International Journal of Advances in Engineering and Management (IJAEM)
Volume 4, Issue 1 Jan 2022, pp: 853-860 www.ijaem.net ISSN: 2395-5252

Land Use Feature Detection from Satellite Imagery using Machine Learning Models

Sushil Chandra[1], Udai Raj[2], Rajeev Sonkar[3], Ujjwal Yadav[4], Pragati Srivastava[5], Deepankar Acharyya[6], Prajjwal Singh[7],

[1]*Scientist-SF, Head, CIP & DM, Remote Sensing Application Centre, Uttar Pradesh, Lucknow, sushil.chandra@rsacup.org.in*
[2]*Scientist-SE, Remote Sensing Application Centre, Uttar Pradesh*
[3]*Project scientist, Remote Sensing Application Centre, Uttar Pradesh, Lucknow,*
[4]*Project Scientist, Remote Sensing Application Centre, Uttar Pradesh, Lucknow,*
[5]*Project Scientist, Remote Sensing Application Centre, Uttar Pradesh, Lucknow,*
[6]*student, Tezpur University, Assam, India*
[7]*student, Tezpur University, Assam, India*

Submitted: 15-01-2022 Revised: 23-01-2022 Accepted: 26-01-2022

ABSTRACT
Land-use feature detection is one of the hot applications of GIS (Geographic Information System). With satellite imagery as the forefront source of updated geographical data, we can use it to observe the land-use feature change and keep up with the latest changes with minimal effort and maximum efficiency. Already, many parties have started deriving and working on different methodologies to achieve this goal. Some of the approaches use the algorithms of Machine Learning and the performance level achieved are quite satisfactory. In this paper, we have explored some of the Machine Learning based approaches (Random Forest, XGBoost, U-Net, Artificial Neural Network) for land-use feature detection. We have used the online platform, google colab and online storage google drive to train our model and perform the prediction.

Keywords
Land use features, Satellite Imagery, Machine Learning, Deep Learning, Random Forest, XGBoost, U-Net, Artificial Neural Network(ANN).

I. INTRODUCTION

Popularity and demand for location based applications are increasing daily. The domain of GIS (Geographical Information System) has seen a wide range of developments in the recent time. With easy access to the vast amount of geo-spatial satellite data, many new applications have now been possible. The processes that required manual survey and analysis of the land now can be performed remotely and frequently with the help of satellite data. In this paper, we have tried to put forward some methods for one such application.

The term "Land-use" refers to the management and modification of natural environment or wilderness into built environments such as settlements and semi-natural habitats such as arable fields, pastures, and managed woods[1]. So in-short the land-use features refers to how a particular piece of land is being managed or utilized. The study of the land-use features is very important as it helps us understand the way how the world has been adapted to our needs and in what patterns. This will also help us to predict any future consequences that may take place due to our course of actions. Based on some basic characteristics, the land use features have been categorised into many different categories. Some of them include : Recreational, Agricultural, Impervious, Residential, Commercial, etc.

In this paper, we have restrained ourselves to detecting a limited number of land-use features due to some of our resource constraints.

II. OBJECTIVES

The primary objective is to propose and implement some machine learning models to detect different land-use features and present their performance levels. The training process requires high-end hardware and therefore we have used the online platform google colab (**RAM : 12.72GB,GPU backend**)[2] for the purpose. For storing purposes, we have used google drive, by linking it to the google colab platform.

International Journal of Advances in Engineering and Management (IJAEM)
Volume 4, Issue 1 Jan 2022, pp: 853-860 www.ijaem.net ISSN: 2395-5252

III. TOOLS & METHODOLOGY

Tools:

- Coding Platform : Google Colab [2]

Google Colab provides a jupyter notebook like interface to run python scripts.

- Storage : Google Drive [3]

Methodologies :

In our study, we have explored and used four Machine learning based approaches. These approaches are discussed in detail in the next section. For pre-processing and implementing the models, we have used different libraries which are listed along with the models in the next section. Some of the standard libraries are pre-installed in the google colab environment while others are required to be manually installed at the start of the session. The installation and all the relevant points has been covered and explained in the following sections.

NOTE: The full code for the implementation of the models are not mentioned in this paper. But the links for the code are provided in the corresponding sections and some code snippets are mentioned along with their explanation.

Model Concept and Architecture :

This section of the paper aims at providing a conceptual view to the readers regarding the model architectures that are going to be used for land-use feature detection.

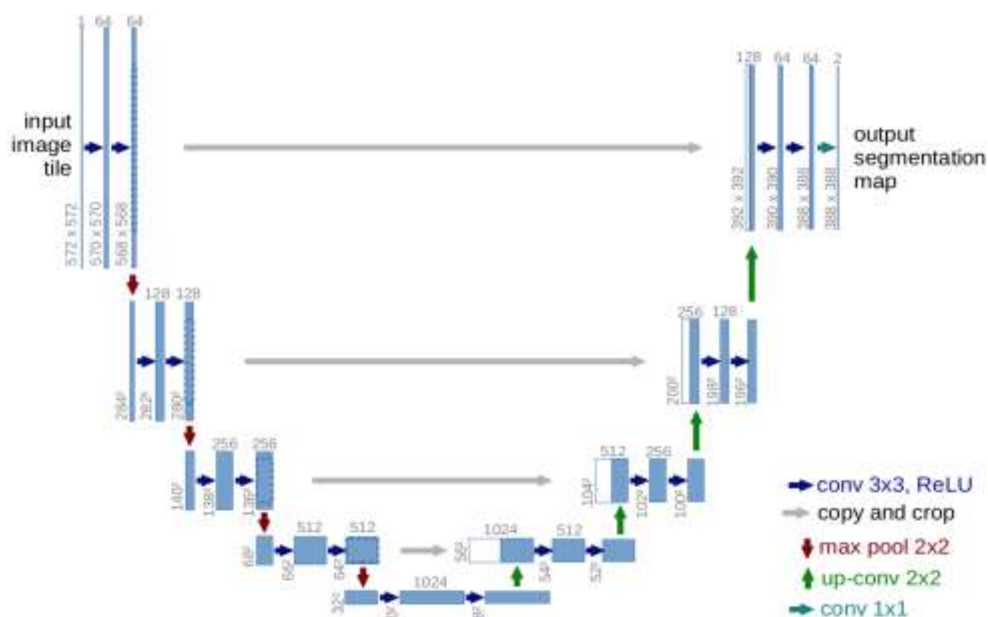
I. The U-Net Architecture:

The U-Net architecture was first designed and applied in the year 2015 to process biomedical images [5]. The model performed semantic segmentation on the bio-medical images, localizing the area of abnormality. But this network architecture didn't remain exclusive to the biomedical field only. Basically it is an image segmentation model, i.e. the model group together the pixels that have similar attributes[4]. It creates a pixel-wise mask for each object present in the input image. In a way, we can say that the model performs classification on every pixel of the input. Currently, U-Net is counted among the best models for semantic segmentation and is commonly used on satellite images.

Some of the basic characteristics of the model are:

- The input and the output shape of the image remains the same.
- The architecture is of U-shape and is symmetric (this point will be clarified in the later part of this section).
- The network is strong enough to predict well based on even few data sets by using excessive data augmentation techniques.

Diving deep into the architecture, below is a diagrammatic representation of the U-Net architecture. The U-Net is a Fully Convolutional architecture, i.e. there are no dense/fully connected layers in the model architecture.



The above diagram shows the model's basic foundation. As can be clearly seen, the U shape of the network and hence the name, U-Net. The architecture is fully symmetrical and can be divided into 2 parts: the contracting path and the expanding path. The left part of the model, also known as the contracting path is responsible for feature extraction from the input. The left part is constituted by the general convolution process and max pooling. Whereas the right part of the network also known as the expanding path is constituted by a number of upsampling processes (transposed 2d convolutions) and convolution layers. The right part of the network makes sure that once the features are identified by the left part of the network, it is scaled up to the original input size. The contracting and the expanding paths are also connected via the skip connections. These skip connections help in preserving the shape data of different objects present in the image. The network combines information from the downsampling/contraction path with the contextual information in the upsampling/expanding path to finally obtain a general information combining localization and context, which is necessary to predict a good segmentation map.

Model Overview:
This U-Net based model takes in a 3 bands (RGB) input and outputs a binary mask
localizing the buildings in the image.

Dataset:
The dataset contains a set of aerial images. These images were collected with the help of drones and are available in png format. Corresponding to every aerial image, there is a binary mask file in the label folder that highlights the buildings present in the image. This dataset that we have used for training the model is not satellite data but rather a colored aerial image. But we were still able to predict on the satellite images considering only the RGB bands.

Libraries:
- Numpy
- Rasterio
- Matplotlib
- Os
- Skimage
- Fastai

All the libraries are installed in the google colab environment by default except the rasterio library. This can be installed by executing the following command :

```
!pip install rasterio
```

The Training Process:

The whole process of training the model can be divided into several stages. The first stage is the pre-processing stage. In the pre-processing stage, we split the whole dataset into training and testing data sets and apply data augmentation operations on it.

In the next stage, we define the model and start the training. For this model, we have used the fastai library. The U-Net model is already implemented in fastai library and we can also define the architecture on which the U-Net architecture will be based on.

```
learn = unet_learner(data, models.resnet34, metrics=metrics)
```

This creates a unet architecture based on the resnet model. For the metrics part, we can define our own metrics or use predefined functions.

For training the model, we will use the fit_one_cycle() function. This function uses large, cyclical learning rates to train models significantly quicker with higher accuracy. When training Deep Learning models with Fastai it is recommended to use the fit_one_cycle() method, due to its better performance in speed and accuracy, over the fit() method.Instead of using a fixed, or a decreasing learning rate, the CLR method allows learning rate to continuously oscillate between reasonable minimum and maximum bounds.

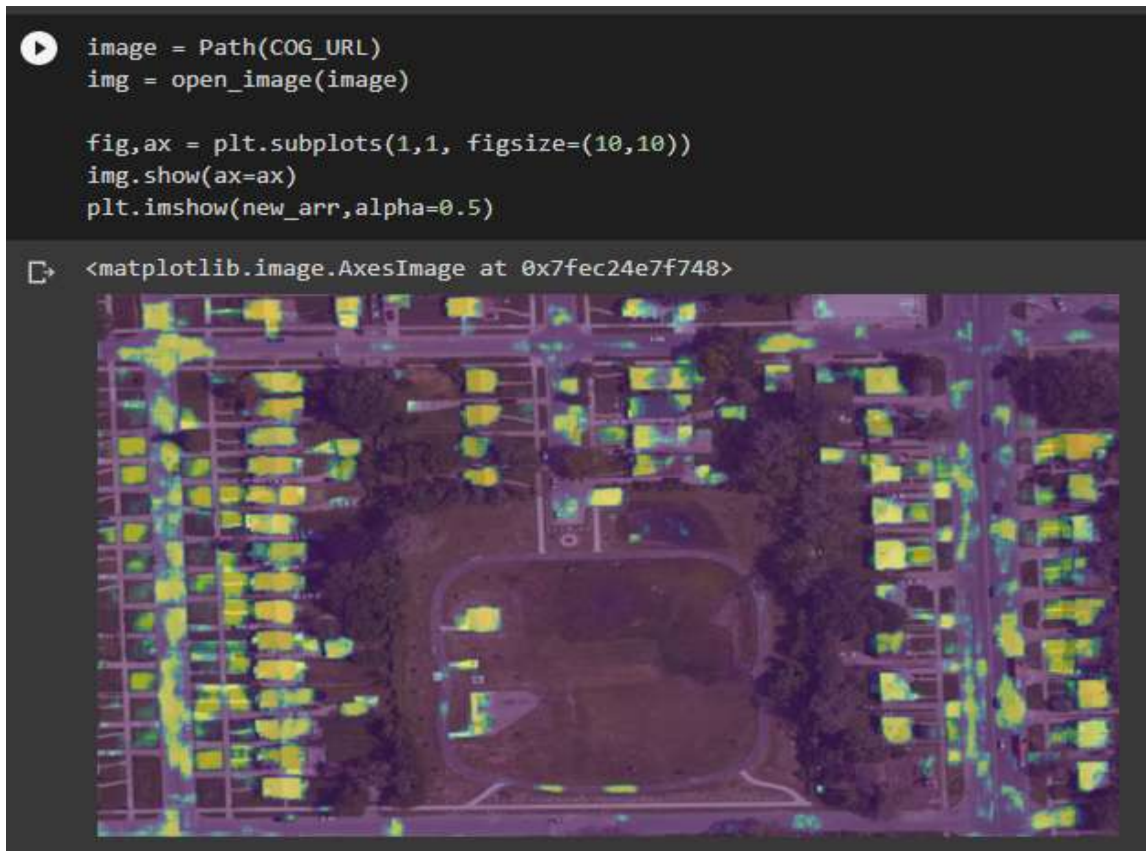
```
learn.fit_one_cycle(13, max_lr=lr,
          callbacks=[
            SaveModelCallback(learn,
                  monitor='dice',
                  mode='max',
      name='20190108-rn34unet-comboloss-alldata-512-best')
      ]          )
```

The Prediction Process:
The process of prediction follows a similar sequence of stages as in the training process. For the pre-processing part, we divide the whole input into tiles of size of the UNet input layer.The prediction will be performed on the tiles and the prediction output is stitched together.
The prediction is performed by the following function:

```
def get_pred(learn, tile):
  t_img = Image(pil2tensor(tile,np.float32))
  outputs = learn.predict(t_img)
  im = (outputs[2][1]).numpy()
  return im
```

The final prediction when overlaid upon the input gives the following output:

International Journal of Advances in Engineering and Management (IJAEM)
Volume 4, Issue 1 Jan 2022, pp: 853-860 www.ijaem.net ISSN: 2395-5252



II. Random Forest:

Random forest is a type of supervised learning algorithm. It is an ensemble of decision trees, usually trained with the bagging method (the general idea of the bagging method is that a combination of learning models increases the overall accuracy of the result). Random forests are an ensemble learning method for classification, regression, and other tasks that operate by constructing a multitude of decision trees and each individual tree outputs a class prediction and the class with the most votes. The key in the model of random forests is the low correlation between the individual constituent decision tree models. This ensures that the trees protect each other from individual errors. The Bagging (Bootstrap Aggregation) process ensures that the models are trained on different sets of data which leads to diversification of each individual constituent model. Along with different training data sets, feature randomness is also ensured (trees use different features to make decisions).

Model Overview:

The model is based on the random forest algorithm. It takes in a 5 band (B1, B2, B3, B4, B5 of Landsat 8 data) input and outputs the vegetation cover label for the input data. The output labels for this model are classified into the 5 classes : No Vegetation, Bare Area, Low Vegetation, Moderate Vegetation, High Vegetation.

Dataset:

The dataset to train this model was prepared by us. We used the concept of NDVI (Normalised Difference Vegetation Index) to get the vegetation cover labels. For Landsat 8 data, NDVI is calculated using the band 5 (Near Infrared Band) and band 4 (Red band). Once we calculated the NDVI for each pixel, we binned the NDVI values into 5 bins. The NDVI values only ranges from 0 to +1 but we included the infinity values, since we replaced the missing data with large numeric values.So we downloaded the Landsat 8 data and performed the above process to get the labelled training data from the original data.

Libraries:

- Os
- Numpy
- Matplotlib
- Glob
- Sci-kit learn (Sklearn)
- Earthpy
- Pickle
- Rasterio

International Journal of Advances in Engineering and Management (IJAEM)
Volume 4, Issue 1 Jan 2022, pp: 853-860 www.ijaem.net ISSN: 2395-5252

All the required libraries are installed in the google colab environment by default, only the earthpy and rasterio libraries are needed to be manually installed at the start of the session by executing the following commands:

```
!pip install earthpy
!pip install rasterio
```

The Training Process:

For the implementation of this model, we have to first construct the training data. We have used the concept of NDVI to label the data. The Earthpy library provides a number of useful functionalities to do the processing. First we get the Landsat 8 data. The Earthpy library provides a function to do that. Then we list out the paths of the geo-tif files (for the individual bands) and stack them into a single unit. We have replaced the missing data fields with -9999.

```
arr_st, meta = es.stack(list1, nodata=-9999)
```

Now the NDVI is calculated using the band 4 (the red band) and band 5 (the near infra-red band):

$$NDVI = \frac{\rho_{NIR} - \rho_{red}}{\rho_{NIR} + \rho_{red}}$$

```
ndvi = es.normalized_diff(arr_st[4], arr_st[3])
```

Then we categorised the ndvi values using the process of binning. All the steps till now constitute the data preparation part.

Next we split the data into training and testing sets and train the model. For selecting the number of estimators for our model, we have plotted the oob score v/s number of estimators graph. And from the graph we get an optimal outcome at a number of estimators around 200.

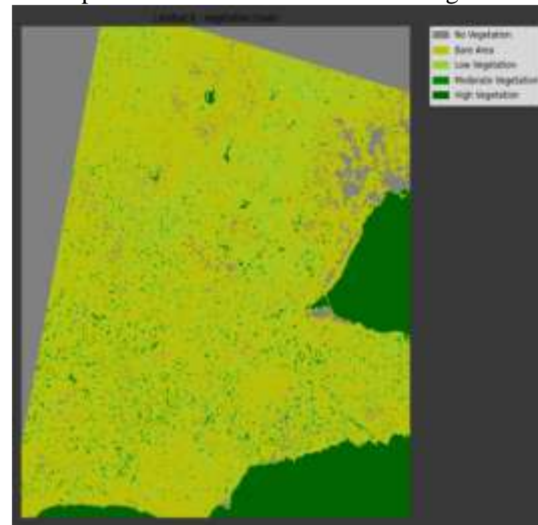
Then we train our model using the following set of code:

```
# Initialize our model with 200 trees
rf = RandomForestClassifier(n_estimators=200, oob_score=True)
# Fit our model to training data
rf.fit(X_train[:800000], y_train[:800000])
```

The Prediction Process:

So for the prediction, we load the geo-tif files and we stack them and reshape them. The prediction is performed using the following code snippet:

The output of the model is of the following form:



III. Artificial Neural Network:

Artificial neural networks/neural networks/feed-forward networks are a system/network of computing units connected together, arranged in a layer. The computing units constituting the neural network are known as neurons. And the connections between the neurons are characterized by a weight value. During the training of the model, the weight values are updated such as to reach an optimal configuration such that the difference between the model output and the actual result is as small as possible. This is achieved via various optimization algorithms such as backpropagation. Here in the above figure, we can see that every neuron of the previous layer is connected to each neuron of the next layer. Such a type of configuration is called a fully connected layer or a dense layer. There are other types of layers available such as a convolutional layer that performs the convolutional operation on the input or a max-pooling layer that performs the downsampling operation. It totally depends upon the designer as well as the objective, the model is trying to achieve.

Model Overview:

This model takes in a satellite image and identifies the built-up surfaces. By built-up surfaces, we refer to the part of the earth's surface that is covered by man-made/artificial surfaces.

Dataset:

The Dataset is a labelled tif file. It is Landsat 5 data containing information from the 6 bands (B1, B2, B3, B4, B5, B6, B7). The data was annotated for the built-up areas. For the testing part, we have the Landsat 5 data.

International Journal of Advances in Engineering and Management (IJAEM)
Volume 4, Issue 1 Jan 2022, pp: 853-860 www.ijaem.net ISSN: 2395-5252

Libraries:

- Pyrsgis - Numpy
- Tensorflow (Keras)
- Sci-kit learn (Sklearn) - Raterio
- Matplotlib
- Glob - Os

All the required libraries are installed in the google colab environment by default, only the pyrsgis and rasterio libraries are needed to be manually installed at the start of the session by executing the following commands:

```
!pip install pyrsgis
!pip install rasterio
```

The Training Process:

The purpose of the pre-processing step is to read in the raw Landsat 5 band data and format it into numpy arrays, which are easy to manipulate and process. In our program, we have used the pyrsgis library to handle the pre-processing operations during the training phase. So, first we read in the tif files using the pyrsgis's raster.read() function. This function readily reads the data into numpy arrays. We read in a total of 2 files: the multi-spectral image (the training image), the Bangalore built-up image (the training label). We observed that it is a 3D numpy array, but for training the model we need a 2D numpy array (pixel, corresponding band values). So we use the changeDimension() to perform the necessary transformation.

The next stage of the pre-processing step includes the splitting of data into testing and training sets and normalizing them.

Then we define the structure of our ANN model. Here we have used the Keras library to implement the model.

```
model = keras.Sequential([
    keras.layers.Flatten(input_shape=(1, nBands)),
    keras.layers.Dense(14, activation='relu'),
    keras.layers.Dense(2, activation='softmax')])

model.compile(optimizer="adam",
loss="sparse_categorical_crossentropy",
metrics=["accuracy"])
```

And proceed with the training:

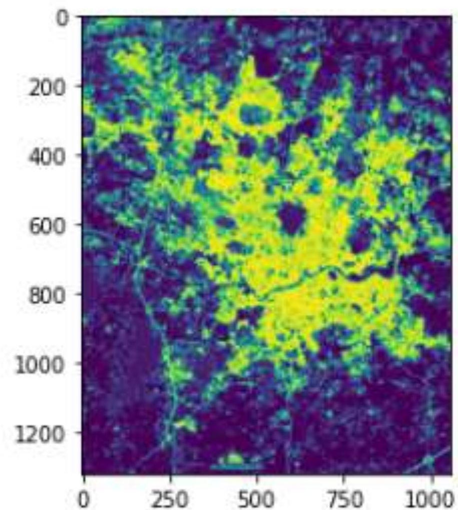
```
model.fit(xTrain, yTrain, epochs=2)
```

We get an accuracy of 95.88% on the training data . Then we make the model predict on the test data and found out the accuracy score to be 95.9% .

The Prediction Process:

Landsat 5 data, the bands are by default not well stacked into a single geo-tif file. So first we have to create a single stacked geo-tif file. For this purpose, we have used the rasterio library.Then we read in the stacked tif file and perform the pre-processing steps. For prediction, we can directly call the predict() function upon the testing data.

This gives us the predictions. In-order to visualize the prediction, we have to reshape the prediction array to the original image shape and use the matplotlib.pyplot's imshow() function. The output is of the following form, where the light areas refers to the built-up surfaces.



IV. XGBoost:

XGBoost is an open source library of highly-performance implementation of gradient boosted decision trees. So in short it is a decision tree based ensemble machine learning algorithm that uses a gradient boosting framework. Boosting is a type of ensemble technique in which models are trained in succession, with each new model being trained to correct the error made by the previous ones. Models are added sequentially until no further improvements can be made. This technique overcomes the possible situation of the other ensemble techniques (in which the models are trained in isolation) in which all the models end up making the same mistake. Gradient Boosting specifically is an approach in which new models are trained to predict the residuals/errors of the prior models. Gradient Boosting is a special case of boosting where errors are minimized by gradient descent algorithm.

Model Overview:
The model is a XGBoost based model, that utilizes the given 3 bands' information and predicts the 8 LULC features (if present) : Buildings, Misc, Road, Track, Trees, Crops,Waterway, Standing water. Although the dataset contained annotations for 10 classes/LULC features, we limit our model to be trained for 8 features only due to the limitations on our processing power.

Dataset:
The dataset used for training this model is from a Kaggle competition named DSTL Satellite Imagery Feature Detection [7]. The goal of this competition is to detect and classify 10 types of objects in the given regions (1Km x 1Km) of satellite imagery provided by the Defense Science and Technology Laboratory (DSTL). The dataset consists of 450 images, 25 of them have training labels. The satellite images are provided in both 3-band and 16-band formats. The 16-band format includes Panchromatic (450-800 nm), 8 Multispectral (red, red edge, coastal, blue, green, yellow, near-IR1, and near-IR2)(400 nm - 1040 nm) and 8 SWIR (1195 nm - 2365 nm). In this model, the 3 bands (RGB) are utilized to train the model. Inside the DSTL dataset, there are a number of files, each serving its own purpose.

- Grid_sizes.csv : Purpose : for scaling the polygons
- Train_wkt_v4.csv : Purpose : the polygons for feature
- Sample_submission.csv : Purpose : the test set
- Three_band : 3 band images (training RGB images)

Libraries:
- Pandas - XGBoost - Numpy - OpenCV (cv2)
- Matplotlib - Shapely - Sci-kit learn - Tifffile
- Os - Pickle

In this case, all the required libraries are installed in the google colab environment by default.

The Training Process:
For the pre-processing step, we first convert the given polygons to masks with proper scaling. Once the masks are available for the training data, we divide the whole training data into the training and testing tests.
Next we define the pipeline including the StandardScaler followed by XGBClassifier and then we can start training our model. This is represented in the following code snippet:

```
clf = make_pipeline(StandardScaler(), xgb.XGBClassifier(nthread=-1, learning_rate = 0.1, n_estimators=100, tree_method='exact'))
clf.fit(train_x, train_y)
```

The Prediction Process:
For the prediction purpose, we have defined a predict_class() function, that reads in the input file, reshape it, makes the predictions, and converts them to masks. The code snippet for this part is shown below:

```
def predict_class(image_id, clf, threshold = 0.05):
   ## Get test data
   test_x, test_image_shape = image_to_test(image_id)
   ## Make predictions
   pred_y = clf.predict_proba(test_x)[:, 1]
   ## Convert predictions to mask
   pred_mask = pred_y.reshape(test_image_shape)
   return pred_y,pred_mask
```

The final output is of the following form:

IV. CONCLUSION

The hardware limitation and the availability of training data did limit our exploration expedition, but still we were able to complete our objectives. We successfully implemented the models to detect land-use features.

One of the major observations that we got from this study is that the model performed well with data that were similar to the training data, or in other words that belonged or are near to the training data location. This is basically due to the fact that the archaeological and land-use planning and pattern varies with location. This statement is true for both natural land use features as well as for the artificial land use features.

REFERENCES

[1]. https://en.wikipedia.org/wiki/Land_use
[2]. https://colab.research.google.com/
[3]. https://www.google.com/intl/en_in/drive/
[4]. https://www.analyticsvidhya.com/blog/2019/04/introduction-image-segmentation-techniques-python/
[5]. https://arxiv.org/abs/1505.04597
[6]. https://towardsdatascience.com/unet-line-by-line-explanation-9b191c76baf5
[7]. https://www.kaggle.com/c/dstl-satellite-imagery-feature-detection
[8]. https://www.fast.ai/about/
[9]. https://keras.io
[10]. https://en.wikipedia.org/wiki/Normalized_difference_vegetation_index#:~:text=The%20normalized%20difference%20vegetation%20index,observed%20contains%20live%20green%20vegetation.
[11]. https://www.usgs.gov/core-science-systems/nli/landsat/landsat-8
[12]. https://www.usgs.gov/core-science-systems/nli/landsat/landsat-5?qt-science_support_page_related_con=0#
[13]. https://earthpy.readthedocs.io/en/latest/get-started.html
[14]. https://github.com/PratyushTripathy/pyrsgis/releases
[15]. https://rasterio.readthedocs.io/en/latest/