Classify Cloud Structures from Satellites

Hanqi Wang
Department of Electrical and Computer Engineering, Master of Engineering in Electrical and Computer Engineering,
hanqiw3@illinois.edu

Yi Jiang
Department of ISchool, Master of Information Management
yij6@illinois.edu

ABSTRACT

The task of the project is to build a model to classify cloud organization patterns from satellite data. This will make scientists better able to create models that will aid our understanding of climate change. In this paper, we proposed a model to deal with the segmentation task and compare the result with different baseline methods. The result shows that our method have better performance even though the time it cost is relatively more.

1. The dataset

The task of the project is to build a model to classify cloud organization patterns from satellite data. This will make scientists better able to create models that will aid our understanding of climate change.

Input: satellite images that contain cloud formations, with label names: Fish, Flower, Gravel, Sugar.

Output: cloud patterns in each image

2. Method

2.1 Model Overview

In this paper, we use the Unet framework [1]to execute segmentation. The model generates bounding boxes and segmentation masks for each instance of an object in the image. It's based on the ResNet50 backbone. Firstly, we conducted data analysis and processing, and converted the mask value in the training set into the mask on the image for observation and analysis. Augmentation of the training images was followed, increasing the richness of the dataset. Then, the corresponding dataset class was established and the Unet framework was created for training. In the Unet framework, ResNet50 is considered as an encoder. Finally, select the appropriate threshold value to predict the test set.

2.2 Unet Framework

The U-shaped structure of U-NET [2]is shown in Figure 2.1. The network is a classical full convolutional network (i.e., there is no full connection operation in the network). The input of the network is an input image tile whose edge of 572 * 572 is mirrored. The left side of the network (red dotted line) is a series of de-sampling operations composed of convolution and Max Pooling. This part of the paper is called the Contracting Path. The compression path is composed of 4 blocks, and each block uses 3 effective convolution and 1 Max Pooling drop sampling. After each drop sampling, the number of Feature Map is multiplied by 2, so the size change of Feature Map as shown in the figure is obtained. Finally, the Feature Map with size 32 * 32 was obtained.

The right-hand side of the network (green dotted line) is called in the paper the expansive Path. It is also composed of 4 blocks. Before the start of each block, the size of the Feature Map is multiplied by 2 through deconvolution, while the number of Feature Map is halved (the last layer is slightly different), and then it is merged with the Feature Map of the symmetric compression path on the left. Since the size of the Feature Map of the compression path on the left and the extension path on the right are different, U-Net normalized the Feature Map of the compression path to the Feature Map of the same size as the extension path (that is, the dotted line on the left in Figure 2.1). The convolution operation of the extended path still uses effective convolution operation, and the size of the finally obtained Feature Map is 388 * 388. Since this task is a binary task, the network has two output Feature maps.

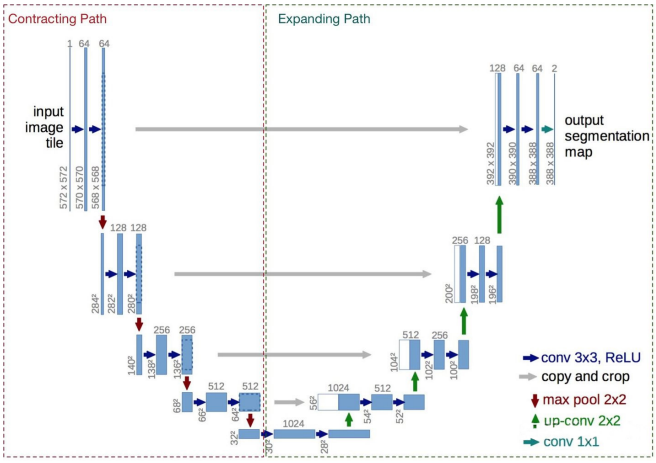

Figure 2.1

2.3 ResNet50 Backbone

With the deepening of the network [3], due to the gradient direction propagation in the model training process, multi-level derivative will be multiplied, which may lead to the gradient approaching 0, thus the gradient dissipates, resulting in the network is difficult to convergence or even unable to convergence. There are many solutions to the gradient dispersion problem, including initial network standardization, data standardization, and Batch Normalization. However, the deepening of the network will bring another problem: with the deepening of the network, the accuracy of both training set and test set will decrease.

But residual learning can solve the problem of "Decreased accuracy of network deepening" For a neural network model, if the model is optimal, the training will easily optimize residual mapping to 0, and then only identity Mapping is left. Therefore, no matter how to increase the depth, theoretically the network will always be in the optimal state. As all the networks added later will transmit information along identity mapping (itself), it can be understood that the layers behind the optimal network are useless (they do not have the ability of feature extraction) and have no effect. In this way, the performance of the network does not degrade with increasing depth.

ResNet50 is divided into 5 stages, of which Stage 0 has a relatively simple structure and can be regarded as the pre-processing of INPUT. The last 4 stages are composed of Bottleneck and have similar structures. Stage 1 contains three blocks, and the remaining three stages contain four, six, and three, respectively.

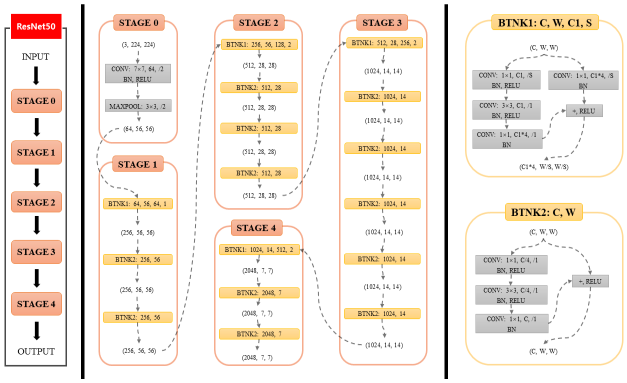

Figure 2.2 ResNet50 Backbone

3. Experimental

We evaluate the proposed model in the following aspects: time it spent and the final score in Kaggle.

3.1 Experimental Setup

As shown in Figure 3.1, the training data have 4 kinds: Sugar, Fish, Gravel and Flower. The total number of each kind is 5546. Since there are some samples not having corresponding pixels even though they have labels. Thus, we just use the samples with exact pixels as training data. Then, we would apply the trained model for the test dataset and submit the result file on Kaggle to get the final score. The score can evaluate the performance of different models. In total, we'd like to execute them 3 times and take the average as the final result.

	# of 'Sugar'	# of 'Fish'	# of 'Gravel'	# of 'Flower'
Total Labels	5546	5546	5546	5546
Labels with pixels	3751	2939	2781	2365

Figure 3.1 Training data

3.2 Model Parameters Setting

The initial parameters are as shown in Figure 3.2.

Parameter	**Initial Value**
optimizer	Adam
scheduler	ReduceLR
criterion	DiceLoss
epoch	19
batch size	10
encoder learning rate	0.001
decoder learning rate	0.01
momentum	0.9

Figure 3.2 Initial Parameters

3.3 Baselines overview

3.3.1 MaskRCNN:

MaskRCNN [4]is a Convolutional Neural Network (CNN) and state-of-the-art in terms of image segmentation. The model generates bounding boxes and segmentation masks for each instance of an object in the image. It's based on Feature Pyramid

Network (FPN) and a ResNet101 backbone. I implement MaskRCNN on Python3, Keras, and TensorFlow.

MaskRCNN is inherited from ftYL-RCNN, which was proposed by He Keming et al. In 2017, it improves ftYL-RCNN and adds Mask branch, which can realize pixel-level classification and extend target detection to instance segmentation. Mask-rcnn consists of backbone network, RPN, and HEAD branch, as shown in the following figure. The backbone network also has many combination types, mainly including ResNet50+FPN and ResNet101+FPN.

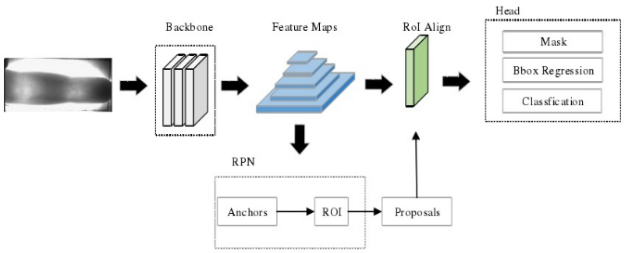

Figure 3.3 MaskRCNN Backbone

Feature Pyramid Network (FPN) is similar to TDM (Top-down Modulation). FPN is a top-down Feature fusion method, but FPN is a multi-scale target detection algorithm, that is, there is not only one Feature prediction layer. Although some algorithms also use multi-scale feature fusion for target detection, they usually only use the features of one scale after fusion. Although this method can combine the semantic information of top-level feature and the detail information of low-level feature, some deviations will be caused in the process of feature deconvolution. Only using the fusion features to predict will have a bad effect on the detection accuracy. FPN method can predict multiple fusion features at different scales and maximize the detection accuracy.

The model training method of FPN is different from the traditional Faster R-CNN method, because FPN has multiple prediction layers. Similar to the SSD approach, the early features in FPN have higher resolution, and we want to use high resolution features to predict small targets. Each fusion feature in FPN is connected to a Region Proposal Network (RPN) Network to generate candidate boxes, and then all generated candidate boxes are gathered together. Because we want smaller targets to be predicted on the convolutional feature graph with higher resolution, we allocate these candidate boxes, and the allocation principle is shown in the following formula:

$$k = k_0 + \log_2 (\sqrt{wh} / 244)$$

Where, k-0 is the prediction layer to which the candidate box is assigned, k-0 is a constant, W and H represent the width and height of the candidate box, and "represents the next step of taking the value. For example, there are 4 prediction layers (2,3,4,5), respectively representing 4 fusion feature layers from large resolution to small resolution. K0 =4, a candidate box whose width and height product is 112^2 will be assigned to the prediction layer k=3(5 when k is greater than 5, 2 when k is less than 2), and the subsequent detection will be carried out. It can be seen from the formula that the larger the area of candidate box is, the larger the number of prediction layers will be assigned, and the large target will be assigned to prediction layers with lower resolution for prediction. After allocation, these candidate boxes are sent into the corresponding RoI pooling layer, and the output results of RoI pooling layer are cascaded together, and then through two fully connected layers, target classification and target location regression are carried out. As the following picture, FPN with Fast R-CNN:

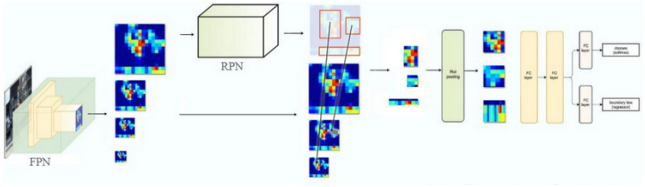

Figure 3.4 FPN with Fast R-CNN

FPN uses feature maps of different resolutions to sense objects of different sizes and makes output features have both low-level visual information and high-level semantic information through continuous up-sampling and cross-layer fusion mechanism. Low-level feature maps are not semantic rich enough to be directly used for classification, while deep features are more trustworthy. By combining the lateral connection with the top-down connection, different scores can be obtained.

3.3.2 Fastai

Fastai [5] library is based on the Deep Learning of Deep Learning courses developed by its founder Jeremy Howard et al. It provides a single and consistent interface for common Deep Learning applications such as computer vision, text, table data, time series and collaborative filtering, which can be done out of the box. Different from other top-level deep learning frameworks, Fastai modules are divided by installation task types. For example, fastai.vison module contains apis for visual aspects including data preparation, model building, training and so on. Fastai is designed to be used in four areas, with four modules named Collab (Collaborative filtering problems), Tabular (tabular or structured data problems), Text (Natural language Processing problems) and Vision (computer vision problems).

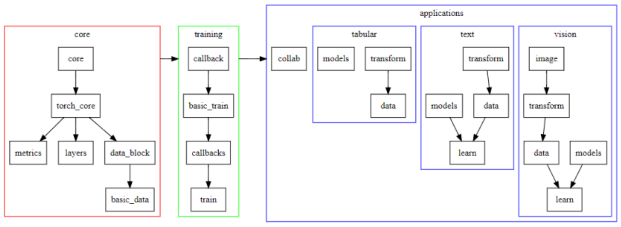

Figure 3.5 Fastai Backbone

In practice, there are three main steps: 1. Data set loading; 2. Pre-training model; 3. Validation set and test set evaluation. Fastai's core dataset loading class is ImageDataBunch, which has several methods that handle common dataset storage patterns (such as CSV +train+test). The underlying class is the Dataset class that DatasetBase inherits from Pytorch. Fastai also has significant improvements in data preprocessing, such as optimization of data

enhancement. Fastai includes pre-training models of TorchVision in PyTorch, including Resnet, VGG, Densenet, etc., as well as some models provided by Fastai, such as Resnext, Inception, etc.

3.4 Result and analysis

3.4.1 Result

The evaluation of all three models is shown in Figure 3.6. From the table, we can see that all three proposed methods can effectively classify the cloud organization patterns to some extent. According to the result, we can find that our proposed model has better performance and gets a higher score in the testing dataset even though it needs more time to train the model itself.

Model Name	**Time**	**Private Score in Kaggle**	**Public Score in Kaggle**
Unet+Resnet50	184320	0.46168	0.45097
MaskRCNN	94321	0.23962	0.23754
Fastai	134357	0.38980	0.39312

Figure 3.6 Results of different models

3.4.2. Ablation study

In our whole procedure, one of the important parts is image augmentation. Data augmentation comprises two terms data and augmentation. It is a popular technique that is used to increase the generalizability of an overfitted data model. Data augmentation is a part of data analysis. It is the set of techniques that is used to increase the amount of data by adding modified copies of already existing data. Data augmentation acts as a regularizer and assists in managing the overfitting of data. Data augmentation increases the possibility of overfitting the model by generating additional training data and exposing the model to different versions of data. In our data pre-processing, we use several augmentation methods to help the model train better. To see how the augmentation affects the overall performance of the model. We did a comparison on the performance of the model with and without augmentation. As shown in Figure 3.7, we can find that the model with augmentation has a higher score in Kaggle. Training time has been reduced to some extent but not by much.

This is because data enhancement will increase the number and diversity of samples, thus increasing the training time of the model. But because of this increase in diversity, underfitting of the model is greatly reduced, improving the performance of the model. This is one of the reasons why many image segmentations use image augmentation as preprocessing.

Scenarios	**Time**	**Private Score in Kaggle**	**Public Score in Kaggle**
with augmentation	184320	0.46168	0.45097
without augmentation	163980	0.45732	0.45184

Figure 3.7 Comparison on the performance of the model with and without augmentation

3.4.3 Implication

Unlike other training models that require many training samples, the Unet model performs well in the case of a small amount of data. However, the training samples of this classification task are not large, thus highlighting the advantages of Unet. At the same time, we added the data enhancement part in our model, so the diversity of training samples will be improved, but at the same time, some training time will be increased.

4. Future work

Unet performs well in the case of small amounts of data, but when the training model is large, its performance may not be as good as other models. So, in the future we can introduce other components in this model to cope, such as self-attention model and so on. At the same time, for a specific task, we do not extract the unique pattern of this kind of classification task, but as a common image segmentation task. As a result, the model is not specific enough and there is room for improvement in performance. We can consider introducing traditional signal processing methods to analyze images and obtain unique features of such images and strengthen the training of such features in subsequent models.

5. Contributions of each group member

Yi Jiang: realize the new method, analyze the result, and complete the report

Hanqi Wang: realize two baseline method, analyze the result, and complete the report

REFERENCES

[1] Ronneberger O, Fischer P, Brox T. U-net: Convolutional networks for biomedical image segmentation[C]//International Conference on Medical image computing and computer-assisted intervention. Springer, Cham, 2015: 234-241.

[2] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 770-778).

[3] Van Dyk, D. A., & Meng, X. L. (2001). The art of data augmentation. Journal of Computational and Graphical Statistics, 10(1), 1-50.

[4] Fastai. (n.d.). FASTAI/fastai: The FASTAI Deep Learning Library. GitHub. Retrieved May 4, 2022, from https://github.com/fastai/fastai

[5] Matterport. (n.d.). Matterport/MASK_RCNN: Mask R-CNN for object detection and instance segmentation on Keras and tensorflow. GitHub. Retrieved May 4, 2022, from https://github.com/matterport/Mask_RCNN