size()[2] - x1.size()[2]
    diffX = x2.size()[3] - x1.size()[3]

    x1 = F.pad(x1, [diffX // 2, diffX - diffX // 2,
                    diffY // 2, diffY - diffY // 2])

    x = torch.cat([x2, x1], dim=1)
    return self.conv(x)

class OutConv(nn.Module):
    def __init__(self, in_channels, out_channels):
        super(OutConv, self).__init__()
        self.conv = nn.Conv2d(in_channels, out_channels, kernel_size=1)

    def forward(self, x):
        return self.conv(x)

```

Utils

Data_vis.py

```

import matplotlib.pyplot as plt

def plot_img_and_mask(img, mask):
    classes = mask.shape[2] if len(mask.shape) > 2 else 1
    fig, ax = plt.subplots(1, classes + 1)
    ax[0].set_title('Input image')
    ax[0].imshow(img)
    if classes > 1:
        for i in range(classes):
            ax[i+1].set_title(f'Output mask (class {i+1})')
            ax[i+1].imshow(mask[:, :, i])
    else:
        ax[1].set_title(f'Output mask')
        ax[1].imshow(mask)
    plt.xticks([], plt.yticks([]))
    plt.show()

```

Dataset.py

```
from os.path import splitext
from os import listdir
import numpy as np

from glob import glob
import torch
from torch.utils.data import Dataset
import logging
from PIL import Image

class BasicDataset(Dataset):
    def __init__(self, imgs_dir, masks_dir, scale=1, mask_suffix=''):
        self.imgs_dir = imgs_dir
        self.masks_dir = masks_dir
        self.scale = scale
        self.mask_suffix = mask_suffix
        # self.dic = np.array(7,4)
        assert 0 < scale <= 1, 'Scale must be between 0 and 1'

        self.ids = [splitext(file)[0] for file in listdir(imgs_dir)
                     if not file.startswith('.')]
        logging.info(f'Creating dataset with {len(self.ids)} examples')

    def __len__(self):
        return len(self.ids)

    @classmethod
    def preprocess(cls, pil_img, scale):
        w, h = pil_img.size
        newW, newH = int(scale * w), int(scale * h)
        assert newW > 0 and newH > 0, 'Scale is too small'
        pil_img = pil_img.resize((newW, newH))

        img_nd = np.array(pil_img)

        if len(img_nd.shape) == 2:
            img_nd = np.expand_dims(img_nd, axis=2)

        # HWC to CHW
        img_trans = img_nd.transpose((2, 0, 1))
        if img_trans.max() > 1:
            img_trans = img_trans / 255
        return img_trans
```

```

def preprocess_mask(cls, pil_img, scale):
    w, h = pil_img.size
    newW, newH = int(scale * w), int(scale * h)
    assert newW > 0 and newH > 0, 'Scale is too small'
    pil_img = pil_img.resize((newW, newH))

    img_nd = np.array(pil_img)

    if len(img_nd.shape) == 2:
        img_nd = np.expand_dims(img_nd, axis=2)

    # HWC to CHW
    img_trans = img_nd.transpose((2, 0, 1))
    torch.set_printoptions(edgeitems=10)
    return img_trans

def RGB_2_class_idx(cls, mask_to_be_converted):
    mapping = {(0, 255, 255): 0, #urban_land
               (255, 255, 0): 1, #agriculture
               (255, 0, 255): 2, #rangeland
               (0, 255, 0): 3, #forest_land
               (0, 0, 255): 4, #water
               (255, 255, 255): 5, #barren_land
               (0, 0, 0): 6} #unknown

    temp = np.array(mask_to_be_converted)
    temp = np.where(temp >= 128, 255, 0)

    class_mask = torch.from_numpy(temp)
    h, w = class_mask.shape[1], class_mask.shape[2]
    mask_out = torch.zeros(h, w, dtype=torch.long)
    for k in mapping:
        idx = (class_mask == torch.tensor(k, dtype=torch.uint8)).unsqueeze(1).unsqueeze(2)
        valid_idx = (idx.sum(0) == 3)
        mask_out[valid_idx] = torch.tensor(mapping[k], dtype=torch.long)

    return mask_out

def __getitem__(self, i):
    idx = self.ids[i]
    idx = idx[:len(idx) - 4]
    mask_file = glob(self.masks_dir + idx + '_mask' + '.png')
    img_file = glob(self.imgs_dir + idx + '_sat' + '.jpg')

    assert len(mask_file) == 1, \

```

```

        f'Either no mask or multiple masks found for the ID {idx}: {
mask_file}'
        assert len(img_file) == 1, \
            f'Either no image or multiple images found for the ID {idx}:
{img_file}'
        mask = Image.open(mask_file[0])
        img = Image.open(img_file[0])

        assert img.size == mask.size, \
            f'Image and mask {idx} should be the same size, but are {img
.size} and {mask.size}'

        img = self.preprocess(img, self.scale)
        mask = self.preprocess_mask(mask, self.scale)
        mask = self.RGB_2_class_idx(mask)
        return {
            'image': torch.from_numpy(img).type(torch.FloatTensor),
            # 'mask': torch.from_numpy(mask).type(torch.FloatTensor)
            'mask': mask
        }

class CarvanaDataset(BasicDataset):
    def __init__(self, imgs_dir, masks_dir, scale=1):
        super().__init__(imgs_dir, masks_dir, scale, mask_suffix='_mask'
)

```

classcount.py

```

import numpy as np
from tqdm import tqdm

def classcount(loader):
    n_train = len(loader)

    class_weight = np.array([0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0])

    with tqdm(total=n_train, desc='Class Count Assessment', unit='batch'
, disable = False, leave=True) as pbar:
        for batch in loader:
            imgs, true_masks = batch['image'], batch['mask']
            (unique, counts) = np.unique(true_masks, return_counts=True)
            frequencies = np.asarray((unique, counts))
            # print(frequencies.shape)
            for i in range(frequencies.shape[1]):

```

```

        class_weight[frequencies[0,i]] += frequencies[1,i]
    pbar.update()

    # print(class_weight)
    class_weight = class_weight[:-1].min()/class_weight
    class_weight[-1]=0

    return class_weight

```

Diceloss.py

```

import torch

def dice_coef_9cat(y_true, y_pred, smooth=1e-7):

    y_pred = torch.softmax(y_pred, dim=1)
    y_true_f = torch.flatten(torch.nn.functional.one_hot(y_true.to(torch.int64), num_classes=7))
    y_pred_f = torch.flatten(y_pred.permute(0,2,3,1))
    intersect = torch.sum(y_true_f * y_pred_f, axis=-1)
    denom = torch.sum(y_true_f + y_pred_f, axis=-1)
    return torch.mean((2. * intersect / (denom + smooth)))

def dice_coef_9cat_loss(y_true, y_pred):
    return 1 - dice_coef_9cat(y_true, y_pred)

```

Distribution.py

```

import cv2
import os
import glob
import skbuild
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

img_dir = "data/training_data" # Enter Directory of all images
data_path = os.path.join(img_dir, '*g')
files = glob.glob(data_path)
data = []

```

```

yellow_count =0
magenta_count=0
cyan_count =0
black_count =0
blue_count=0
white_count=0
green_count=0

for f1 in files:
    # print(f1)
    img = cv2.imread(f1)
    img=cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    cyan_count += np.count_nonzero((img == [0, 255, 255]).all(axis=2))
    #urban_land
    yellow_count += np.count_nonzero((img == [255,255, 0]).all(axis=2))
    #agriculture
    magenta_count += np.count_nonzero((img == [255, 0,255]).all(axis=2))
    #range_land
    green_count += np.count_nonzero((img == [0, 255,0]).all(axis=2))
    #forest_land
    blue_count += np.count_nonzero((img == [0, 0, 255]).all(axis=2))
    #water
    white_count += np.count_nonzero((img == [255, 255, 255]).all(axis=2))
    #barren_land
    black_count += np.count_nonzero((img == [0, 0, 0]).all(axis=2))
    #unknown

print("yellow:",yellow_count/1e6)
print("magenta:",magenta_count/1e6)
print("cyan:",cyan_count/1e6)
print("black:",black_count/1e6)
print("white:",white_count/1e6)
print("blue:",blue_count/1e6)
print("green:",green_count/1e6)
n_bins=7

df = pd.DataFrame({'Land Type':['urban_land','agriculture','rangeland','forest_land','water','barren_land','unknown'],
                   'Pixels':[cyan_count,yellow_count,magenta_count,green_count,blue_count,white_count,black_count]})

ax = df.plot.bar(x='Land Type', y='Pixels', width=1)
plt.show()

```

eval.py

```

import torch
import torch.nn.functional as F
from tqdm import tqdm

from diceloss import dice_coef_9cat_loss

def eval_net(net, loader, device):
    """Evaluation without the densecrf with the dice coefficient"""
    net.eval()
    mask_type = torch.float32 if net.n_classes == 1 else torch.long
    n_val = len(loader)  # the number of batch
    tot = 0

    with tqdm(total=n_val, desc='Validation round', unit='batch', disable
= True, leave=True) as pbar:
        for batch in loader:
            imgs, true_masks = batch['image'], batch['mask']
            imgs = imgs.to(device=device, dtype=torch.float32)
            true_masks = true_masks.to(device=device, dtype=mask_type)

            with torch.no_grad():
                mask_pred = net(imgs)

            if net.n_classes > 1:
                tot += F.cross_entropy(mask_pred, true_masks).item()
                # tot += dice_coef_9cat_loss(true_masks,mask_pred).item(
)
            else:
                pred = torch.sigmoid(mask_pred)
                pred = (pred > 0.5).float()
                tot += dice_coef(pred, true_masks).item()
            pbar.update()
    net.train()
    return tot/n_val

```

Train.py

```

import argparse
import logging
import os
import sys

import numpy as np

```

```

import torch
import torch.nn as nn
from torch import optim
from tqdm import tqdm
from tensorflow.keras.applications.resnet50 import ResNet50

from eval import eval_net
from unet import UNet

from torch.utils.tensorboard import SummaryWriter
from utils.dataset import BasicDataset
from torch.utils.data import DataLoader, random_split
from diceloss import dice_coef_9cat_loss
from classcount import classcount

torch.autograd.set_detect_anomaly(True)

# Comment/Uncomment to toggle subset for training
dir_img = 'data/img_subset/'
dir_mask = 'data/masks_subset/'

## Comment/Uncomment to toggle subset for training
# dir_img = 'data/training_data/images'
# dir_mask = 'data/training_data/masks'

dir_checkpoint = 'checkpoints/'
res = ResNet50(weights="imagenet", include_top=False,
               input_tensor=Input(shape=(224, 224, 3)))

def train_net(net,
              device,
              epochs=5,
              batch_size=1,
              lr=0.001,
              val_percent=0.1,
              save_cp=True,
              img_scale=0.5):

    dataset = BasicDataset(dir_img, dir_mask, img_scale)
    n_val = int(len(dataset) * val_percent)
    n_train = len(dataset) - n_val
    train, val = random_split(dataset, [n_train, n_val], generator=torch.
Generator().manual_seed(42))
    train_loader = DataLoader(train, batch_size=batch_size, shuffle=True,
, num_workers=8, pin_memory=True)
    val_loader = DataLoader(val, batch_size=batch_size, shuffle=False, n
um_workers=8, pin_memory=True, drop_last=True)

```

```

writer = SummaryWriter(comment=f'LR_{lr}_BS_{batch_size}_SCALE_{img_
scale}')
global_step = 0

logging.info(f'''Starting training:
    Epochs:          {epochs}
    Batch size:      {batch_size}
    Learning rate:   {lr}
    Training size:   {n_train}
    Validation size: {n_val}
    Checkpoints:     {save_cp}
    Device:          {device.type}
    Images scaling:  {img_scale}
''')

optimizer = optim.Adam(net.parameters(), lr=lr, weight_decay=1e-8)
weights_classes = torch.from_numpy(classcount(train_loader))
weights_classes = weights_classes.to(device=device, dtype=torch.float32)

print("Class Distribution", weights_classes)

if net.n_classes > 1:
    criterion = nn.CrossEntropyLoss(weight = weights_classes)
else:
    criterion = nn.BCEWithLogitsLoss()

for epoch in range(epochs):
    net.train()

    epoch_loss = 0
    pseudo_batch_loss=0 ##remove when not pruning for lr
    with tqdm(total=n_train, desc=f'Epoch {epoch + 1}/{epochs}', unit='img') as pbar:
        for batch in train_loader:
            # Use half precision model for training
            net.half()

            imgs = batch['image']
            true_masks = batch['mask']
            assert imgs.shape[1] == net.n_channels, \
                f'Network has been defined with {net.n_channels} input channels, ' \
                f'but loaded images have {imgs.shape[1]} channels. Please check that ' \
                'the images are loaded correctly.'

```

```

        imgs = imgs.to(device=device, dtype=torch.float16)
        mask_type = torch.float32 if net.n_classes == 1 else torch.long
        true_masks = true_masks.to(device=device, dtype=mask_type)

        masks_pred = net(imgs)

        # convert the prediction to float32 for avoiding nan in loss calculation
        masks_pred = masks_pred.type(torch.float32)

        ## Cross Entropy Loss
        loss = criterion(masks_pred, true_masks)
        epoch_loss += loss.item()

        ## Dice Loss
        # loss=dice_coef_9cat_loss(true_masks,masks_pred)
        # epoch_loss += loss.item()

        pbar.set_postfix(**{'Epoch Loss': epoch_loss/n_train})

        net.float()
        optimizer.zero_grad()
        loss.backward()
        nn.utils.clip_grad_value_(net.parameters(), 0.1)
        optimizer.step()

        pbar.update(imgs.shape[0])
        global_step += 1

        pseudo_batch_loss += loss.item()

        if (global_step) % 16 == 0:
            writer.add_scalar('Batch Loss/train',pseudo_batch_loss, global_step)
            pseudo_batch_loss = 0

        writer.add_scalar('Loss/train', epoch_loss/n_train, epoch+1)

        for tag, value in net.named_parameters():
            tag = tag.replace('.', '/')
            writer.add_histogram('weights/' + tag, value.data.cpu().numpy(), epoch+1)
            writer.add_histogram('grads/' + tag, value.grad.data.cpu().numpy(), epoch+1)

        val_score = eval_net(net, val_loader, device)

```

```

if net.n_classes > 1:
    logging.info('Validation CE Loss: {}'.format(val_score))
    writer.add_scalar('Loss/test', val_score, epoch+1)
else:
    logging.info('Validation Dice Coeff: {}'.format(val_score))
    writer.add_scalar('Dice/test', val_score, epoch+1)

writer.add_images('images', imgs, epoch+1)
if net.n_classes == 1:
    writer.add_images('masks/true', true_masks, epoch+1)
    writer.add_images('masks/pred', torch.sigmoid(masks_pred) >
0.5, epoch+1)

if (epoch+1) % 5 == 0:
    if save_cp:
        try:
            os.mkdir(dir_checkpoint)
            logging.info('Created checkpoint directory')
        except OSError:
            pass
        torch.save(net.state_dict(),
                    dir_checkpoint + f'CP_epoch{epoch + 1}.pth')
        logging.info(f'Checkpoint {epoch + 1} saved !')

def get_args():
    parser = argparse.ArgumentParser(description='Train the UNet on images
and target masks',
                                     formatter_class=argparse.ArgumentDe
faultsHelpFormatter)
    parser.add_argument('-e', '--
epochs', metavar='E', type=int, default=5,
                        help='Number of epochs', dest='epochs')
    parser.add_argument('-b', '--batch-
size', metavar='B', type=int, nargs='?', default=1,
                        help='Batch size', dest='batchsize')
    parser.add_argument('-l', '--learning-
rate', metavar='LR', type=float, nargs='?', default=4e-5,
                        help='Learning rate', dest='lr')
    parser.add_argument('-f', '--
load', dest='load', type=str, default=False,
                        help='Load model from a .pth file')
    parser.add_argument('-s', '--
scale', dest='scale', type=float, default=1,
                        help='Downscaling factor of the images')

```

```

        parser.add_argument('-v', '--
validation', dest='val', type=float, default=10.0,
                           help='Percent of the data that is used as valida
tion (0-100)')

    return parser.parse_args()

if __name__ == '__main__':
    logging.basicConfig(level=logging.INFO, format='%(levelname)s: %(mes
sage)s')
    args = get_args()
    device = torch.device('cuda' if torch.cuda.is_available() else 'cpu'
)

    # device = torch.device('cpu')
    logging.info(f'Using device {device}')

    net = UNet(n_channels=3, n_classes=7, bilinear=True)

    logging.info(f'Network:\n'
                 f'\t{net.n_channels} input channels\n'
                 f'\t{net.n_classes} output channels (classes)\n'
                 f'\t{"Bilinear" if net.bilinear else "Transposed conv"}
upsampling')

    if args.load:
        net.load_state_dict(
            torch.load(args.load, map_location=device)
        )
        logging.info(f'Model loaded from {args.load}')

    net.to(device=device)
    # faster convolutions, but more memory
    try:
        train_net(net=net,
                  epochs=args.epochs,
                  batch_size=args.batchsize,
                  lr=args.lr,
                  device=device,
                  img_scale=args.scale,
                  val_percent=args.val / 100)
    except KeyboardInterrupt:
        torch.save(net.state_dict(), 'INTERRUPTED.pth')
        logging.info('Saved interrupt')
        try:
            sys.exit(0)
        except SystemExit:
            os._exit(0)

```

```
import sys
import os

py_file_location = "/content/drive/MyDrive/land_cover_classification"
sys.path.append(os.path.abspath(py_file_location))
```

```
!python /content/drive/MyDrive/land_cover_classification/predict.py -
i /content/drive/MyDrive/land_cover_classification/data/180902_sat.jpg -
o /content/drive/MyDrive/land_cover_classification/predictions/ -s .8 -viz
```

CHAPTER-12

OUTPUT

