Semantic Segmentation of Aerial Imagery

Nikita Jayakar[1] **and Richanshu Jha**[2]

[1]**nmj303@nyu.edu**
[2]**rj1469@nyu.edu**

1 PROBLEM STATEMENT

These days, there are many applications which require the involvement of creation of geospatial representation of cities. It is possible to capture high resolution satellite images from satellites or drones for the goal of collecting this data. However, if one has to manually classify or segment the cities based on roads, buildings and green cover, it becomes a really time consuming task prone to human error. This applies not only to roads and buildings (The primary scope of this project), but also vegetation cover. This information is essential to generate the mapping of buildings and roads, and vegetation data is used to calculate the percentage of greenery in any city/geographical area and essential in forming various environmental metrics for the area. While vegetation in a city/geographical area may be found out reliably by using HSV-based thresholding algorithms, this is not true for roads/buildings. Due to this, deep learning image classification algorithms, such as Convolutional Neural Networks(CNN), become essential tools for satisfactory and feasible classification of roads and buildings.

2 LITERATURE SURVEY

The aim of this project is to target especially the urban geographical areas to analyze the mapping of roads, buildings and green cover. (Muruganandham S., 2016) It is a very useful technique to use deep learning to build image classifiers based on Fully Convolutional Networks(FCNs). Modified versions of CNNs can also be used for regularization, optimization and achieving higher model accuracies of the image classifier. Hence, an image classifier can be built on satellite images using deep learning techniques like CNNs. (Wurm and Taubenböck, 2019) Since analysis is to be done on urban areas, there is a journal paper which focused on a very specific use-case for analysis of urban cities. It focused on semantically segmenting the images of slum-areas which is also a big problem in some of the urban cities in the world. This segmentation on a very specific domain was performed using transfer learning on deep learning FCNs from one dataset to another. (Li and Yu, 2019) U-Net-based semantic segmentation can be used to get very less difference between the validation dataset and the actual required output of the building footprint extraction from the satellite images data. Building footprints can be classified and extracted quite accurately from the given images dataset. (Ishii T. and R., 2016) It is also vital to perform tuning on the threshold to be applied for getting optimal performance on the semantic segmentation of buildings after applying FCNs on the images.

3 DATA SET GENERATION

3.1 Why generate data

We did a fair amount of research on the internet with regard to the data sets available for our project. While segmented Geo spatial imagery data is readily available online, there are lots of issues with combining the (relative to the task) small data sets. Some are focused on rural areas, some on urban areas, and there is a lot of variation on file formats, sizes and resolutions.We have opted to stray from the convention of using pre-existing data and will be creating the data set as part of the project. For the task of creating the data set, our ground-truth will be extracted from google maps. For this, we have developed a data extraction pipeline which has 3 modules: The Google maps module, the Image Extraction module, and the data preparation module. These are presented below.

3.2 Google maps module

The first module will be a browser-based google maps module that will have two instances of Google Maps open on the same screen. This will be done by calling Google APIs on a browser window. One instance of maps displays the satellite image of a predetermined area, and the other instance of the map is programmed to show the roads and buildings of the same location. Segmenting all buildings into a single color channel was not possible using Google APIs. However, it is possible to configure the map such that they can be extracting via thresholding RGB values. These maps are configured to follow a predetermined straight line path through a selected area. Both these maps run simultaneously giving the satellite imagery and labelled roads/buildings view side by side.

3.3 Data extraction module

The image extraction module is a simple Python command line program with the capability of taking screenshots or window captures of the Google maps module. These images are taken at regular intervals based on the the panning speed of the google maps module and would perform basic image splitting and cropping to provide two raw images per screenshot (One for the satellite imagery and one for the road/buildings view). The image extraction and google maps modules are run simultaneously to generate the raw data set. The images that are generated are saved in JPEG format.

3.4 Data preparation module

The data set preparation module is run after a significant amount of raw data has been collected. In this module, the raw pairs of images generated from the previous step are processed as follows. The images and labels are both cropped according to the image dimensions required by the model, which is (464, 464). The 'road/buildings views' image are threshold against various sets of values to extract the data of roads and buildings from the raw label. This data is stored in the as one-hot encoded channels in the final label. The second component of the Data preparation module is the Vegetation segmentation.

For getting information about vegetation, the process involves generating a mask of all trees in the input image, and merging that data with the corresponding label of the image. Initially, we tried to threshold the RGB intensity values of the image with the consideration that the intensity value of green would be relatively higher than red and blue. This method was unable to give a reliable segmentation. We experimented with a number of techniques and finalized on thresholding the Hue, Saturation and Brightness values of each pixel to segment those with vegetation. For this, first the final input image is converted to an HSV format. It is then masked with configurable set threshold and sensitivity parameters. These parameters have been obtained after rigorous testing with various images in our dataset and the segmented outputs were reliable as ground truth for training. In the data preparation process, this generated layer of vegetation is added as a third channel to the already generated roads/buildings label which now has a shape of (464,464,3).

The images and corresponding labels then undergo a process of data augmentation where the images are rotated, flipped etc. which further increases the data size. The augmented final images are saved in JPEG format. For the final labels, the format was initially a grayscale image where pixels containing roads, buildings and vegetation were be encoded as two discrete intensity values. However, JPEG compression was causing errors in the magnitude of 0-5 intensity values. We cannot risk altered pixel values while converting and saving labels. Due to this, the final labels are being saved as binary numpy files (.npy) which reliably store the encoded labels for each pixel. The image below shows one such pair of augmented jpeg image and corresponding .npy label which has been encoded as a greyscale image.

Figure 1. Augmented final image

Figure 2. Augmented final label

4 MODEL AND TRAINING

4.1 Model selection

The primary deep learning task that this project is involved with is semantic segmentation of images. Thus, after considering multiple options such as Gated-SCNN, YOLACT, U-NET, Fast FCN etc. U-Net architecture allows for multiple advantages; it is computationally cheaper than its alternatives, it works well with smaller datasets making it inherently easier to test with, and we found multiple sources where the model was used for semantic segmentation of bio-medical imagery which is a similar paradigm to this project. Due to these, we decided to opt for the U-NET Architecture. Keras with Tensorflow has been used to implement this model. The architecture and details are follows.

4.2 Model architecture

The input to the U-net model is the self-generated dataset of final images along with their labels in .npy format. The input format of the images for the constructed U-Net Model is of the dimension (464, 464) for 3 input channels since the dataset is of colored (RGB) satellite images. The predicted labels are of the format .npy with the same dimensions as the images. For the construction of the U-Net model, ReLU activation function is applied for every down-convolution. The starting number of neurons for this model is then set to be 8. In the U-Net model, every down-convolution in 2D is performed twice before using the MaxPooling operation. MaxPooling at every down-convolution step is performed to down-size the input data dimensions into almost half. In the model which is built, double down-convolution with MaxPooling is used four times. At the middle of the U-net model, the down-convolution is again performed twice. Since the number of down-convolutions except for the down-convolution in the middle is performed four times, the number of up-convolutions to be performed to complete the U-shape of the U-net model is also performed four times. It was noticed that the down-convolution with the 'same' padding preserves the dimensions of the data at every step after performing up-convolution. Hence, 'same' padding is chosen for this model over 'valid' padding. Performing up-convolution in this model ensures that the output of this model will have the same dimensions as the input data of images.

4.3 Model training and hyperparameters

To compile and test this model, the Adam optimizer is used for the gradient descent. The hyperparameters adjusted for this model are especially the learning rate which is set to $5 * (1e-4)$ for better accuracy of the model. The loss function used for training this model is 'Categorical crossentropy'. This goes hand-in-hand with the Adam optimizer for the gradient descent of the model. Categorical crossentropy is used since the number of labels for segmentation of the image is 3: roads, buildings, and vegetation cover. This type of loss function computes the cross-entropy loss between the predictions and the actual output for two or more classes(categories). The output predicted labels for this loss function in Keras with Tensorflow have to be provided in one-hot encoding format. Hence, the actual output labels of the model using this loss function is in the form of One-hot encoding. Generally, this loss function is used when the output should have labelled image segmentation of either two or more classes. One of the most important hyperparameters for training the model with self-generated dataset is setting callbacks for using optimal number of epochs for getting the best possible accuraries after training the model. Using callbacks like ModelCheckpoint, ReduceLROnPlateau, and EarlyStopping can possibly give the better results and avoid both underfitting and overfitting of the input data. The diagram of the model architeture is below, the total number of trainable parameters are: 540,235.

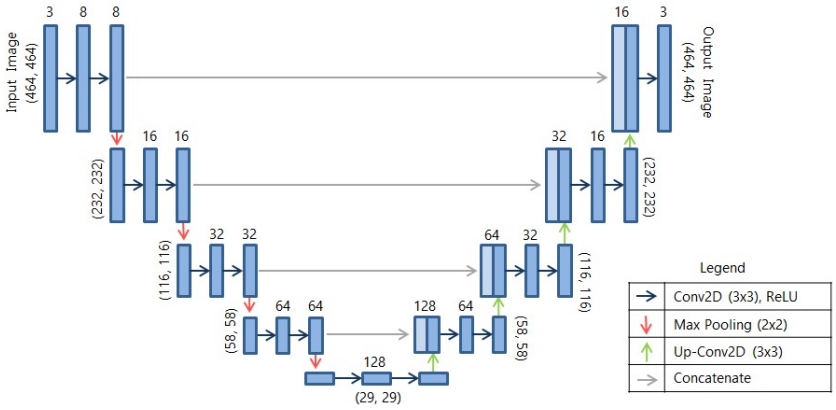

Figure 3. Architecture of the model

The first callback function, Early Stopping stops the training if the loss after every epoch does not decrease or in some cases, keeps on increasing. The second callback function, ReduceLROnPlateau is used to ensure that the learning rate should be reduced if the validation loss does not decrease over a certain number of epochs while training the data. The number of epochs over which it is decided whether or not to reduce the learning rate after stagnation of the model is called patience. Validation loss, in this model, is the quantity which is monitored for deciding whether or not to reduce the learning rate over a patience number of epochs. The third callback function, ModelCheckpoint is used to ensure that the best version of the model compiled so far over a certain number of epochs is saved along with the model weights.

5 PRELIMINARY RESULTS, CHALLENGES, AND REMAINING WORK

The development of the data extraction/processing pipeline and module for training/validating the model is complete. We have trained various models with increasing number of data points and have till now, have completed the labelled data generation/processing for 1680 data-points which is roughly 1GB of data. Our preliminary model has been trained with the aforementioned 1680-datapoint dataset with a batch size of 6 for 12 epochs. From the validation output as shown below, it is evident that further training is need for vegetation, and also that it is unable to identify the space between buildings. This is because the output of the model is a one-hot encoded vector of size 3 (buildings, roads, vegetation). The solution to this will be to have a 4th channel corresponding to a 'none of these' class. We are also developing a custom data generator module and (time permitting) Amazon s3 integration to reduce the memory and storage footprint. This allows us to increase the number of neurons per layer. The final model would be trained for a much larger generated dataset, include the aforementioned 4^{th} channel and have improved accuracy.

Figure 4. Validation image

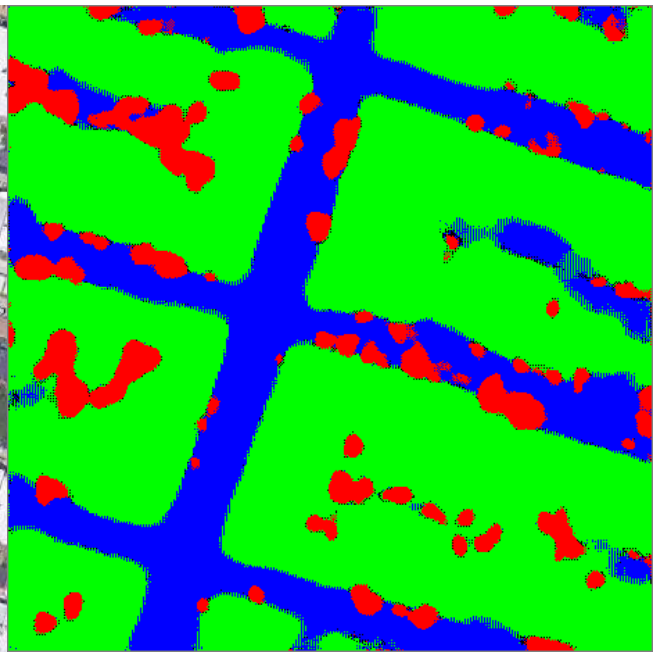

Figure 5. Validation label

REFERENCES

Ishii T., Simo-Serra E., I. S. M. Y. S. A. I. H. and R., N. (2016). Detection by classification of buildings in multispectral satellite imagery. *23rd International Conference on Pattern Recognition (ICPR) Cancún Center*, (3344):3344–3349.

Li, W., H.-C. F. J. Z. J. F. H. and Yu, L. (2019). Semantic segmentation-based building footprint extraction using very high-resolution satellite images and multi-source gis data. *Remote Sensing*, (403):1–19.

Muruganandham S., R. M. (2016). Semantic segmentation of satellite images using deep learning. *Faculty of Electrical Engineering, Department of Cybernetics*, pages 51–62.

Wurm, M., S. T. Z. X. W. M. and Taubenböck, H. (2019). Semantic segmentation of slums in satellite images using transfer learning on fully convolutional neural networks. *ISPRS Journal of Photogrammetry and Remote Sensing*, (150):59–69.