Image Segmentation for Road Detection

Vimal Gaur
Computer Science and engineering
MSIT
New Delhi, India
vimalgaur@msit.in

Abhishek Mishra
Computer Science and engineering
MSIT
New Delhi, India
Abhishekmishra@msit.in

Abhay Chauhan
Computer Science and engineering
MSIT
New Delhi, India
Abhay.498@gmail.com

Akshit Singh
Computer Science and engineering
MSIT
New Delhi, India
Akshitsingh2305@gmail.com

Abhinav Raghav
Computer Science and Engineering
MSIT
New Delhi, India
abhinavr6293@gmail.com

***Abstract*—Deep learning as technology is offering endless opportunities to develop new types of digital services. Pertaining to such technology detecting buildings, roads and terrain from satellite images offer a lot of potentials, from tracking the migration of large chunk of the population to helping in city planning for the administration of the country. We have created a Convolutional neural network for extracting road maps and the structure of cities from satellite imaging with the help of U-Net. With the constantly growing world, population structured settlement becomes a need of the hour and with an increase in the number of natural calamities per year, rescue operations need to be streamlined. The purpose of this paper would be to develop a model with reasonable accuracy, which might not be impeccable but provide ideas for further improvement.**

I. INTRODUCTION

Drawing out most recent layout of the populous areas in the city or a remote area can be laborious task if done outdated methods. These parts exist in metropolitan cities and are rapidly changing by the movement of humans or even in remote areas where attention of the authorities is minimum. During a natural disaster, it can be a promising tool in locating the damaged building of the area and can be helpful in finding routes to provide aide as early as possible. As in 2020 outbreak of COVID - 19 tested the readiness of the government, mapping out of hotspots and distribution of test kits could have been fairly easy task by the likes of this project. Especially in a country like India, where people travel great distances from their homeland in search of work during a pandemic like this needed to be moved amidst the lockdown.

This project addresses the issue of segmentation of images obtained by satellite imaging. It dissects the image pixel by pixel and classifying each pixel as part of road.

II. RELATED WORK

Extraction objects namely buildings, lakes, ships etcetera from satellite images is not a pretty recent field of investigation. There has been tremendous work done on this in the past few years. We can cite [1] and [2]. The first paper uses the SVM algorithm to extract buildings from high-resolution images provided to it and the second one uses a different approach called a full convolution network which is not that old method and is very prevalent in today's image segmentation projects.

Extrapolation and building more on FCN we get a specific type of neural network which is called U-Net, which has proven its worth in bio-medical research and works even on a small number of datasets. When it was used in satellite images for segmentation it received lots of appreciation. We have mainly focused on [3] for our model and development of this project.

III. DATASET

A. *Data Selection*

The dataset was collected from [4]. This dataset was the collection of 1113 satellite images along with their masks. Images in this dataset were of high resolution and their masks were black and white, where white pixel represents road and black pixel represents the rest of the area.

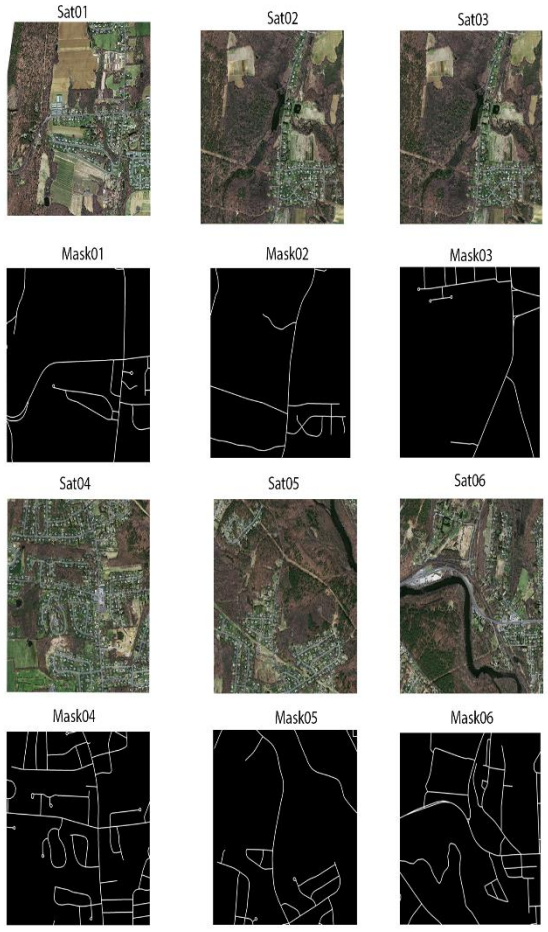

Fig 1: Images before pre-processing.

B. *PREPROCESSING*

The problem with this dataset was that its images were not complete so after manually removing images dataset was reduced to just 979 images. Another big issue was the size of its images(1500X1500), which was not suitable for our neural network. So, we used a script to crop images into smaller images of size 256X256 which not only reduced the size of images but also increased out training dataset size to 35244 which was later reduced to 27588 after rejection of some more images. After pre-processing, the dataset was suitable for training our model.

Fig 2: Images after pre-processing.

IV. METHOD

The method we have used is very simple and yields a pretty good result, but before discussing method let's see what is CNN, U-NET, TensorFlow, and Keras.

CNN stands for convolution neural network. CNN automatically extracts useful features from an image by performing several pooling and convolution operations on the image which is fed to the model as input.

U-NET is a special case for CNN in which convolution and deconvolution is used simultaneously. It is a very powerful and precise model for image segmentation tasks and works on even a limited number of inputs.

TensorFlow is open-source software and It interacts with GPU to maintain dataflow and differential programming for range of tasks. TensorFlow is generally used for math library and machine learning applications such as developing neural networks.

Keras is also an open-source software which is written in python. It works on top of TensorFlow. It comes in handy in experimentation with deep neural network which can be a tedious task on TensorFlow as It contains different libraries and functions which works as building blocks for neural network.

A. *ARCITECTURE OF THE MODEL*

Instead of using someone else developed model we decided to develop our own model taking inspiration from model originally developed by [3].
U-Net is basically a pair of down-sampling and up-sampling. During contraction path features of down-sampling activation function and max-pooling extracts features from images and symmetric expanding path up-samples the result from the previous step and increases the resolution of the extracted feature. Along with expanding and contracting path skip connections are used which provides context and localization. The U-Net figure is displayed below.

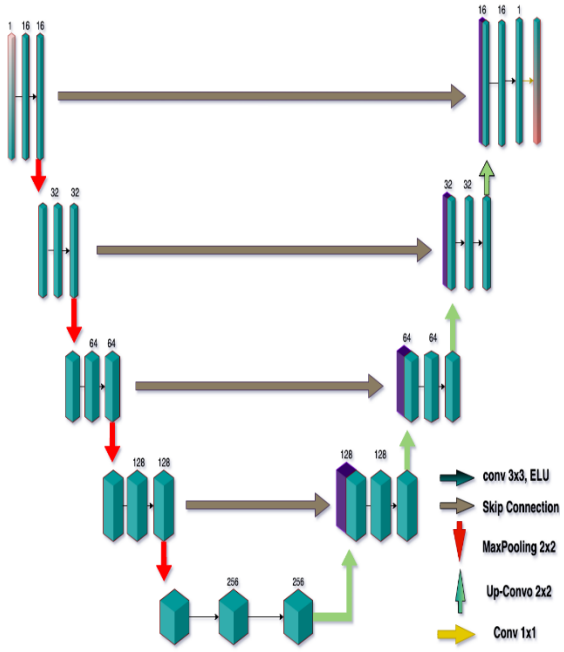

Fig 3: U-Net Architecture.

B. *MODEL SUMMARY*

The above architecture was implemented using keras and below is summary of that model.

Layer (type)	Output Shape	Param #	Connected to
input_1 (InputLayer)	(None, 256, 256, 3)	0	
conv2d_1 (Conv2D)	(None, 256, 256, 16)	448	input_1[0][0]
batch_normalization_1 (BatchNor	(None, 256, 256, 16)	64	conv2d_1[0][0]
dropout_1 (Dropout)	(None, 256, 256, 16)	0	batch_normalization_1[0][0]
conv2d_2 (Conv2D)	(None, 256, 256, 16)	2320	dropout_1[0][0]
batch_normalization_2 (BatchNor	(None, 256, 256, 16)	64	conv2d_2[0][0]
max_pooling2d_1 (MaxPooling2D)	(None, 128, 128, 16)	0	batch_normalization_2[0][0]
conv2d_3 (Conv2D)	(None, 128, 128, 32)	4640	max_pooling2d_1[0][0]
batch_normalization_3 (BatchNor	(None, 128, 128, 32)	128	conv2d_3[0][0]
dropout_2 (Dropout)	(None, 128, 128, 32)	0	batch_normalization_3[0][0]
conv2d_4 (Conv2D)	(None, 128, 128, 32)	9248	dropout_2[0][0]
batch_normalization_4 (BatchNor	(None, 128, 128, 32)	128	conv2d_4[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 64, 64, 32)	0	batch_normalization_4[0][0]
conv2d_5 (Conv2D)	(None, 64, 64, 64)	18496	max_pooling2d_2[0][0]
batch_normalization_5 (BatchNor	(None, 64, 64, 64)	256	conv2d_5[0][0]
dropout_3 (Dropout)	(None, 64, 64, 64)	0	batch_normalization_5[0][0]
conv2d_6 (Conv2D)	(None, 64, 64, 64)	36928	dropout_3[0][0]
batch_normalization_6 (BatchNor	(None, 64, 64, 64)	256	conv2d_6[0][0]
max_pooling2d_3 (MaxPooling2D)	(None, 32, 32, 64)	0	batch_normalization_6[0][0]
conv2d_7 (Conv2D)	(None, 32, 32, 128)	73856	max_pooling2d_3[0][0]
batch_normalization_7 (BatchNor	(None, 32, 32, 128)	512	conv2d_7[0][0]
dropout_4 (Dropout)	(None, 32, 32, 128)	0	batch_normalization_7[0][0]
conv2d_8 (Conv2D)	(None, 32, 32, 128)	147584	dropout_4[0][0]
batch_normalization_8 (BatchNor	(None, 32, 32, 128)	512	conv2d_8[0][0]
max_pooling2d_4 (MaxPooling2D)	(None, 16, 16, 128)	0	batch_normalization_8[0][0]
conv2d_9 (Conv2D)	(None, 16, 16, 256)	295168	max_pooling2d_4[0][0]
batch_normalization_9 (BatchNor	(None, 16, 16, 256)	1024	conv2d_9[0][0]
dropout_5 (Dropout)	(None, 16, 16, 256)	0	batch_normalization_9[0][0]
conv2d_10 (Conv2D)	(None, 16, 16, 256)	590080	dropout_5[0][0]
batch_normalization_10 (BatchNo	(None, 16, 16, 256)	1024	conv2d_10[0][0]
conv2d_transpose_1 (Conv2DTrans	(None, 32, 32, 128)	131200	batch_normalization_10[0][0]
concatenate_1 (Concatenate)	(None, 32, 32, 256)	0	conv2d_transpose_1[0][0] batch_normalization_8[0][0]
conv2d_11 (Conv2D)	(None, 32, 32, 128)	295040	concatenate_1[0][0]
batch_normalization_11 (BatchNo	(None, 32, 32, 128)	512	conv2d_11[0][0]
dropout_6 (Dropout)	(None, 32, 32, 128)	0	batch_normalization_11[0][0]
conv2d_12 (Conv2D)	(None, 32, 32, 128)	147584	dropout_6[0][0]
batch_normalization_12 (BatchNo	(None, 32, 32, 128)	512	conv2d_12[0][0]
conv2d_transpose_2 (Conv2DTrans	(None, 64, 64, 64)	32832	batch_normalization_12[0][0]
concatenate_2 (Concatenate)	(None, 64, 64, 128)	0	conv2d_transpose_2[0][0] batch_normalization_6[0][0]
conv2d_13 (Conv2D)	(None, 64, 64, 64)	73792	concatenate_2[0][0]
batch_normalization_13 (BatchNo	(None, 64, 64, 64)	256	conv2d_13[0][0]
dropout_7 (Dropout)	(None, 64, 64, 64)	0	batch_normalization_13[0][0]
conv2d_14 (Conv2D)	(None, 64, 64, 64)	36928	dropout_7[0][0]
batch_normalization_14 (BatchNo	(None, 64, 64, 64)	256	conv2d_14[0][0]
conv2d_transpose_3 (Conv2DTrans	(None, 128, 128, 32)	8224	batch_normalization_14[0][0]
concatenate_3 (Concatenate)	(None, 128, 128, 64)	0	conv2d_transpose_3[0][0] batch_normalization_4[0][0]
conv2d_15 (Conv2D)	(None, 128, 128, 32)	18464	concatenate_3[0][0]
batch_normalization_15 (BatchNo	(None, 128, 128, 32)	128	conv2d_15[0][0]
dropout_8 (Dropout)	(None, 128, 128, 32)	0	batch_normalization_15[0][0]
conv2d_16 (Conv2D)	(None, 128, 128, 32)	9248	dropout_8[0][0]
batch_normalization_16 (BatchNo	(None, 128, 128, 32)	128	conv2d_16[0][0]
conv2d_transpose_4 (Conv2DTrans	(None, 256, 256, 16)	2064	batch_normalization_16[0][0]
concatenate_4 (Concatenate)	(None, 256, 256, 32)	0	conv2d_transpose_4[0][0] batch_normalization_2[0][0]
conv2d_17 (Conv2D)	(None, 256, 256, 16)	4624	concatenate_4[0][0]
batch_normalization_17 (BatchNo	(None, 256, 256, 16)	64	conv2d_17[0][0]
dropout_9 (Dropout)	(None, 256, 256, 16)	0	batch_normalization_17[0][0]
conv2d_18 (Conv2D)	(None, 256, 256, 16)	2320	dropout_9[0][0]
batch_normalization_18 (BatchNo	(None, 256, 256, 16)	64	conv2d_18[0][0]
conv2d_19 (Conv2D)	(None, 256, 256, 1)	17	batch_normalization_18[0][0]

Total params: 1,946,993
Trainable params: 1,944,049
Non-trainable params: 2,944

Fig 4: Model Summary

We have used accuracy and loss metrics from [5].

```
#accuracy Metric
from keras import backend as K
def iou_coef(y_true, y_pred, smooth=1):
    intersection = K.sum(K.abs(y_true * y_pred), axis=[1,2,3])
    union = K.sum(y_true,[1,2,3])+K.sum(y_pred,[1,2,3])-intersecti
    iou = K.mean((intersection + smooth) / (union + smooth), axis=

    return iou
```

Fig 5: Accuracy Metric

```
#Loss function
def dice_coef(y_true, y_pred, smooth = 1):
    y_true_f = K.flatten(y_true)
    y_pred_f = K.flatten(y_pred)
    intersection = K.sum(y_true_f * y_pred_f)
    return (2. * intersection + smooth) / (K.sum(y_true_f) + K.sum(y_pred_f) + smo

def soft_dice_loss(y_true, y_pred):
    return 1-dice_coef(y_true, y_pred)
```

Fig 6: Loss Function

C. *SPLITTING AND TRAINING*

After pre-processing the dataset and defining our model but before training our model we need to split our data in the train and test sample. We have split our data training (80%) images and test (20%) data. We have also defined training epochs, learning rate, and batch size.

```
TRAIN SET
(22070, 256, 256, 3)
(22070, 256, 256, 1)
TEST SET
(5518, 256, 256, 3)
(5518, 256, 256, 1)
```

Fig 7: Train and Test set

```
EPOCHS = 100
LEARNING_RATE = 0.0001
BATCH_SIZE = 16
```

Fig 8: Hyperparameters

During the training of our model, training stopped after 48th epochs as the loss value was not improving leading to early stopping, which was absolutely fine.

V. RESULT

The training and early stopping of our model yielded loss of 24.462% and accuracy of 60.62%, this result was neither good nor bad and we accepted the accuracy of around 65%.

Following is the prediction obtained on a separate set of images that were not the part of train and test split and from predicted images it can be seen results are fairly good.

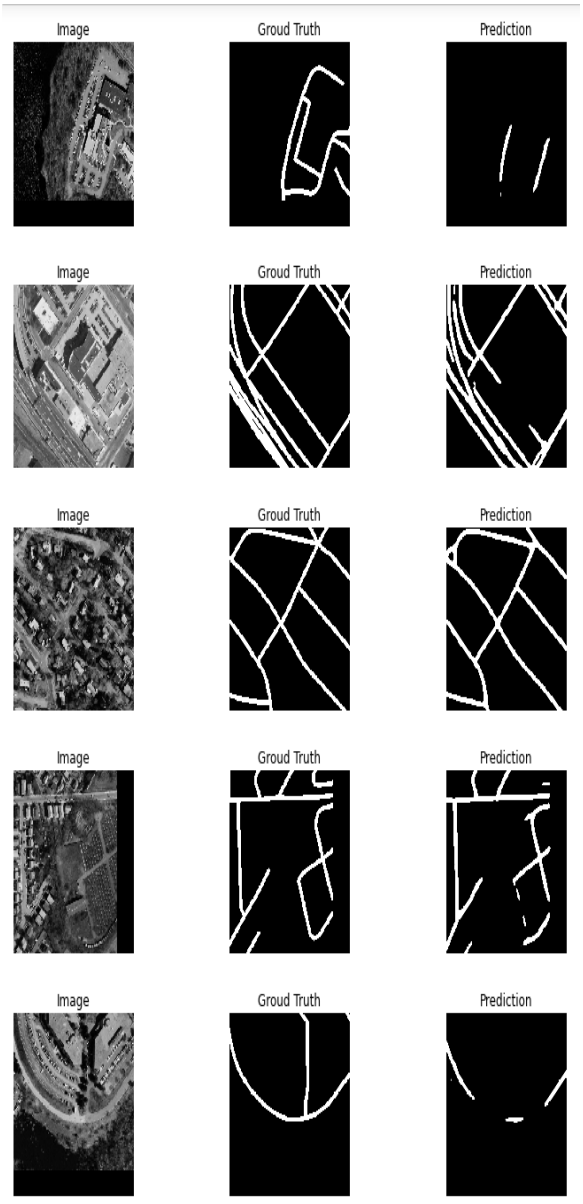

Fig 9: Predictions

From [6] we also found that the accuracy of prediction is highly dependent on the resolution of the images. Moreover, from [2] we found that accuracy can further be improved when the above network is paired with other pre-trained models like VGG16 encoder, SegNet etcetera.

VI. LIMITATIONS & FUTURE WORK

As we wandered through this project, again and again, we found out some limitations, and we thought it is important to address them. Firstly, this image segmentation task is highly dependent on the resolution of images provided to it. The accuracy of prediction is directly proportional to the resolution of the image which is further supported by [2].

Secondly, the amount of computational power required for training the model, although is it not that big of an issue because of increasing computational power but that power is limited to certain organizations and not available to all.

Learning from the past one thing we can do first is collecting a high-resolution dataset through which feature extraction becomes easy, as we have seen above resolution of images can make lot of difference in accuracy and consequently in predictions.

Secondly, we can focus on detecting road step by step, as some roads are wide and some roads are narrow. This classification has the potential to turn image segmentation for road detection up-side-down as in case of the wide road we will have freedom or flexibility to detect its boundary first, and detecting boundary is a fairly easy task as compared to the whole road.

VII. CONCLUSION

In the contemporary world where climate change is the most certain enemy of humankind and natural disasters are almost cyclic. We must turn to our faithful friend technology and its latest gift to us, machine learning for tackling this issue. In this project, With the help of a dataset containing 2-D dimensional images captured through satellite imaging trained our model after masking those images. We developed our CNN architecture with the help of U-NET.

With the help of Keras libraries we able to approach the problem of semantic segmentation of images or motion for that matter with efficiency. Keras working on top of TensorFlow creating a robust environment, maybe not a new-fangled model but with reasonable accuracy.
The lapses in scalability and accuracy still exist but it opens new doors to future development in resolving the problem by greater understanding.

REFERENCES

[1] O. Benarchid and N. Raissouni, "support vector machines for object based building extraction in suburban area using very high resolution satellite images.," *IAES International journal of AI,* vol. 3, pp. 43-50, 2013.

[2] B. Bischke, P. Helber, J. Folz and A. Damian Borth, "multi-task learning for segmentation of building footprints with deep neural network," in *IEEE*, 2015.

[3] O. Ronneberger, P. F. and T. Brox, "U-Net: Convolutional Networks for biomedical image segmentation," in *IEEE*, 2015.

[4] V. Mnih, "Machine Learning for Aerial Image Labeling," in *University of Toronto*, 2013.

[5] E. Tiu, "Metrics to Evaluate your Semantic Segmentation Model," 9 august 2019.

[6] Z. Yan, X. Han, C. Wang, Y. Qiu, Z. Xiong and S. Cu, "LEARNING MUTUALLY LOCAL-GLOBAL U-NETS FOR HIGH RESOLUTION RETINAL SEGMENTATION IN FUNDUS IMAGES," in *IEEE*, 2019.

[7] H. Monajemi, D. L. Donoho and V. Stodden, "making massive computational experiment painlessly," in *Proc. of IEEE Intl. Conf on Big Data*, 2016.

[8] Y. Chen, L. Cheng, M. Li, J. Wang, L. Tong and K. Yang, "Multiscale Grid Method for Detection and Reconstruction of rooftop usinf LiDAR Data," in *IEEE*, 2014.

[9] G. Chho, C. B. Aramburu and I. Bougdal-Lambert, "Satellite Image Segmentation for Building Detection using U-Net," in *IEEE*, 2014.

[10] J. Long, E. Shelhamer and T. Darrell, "Fully Convolutional Networks for sematic segmentation," in *IEEE*, 2015.