Satellite Imagery Feature Detection using Deep Convolutional Neural Network, U-NET

(Final Report)

Ardavan Sassani
Department of Computer Science
Georgia State University
asassani1@gsu.edu

Abstract

We implemented a U-net, a deep fully convolutional network to segment the input satellite imageries into ten classified classes based on multispectrum channels. We tried two different training methods; first, we trained all ten classes simultaneously with an IOU accuracy of 0.07. then, we trained ten models for each class individually and got more accurate results of 0.15. The accuracy of the model is varied among classes. The highest values belong to the building, trees, and corps classes and the lowest values are for cars and trucks. The main reasons for this error are the small dataset size, which is only 25 annotated images, and the heavily imbalanced distribution of classes.

1. Introduction

Semantic segmentation for images can be defined as partitioning and classifying the image into meaningful parts and classifying each part at the pixel level into one of the pre-defined classes [1]. Semantic segmentation faces an inherent tension between semantics and location: global information resolves what, while local information resolves where [2]. While convolutional networks have existed for a long time, their success was limited due to the size of the available training sets and the size of the considered networks [3].

One issue of this project is the small training dataset size with the total number of training images of 25 object-labeled images. To overcome this problem, we use U-Net architecture, an extended form of Fully Connected Network FCN. The main idea of this approach is to use a CNN as a powerful feature extractor while replacing the fully connected layers with convolution ones to output spatial maps instead of classification scores. Those maps are upsampled to produce dense per-pixel output. This method allows training CNN in the end-to-end manner for segmentation with input images of arbitrary sizes [1].

U-net is a neural network architecture designed primarily for image segmentation to have good localization and use context simultaneously [3]. The basic structure of a U-net architecture consists of two paths. The first path is the contracting path, also known as the encoder or the analysis path, which is similar to a regular convolution network and provides classification information. The second is an expansion path, also known as the decoder or the synthesis path, consisting of up-convolutions and concatenations with features from the contracting path. This expansion allows the network to learn localized classification information [4].

2. Methodology

2.1. Preprocessing Data

2.1.1. Exploring data

We have two types of data. Satellite imagery files are saved in GeoTiff format, and ground-true labeled segments are saved in WKT (well-known text) format. Satellite images are divided into four categories: RGB, Panchromatic, Multispectral 8-bands, and SWIR 8-bands (see Table 1). All images are captured from 18 locations from the same region of the earth. Each location contains 25 pictures (5x5 images) and are indexed by their relative locations. For example, image id 6010-1-3 is an image in the second row and fourth column of location id 6010 (see Figure 1). In total, we have 450 images per type (the total number of tiff files is 450*4=1800). Twenty-five of these 450 images are segmented and labeled in 10 classes. We use the open-source libraries *tifffile* and *pandas* to read and manipulate GeoTiff files. Also, we are using the *seaborn* library to visualize our data.

We have an issue with the distribution of classes among images. Figure 3 shows that target classes are heavily imbalanced within each set of images, e.g., %59.61 for crops against %0.008 for trucks. One solution to this problem is classifying each class separately instead of using one multi-classifier [1].

Table 1: Satellite imagery specifications

Image type	Size	Resolution (m)	Spectral range (nm)	Dynamic Range (bit/pixel)	W:H ratio
RGB	3 x 3350 x 3338			8	1:1.0036
Panchromatic	1 x 3349 x 3338	0.31	450-800	11	1:1.0033
Multispectral	8 x 838 x 835	1.24	400-1040	11	1:1.0036
SWIR	8 x 134 x 134	7.5	1195-2365	14	1:1

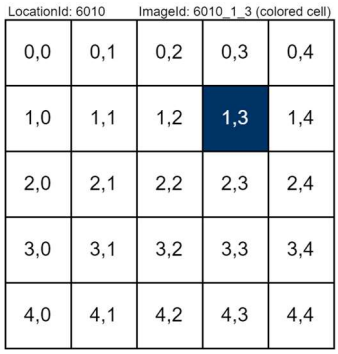

Figure 1:Addressing images based on ImageId and ImageLocationId

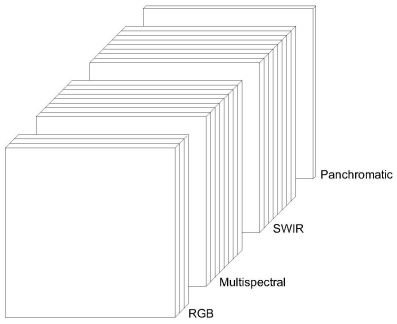

Figure 2: All layers of data for each ImageId

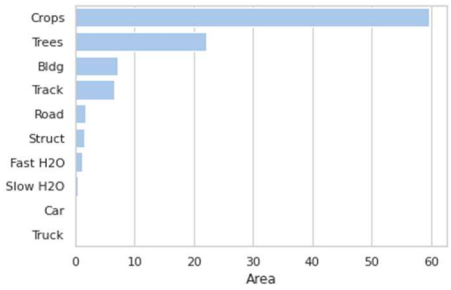

Figure 3: Distribution of classes as the percentage of the area for each class

2.1.2. Data Alignment

We need to extract data from GeoTiff files based on locations addressed by the shapefiles in wkt format. To do that, we use the *shapely* library. This library contains a wkt class to read and write shapefiles in several formats like text, JSON, and binary.

We implemented DataUtil and DataImage classes to make everything reusable and easy to debug. DataUtil class is responsible for reading data from files and creating data frames. Dataimage class is responsible for creating, resizing, aligning all images, and converting them into one tensor as network inputs.

The alignment is challenging because we have three different resolutions. To solve this issue, we use *resize* method in the *cv2* library to increase the size of all images to the largest resolution, which is RGB files (3350 x 3338). Also, we used the pansharpening technique to improve the resolution of the multiband images.

We will crop images into 112 x 112 baches to train and test the model.

Also, we have a util package to handle loss functions and metrics that will use inside the model.

2.2. Model Architecture

2.2.1. Network design

We decided to use a U-Net model in this project, which is common in problems with small training datasets like ours. We have implemented a class UnetModel using TensorFlow and Keras libraries with all needed function signatures. The primary architecture is shown in Figure 4 and Table Table 2. We will use Relu as our activation function and Adam optimizer with learning rates1e-3 and1e-4. All conv_layers are 3x3 with the same size padding. All max-pool layers are 2x2. We also used batch-normalization and drop-out(25%) to avoid overfitting.

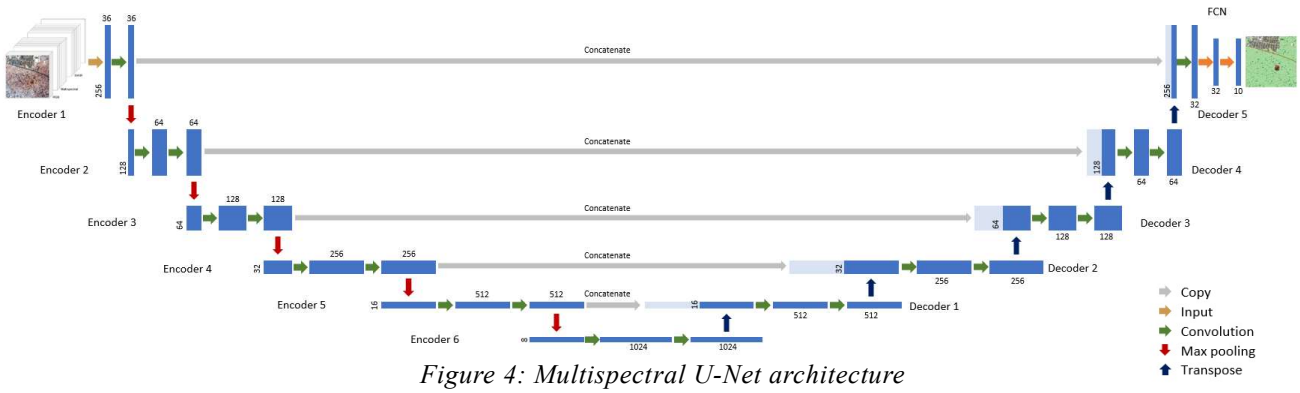

Figure 4: Multispectral U-Net architecture

Table 2: Network design summary.

Layer (type)	Output Shape	Param #	Connected to
input_2 (InputLayer)	(None, 256, 256, 36)	0	[]
conv2d_23 (Conv2D)	(None, 256, 256, 32)	10400	['input_2[0][0]']
conv2d_24 (Conv2D)	(None, 256, 256, 32)	9248	['conv2d_23[0][0]']
max_pooling2d_5 (MaxPooling2D)	(None, 128, 128, 32)	0	['conv2d_24[0][0]']
batch_normalization_8 (BatchNormalization)	(None, 128, 128, 32)	128	['max_pooling2d_5[0][0]']
conv2d_25 (Conv2D)	(None, 128, 128, 64)	18496	['batch_normalization_8[0][0]']
conv2d_26 (Conv2D)	(None, 128, 128, 64)	36928	['conv2d_25[0][0]']
max_pooling2d_6 (MaxPooling2D)	(None, 64, 64, 64)	0	['conv2d_26[0][0]']
dropout_8 (Dropout)	(None, 64, 64, 64)	0	['max_pooling2d_6[0][0]']
batch_normalization_9 (BatchNormalization)	(None, 64, 64, 64)	256	['dropout_8[0][0]']
conv2d_27 (Conv2D)	(None, 64, 64, 128)	73856	['batch_normalization_9[0][0]']
conv2d_28 (Conv2D)	(None, 64, 64, 128)	147584	['conv2d_27[0][0]']
max_pooling2d_7 (MaxPooling2D)	(None, 32, 32, 128)	0	['conv2d_28[0][0]']
dropout_9 (Dropout)	(None, 32, 32, 128)	0	['max_pooling2d_7[0][0]']
batch_normalization_10 (BatchNormalization)	(None, 32, 32, 128)	512	['dropout_9[0][0]']
conv2d_29 (Conv2D)	(None, 32, 32, 256)	295168	['batch_normalization_10[0][0]']
conv2d_30 (Conv2D)	(None, 32, 32, 256)	590080	['conv2d_29[0][0]']
max_pooling2d_8 (MaxPooling2D)	(None, 16, 16, 256)	0	['conv2d_30[0][0]']
dropout_10 (Dropout)	(None, 16, 16, 256)	0	['max_pooling2d_8[0][0]']
batch_normalization_11 (BatchNormalization)	(None, 16, 16, 256)	1024	['dropout_10[0][0]']
conv2d_31 (Conv2D)	(None, 16, 16, 512)	1180160	['batch_normalization_11[0][0]']
conv2d_32 (Conv2D)	(None, 16, 16, 512)	2359808	['conv2d_31[0][0]']
max_pooling2d_9 (MaxPooling2D)	(None, 8, 8, 512)	0	['conv2d_32[0][0]']
dropout_11 (Dropout)	(None, 8, 8, 512)	0	['max_pooling2d_9[0][0]']
conv2d_33 (Conv2D)	(None, 8, 8, 1024)	4719616	['dropout_11[0][0]']
conv2d_34 (Conv2D)	(None, 8, 8, 1024)	9438208	['conv2d_33[0][0]']
conv2d_transpose_5 (Conv2DTranspose)	(None, 16, 16, 512)	2097664	['conv2d_34[0][0]']
concatenate_5 (Concatenate)	(None, 16, 16, 1024)	0	['conv2d_transpose_5[0][0]', 'conv2d_32[0][0]']
batch_normalization_12 (BatchNormalization)	(None, 16, 16, 1024)	4096	['concatenate_5[0][0]']
conv2d_35 (Conv2D)	(None, 16, 16, 512)	4719104	['batch_normalization_12[0][0]']
conv2d_36 (Conv2D)	(None, 16, 16, 512)	2359808	['conv2d_35[0][0]']
dropout_12 (Dropout)	(None, 16, 16, 512)	0	['conv2d_36[0][0]']
conv2d_transpose_6 (Conv2DTranspose)	(None, 32, 32, 256)	524544	['dropout_12[0][0]']
concatenate_6 (Concatenate)	(None, 32, 32, 512)	0	['conv2d_transpose_6[0][0]', 'conv2d_30[0][0]']
batch_normalization_13 (BatchNormalization)	(None, 32, 32, 512)	2048	['concatenate_6[0][0]']
conv2d_37 (Conv2D)	(None, 32, 32, 256)	1179904	['batch_normalization_13[0][0]']
conv2d_38 (Conv2D)	(None, 32, 32, 256)	590080	['conv2d_37[0][0]']
dropout_13 (Dropout)	(None, 32, 32, 256)	0	['conv2d_38[0][0]']
conv2d_transpose_7 (Conv2DTranspose)	(None, 64, 64, 128)	131200	['dropout_13[0][0]']
concatenate_7 (Concatenate)	(None, 64, 64, 256)	0	['conv2d_transpose_7[0][0]', 'conv2d_28[0][0]']
batch_normalization_14 (BatchNormalization)	(None, 64, 64, 256)	1024	['concatenate_7[0][0]']
conv2d_39 (Conv2D)	(None, 64, 64, 128)	295040	['batch_normalization_14[0][0]']
conv2d_40 (Conv2D)	(None, 64, 64, 128)	147584	['conv2d_39[0][0]']
dropout_14 (Dropout)	(None, 64, 64, 128)	0	['conv2d_40[0][0]']
conv2d_transpose_8 (Conv2DTranspose)	(None, 128, 128, 64	32832	['dropout_14[0][0]']
concatenate_8 (Concatenate)	(None, 128, 128, 128)	0	['conv2d_transpose_8[0][0]', 'conv2d_26[0][0]']
batch_normalization_15 (BatchNormalization)	(None, 128, 128, 128)	512	['concatenate_8[0][0]']
conv2d_41 (Conv2D)	(None, 128, 128, 64)	73792	['batch_normalization_15[0][0]']
conv2d_42 (Conv2D)	(None, 128, 128, 64)	36928	['conv2d_41[0][0]']
dropout_15 (Dropout)	(None, 128, 128, 64)	0	['conv2d_42[0][0]']
conv2d_transpose_9 (Conv2DTranspose)	(None, 256, 256, 32)	8224	['dropout_15[0][0]']
concatenate_9 (Concatenate)	(None, 256, 256, 64)	0	['conv2d_transpose_9[0][0]', 'conv2d_24[0][0]']
conv2d_43 (Conv2D)	(None, 256, 256, 32)	18464	['concatenate_9[0][0]']
FCN (softmax)	(None, 256, 256, 32)	9248	['conv2d_43[0][0]']
FCN (sigmoid)	(None, 256, 256, 10)	330	['FCN[0][0]']
Total params: 31,113,898 Trainable params: **31,109,098** Non-trainable params: 4,800			

2.2.2. Metrics and loss functions

Since the shapefiles are not mutually exclusive (due to overlapping between multiple classes), using categorical cross-entropy does not work in this case. Instead, we will use binary cross-entropy (BCE-Dice Loss) as our classification loss function (equation 1). Since we have unbalanced distribution in our labeled dataset, we decided to train the model individually for each class as a model improvement. Thus, we have two types of trained models; one model with all ten classes and ten models with one binary classifier.

$$Loss(w) = \frac{1}{N}\sum_{n=1}^{N} H(p_n, q_n) = -\frac{1}{N}\sum_{n=1}^{N}[y_n log\hat{y}_n + (1 - y_n)log(1 - \hat{y}_n)] \qquad \text{Equation 1}$$

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} = \frac{|A \cap B|}{|A| + |B| - |A \cap B|} \ , \ 0 \leq J(A,B) \leq 1 \qquad \text{Equation 2}$$

We need to use the Jaccard index [5] as mentioned on the Kaggle website [6] as our evaluation metric (equation 2). We implemented both of them inside the utils package:

```
jaccard_distance(y_true, y_pred, smooth=100)
    intersection = K.sum(K.abs(y_true * y_pred), axis=-1)
    sum_ = K.sum(K.abs(y_true) + K.abs(y_pred), axis=-1)
    jac = (intersection + smooth) / (sum_ - intersection + smooth)
    return (1 - jac) * smooth

dice_coefficient(y_true, y_pred)
    intersection = tf.reduce_sum(tf.multiply(y_true, y_pred))
    union = tf.reduce_sum(tf.square(y_true)) +
tf.reduce_sum(tf.square(y_pred))
    loss = 1. - 2 * intersection / (union +
tf.constant(tf.keras.backend.epsilon()))
    return loss
```

3. Results

3.1. One model for all classes

In this scenario, we trained the model with all classes at once. The batch size was 64, and the input size was 256x256 pixels.

Figure 5 shows the training and validation loss during the training. The maximum epoch number was 50. Figure 6 shows the prediction results of one random tile of imageId 6100_1_3. It works pretty well only for buildings and trees. Overall IOU accuracy is 0.07, which is very low.

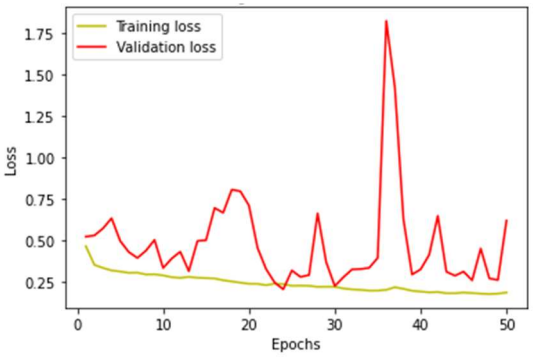

Figure 5: Training and validation loss diagram

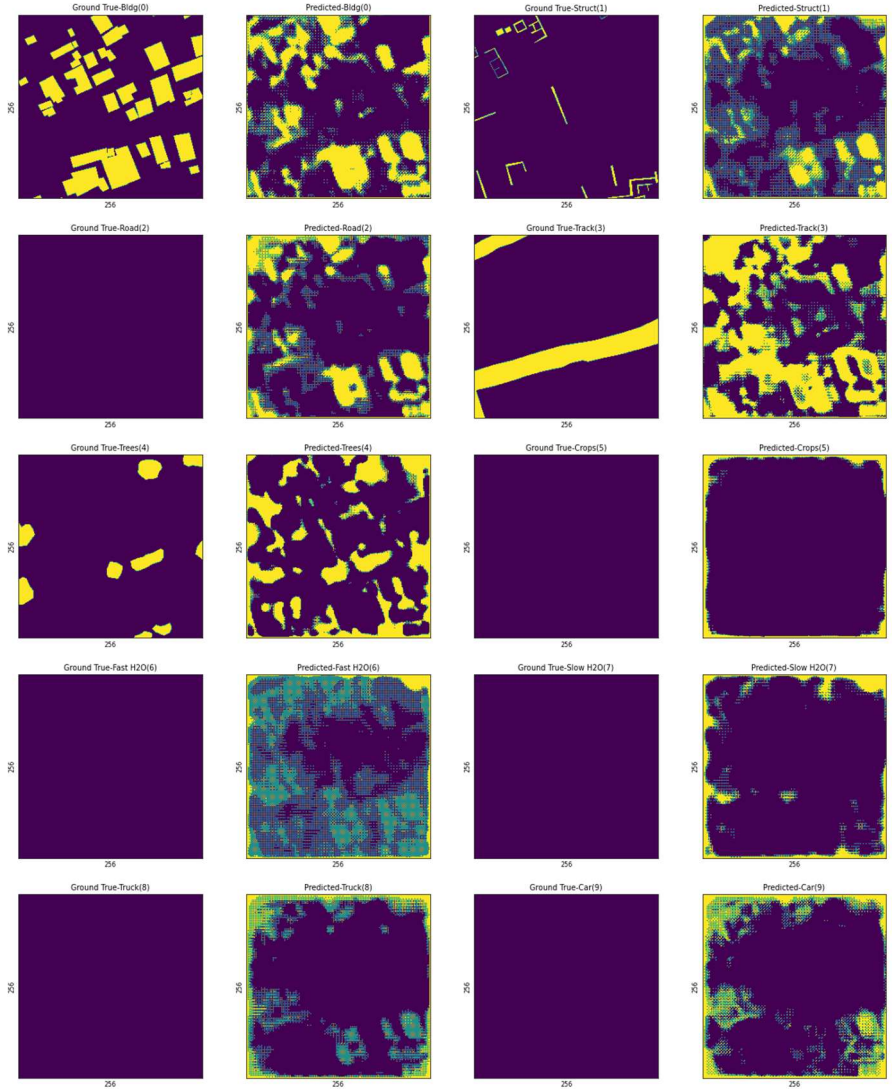

Figure 6: Results of one model for ten classes

3.2. Ten binary models were trained individually for each class

To solve the problem of the imbalanced labeled dataset, we tried to train the model for each class individually. Figure 6 shows a slight improvement in segmentation. But the overall IOU accuracy is still as low as 0.15.

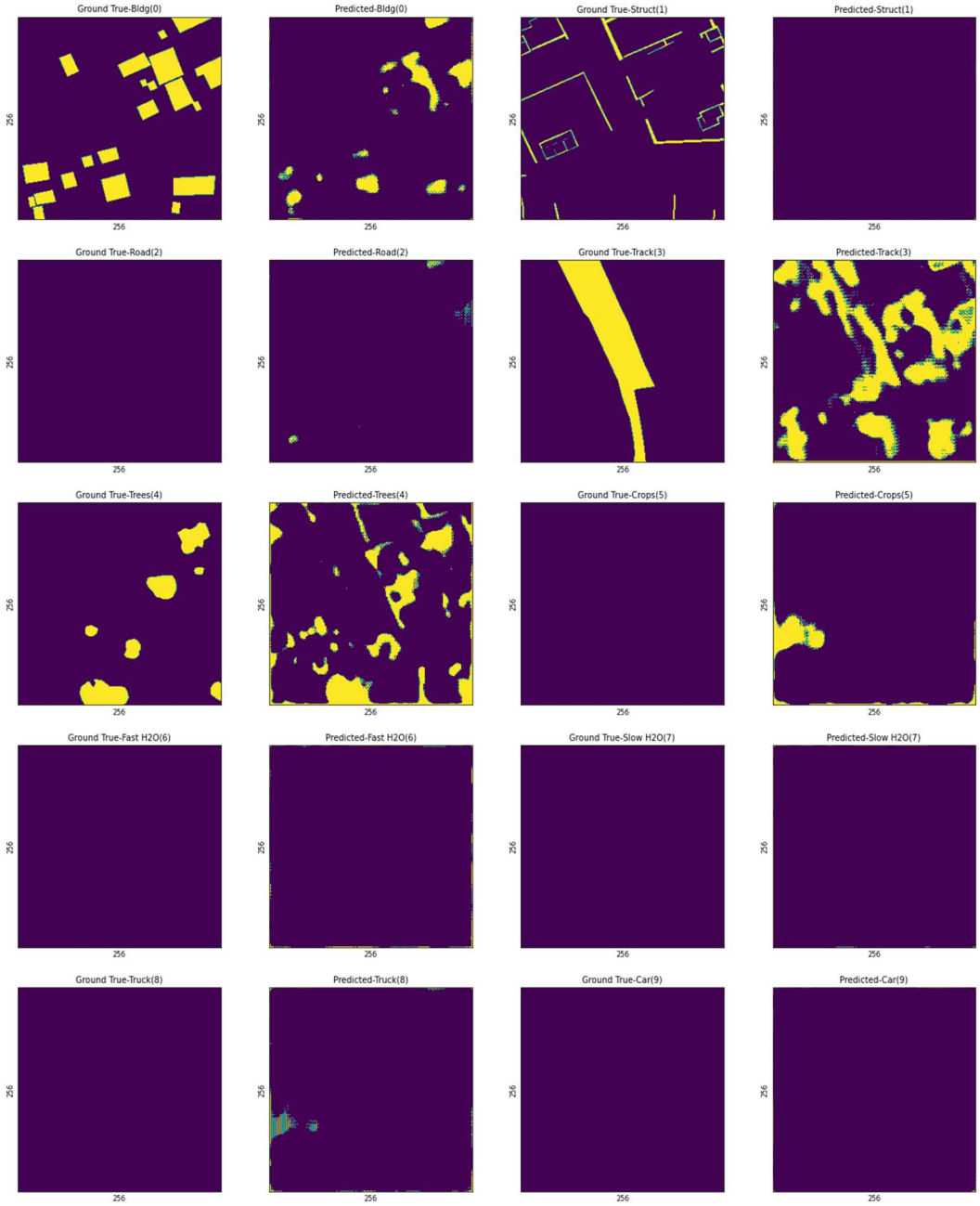

Figure 7: Results of one model for individual class

4. Conclusion

We implemented a U-net fully convolutional network to segment the input satellite imageries into classified classes in this project. At first, we trained all ten classes simultaneously with an IOU accuracy of 0.07. We found that the main reasons for this error are the small size of the dataset and the heavily imbalanced distribution of classes. To overcome this problem, we trained ten models for each class individually and got more accurate results of 0.15, which is still considered low. The accuracy of the model is varied among classes. The highest values belong to the building, trees, and corps classes and the lowest values are for cars and trucks.

There are two other possible improvements; Data augmentation and stratified sampling, which we are trying to implement in future works.

5. Acknowledgment

We genuinely wish to thank Dr. Ji for his guidance and special thanks to the DICE lab for their support.

6. References

[1] V. Iglovikov, S. Mushinskiy, and V. Osin, "Satellite Imagery Feature Detection using Deep Convolutional Neural Network: A Kaggle Competition," 6 2017.

[2] E. Shelhamer, J. Long, and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 39, no. 4, pp. 640-651, 4 2017.

[3] O. Ronneberger, P. Fischer and T. Brox, "U-Net: Convolutional Networks for Biomedical Image Segmentation," 5 2015.

[4] N. Siddique, S. Paheding, C. P. Elkin and V. Devabhaktuni, "U-net and its variants for medical image segmentation: A review of theory and applications," *IEEE Access,* 2021.

[5] "Wikkipedia," [Online]. Available: https://en.wikipedia.org/wiki/Jaccard_index. [Accessed 22 03 2022].

[6] "Kaggle," [Online]. Available: https://www.kaggle.com/c/dstl-satellite-imagery-feature-detection. [Accessed 22 03 2022].