Semantic Segmentation of Remote Sensing Images Based on Deep Learning Techniques

Jianlong LI Draco
Data Science
BNU-HKBU
United International College
n830026061@mail.uic.edu.cn

Siyuan WANG Ryan
Data Science
BNU-HKBU
United International College
n830026113@mail.uic.edu.cn

Yuanhao LI Erwin
Data Science
BNU-HKBU
United International College
n830026070@mail.uic.edu.cn

Abstract

In recent years, the success of deep learning in natural scene image processing boosted its application in the analysis of remote sensing images. High resolution remote sensing image contains complex object information, but the applications of traditional segmentation methods are greatly limited. The segmentation method, represented by the deep convolution neural network, has made a breakthrough in many fields. Aiming at the problem of high-resolution remote sensing image segmentation, our project proposes a deep convolution neural network based on U-Net, which make them suitable for multi-targets semantic segmentation of remote sensing images. The results show that this model have its own advantages on the segmentation of different remote sensing objects.

Index Terms—Neural Network, Satellite Image Processing, Multi-class Classification, U-Net Models.

1 Introduction

Satellite images can provide as a spatial frame for modelling and understanding earth from above. With proper data processing, analytics and cutting-edge technologies like deep learning can generate great insights. The idea of semantic segmentation is to label each pixel with a corresponding class of what it is representing.

The four Orbita Hyper Spectral (OHS) satellites of "Zhuhai NO.1" are important members of the global family of 25 OHS satellites and the only commercial OHS satellites in China that have been launched and networked. The characteristic OHS data of "Zhuhai NO.1" is of world-class level and capable of carrying out accurate and quantitative analysis of vegetation, water body, ocean and other ground features. In our project we contrived to find the bare land, water, building and vegetation part in each satellite remote sensing image shotted by "Zhuhai NO.1" OHS satellite base on semantic segmentation.

2 Related Works

In these kinds of image semantic segmentation task, we can select a lot of classic networks, such as FCN, U-net, Segnet, RCNN, etc., which are very classic and widely used in many fields.
While going through different architectures, we figure out that most of the research studies end up in selecting U-Net architecture. (Cui et al., 2020) It is understood that U-Net architecture is well-designed for semantic segmentation. (Su et al., 2019)
U-net is a very elegant network architecture and has many advantages. The biggest selling point is that it can train a good model well based on a relatively small data set. This advantage is quite suitable for our task. Moreover, the training speed of U-net is also very fast.

3 Dataset and Features

3.1 Dataset Summary

The contest council provide 90 satellite remote sensing images with labels to train the model. The shape of each images is $32 \times 500 \times 500$, which means each images' width is 500, height is 500, dimension is 32. The normal image library of python like "opencv" or "matplotlib" can't read such high dimension data, so we select the library *gdal* to process these data. GDAL is a translator library for raster and vector geospatial data formats that is released under an X/MIT style Open-Source License by the Open-Source Geospatial Foundation.

3.2 Features Statements

For each pixels of images, there are a corresponding label. A data image has 250,000 pixels with 250000 labels. And these labels are stored in an image too, the shape of the label image is $1 \times 500 \times 500$. Label “1” means vegetation, “2” means building, “3” means bare land, “4” means water and “255” means others. In the *figure 1*, we consider the label as gray scale and plot it out, the gray scale of 1,2,3 and 4 are very close in the original picture, cannot be distinguished directly by human eyes.

Fig. 1: Raw label image

3.3 Image Cutting

To reduce the memory usage when training, we cut the data images to small sub images. We have four steps to cut image. In first step, we cut most of portion in data images top left corner. The *figure 2* shows the procedure to cut 150×150 sub-images from 500×500 data images. In our project, we cut 256×256 sub-images from the 500×500 data image, and the sub-images are shown in *figure 3*. After image cut, we got 360 256×256 sub-images. Also, after we predict, we will have an inverse procedure to merge the sub-images together.

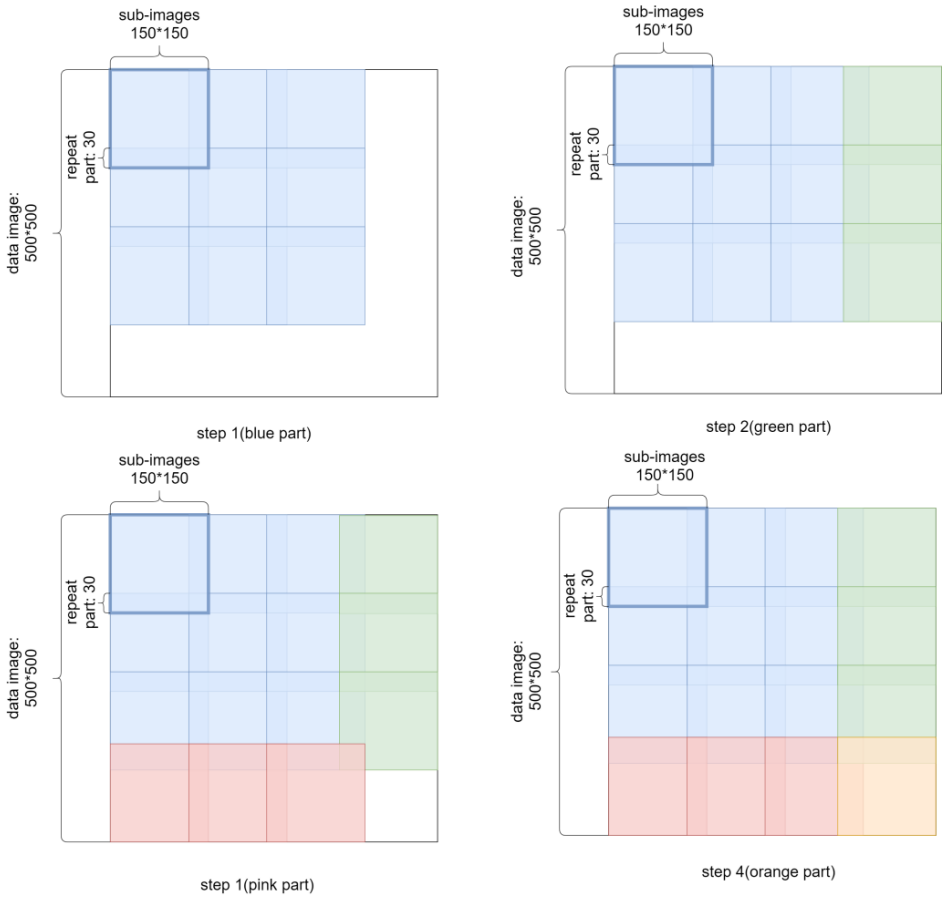

Fig. 2: Image cutting process

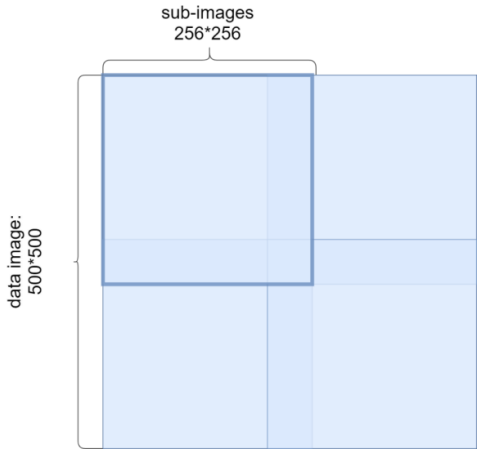

Fig. 3: Image cutting in our project

3.4 Data Enhancement

To have better performance in verge information and enhance accuracy, we transfer image by the three actions, Horizontal flip, Vertical flip, and Diagonal flip. And use the 360 sub-images, 360 horizontal flipped sub-images, 360 vertical flipped sub-images and 360 diagonal flipped sub-images as the training data to next step, totally we will have 1440 image data with shape 256×256×32, and we use 80% of them as training data, 20% of them as validate data.

4 Methods

We choose U-net as our training model.

U-Net is based on fully convolutional networks and its main idea is to add successive layers behind conventional convolution networks, which are designed for up-sampling. In order to locate more accurately, up-sampling combines features of upstream. U-net retains a large number of characteristic channels in the up-sampling part, which propagates context information to a higher resolution layer. So the whole structure looks like an “U”.

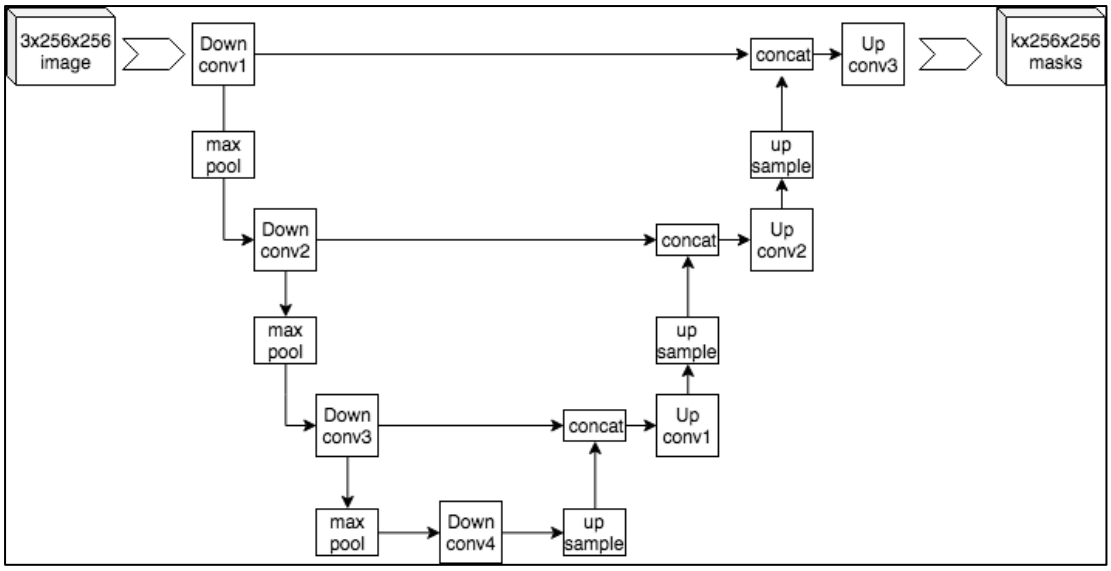

Fig. 4: U-net overview

4.1 Principles of the U-net

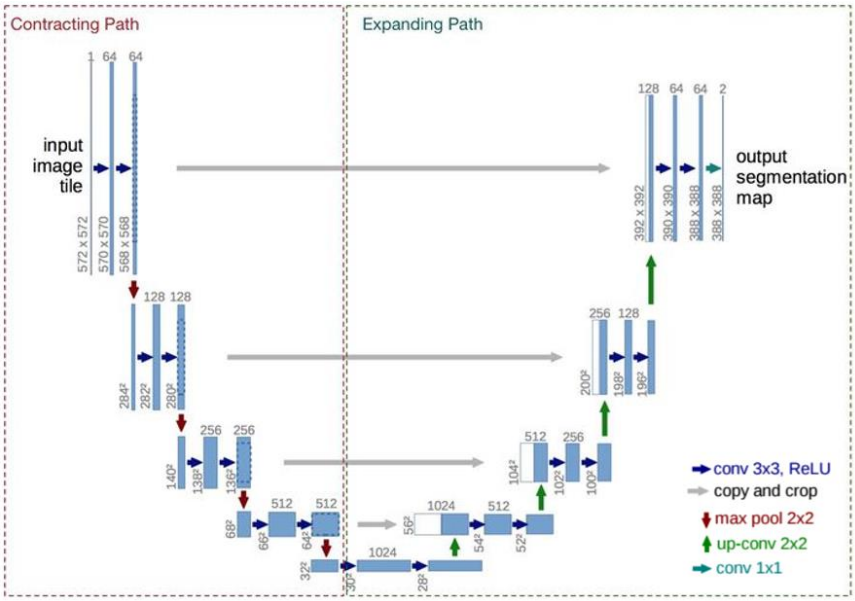

Fig. 5: U-net detailed structure

The network consists of a contracting path and an expanding path, which gives it the U-shaped architecture. The blue and white boxes represent the feature map; The blue arrow represents 3x3 convolution for feature extraction; The red arrow indicates max pooling, which is used to reduce dimensions; The green arrow indicates up-convolution, which is used to recover dimensions; In the end, the cyan arrow represents 1×1 convolution, which is used to output the result.

The contracting path is a typical convolutional network that consists of repeated application of convolutions, each followed by a rectified linear unit (ReLU) and a max pooling operation. During the contraction (the left part), the spatial information is reduced while feature information is increased. The expanding path on the right-hand side of figure 5 combines the feature and spatial information through a sequence of up-convolutions and concatenations with high-resolution features from the contracting path.

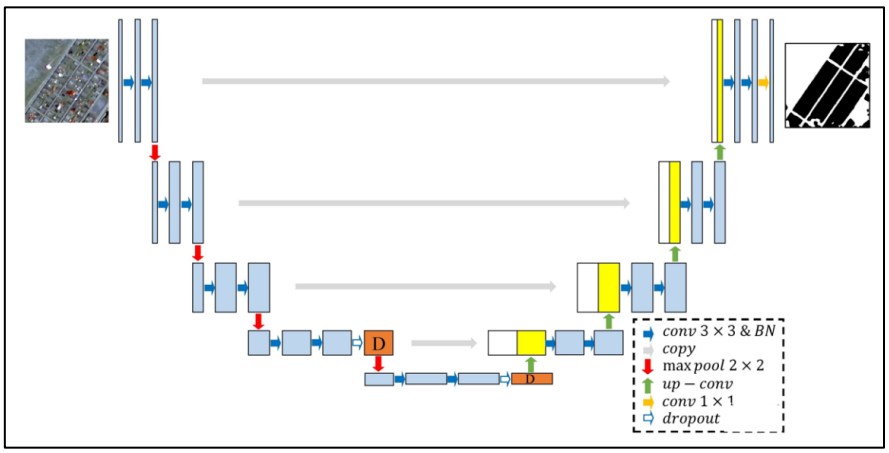

Fig. 6: U-net adding dropout layer

In our modified U-net, we add a dropout layer to do the regularization and avoid overfitting.

4.2 U-net Model Implementation

```
def unet(pretrained_weights = None, input_size = (256, 256, 4), classNum = 2, learning_rate = 1e-5):
    inputs = Input(input_size)
    #  2-Dimensional Convolution Layer
    conv1 = BatchNormalization()(Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(inputs))
    conv1 = BatchNormalization()(Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv1))
    #  Max Pooling to the data.
    pool1 = MaxPooling2D(pool_size=(2, 2))(conv1)
    conv2 = BatchNormalization()(Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(pool1))
    conv2 = BatchNormalization()(Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv2))
    pool2 = MaxPooling2D(pool_size=(2, 2))(conv2)
    conv3 = BatchNormalization()(Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(pool2))
    conv3 = BatchNormalization()(Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv3))
    pool3 = MaxPooling2D(pool_size=(2, 2))(conv3)
    conv4 = BatchNormalization()(Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(pool3))
    conv4 = BatchNormalization()(Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv4))
    #  Dropout regularization and avoid overfitting.
    drop4 = Dropout(0.5)(conv4)
    pool4 = MaxPooling2D(pool_size=(2, 2))(drop4)

    conv5 = BatchNormalization()(Conv2D(1024, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(pool4))
    conv5 = BatchNormalization()(Conv2D(1024, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv5))
    drop5 = Dropout(0.5)(conv5)
```

Fig. 7: Code for U-net (contracting path part)

Above code are mainly the contracting path (the left half part of the U-net), conducting the convolution and max pooling operations.

```
up6 = Conv2D(512, 2, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(UpSampling2D(size = (2,2))(drop5))
try:
    merge6 = concatenate([drop4,up6],axis = 3)
except:
    merge6 = merge([drop4,up6], mode = 'concat', concat_axis = 3)
conv6 = BatchNormalization()(Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(merge6))
conv6 = BatchNormalization()(Conv2D(512, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv6))

up7 = Conv2D(256, 2, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(UpSampling2D(size = (2,2))(conv6))
try:
    merge7 = concatenate([conv3,up7],axis = 3)
except:
    merge7 = merge([conv3,up7], mode = 'concat', concat_axis = 3)
conv7 = BatchNormalization()(Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(merge7))
conv7 = BatchNormalization()(Conv2D(256, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv7))

up8 = Conv2D(128, 2, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(UpSampling2D(size = (2,2))(conv7))
try:
    merge8 = concatenate([conv2,up8],axis = 3)
except:
    merge8 = merge([conv2,up8],mode = 'concat', concat_axis = 3)
conv8 = BatchNormalization()(Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(merge8))
conv8 = BatchNormalization()(Conv2D(128, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv8))

up9 = Conv2D(64, 2, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(UpSampling2D(size = (2,2))(conv8))
try:
    merge9 = concatenate([conv1,up9],axis = 3)
except:
    merge9 = merge([conv1,up9],mode = 'concat', concat_axis = 3)
conv9 = BatchNormalization()(Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(merge9))
conv9 = BatchNormalization()(Conv2D(64, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv9))
conv9 = Conv2D(2, 3, activation = 'relu', padding = 'same', kernel_initializer = 'he_normal')(conv9)
conv10 = Conv2D(classNum, 1, activation = 'softmax')(conv9)

model = Model(inputs = inputs, outputs = conv10)

#  Used for training the model(optimizer, target function and model assessment)
model.compile(optimizer = Adam(lr = learning_rate), loss = 'categorical_crossentropy', metrics = ['accuracy'])

#  if having the pre-training weights.
if(pretrained_weights):
    model.load_weights(pretrained_weights)

return model
```

Fig. 8: Code for U-net (expanding path part)

Above are the expanding path (the right half part of the U-net), conducting the convolution and up-convolution operations which is used to recover the dimensions.

5 Result Analysis

After we define the U-net, we can start training the model. But again, a hardware problem needs to be solved before start training.

When memory is limited, it is often impossible to read the huge training set into the memory at once. Therefore, we use a generator, which means that we only read and only read this batch of data before training on this batch of data.

The next step is to train the model. We divide 80% of the total images into the training set, and the remaining 20% into the validation set.

We have enhanced the image, resulting in a larger training set, so training process is very time-consuming. Therefore, during the training process, we constantly adjust the training parameters such as the learning rate and batch size to speed up the training. But overall, the learning rate is 1e-4 and the batch size is 8 and 50 epochs of training in total.

The following figure shows the training accuracy and verification accuracy of the last 10 epochs.

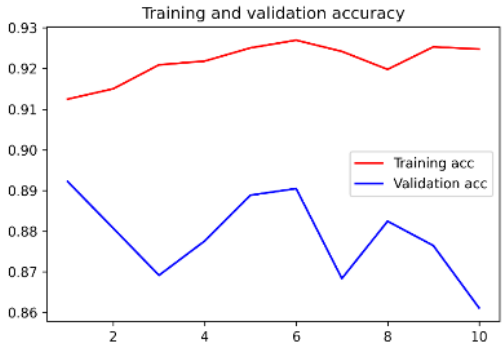

Fig. 9: Training and validation accuracy of the last ten epochs

It can be seen that the validation accuracy no longer increases with the improvement of the training accuracy. And the improvement of training accuracy has also become very difficult.

Since the test set of the data given by the competition has no labels, and we need as much data as possible for training to achieve high overall accuracy, the data for the result analysis comes from part of the validation set.

Since the input size of our model is the cropped image size, before predicting the image, the image must be cropped. After cropping, put all small datasets into the model for prediction. Then, merge the images after getting the results.

After we get the results, we need to analyze the results. A confusion matrix can be generated based on the result of the model prediction and the real label.

label\prediction	**Vegetation**	**Building**	**Bare land**	**Water**	**Other**
Vegetation	1179532	1407	0	1625	44805
Building	157	37725	0	0	19925
Bare land	123	10556	0	3	2887
Water	2323	1313	0	111430	14793
Other	103732	22175	0	4705	690784

According to the confusion matrix, by calculating the accuracy rate, we can further analyze each label precision, recall, F1-Score, overall accuracy, IoU, mIoU and FWIoU.

name	**Vegetation**	**Building**	**Bare land**	**Water**	**Other**
precision	0.92	0.52	none	0.95	0.89
recall	0.96	0.65	0.00	0.86	0.84
F1-Score	0.94	0.58	none	0.90	0.87
IoU	0.88	0.41	0.00	0.82	0.76

overall accuracy	**0.90**
mIoU	0.57
FWIoU	0.82

The overall accuracy is acceptable. The prediction of vegetation, water and background is good, but the prediction of buildings and bare land are not ideal. That's probably because the overall area of bare land is small, and the accuracy of the model is not high enough to predict the label of bare land.

To visualize prediction results, we can select an image from the validation set.

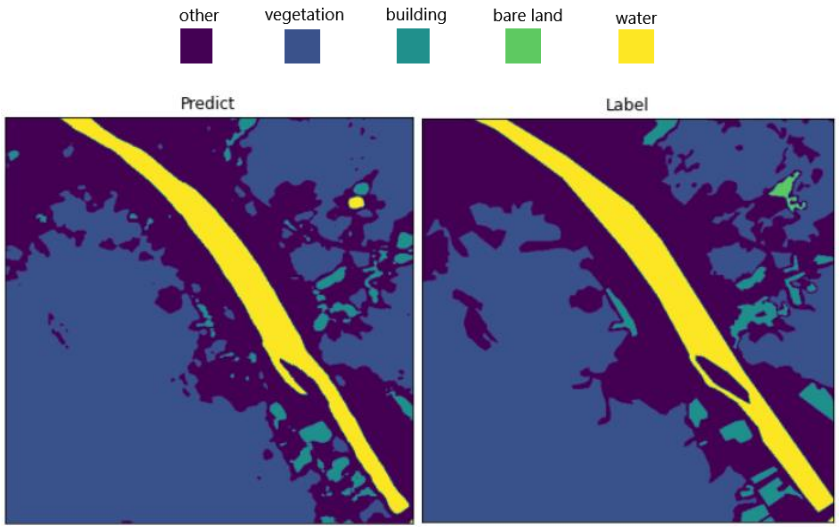

Fig. 10: prediction and true label for image 0089.tif

Combined with the visualize result, it can be seen more intuitively that the model prediction is basically correct.

In the prediction of this image, the model incorrectly classified the 'bare land' as 'building' and 'other'.

If we observe the results in combination with remote sensing images, we will find that 'water' and 'vegetation' have more obvious feature. The feature of the 'building' and the 'bare land' are relatively similar. The feature of the 'other' label appears to be more random, unpredictable, but thanks to its large training set, its accuracy is not that bad. The training data of the 'bare land' label is smaller than that of the 'building' and both of them do not have enough label that belong to them.

6 Conclusion

In the case of sufficient data sets, labels with more obvious features are easier to classify. However, labels that occupy less image area and have unobvious features are more difficult for the model to classify. Adjusting the training parameters such as learning rate and batch size can continue to improve the accuracy of the model.

7 Future Work

We can add weight to different labels. Besides, we can conduct some more operations to enhance the image, such as adding gaussian noise and adjust the illumination of satellite remote sensing photos. Lastly, we may continue to adjust the learning rate and batch size.

8 References

[1] Iglovikov, Vladimir; Shvets, Alexey (2018). "TernausNet: U-Net with VGG11 Encoder Pre-Trained on ImageNet for Image Segmentation".

[2] Ronneberger, Olaf; Fischer, Philipp; Brox, Thomas (2015). "U-Net: Convolutional Networks for Biomedical Image Segmentation".

[3] Wikimedia Foundation. (2021, February 16). U-Net. Wikipedia. https://en.wikipedia.org/wiki/U-Net.

[4] Akeret, Joel (2018-12-24), Generic U-Net Tensorflow implementation for image segmentation: jakeret/tf_unet

[5] SU Jianmin,YANG Lanxin,JING Weipeng. U-Net Based Semantic Segmentation Method for High Resolution Remote Sensing Image[J]. CEA, 2019, 55(7): 207-213.

[6] B. Cui, X. Chen and Y. Lu, "Semantic Segmentation of Remote Sensing Images Using Transfer Learning and Deep Convolutional Neural Network With Dense Connection," in IEEE Access, vol. 8, pp. 116744-116755, 2020, doi: 10.1109/ACCESS.2020.3003914.