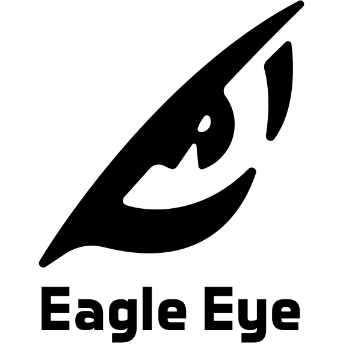


Cairo University

Faculty of Engineering

Department of Computer Engineering

**Eagle Eye**



A Graduation Project Report Submitted

to

Faculty of Engineering, Cairo University

in Partial Fulfillment of the requirements of the degree

of

Bachelor of Science in Computer Engineering.

**Presented by**

Omar Mohammad Walid Mohammad

Yahia Ali

**Supervised by**

Prof. Hoda Baraka

2020

All rights reserved. This report may not be reproduced in whole or in part, by photocopying or other means, without the permission of the authors/department.

**Abstract**

There is a continuous stream of new satellite imagery data that is generated on a daily basis. There can never be enough manual workers working on this data to maintain a perpetually up-to-date polygon map. We explore the value of using Machine Learning algorithms to automate this manual and tedious process.

The core objective of Eagle Eye is to process an input high-resolution satellite image and produce a polygon map that meaningfully describes the image.

Eagle Eye implements two different learning algorithms to realize this objective. The first one involves using the U-Net structure. We also implemented a secondary, more “traditional” Machine Learning algorithm to showcase the advantages of convolutional networks in the task of semantic segmentation. This approach uses the Deep Belief Net structure.

The programming language used in this project is Python. The network structures are implemented using the TensorFlow platform for its high performance and flexibility. The system was tested using the PyTest library.

The outcomes of the project are fairly satisfactory considering the limited computational resources we had to work with. Our best model achieves an accuracy of 80% at 11 label classes.

**الملخص**

هناك تدفق مستمر لبيانات صور الأقمار الصناعية الجديدة التي يتم إنشاؤها على أساس يومي. لا يمكن أن يكون هناك عدد كافٍ من العمال اليدويين الذين يعملون على هذه البيانات للحفاظ على خريطة محدثة بشكل دائم. نستكشف قيمة استخدام خوارزميات التعلم الآلي لأتمتة هذه العملية اليدوية الشاقة.

الهدف الأساسي من إيجل أى هو معالجة صورة القمر الصناعي عالية الدقة المدخلة وإنتاج خريطة تصف الصورة بشكل هادف.

يطبق إيجل أى خوارزميات تعليمية مختلفة لتحقيق هذا الهدف. الأول يتضمن استخدام هيكل يو-نت. قمنا أيضًا بتنفيذ خوارزمية ثانوية أكثر "تقليدية" للتعلم الآلي لعرض مزايا الشبكات العصبونية الالتفافية في مهمة التجزئة الدلالية. يستخدم هذا النهج بنية شبكة المعتقد العميقة.

لغة البرمجة المستخدمة فى هذا المشروع هى لغة بايثون. يتم تنفيذ هياكل الشبكة باستخدام منصة تنسور فلو لأدائها العالي ومرونتها. تم اختبار النظام باستخدام مكتبة باي-تست.

نتائج المشروع مرضية إلى حد ما بالنظر إلى الموارد الحسابية المحدودة التي كان علينا العمل معها. يحقق أفضل نموذج لدينا دقة تصل إلى 80٪ في 11 فئة تصنيف.

**Acknowledgment**

In this page, we would like to express our deepest gratitude to our project supervisor, Prof. Hoda Baraka for going out of her way to support and encourage us from the very beginning. We would also like to thank the faculty as whole for providing us with this unique learning experience that rewards creativity.

**Table of Contents**

[Abstract ii](#_30j0zll)

[الملخص](#_1fob9te) iii

[Acknowledgment iv](#_3znysh7)

[Table of Contents v](#_2et92p0)

[List of Figures vii](#_tyjcwt)

[List of Tables viii](#_3dy6vkm)

[List of Abbreviations ix](#_1t3h5sf)

[List of Symbols x](#_4d34og8)

[Contacts xi](#_2s8eyo1)

[Chapter 1: Introduction 1](#_17dp8vu)

[1.1. Motivation and Justification 1](#_3rdcrjn)

[1.2. Project Objectives and Problem Definition 1](#_26in1rg)

[1.3. Project Outcomes 2](#_lnxbz9)

[1.4. Document Organization 2](#_35nkun2)

[Chapter 2: Market Visibility Study 4](#_1ksv4uv)

[2.1. Targeted Customers 4](#_44sinio)

[2.2. Market Survey 4](#_2jxsxqh)

[2.2.1. Ecopia Tech 4](#_z337ya)

[2.2.2. Esri ArcGIS 4](#_3j2qqm3)

[2.3. Business Case and Financial Analysis 5](#_1y810tw)

[2.3.1 Business Case 5](#_4i7ojhp)

[2.3.2 Financial Analysis 5](#_2xcytpi)

[Chapter 3: Literature Survey 7](#_1ci93xb)

[3.1. Background on Color Representations 7](#_3whwml4)

[3.2. Background on Semantic Segmentation 8](#_2bn6wsx)

[3.3. Comparative Study of Previous Work 8](#_qsh70q)

[3.4. Implemented Approach 9](#_3as4poj)

[Chapter 4: System Design and Architecture 10](#_1pxezwc)

[4.1. Overview and Assumptions 10](#_49x2ik5)

[4.2. System Architecture 10](#_2p2csry)

[4.2.1. Block Diagram 10](#_147n2zr)

[4.2.2. Preprocessing Module 12](#_3o7alnk)

[4.2.3. U-Net Module 12](#_23ckvvd)

[4.2.4. Feature Extraction Module 12](#_ihv636)

[4.2.5. DBN Module 12](#_32hioqz)

[4.2.6. Application Module 12](#_1hmsyys)

[4.3. Preprocessing Module 13](#_41mghml)

[4.3.1. Functional Description 13](#_2grqrue)

[4.3.2. Modular Decomposition 13](#_vx1227)

[4.3.3. Design Constraints 14](#_3fwokq0)

[4.3.4. Other Description 14](#_1v1yuxt)

[4.4. U-Net Module 15](#_4f1mdlm)

[4.4.1. Functional Description 15](#_2u6wntf)

[4.4.2. Modular Decomposition 15](#_19c6y18)

[4.4.3. Design Constraints 18](#_3tbugp1)

[4.4.4. Other Description 19](#_28h4qwu)

[4.5. Feature Extraction Module 20](#_nmf14n)

[4.5.1. Functional Description 20](#_37m2jsg)

[4.5.2. Modular Decomposition 20](#_1mrcu09)

[4.5.3. Design Constraints 24](#_46r0co2)

[4.6. DBN Module 25](#_2lwamvv)

[4.6.1. Functional Description 25](#_111kx3o)

[4.6.2. Modular Decomposition 25](#_3l18frh)

[4.6.3. Design Constraints 30](#_206ipza)

[4.7. Application Module 31](#_4k668n3)

[4.7.1. Functional Description 31](#_2zbgiuw)

[4.7.2. Modular Decomposition 31](#_1egqt2p)

[4.7.3. Design Constraints 32](#_3ygebqi)

[Chapter 5: System Testing and Verification 33](#_2dlolyb)

[5.1. Testing Setup 33](#_sqyw64)

[5.2. Testing Plan and Strategy 33](#_3cqmetx)

[5.2.1. Module Testing 33](#_1rvwp1q)

[5.2.2. Integration Testing 34](#_4bvk7pj)

[5.3. Testing Results 34](#_2r0uhxc)

[Chapter 6: Conclusions and Future Work 36](#_1664s55)

[6.1. Faced Challenges 36](#_3q5sasy)

[6.2. Gained Experience 37](#_25b2l0r)

[6.3. Conclusions 37](#_kgcv8k)

[6.4. Future Work 38](#_34g0dwd)

[References 39](#_1jlao46)

[Appendix A: Development Platform and Tools 40](#_43ky6rz)

[A.1. Software Tools 40](#_2iq8gzs)

[A.1.1. Development Platform 40](#_xvir7l)

[A.1.2. Preprocessing Libraries 40](#_3hv69ve)

[A.1.3. Machine Learning Libraries 40](#_1x0gk37)

[A.1.4. Graphical User Interface Libraries 40](#_4h042r0)

[A.1.5. Unit Testing Library 40](#_2w5ecyt)

[Appendix B: Use Cases 41](#_1baon6m)

[Appendix C: User Guide 42](#_3vac5uf)

**List of Figures**

[Figure 3.1. HSV Color Representation 7](#_2afmg28)

[Figure 4.1. System Block Diagram 11](#_pkwqa1)

[Figure 4.2. Reference U-Net architecture 15](#_39kk8xu)

[Figure 4.3. Our Custom U-Net Architecture, part 1 16](#_1opuj5n)

[Figure 4.4. Our Custom U-Net Architecture, part 2 17](#_48pi1tg)

[Figure 4.5. Feature Extraction Flowchart 20](#_2nusc19)

[Figure 4.6. Example RBM showing just one hidden layer output 25](#_1302m92)

[Figure 4.7. Example RBM showing all connections 26](#_3mzq4wv)

[Figure 4.8. Reconstruction phase in our example RBM 27](#_2250f4o)

[Figure 4.9. Example on RBM stacking in the unsupervised learning phase 28](#_haapch)

[Figure 4.10. Final DBN Structure 29](#_319y80a)

[Figure 5.1. Example Input Image 35](#_1gf8i83)

[Figure 5.2. Example Output Image 35](#_40ew0vw)

[Figure B.1. Use Case Diagram 41](#_2fk6b3p)

[Figure C.1. Welcome Screen 42](#_upglbi)

[Figure C.2. Main Window 42](#_3ep43zb)

[Figure C.3. Processing Screen 43](#_1tuee74)

[Figure C.4. Results Window 43](#_4du1wux)

[Figure C.5. Final Step Screen 44](#_2szc72q)

**List of Tables**

[Table 2.1. Cash Flow Table 6](#_184mhaj)

[Table 4.1. Image feature list 22](#_3s49zyc)

[Table 5.1. Dataset Class Types 34](#_279ka65)

[Table 5.2. Accuracy of each model 34](#_meukdy)

[Table 5.3. Comparison between two training set sizes 34](#_36ei31r)

[Table C.1. Result Color Legend 44](#_1ljsd9k)

**List of Abbreviations**

CCM Color Co-occurrence Matrix

CD Contrastive Divergence

CNN Convolutional Neural Network

DBN Deep Belief Net

GIS Geographic Information System

GPU Graphics Processing Unit

GUI Graphical User Interface

HSV Hue, Saturation and Value

NIR Near Infrared

RBM Restricted Boltzmann Machine

ReLU Rectified Linear Unit

RGB Red, Green and Blue

SSD Sum of Squared Differences

STD Standard Deviation

**List of Symbols**

Maximum RGB value

Minimum RGB value

Difference between the maximum and minimum RGB values

Hue Color Co-occurrence Matrix

Saturation Color Co-occurrence Matrix

Value Color Co-occurrence Matrix

Variance

Standard Deviation

Number of visible nodes

Number of hidden nodes

Weight Matrix

Hidden layer bias vector

Visible layer bias vector

Hidden node value in the current iteration

Visible node value in the current iteration

Hidden node value in the next iteration

Visible node value in the next iteration

Number of classes

Learning rate

Loss coefficient

**Contacts**

**Team Members**

|  |  |  |
| --- | --- | --- |
| **Name** | **Email** | **Phone Number** |
| Omar Mohammad | [omarkimo80@gmail.com](mailto:omarkimo80@gmail.com) | +2 01144521964 |
| Walid Mohammad | [walidhella@disroot.org](mailto:walidhella@disroot.org) | +2 01555910380 |
| Yahia Ali | [yahiaali2014.ya@gmail.com](mailto:yahiaali2014.ya@gmail.com) | +2 01030344226 |

**Supervisor**

|  |  |  |
| --- | --- | --- |
| **Name** | **Email** | **Number** |
| Hoda Baraka | [hodabaraka60@gmail.com](mailto:hodabaraka60@gmail.com) | +2 01005411083 |

This page is left intentionally empty

**Chapter 1: Introduction**

Eagle Eye is a Machine Learning project whose aim is to perform semantic segmentation on satellite images to produce meaningful maps that describe the images and provide context that is human-readable.

**1.1. Motivation and Justification**

The Earth is a big place, and there are hundreds of satellites orbiting the planet, constantly taking new pictures. There’s simply not enough people to manually go through all this data and observe every little change. Artificial Intelligence is needed to maintain an up-to-date polygon map.

Polygon maps have a variety of uses. There is, however, one main use that motivated us to build Eagle Eye; detection of illegal buildings on agricultural lands.

It is illegal in Egypt for farmers to remove native vegetation on agricultural lands and build houses instead. This is because of how dangerous the issue of land clearing is.

In 2017, the government reported that 1.8 million infringing buildings have been detected in just the previous 6 years. This amounted to more than of land. Of course, this number only represents what the government detected. The real numbers may be higher.

The main motivation of Eagle Eye is to help the government in the detection of such illegal buildings by automating the process. However, it can be used for any number of general cases.

**1.2. Project Objectives and Problem Definition**

Our project has a core objective which is to process an input high-resolution satellite image and produce a polygon map that meaningfully describes the image.

We pursue another objective in our project with the aim of providing a solution to the problem stated in the previous section. Process the produced polygon map and produce a list of buildings that are surrounded (or nearly surrounded) by agricultural lands.

**1.3. Project Outcomes**

The outcomes of the project are explained in this section.

* A software module that performs necessary preprocessing on the dstl satellite image dataset. This dataset contains images of 20 spectral bands.
* A software module that implements the U-Net machine learning algorithm to produce a model that is trained on the dstl dataset.
* A software module that performs feature extraction operations on 4-band images (Red, Green, Blue and Near Infrared) to extract relevant features.
* A software module that implements a Deep Belief Net to produce a model trained on the transformed dataset produced from the feature extraction module.
* A software module that implements a user-facing application that can accept user input and use any of the generated models to provide the user with a prediction as well as a list of infringing buildings.

**1.4. Document Organization**

The report consists of five chapters after this one for a total of six chapters. All references cited throughout the report are listed in the References section after Chapter 6. Finally, there are appendices at the end which include further information about the project.

The following is a description of each chapter.

* Chapter 2:

In which we discuss our market survey regarding similar projects to ours and potential competitors. We discuss the approaches of our competitors as well as their advantages and disadvantages. This is followed by a discussion of our business case and financial analysis.

* Chapter 3:

In which we provide important context on the problems we set out to solve as well as background on the most important topics to explaining our work. Previous works in this area are also discussed and compared. Finally, we describe our implemented approach which is fully explained in Chapter 4.

* Chapter 4:

In which we describe the project in full detail. This chapter represents the main body of this report. We highlight and discuss our scientific approaches and methodologies.

* Chapter 5:

In which we explain our testing and verification methodology. This includes the testing setup, strategy and environment. Results from different testing scenarios are discussed.

* Chapter 6:

In which we summarize the entire project, its features and limitations. This includes the challenges we faced to realize the objectives and the experience gained from working on the project. We give directions for future work in this area that extends beyond our project’s scope.

The following is a description of each appendix.

* Appendix A:

This appendix explains the tools and platforms we used.

* Appendix B:

This appendix shows the use cases for our project.

* Appendix C:

This appendix provides a clear and detailed guide for customers to properly use our project.

**Chapter 2: Market Visibility Study**

In this chapter, we survey the related market to our project and discuss similar projects to ours and their advantages and disadvantages. This is followed by a discussion of our business case and financial analysis.

**2.1. Targeted Customers**

As the main inspiration to develop Eagle Eye was to help with the problem of Agricultural Land Infringement, we can consider the target customer of the project to be the Egyptian Government. The proper authorities can use our data to quickly identify illegal buildings.

As stated in the previous chapter, however, our project can have many uses and isn’t limited to the problem of agricultural lands. Thus, we can expand our perspective to include more customers to financially support the project.

Since the scope of our project is limited to simply extracting the useful data from satellite images, we must resort to selling our data to established companies who can derive benefits from the data for profit.

One possible consideration is companies who specialize in Geographic Information System (GIS) software such as Esri Northeast Africa. GIS companies have employees who segment the data manually on customer demand. With access to automatically generated segmentations, they can upscale their operation greatly. This makes them viable customers to us.

**2.2. Market Survey**

**2.2.1. Ecopia Tech**

Ecopia Tech is a technology company based in Canada. They provide a service similar to ours. They advertise that their machine learning algorithms are used to generate accurate vector maps for satellite images.

They do not publicize exact details about their algorithms or whether their solution is completely end-to-end. They may be using machine learning only partially and relying on employees to manually finalize the vector maps.

Their approach does reduce the human workload, but can’t be considered a complete solution to the problem.

**2.2.2. Esri ArcGIS**

Esri is an international supplier of GIS software based in California, USA. They develop professional GIS software (ArcGIS) that can be used to manually draw vector maps for satellite images.

They provide the same service advertised by Ecopia Tech, but with one crucial difference; the work is entirely manual. Their solution to the problem has already been explained; they rely on their employees to manually draw the vector maps, which greatly limits the size of their operation.

**2.3. Business Case and Financial Analysis**

**2.3.1 Business Case**

As stated above, Eagle Eye is targeted towards the enterprise market. We propose deal with a GIS company like Esri Northeast Africa.

When they are commissioned with drawing polygon maps over their employee capacity, they can send us the images instead of refusing the commission.

After we receive the commissioned images, we use Eagle Eye to provide them with polygon maps. Since our process is automated, it is extremely scalable; using the machine learning model to draw a polygon map takes less than a minute.

The amount of data we’ll be able to sell will depend on the commissions of other companies such as Esri and, of course, on the number of deals we can make.

Manual workers in GIS companies are basically our only competition here in Egypt. Since we can automate most of the work they do, our prices can be quite high (relative to the initial investment) so long as it’s cheaper for companies to use our services instead of hiring and training more workers.

**2.3.2 Financial Analysis**

**Capital Expenditure (Capex)**

The following is a list of items that will be purchased as an initial investment.

* 3 Mid-Range Desktop Computers: 3 x 7,000 = 21,000EGP
* 3 Desks and Office Chairs: 3 x 5,000 = 15,000EGP
* 1 Air Conditioning unit: 11,000EGP
* 1 High-End Server with a High-End GPU = 22,000EGP

**Operational Expenses (Opex)**

The following is a list of recurring expenses on a monthly basis.

* Workspace rent: 5,000EGP per month
* Developer salaries: 3 x 9,000 = 27,000EGP per month
* Utility bills (Internet, Electricity, Water): 1500EGP per month
* Legal representation: 2,000EGP per month
* Business name certificate: 750EGP per month

**Cash Flow Analysis**

Revenue will depend on how many commissions we receive. We assume that the revenue will be 20,000EGP per month, then rise to 40,000EGP. Finally, the revenue stabilizes at 50,000EGP. We reach the break-even point in 21 months.



**Table 2.1. Cash Flow Table**

**Chapter 3: Literature Survey**

This chapter provides important context on the problems we outlined in Chapter 1. Particularly, we provide a background on the subjects of color spaces, satellite images and semantic segmentation. We then proceed to compare previous works in this area and describe our implemented approach which is fully explained in the following chapter.

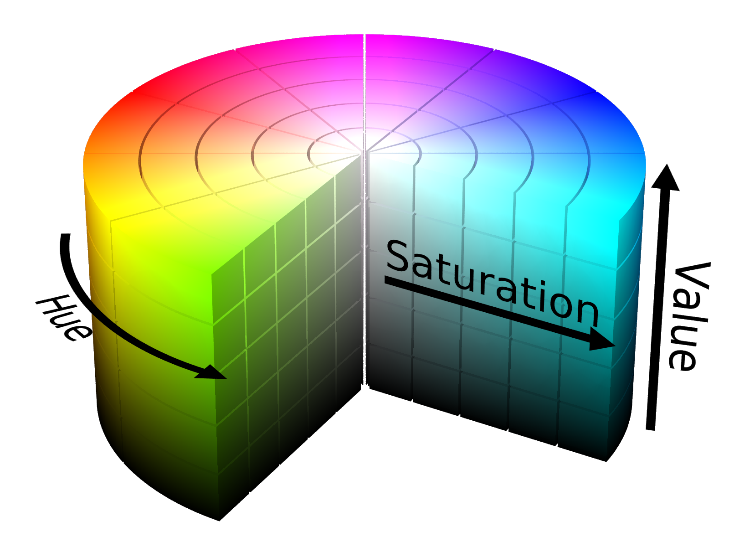
**3.1. Background on Color Representations**

Computers use various color representations to model images, different color representations are more suitable for some calculations than others. There isn’t one particular representation that can be considered “better” than the others.

The most commonly used color representations that many people are familiar with is the RGB model. RGB stands for Red, Green and Blue. In this model, any pixel is defined by a mix of those three colors. For example, the color yellow is a mix of red and green, so it can be written as (255,255,0). This examples assume an 8-bit value per color which amounts to a maximum value of 255.

An RGB image is stored as a 3-dimensional matrix in computer programs. The first two dimensions are the width and height, respectively. The 3rd dimension is the RGB color representation of each pixel.

For our purposes, there is another important color representation that must be discussed, and that is HSV. It stands for Hue, Saturation and Value. Hue is an angle with values in the range [0:360], it specifies the color on the cylinder. Saturation is a percentage in the range [0:1] that specifies light pollution. Value is also a percentage in the range [0:1], and it specifies the brightness of the pixel.



**Figure 3.1. HSV Color Representation**

**3.2. Background on Semantic Segmentation**

The classification of satellite images is considered a harder problem than regular photographs. Traditional supervised learning methods do not generalize well for such a large-scale learning problem. The problem of detecting various land cover classes in general is a difficult problem considering the significantly higher intra-class variability in land cover types such as trees, grasslands, barren lands, water bodies, etc. as compared to that of roads. Traditional supervised learning approaches require carefully selected hand-crafted features and substantial amounts of labeled data. On the other hand, purely unsupervised approaches are not able to learn the higher order dependencies inherent in the land cover classification problem.

Recently, deep convolutional networks have been used successfully in many visual recognition tasks. They have been around for a long time, but they only recently improved because large datasets with labels have become available. The typical use of convolutional networks is on classification tasks, where one class label describes the whole image. In many visual tasks, however, the output is required to be localized, i.e., a class label assigned to each pixel in the image.

**3.3. Comparative Study of Previous Work**

Ciresan et al. ‎[1] trained a network in a sliding-window setup to predict the class label of each pixel by providing a local region (patch) around that pixel as input. This network can localize well. And the training data in terms of patches is much larger than the number of training images. This strategy was successful, but it has two clear disadvantages. First, it’s slow because the network runs separately for every patch, and the patches are overlapping, which introduces a lot of redundancy. The second disadvantage is the trade-off between localization accuracy and context. Larger patches reduce the localization accuracy, while smaller ones reduce the context. More recent approaches have achieved a good balance between the two.

Long et al. ‎[2] proposed an architecture based on the concept of Convolutional Neural Networks (CNN). This approach is named the Fully Connected Network (FCN). The key idea in this approach is to build a convolutional network that can be trained end-to-end, pixels-to-pixels. This approach was successful, but it required a very large amount of training images.

Ronneberger et al. ‎[3] proposed a modification on the FCN such that it works with few training images and yields more precise segmentations. This approach is quite successful as it improved upon FCN. It was named U-Net due to its architecture being shaped like a U.

**3.4. Implemented Approach**

In our project, we implemented the U-Net architecture proposed in 2015. U-Net was chosen because it shows how much better at segmentation CNNs are. We also implemented a secondary, more “traditional” Machine Learning algorithm to showcase the advantages of convolutional networks in the task of semantic segmentation.

The secondary approach comprises a manual feature extraction module and a Deep Belief Net (DBN) consisting of an unsupervised phase and a Softmax Linear Classifier as the output layer. This approach is based on the work of Geoffrey Hinton ‎[4] and it was chosen because it’s relatively fast compared to other Machine Learning algorithms

This secondary approach was used specifically with satellite images in Basu et al. ‎[5], where some more parameters are detailed. We implemented this approach according to the recommendations in Basu et al. ‎[5]

**Chapter 4: System Design and Architecture**

In this chapter, we discuss the design and methodology of our system. The high level system architecture is presented. Each module is then discussed in detail in the following sections.

**4.1. Overview and Assumptions**

Eagle Eye is developed using the Python programming language. Overall, there are five different modules in this project. The project is designed in such a way that allows each individual team member to work on a module (or a submodule) separately and independently from other modules.

In this project, we train the machine learning models with an image dataset from dstl. The preprocessing and feature extraction modules assume that the input is from this dataset. In order to use other datasets for training, it may be necessary to tweak those modules.

The system is designed to create two different machine learning structures and train them both on the available dataset to generate different models.

**4.2. System Architecture**

**4.2.1. Block Diagram**

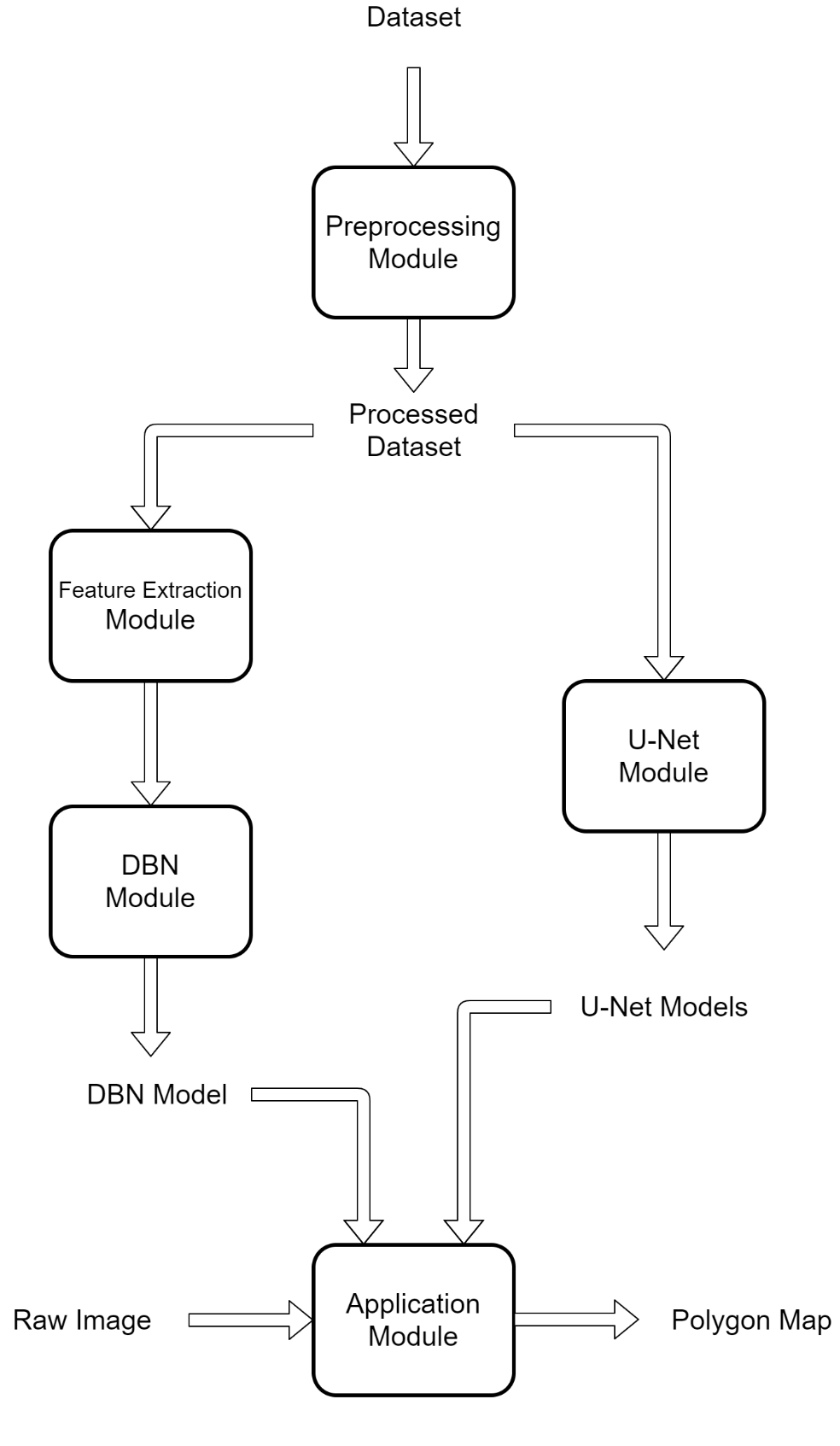
The first module takes the dstl dataset and performs the necessary preprocessing. There are two different paths representing the two approaches to solving the problem.

The first path (U-Net) consists of one module which takes the preprocessed dataset and constructs the neural network and trains it.

The second path consists of two modules; the Feature Extraction module and the Deep Belief Net module.

The Application module uses the models that were generated from the two paths to perform a prediction according to user input. This module provides a graphical user interface to make Eagle Eye more user-friendly.

The block diagram of the project is presented in the following page.



**Figure 4.1. System Block Diagram**

**4.2.2. Preprocessing Module**

This module performs some required techniques to get the images ready for training and for segmentation.

**4.2.3. U-Net Module**

This module is responsible for the U-Net structure path in our semantic segmentation process. This includes building the CNNs of which the model consists, training on the training set and validating on the validation set to compute the accuracy of the model, and lastly, segmenting input test data by producing masks for the classes in the input image.

**4.2.4. Feature Extraction Module**

This module computes the required features from the input image to reduce dimensionality. It takes an image consisting of 4 channels (Red, Green, Blue and NIR) and performs the necessary calculations to produce 16 features that are considered to be the most relevant to segmentation.

**4.2.5. DBN Module**

This module is responsible for creating the Deep Belief Net and training it using the training subset of the dataset. It produces a new trained model that can be used to predict the labels of any input raw image.

**4.2.6. Application Module**

This module includes the user interface. It uses the trained models to produce a polygon map for raw non-labeled user-input images. Also, it produces a list of potentially infringing buildings.

**4.3. Preprocessing Module**

**4.3.1. Functional Description**

This module performs some required techniques to get images ready for training and for segmentation. Since the amount of training data in satellite images is low as compared to traditional image segmentation datasets, the individual images are of high resolution and this can be a trade-off between the total number of training images and the resolution of the training images, so the training images must be pre-processed well and carefully to achieve good results.

**4.3.2. Modular Decomposition**

We use preprocessing in two different ways: one for preparing the training data for training, and the other for preparing the image before segmenting it. There is an overlap between the two ways in some basic information.

In the dataset, each “image object” (with particular image id) has four images made with different wave lengths, in total 20 channels. (3 RGB, 1 Panchromatic, 8 Multispectral, 8 Short Wave Infrared)

Note that we trained several models. The first was trained using the 20-band labeled dataset. The second was trained using just 4 bands from the dataset. The third and last was trained using only 3 bands. In every model, we train only the images of interest according to the case. In the rest of this section, however, we will consider only the 20-band case.

**Preparing images for training**

There is a total of 25 images available with labels which makes them suitable for training. The U-Net model requires some order of axes in images, so for each image, we roll the first axis to be the last axis so that the training is done smoothly.

First, we resize all images to 800x800 pixels which are closest to the Multispectral image resolution (M), then join them along the axis. So at the end we had a tensor of following shape: 20x800x800. We create the pixel masks from given polygons with same size: 800x800 pixels.

Secondly, we perform “Contrast Enhancement” on the images. The 5th and 95th percentile are used as parameters.

Thirdly, we stack all images in an array with the dimensions of 5 images wide, 5 images high and 20 Channels, and stack all the masks for images using the information in the training data in another array but with the dimensions of 5 images wide, 5 images high and 10 classes of images.

Finally, we save those two arrays in two files for further need.

**Preparing images for segmentation**

Each training image is processed separately. Each image is resized and normalized the same way as training data to 20x800x800 tensor.

Split this tensor to multiple tensors each with the input layer size of the U-Net model which is 160 that add up to five tensors. Predict each tensor separately using the model and then converge the results.

**4.3.3. Design Constraints**

This module assumes, depending on the chosen number of bands, that the input image contains all the bands specified. If the modules expect to have RGB input image and the user entered another format the result will be unexpected.

This module is constrained to the input layer size in U-Net model, so if another model is used this module is needed to be modified as the result will be dummy for that new model.

The training version of this module depends on the size of the available training data which is 25, hence the size of two arrays depend on that as well (5 images wide \* 5 images high).

**4.3.4. Other Description**

While preparing the test image for segmentation, we split the tensor 20x800x800 to five 20x800x160 tensors so that we can predict them using the U-Net model well as the input layer size that we chose for the model (160).

**4.4. U-Net Module**

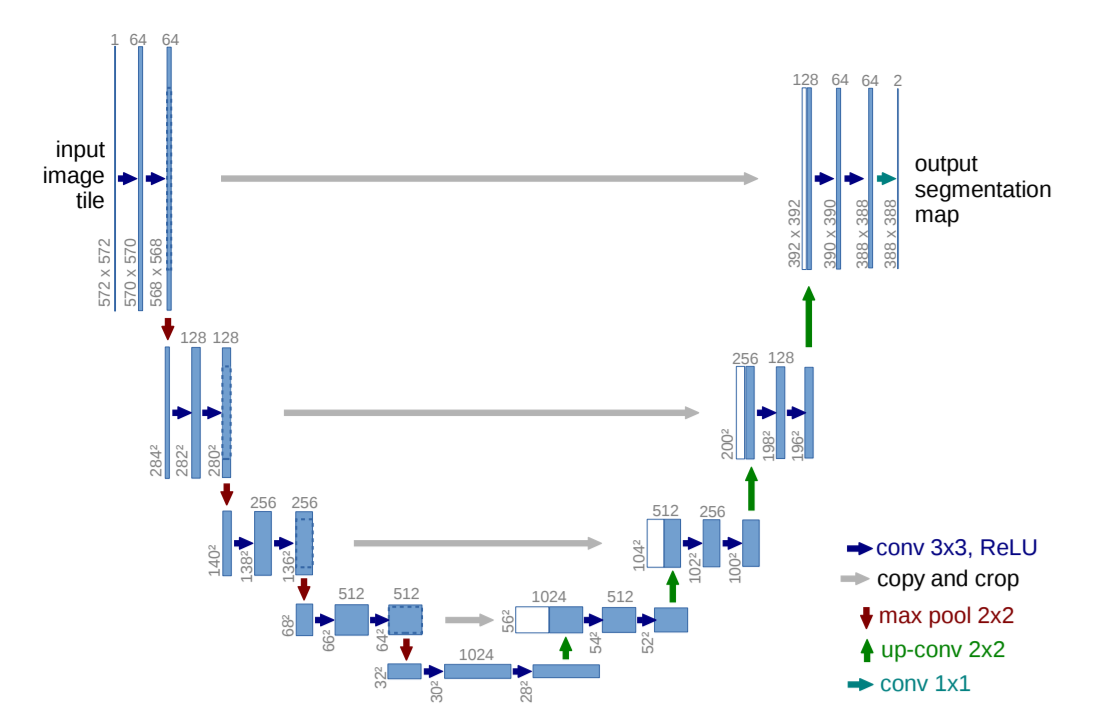
**4.4.1. Functional Description**

This module is responsible for the deep learning part in our semantic segmentation process. This includes building the CNNs of which the model consists, training on the training set and validating on the validation set to compute the accuracy of the model, and lastly segmenting input test data by producing masks for the classes in the input image.

**4.4.2. Modular Decomposition**

**Building the U-Net Model**

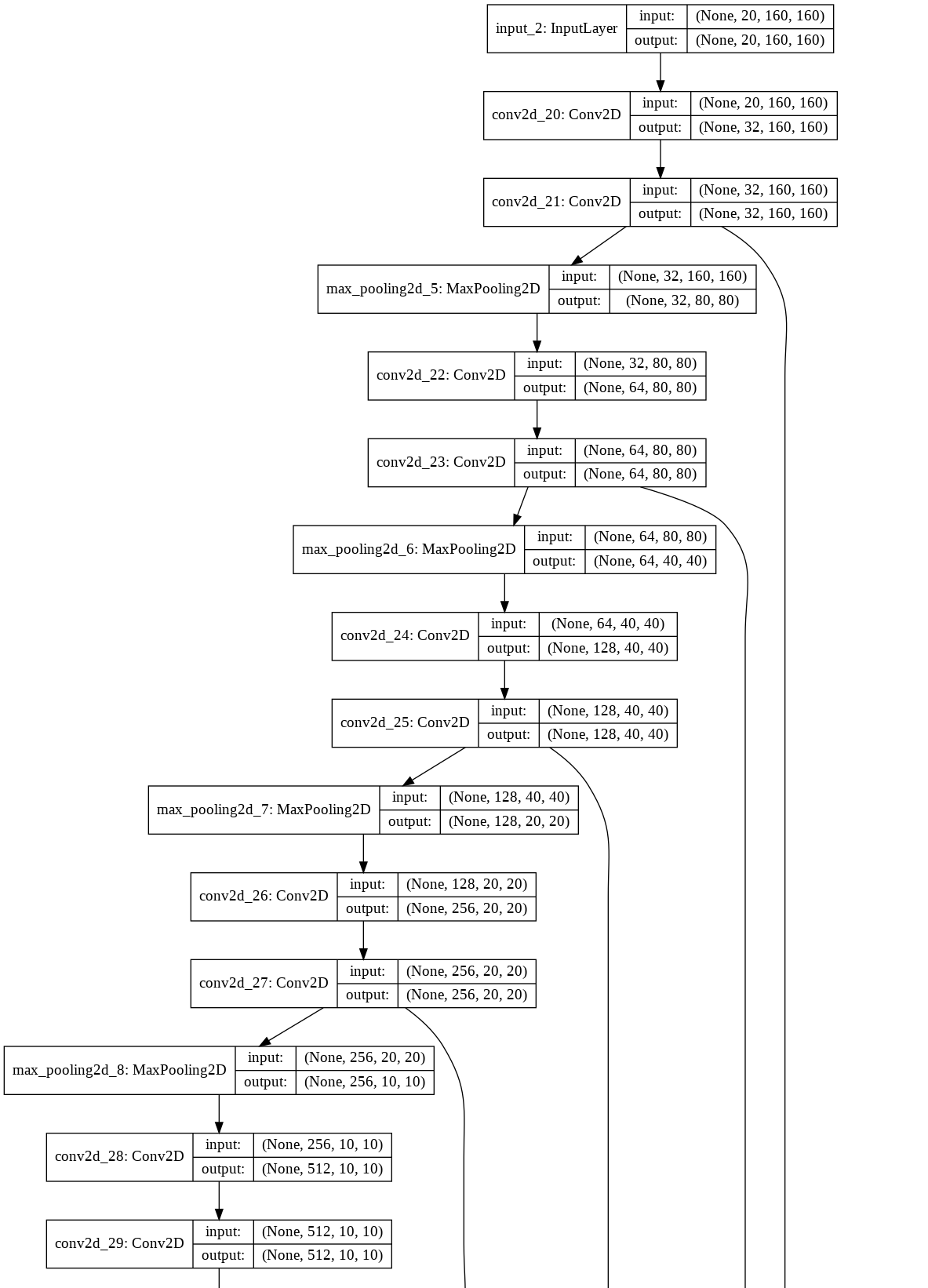
We can see an example (reference) of the U-Net architecture in the following figure:



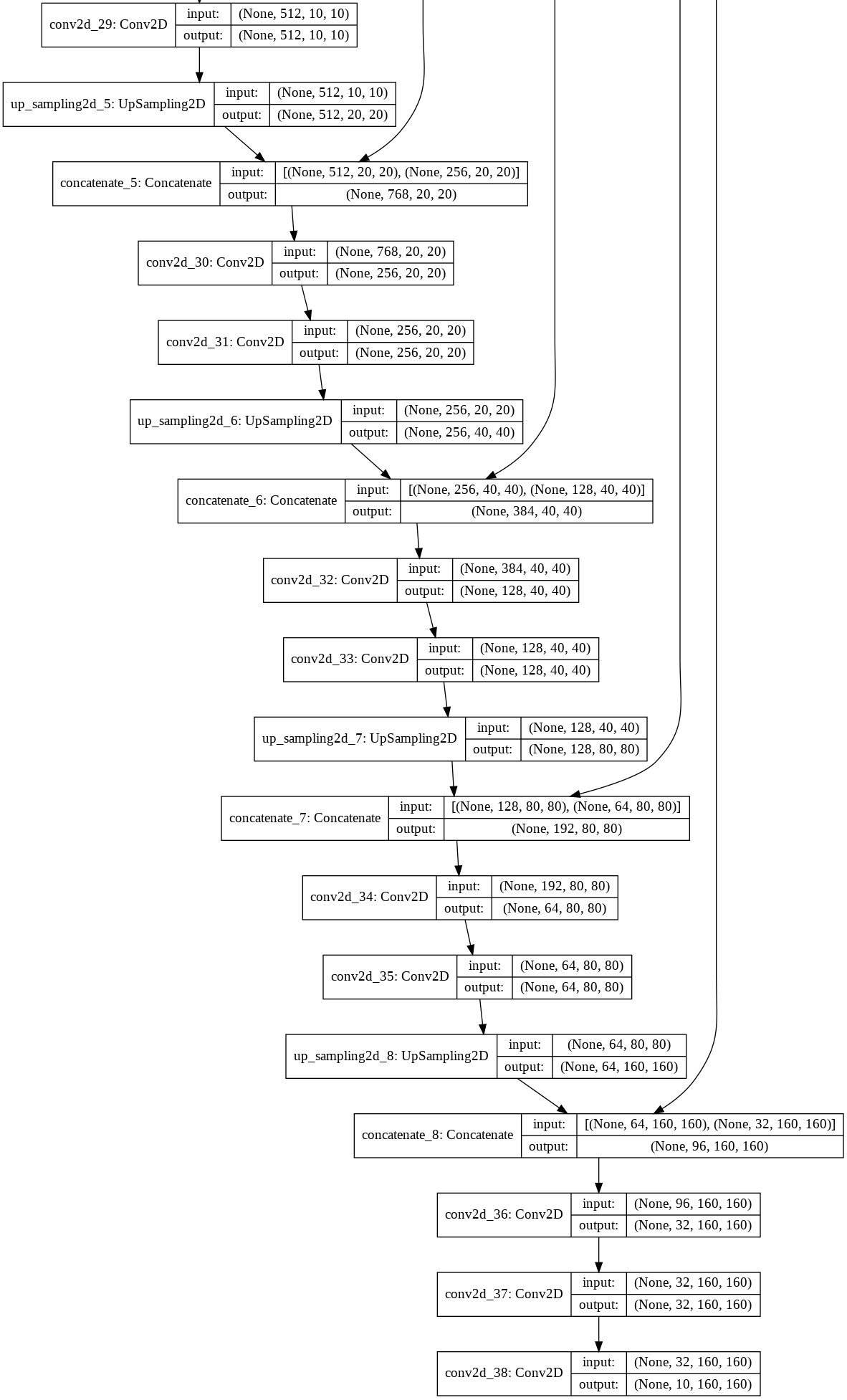
**Figure 4.2. Reference U-Net architecture**

As evident from the figure above, the architecture is shaped like a U, which is the reason for its name.

We modify this architecture a little in our project. The modified version is shown in the following figures.



**Figure 4.3. Our Custom U-Net Architecture, part 1**



**Figure 4.4. Our Custom U-Net Architecture, part 2**

The U-Net network architecture consists of a contracting path (left side) and an expansive path (right side).

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 down sampling. At each down sampling step, we double the number of feature channels.

Every step in the expansive path consists of an up sampling 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.

**Training and Validation**

In this phase, we continuously make a checkpoint of the model so that we don’t start the process from the beginning if some error happens. These are the steps we follow:

1. Load the training data saved in the preprocessing module.
2. Pick some samples from them for validation.
3. We fit the model, calculate the accuracy of the model, and save the model weights.
4. We do the fitting step two times and choose the weights with the higher accuracy.

**4.4.3. Design Constraints**

Our chose of the input layer size (160) restricts the down sampling and up sampling to be done at most four times.

Our hardware limits us to extract at most 5000 patches when picking validation data which limit the accuracy of the model.

We do the fitting step two times as our hardware limit us to do more.

Most of constraints in this module come from the available hardware.

**4.4.4. Other Description**

To allow a seamless tiling of the output segmentation map, it is important to select the input tile size such that all 2x2 max-pooling operations are applied to a layer with an even x- and y-size. In our case we chose 160 as the input size. Since we perform down sampling four times, we will have 160 => 80 => 40 => 20 => 10, which guarantee a seamless tiling of the output segmentation map.

In our modified U-Net architecture, the “20” in the input layer can also be 3 or 4 depending on the number of bands the user chose. The “10” in the output of last layer is the number of classes to which we segment the image.

Every machine learning model must have some sort of loss (cost) function to map samples to some value so that the model can learn and predict on non-training data correctly. The loss function we use is “Binary cross entropy” as the pixel is just belongs to some class (building, water, crops, etc.) or not. And this is the equation of the binary cross entropy loss function:

**4.5. Feature Extraction Module**

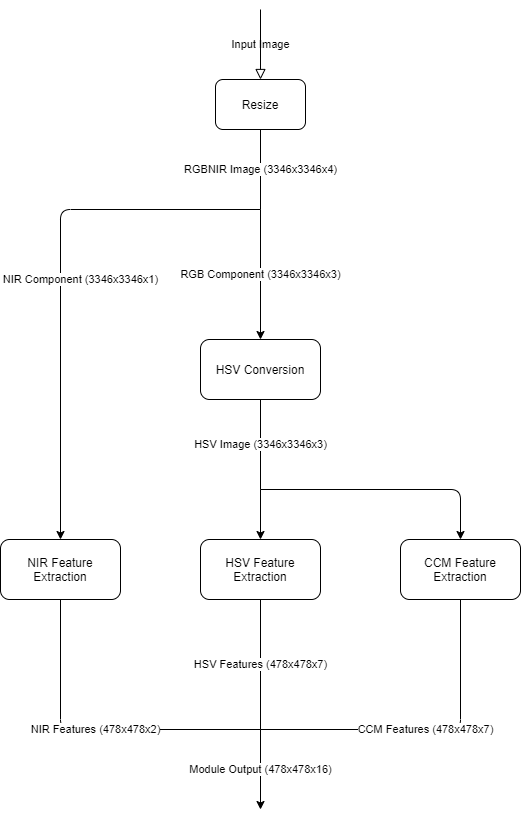
**4.5.1. Functional Description**

This module is responsible for computing the required features from the input image to reduce dimensionality. It takes an image consisting of 4 channels (Red, Green, Blue and NIR) and performs the necessary calculations to produce 16 features that are considered to be the most relevant to segmentation.

**4.5.2. Modular Decomposition**

First, we resize the input image to size 3346x3346. We now have an image of size 3346x3346x4. Then, we convert the RGB component of the image to the HSV (Hue, Saturation and Value) image representation.

Since all of our features are based on aggregation, each feature is computed in a window of 7x7. This means that for every 7x7 pixels, we compute one feature vector and the resulting set of features will be 478x478x16.



**Figure 4.5. Feature Extraction Flowchart**

To convert images from the RGB representation to the HSV representation we apply the following process to each pixel:

1. Divide the RGB values by 1023 (assuming 1024 levels per channel).
2. Calculate the factors:
3. Calculate H (Hue) using the formula:
4. Calculate S (Saturation) using the formula:
5. Set V (Value) to the value of :

**Color Co-occurrence Matrix (CCM)**

It is important to explain the concept of Color Co-occurrence Matrices (CCMs) because our most relevant features depend on this concept. A CCM is a 256x256 matrix (assuming the image has 256 levels) that describes the adjacency of pixels in one channel of the image.

The matrix is first initialized with zeros. Then, for every pixel in the channel, we consider the pixel that is one step right and one step down. The corresponding entry in the CCM is determined by the value of the two pixels. The first pixel’s value determines the row, while the neighboring pixel’s value determines the column. The corresponding entry (initially zero) is then incremented by 1.

For example, if one pixel is 231 and the neighboring pixel is 112, we increment the entry in row 231, column 112 in the CCM.

Pixels in the last row or the last column are only included as neighbors of the previous row or column since their would-be neighbors are out-of-bounds.

To create a CCM, the image channel is normalized such that the minimum value is 0 and the maximum value is 1. Then, it is multiplied by 255 and rounded to integers. Thus, any value in the image matrix can be used as an index for the CCM.

It should be noted that the images from the dstl dataset actually have 1024 levels but they are quantized to 256 levels in this way to reduce the computation time.

**Image Features**

For our project, we extract several features that are considered the most relevant for classification in Basu et al. ‎[5]. The following table enumerates those features:

|  |  |
| --- | --- |
| **Index** | **Feature** |
| 1 | Mean of Hue CCM |
| 2 | Mean of Saturation CCM |
| 3 | Mean of Value CCM |
| 4 | SSD of Hue CCM |
| 5 | Autocorrelation of Hue CCM |
| 6 | 2nd moment of Saturation CCM |
| 7 | 2nd moment of Value CCM |
| 8 | Mean of Hue |
| 9 | Mean of Saturation |
| 10 | Mean of Value |
| 11 | Mean of Near Infrared |
| 12 | 2nd moment of Value |
| 13 | Variance of Value |
| 14 | STD of Hue |
| 15 | STD of Value |
| 16 | STD of Near Infrared |

**Table 4.1. Image feature list**

Each of these features is explained with the appropriate equations that were used in the project as follows:

1. Mean of Hue CCM
2. Mean of Saturation CCM
3. Mean of Value CCM
4. Sum of Squared Differences of Hue CCM
5. Autocorrelation of Hue CCM
6. Second Moment of Saturation CCM
7. Second Moment of Value CCM
8. Mean of Hue
9. Mean of Saturation
10. Mean of Value
11. Mean of NIR
12. Second Moment of Value
13. Variance of Value
14. Standard Deviation of Hue
15. Standard Deviation of Value
16. Standard Deviation of NIR

**4.5.3. Design Constraints**

Since our 16 features are dependent on the Red, Green, Blue, and NIR channels, the input image must contain exactly those 4 channels.

This module was designed to work with the dstl dataset. As such, it assumes that the input image size is at least 3346x3346. It can use smaller images of any size, but the results will be poorer.

More importantly, the input image is assumed to have 1024 levels in the RGB component. This is important because the HSV conversion function begins by dividing the pixel values by 1023. Images with more or less levels than 1024 will corrupt most of the feature calculations.

Computation time is a very limiting factor here. As stated previously, the Color Co-occurrence Matrices have a size of only 256x256 rather than 1024x1024. This is done specifically to reduce computation time.

**4.6. DBN Module**

**4.6.1. Functional Description**

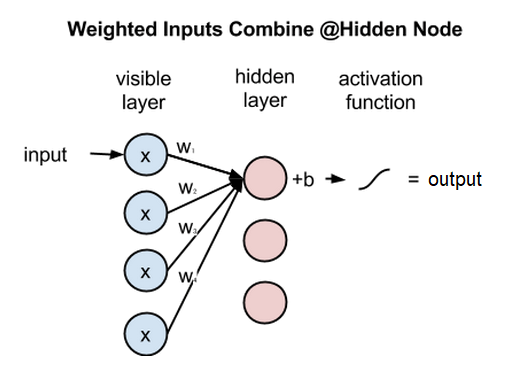
This module is responsible for creating the Deep Belief Net and training it using the dataset after extracting the required features using the Feature Extraction Module as well as the labels from the original dataset. It produces a trained model that can be used to predict the labels of any input raw image.

**4.6.2. Modular Decomposition**

This module consists of two submodules; the Restricted Boltzmann Machine class and the Deep Belief Net class.

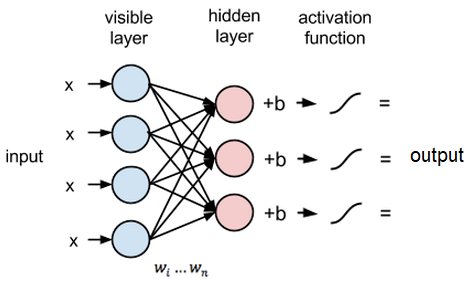
**Restricted Boltzmann Machine**

Boltzmann Machines (BMs) are a form of log-linear Markov Random Field, i.e., for which the energy function is linear in its free parameters. They consist of exactly two layers having visible and hidden nodes. The more hidden units we use, the more we can increase the modeling capacity of the Boltzmann Machine (BM). Restricted Boltzmann Machines restrict BMs to not having visible-visible or hidden-hidden connections i.e. there are no intra-layer connections. The following figure shows an example RBM with 4 visible units and 3 hidden ones.



**Figure 4.6. Example RBM showing just one hidden layer output**

In an RBM, each input (x) is multiplied by a separate weight (w), the products are summed, added to a bias (b), and again the result is passed through an activation function to produce the node’s output (a). The following figure shows the same example RBM with all connections, where n is the number of visible units.



**Figure 4.7. Example RBM showing all connections**

In our project, we use the sigmoid activation function:

Weights and biases are initialized randomly using a normal distribution with standard deviation defined by the following equation:

where is the number of visible nodes.

Weights form a 2-dimensional matrix where the number of rows is the number of hidden nodes and the number of columns is the number of visible nodes.

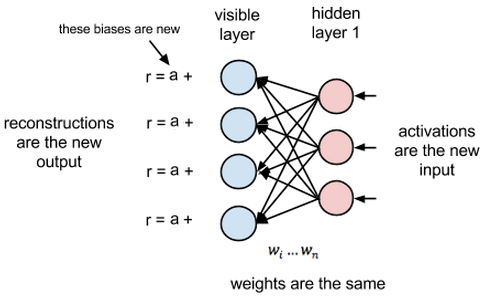
Biases form a vector where the number of elements is the number of visible nodes.

The most important property of RBMs is they can reconstruct the input data by themselves with no labels i.e. in an unsupervised manner. We can make multiple forward and backward passes between the hidden layer and the visible layer.

In the forward pass, an RBM uses the inputs to make predictions about node activations, or the probability of an output (a) given a weighted input (x); .

In the reconstruction phase, the activations of the hidden layer become the input in a backward pass. They are multiplied by the same weights, one per internode edge, just as x was weight-adjusted on the forward pass. The sum of those products is added to a visible-layer bias at each visible node, and the output of those operations is a reconstruction; i.e. an approximation of the original input.

In this backward pass, the RBM estimates the probability of inputs (x) given activations (a) which are weighted using the same coefficients from the forward pass; .



**Figure 4.8. Reconstruction phase in our example RBM**

Reconstruction is making guesses about the probability distribution of the original input; i.e. the values of many varied points at once. This is known as generative learning. To measure the distance between its estimated probability distribution and the ground-truth distribution of the input, RBMs use Contrastive Divergence.

Samples of p(x) can be obtained by running a Markov chain to convergence, using Gibbs sampling as the transition operator.

Gibbs sampling of the joint of N random variables is done through a sequence of N sampling sub-steps of the form where contains the other random variables in , excluding .

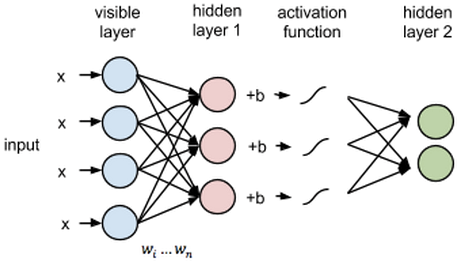
Since visible and hidden units are conditionally independent, we perform block Gibbs sampling, where visible units are sampled simultaneously given fixed values of the hidden units. Likewise, given the visible units, hidden units are also sample simultaneously. A step in the Markov Chain can be written as:

Contrastive Divergence (CD) is used to speed up the sampling process since otherwise, Gibbs sampling would take an infeasible amount of time until convergence. CD does not wait for the chain to converge. Samples are obtained after only one step of Gibbs sampling. CD also initializes the Markov chain with a training example (i.e., from a distribution that is expected to be close to p, so that the chain will be already close to having converged to its final distribution p).

**Deep Belief Net**

The Deep Belief Net consists of two parts; the Unsupervised Learning phase and the Supervised Learning phase.

Unsupervised learning is performed using a stack or RBMs, where the hidden layer of each RBM is used as a visible layer for the following one.



**Figure 4.9. Example on RBM stacking in the unsupervised learning phase**

In our project, we use three RBMs, each with 100 hidden nodes. The visible layer for the first RBM has 16 nodes corresponding to our 16 features. Each RBM is trained for 3 iterations, where each iterations has a forward pass to predict the probabilities of the outputs and a backward pass to approximately reconstruct the input.

The Supervised learning phase adds a neural network consisting of a single Softmax Linear Classifier as a final output layer. This output layer is trained for 100 iterations to produce the final values for the weights and biases.

The output layer has one neuron for each class (11 in our case). The weight matrix W has a row for each class (11 rows) and a column for each unit in the previous hidden layer (100 columns). The bias vector has an entry for each neuron (11 values).

Initially, the weights and biases are assigned randomly according to the standard normal distribution.

Each input is multiplied by their respective weights for each class and added to the bias value of that class. This is followed by the Softmax activation function.

Where i represents the class index, and j represents the hidden node index.

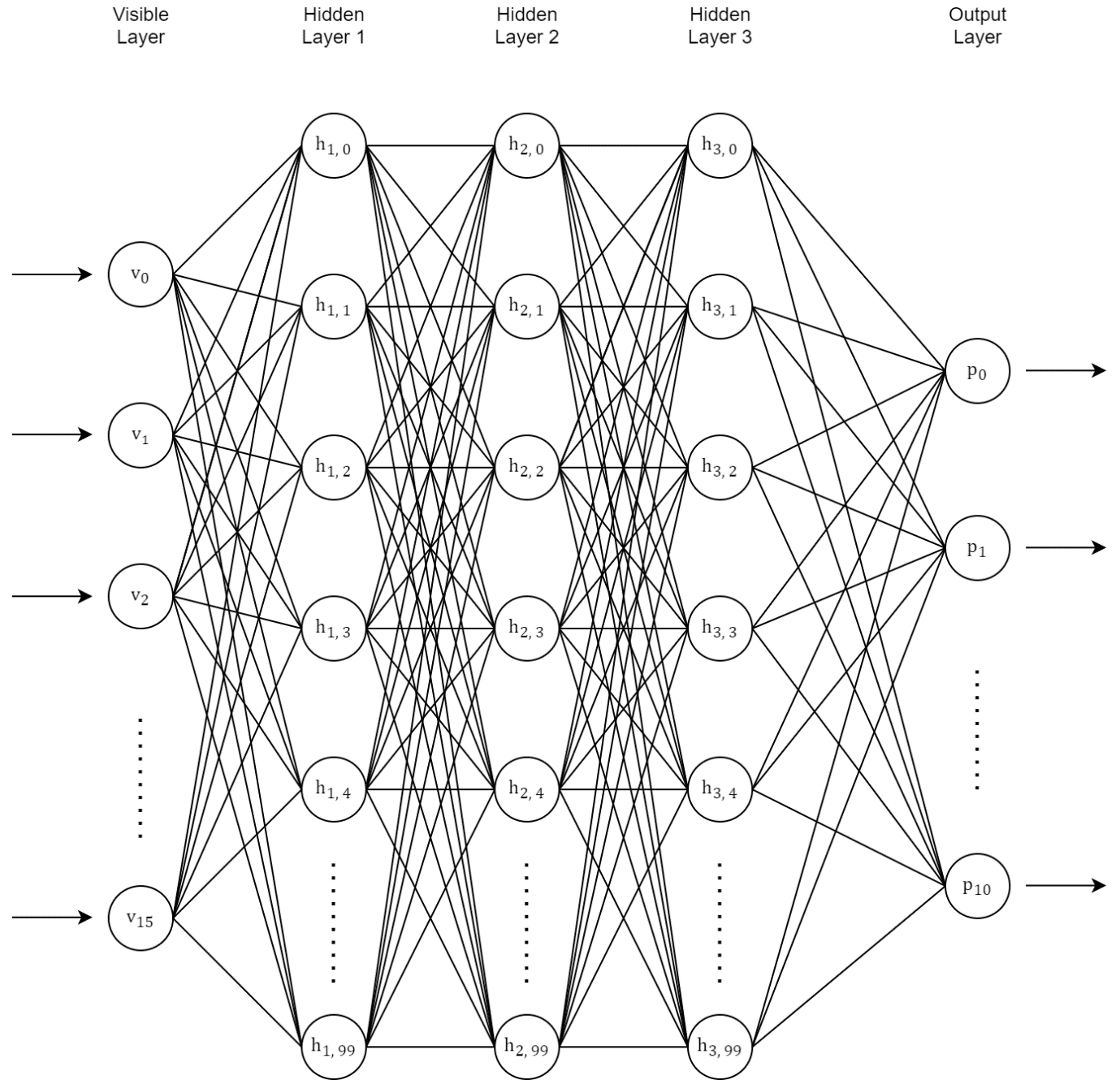
The Softmax activation function is applied on the prediction vector as follows:

For the next iteration, the weights are updated according to the equation:

Where is the learning rate, is the number of samples, is the batch size.

The biases are updated according to the equation:

The figure below represents the overall DBN structure, including the stacked RBMs as well as the Softmax linear classifier layer.



**Figure 4.10. Final DBN Structure**

For the backpropagation step, Stochastic Gradient Descent is used as the optimization function. First, the deltas for each weight and bias are computed. Then, the gradients are computed.

We use Multi-Class Cross Entropy to measure the loss in each epoch. Cross Entropy Loss is calculated using the following equation:

Where is the number of classes, is the actual probability of class x and is the predicted probability of class x.

After training the network, we can predict the class of a given input by passing it through the weighted network and producing probabilities for each class. The chosen class for prediction is the class with the highest probability.

All the weights and biases of the Deep Belief Net are then saved in an external file. This file can be loaded later to run the prediction on any user-input raw image and give a semantically segmented image.

**4.6.3. Design Constraints**

This module uses the sigmoid function as an activation function for both the unsupervised and supervised phases. Stochastic gradient descent is chosen as the optimization function. Changing these functions would require modifying the code base of the project.

It is important to note, however, that the module isn’t hardcoded to run for the specified number of iterations or build a specific number of RBM layers in a DBN. The code was written to allow testing different hyper parameters including:

* Number of stacked RBMs in the DBN.
* Number of hidden nodes in each RBM (separately).
* Learning rate for the RBMs.
* Number of training epochs for each RBM.
* Learning rate for the neural network.
* Batch size and dropout for the neural network.
* Number of iterations for the neural network.

**4.7. Application Module**

**4.7.1. Functional Description**

This module includes the user interface. It uses the trained models to produce a polygon map for raw non-labeled user-input images. Also, it produces a list of potentially infringing buildings.

**4.7.2. Modular Decomposition**

**Graphical User Interface**

A welcome message is displayed to the user on launch. The user is provided with dropdown list containing the possible use cases:

* Run a prediction using a 3-band image.
* Run a prediction using a 4-band image.
* Run a prediction using a 20-band image.

Upon choice of the use case, the user is provided with a window to specify the location of the input image.

The prediction starts to run and a timer is displayed to the user that starts with beginning of processing. When the prediction is ready, the results are showed to the user.

The polygon map displayed to the user uses a different color for each polygon according to the predicted class.

The results of the prediction as well as the detection are saved for the user to view independently of our software.

**Detection of Buildings surrounded by crops**

The following algorithm is used to detect buildings in agricultural lands.

For each building:

1. Calculate the area of the building in the predicted polygon list using the latitude and longitude coordinates.

2. Count the number of pixels in the vicinity of the building that are crops.

3. Multiply this count by a pre-determined coefficient.

4. If the result of the multiplication is larger than the building’s area, the building is flagged as a potential violation.

The coefficient should not be so small that nothing is detected, and not so large that any building adjacent to a crop is flagged. It was determined after many tests using trial and error.

**4.7.3. Design Constraints**

High quality satellite images are typically saved in the TIFF image format as it is the most suitable. This module assumes that the input image will be TIFF as well. In the case of multi-band images, it also expects the images to follow the same naming pattern and be in the same folder. The results are also saved in the same folder, following the same pattern.

**Chapter 5: System Testing and Verification**

In this chapter, we explain the steps we carried out to ensure that the project objectives have been realized and discuss the results.

Each module in the system was tested independently of the others to ensure it works as intended. This afforded extra ease and simplicity in the task of integrating the system because we only had to worry about the interfacing between modules containing potential. The complete system was tested after integration of each module successfully.

**5.1. Testing Setup**

The dataset that was used to feed the machine learning algorithms contains a subset with correct labels. We split this subset further into a training set and a validation set. The data in the training set was used to train the models, while the data in the validation set is used to verify the results.

This guarantees that the verification results are admissible because the training set is completely separate from the validation set.

**5.2. Testing Plan and Strategy**

The testing plan involves feeding the validation set to the trained model and obtaining the prediction. Of course, we already have the correct labels for the validation set. We compare the predicted label to the correct label for each pixel. An accuracy score is measured for each model. The accuracy score function is defined as follows:

Where |A| is the number of correctly predicted classes and |B| is the number of wrongly predicted classes. This score is used to compare the different models we generated in the project.

**5.2.1. Module Testing**

Other than the DBN and U-Net modules, there are some crucial parts in the rest of the system that were tested using a Python library called PyTest. It is a framework that facilitates writing small tests, and can also scale to support complex functional testing for applications.

This type of testing ensures the correctness of various functions in the Preprocessing and Feature Extraction modules.

**5.2.2. Integration Testing**

After development of each module was finished, it was integrated with other modules that interface with it. After each integration, sub-system would be tested to verify there are no issues. This approach has simplified the testing process compared to integrating all the modules at once.

**5.3. Testing Results**

The dataset 10 different classes such that each pixel can only be labeled as one of them. The classes are explained in the following table:

|  |  |  |
| --- | --- | --- |
| **Index** | **Class Type** | **Example** |
| 0 | Buildings | Residential and non-residential buildings, facilities. |
| 1 | Small structures | Barns, stables. |
| 2 | Roads | Highways |
| 3 | Tracks | Dirt track, cart track, footpath, trail. |
| 4 | Trees | woodland, hedgerows, groups of trees, standalone trees |
| 5 | Crops | Contour ploughing, cropland, grain crops, row crops. |
| 6 | Waterways | Rivers, canals. |
| 7 | Standing water | Lakes |
| 8 | Large Vehicles | Lorries, trucks, buses. |
| 9 | Small Vehicles | Cars, vans, motorbikes |
| 10 | Other | Anything not covered by the 10 classes. |

**Table 5.1. Dataset Class Types**

The following are the results of each model:

|  |  |  |
| --- | --- | --- |
| **Model Type** | **Number of bands** | **Accuracy Score** |
| U-Net | 3 bands | 76.25% |
| U-Net | 4 bands | 80.44% |
| U-Net | 20 bands | 79.41% |
| Deep Belief Net | 4 bands | 60.01% |

**Table 5.2. Accuracy of each model**

The model with the best accuracy is the U-Net model that was trained using only the 4-band subset of the dataset. This shows that the extra 16 bands are unnecessary for this task.

We also attempted duplicate the result of the winning model in less time. We ran the machine learning algorithm with a smaller subset of the dataset as the training set. This, however, resulted in poorer accuracy.

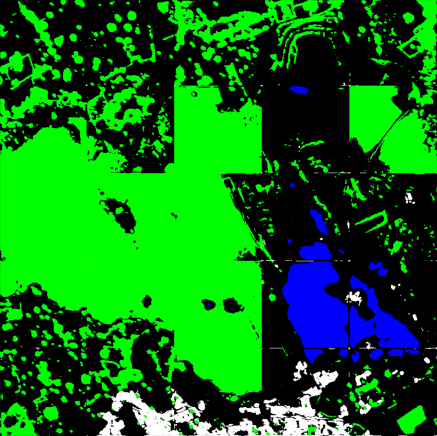
|  |  |  |  |
| --- | --- | --- | --- |
| **Model Type** | **Number of bands** | **Number of patches** | **Accuracy Score** |
| U-Net | 20 bands | 5000 | 80.44% |
| U-Net | 20 bands | 2500 | 75.37% |

**Table 5.3. Comparison between two training set sizes**

An example result of one image is presented in the following page.



**Figure 5.1. Example Input Image**



**Figure 5.2. Example Output Image**

**Chapter 6: Conclusions and Future Work**

In this chapter, we discuss the challenges we faced while building Eagle Eye and the methods we employed to overcome them. We discuss the experience we gained working on this project, and the conclusions we arrived at. A summarization of the whole project as well as its features and limitations is provided. Finally, we discuss potential enhancements and extensions to our work.

**6.1. Faced Challenges**

**Data Availability**

Getting a dataset suitable for the purposes of Eagle Eye was quite a challenge.

We needed a dataset that satisfied the following requirements:

* A large amount of labeled data.
* A high enough resolution to be able to distinguish objects.
* Good spectral band coverage; simple colored images aren’t enough.
* Diversity. We can’t depend on images of rural areas only. We need to include urban areas in the training phase as well. This is to avoid bias.
* Considering our main motivation for Eagle Eye, we needed images of Egyptian areas. This is to make sure that Eagle Eye can indeed be used for the purpose of detecting illegal buildings on agricultural lands.

In 2017, Kaggle hosted a data science contest related to satellite image. This contest was sponsored by the Defence Science and Technology Library (dstl) which is an executive agency of the Ministry of Defence of the United Kingdom. To support the contest, dstl publicly provided a large dataset of 20-band satellite images as well as polygon maps for a number of images for free. After preprocessing, this dataset was suitable for our purposes in every way but one; the images are from the UK and not Egypt.

In the early stages of our project, we contacted a representative of the Egyptian branch of Esri Northeast Africa which is an established GIS company. The company agreed to provide us with multiple 4-band satellite images of Egyptian areas. We used these images to validate the results of training using the dstl dataset.

**Performance and Computation Time**

Another big challenge we faced while building Eagle Eye was computation time and performance. This was a problem in both the training and the feature extraction phases. Using the dataset as-is to train would require infeasible time for computations.

The solution to this problem was two-fold. The first part of the solution was to use other hardware than our laptops. We uploaded the code to the Google Colab platform. Colab is a free service provided by Google that provides users with an interface to remotely run their code on Google servers which have powerful hardware suitable for most machine learning applications.

The second part of the solution involved modifying the dataset. For the U-Net module, each image was resized to a smaller size to reduce dimensionality. For the DBN module, images were quantized to 256 levels instead of the original 1024 levels.

**6.2. Gained Experience**

We derived many benefits from working on this project. Our research capability has improved significantly as we had to read many research papers to arrive at a good approach for solving the problem. Simply choosing the most recent research or the research with the best result was not enough. We had to consider which approaches would provide a good basis for understanding the field.

We gained a lot of knowledge in the fields of machine learning and image processing. We also had to learn new tools for the purpose of machine learning so we can build the project properly.

Working as a team for a long period of time taught us the value of team communication and organization. Evaluating our own abilities and skills as well as those of our teammates’ was crucial for efficiently dividing the work.

Overall, working on Eagle Eye was a great educational experience.

**6.3. Conclusions**

The U-Net module was used to train multiple models using the dataset provided by dstl. The U-Net architecture generates a model capable of predicting the class (among the 10 classes of labels) for each pixel in a given image with good accuracy. The best results are seen in the model which is trained on 20-band images, but the results from the 3-band and the 4-band models are acceptable as well.

The feature extraction module generates 16 features from a given 4-band image. These features have proved to be relevant enough for the purpose of semantic segmentation of satellite images. The module extracts these features in a timely manner, indicating good performance.

The DBN module was used to train a model using a transformed subset of the dataset provided by dstl. Only a subset is used because relevant features depend on 4 bands (Red, Green, Blue and Near Infrared) so the remaining bands aren’t used for training this model. This subset of the dataset is also transformed by going through the feature extraction module. This means that the data that is used to train the model, is very different from the original dataset.

The DBN architecture generates a model that is only somewhat capable of segmenting the input image. Convolutional networks are significantly better than traditional deep learning algorithms when it comes to semantic segmentation.

One of the reasons for this is the lack of consideration of spatial relationships in traditional deep learning algorithms. As explained, U-Net was among the first algorithms to provide localization in the task of semantic segmentation of images.

Finally, the application module is used to provide a class prediction of any user-input image under the specified constraints. The module can use any of the four models generated using the other project modules.

**6.4. Future Work**

The following are potential enhancements to our project:

* Develop a ranking algorithm to determine the most impactful features generated in the feature extraction phase for the Deep Belief Net. This can potentially reduce model training time, but it requires a lot of time to re-train the model on different feature subsets.
* Re-train the models on bigger image sizes. This requires expensive dedicated hardware (GPUs) to be done in a feasible amount of time.

The following are potential extensions to our project:

* Develop a module that benefits from the polygon map such as a pathfinding algorithm to provide users with a path from Point A to Point B on a map.
* Train a model to detect and classify different crops and their health. This requires much more data than what’s available at present.

**References**

1. Ciresan, D.C., Gambardella, L.M., Giusti, A., Schmidhuber, J.: Deep neural networks segment neuronal membranes in electron microscopy images. In: NIPS, 2012. pp.2852–2860.
2. Long, J., Shelhamer, E., Darrell, T.: Fully convolutional networks for semantic segmentation, 2014, arXiv:1411.4038 [cs.CV]
3. Ronneberger, O., Fischer, P., Brox, T.: U-Net: Convolutional Networks for Biomedical Image Segmentation, 2015, arXiv:1505.04597v1 [cs.CV]
4. G. E. Hinton and S. Osindero. A fast learning algorithm for deep belief nets. Neural Computation, 18:2006, 2006.
5. Saikat Basu, Sangram Ganguly, Supratik Mukhopadhyay, Robert DiBiano, Manohar Karki & Ramakrishna Nemani. DeepSat – A learning framework for Satellite Imagery, ResearchGate, 2015.

**Appendix A: Development Platform and Tools**

In this appendix, we explain the tools and platforms we used in building Eagle Eye.

**A.1. Software Tools**

**A.1.1. Development Platform**

Python is an interpreted, high-level, general-purpose programming language. It is the go-to language for fast prototyping as it was built for readability and low complexity.

**A.1.2. Preprocessing Libraries**

OpenCV is a library of programming functions that can be used to read and write images, as well as perform some basic operations on these images.

Shapely is a Python package for manipulation and analysis of planar geometric objects. We use this to handle the label data in the dataset.

NumPy is a library for Python that provides very fast performance in matrix calculations than plain Python functions. It is crucial as computation power is the main bottleneck.

**A.1.3. Machine Learning Libraries**

TensorFlow is a symbolic math library, and is also used for machine learning applications such as neural networks.

Keras is a neural-network library written in Python. It runs on top of TensorFlow. It was designed to enable fast experimentation with deep neural networks. This library is only used for the U-Net module, while the DBN module is implemented using lower-level TensorFlow functions.

**A.1.4. Graphical User Interface Libraries**

PyPlot is a Python library that provides a plotting framework.

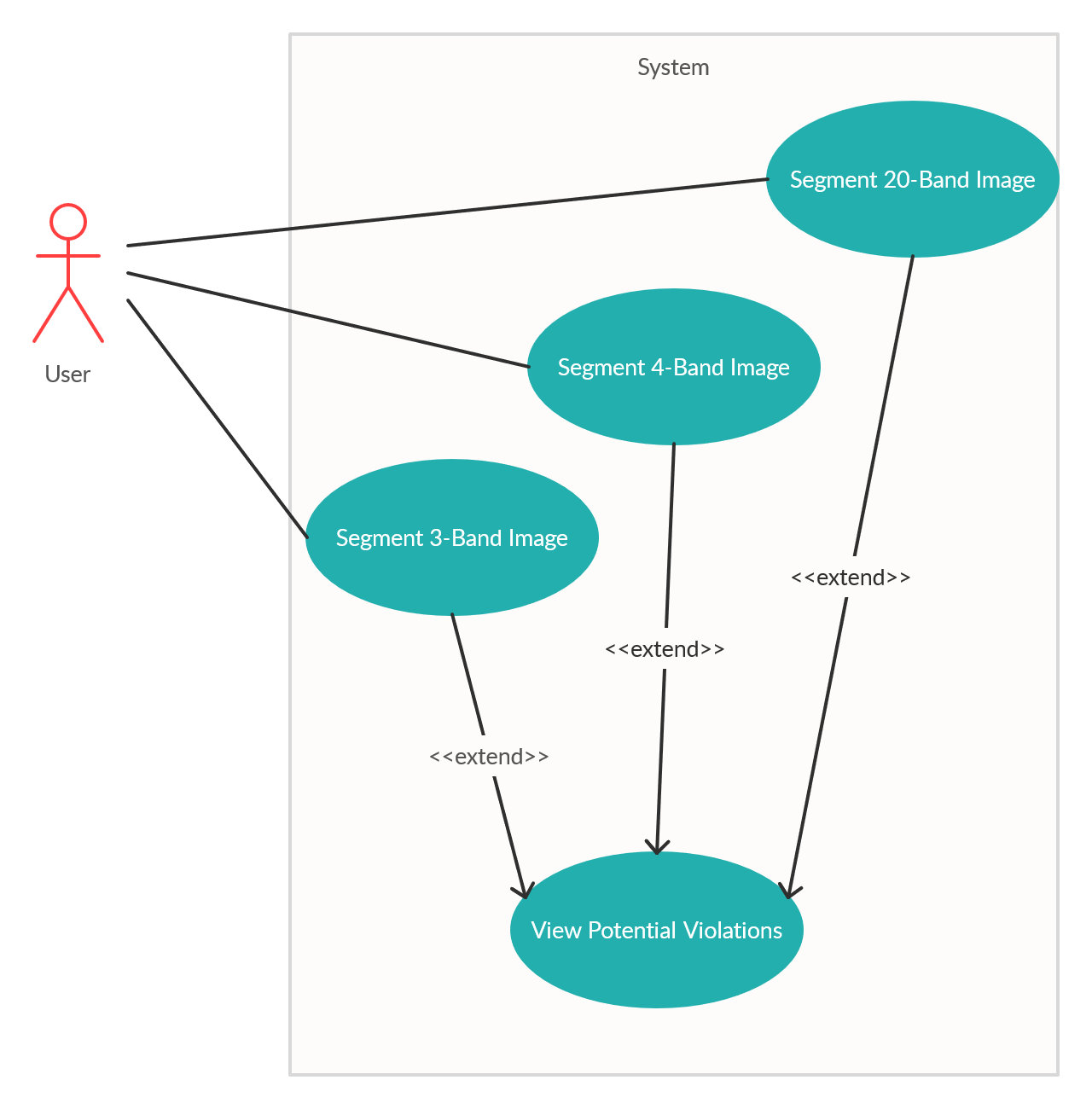
Pillow is a Python library that supports opening, manipulating and saving many different image file formats.

TKinter is the standard Python interface to the Tk GUI (Graphical User Interface) toolkit, which provides basic elements of GUI widgets.

**A.1.5. Unit Testing Library**

PyTest is a framework that facilitates writing small tests, and can also scale to support complex functional testing for applications and libraries.

**Appendix B: Use Cases**



**Figure B.1. Use Case Diagram**

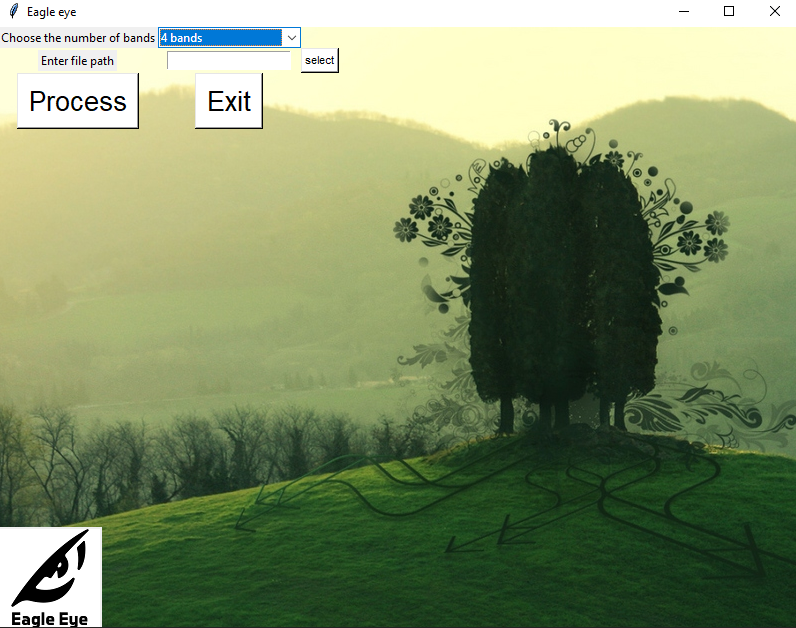
**Appendix C: User Guide**

A welcome message is displayed to the user on launch.



**Figure C.1. Welcome Screen**

The main window is then displayed to the user:



**Figure C.2. Main Window**

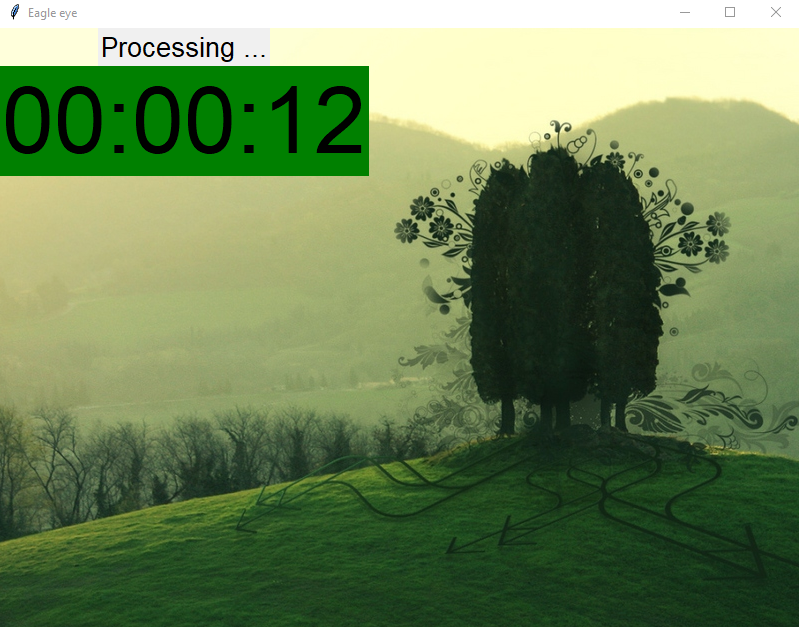
The user is provided with dropdown list containing the possible use cases:

* Run a prediction using a 3-band image.
* Run a prediction using a 4-band image.
* Run a prediction using a 20-band image.

The user can select one of these use cases press browse to choose the input image. After selection, the user can press Process to get the results.

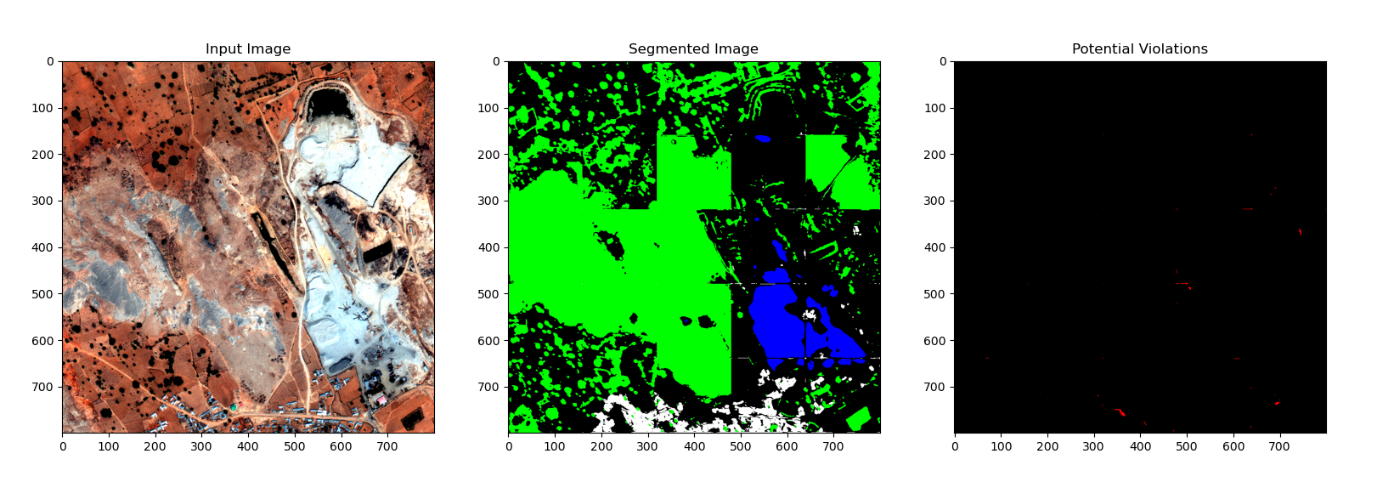
The prediction starts to run and a timer is displayed to the user that starts with beginning of processing.

The main window changes as follows:



**Figure C.3. Processing Screen**

When the processing is finished, the results are showed to the user in a separate window.



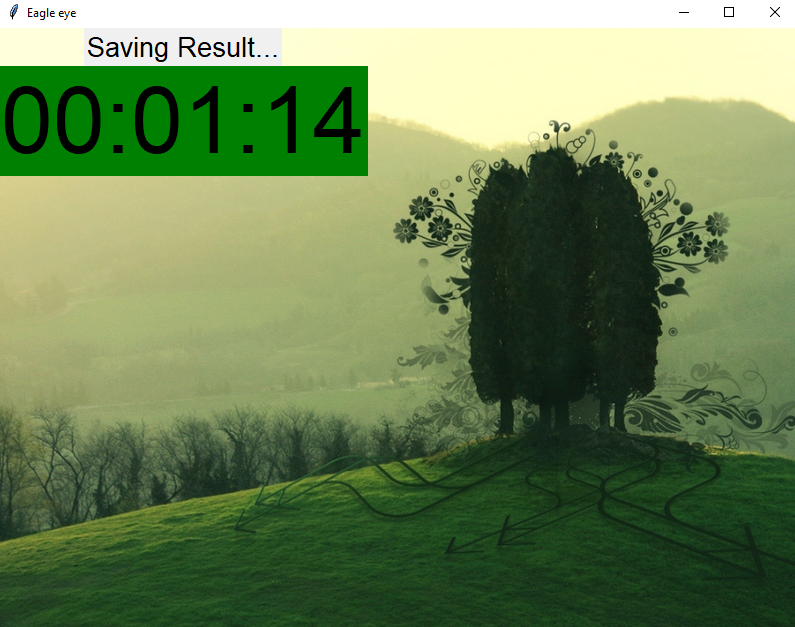
**Figure C.4. Results Window**

The polygon map displayed to the user uses a different color for each polygon according to the predicted class. The used colors are detailed in the following table:

|  |  |
| --- | --- |
| **Color** | **Class** |
| White | Buildings and Structures |
| Red | Potentially Illegal Buildings |
| Gray | Roads and Tracks |
| Green | Crops and Trees |
| Blue | Water |
| Cyan | Vehicles |
| Black | Other |

**Table C.1. Result Color Legend**

The main window is also changed as follows:



**Figure C.5. Final Step Screen**

The results of the prediction as well as the detection are saved for the user to view independently of our software. They are saved in the same folder.