Classifying Clouds Structures using different Semantic Segmentation Architectures

Matthieu Lin
Department of Computer Science
Tsinghua University
lin-yh19@mails.tsinghua.edu.cn

Nathaniel Xu
Department of Computer Science
Tsinghua University
xc-xu19@mails.tsinghua.edu

Abstract

How do different semantic segmentation architectures perform in classifying clouds from satellite images? We are given satellite images that can contain any of these four types of clouds (Sugar, Flower, Fish, Gravel). There are many ways in which clouds can organize, which makes it challenging to build traditional rule based algorithms to separate cloud features. To address these problems, we used two different semantic segmentation architectures, U-net and FPN, with different backbones in order to learn automatic cloud detection and classification. Moreover, to enhance our model insensitivity to noise, we trained our model on augmented images of the original data. Based on our experiments, FPN performed better than U-Net.

1 Introduction

Thanks to Deep Learning, Computer Vision's performance on tasks such as image detection and classification has significantly improved, leading to various applications in real world problems. In recent years climate change has become one of the greatest challenges of our time and shallow clouds play a huge role in determining the Earth's climate. By successfully classifying cloud organization patterns from satellite images using semantic segmentation, we hope to help scientists better understand how clouds will shape our future climate. In this task, we need to first detect the clouds and then classify them, making the use of simple convolutional networks inefficient. In order to solve this task, segmentation is a better approach than object detection as the shapes we want to detect are not rectangular like the bounding boxes in object detection, but rather can be of any shape. Furthermore, segmentation masks are also easier to score than bounding boxes, as overlapping bounding boxes can add another layer of complexity that is not needed. We chose semantic segmentation over instance segmentation as we don't need to differentiate the instances. U-net and FPN are two recent semantic segmentation architectures that include in the desired output a segmentation mask for each class, namely Sugar, Flower, Fish, and Gravel. Our model is trained on images from NASA Worldview and is labeled by scientists of the Max-Planck-Institute for Meteorology.

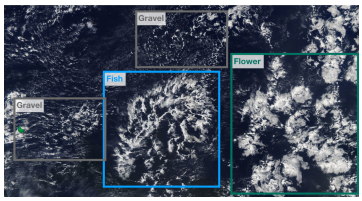


Figure 1: example of types of clouds

Preprint. Work in progress.

As there is very little training data available, we used data augmentation by applying flips, rotations and distortion on the image dataset. This also allows the network to learn invariance deformations without the need to see these transformations in the annotated image corpus. This is particularly important in cloud segmentation, since clouds can organize themselves in many ways. Our goal was to try these two architectures with different backbones; namely resnet50, inceptionresnetv2, efficient-b3, and compare how they performed. Our classification pipeline consist of 3 stages, first one to get the optimal model, then the optimal mask threshold and finally the semantic segmentation. We will begin paper by introducing the dataset, then we will introduce our methodology by introducing the image pre-processing, the model architecture, the loss function and the model inference, as a conclusion we will show our training results.

2 Exploratory Data Analysis

We are given a dataset (X, Y) where $X \in [0,1]^{n \times h \times w \times 3}$ is the tensor storing the RGB representation of each of the images in the database, and $Y \in [0,1]^{n \times h \times w \times 4}$ is the tensor storing the segmentation masks for each of the 4 different classes of each image. The given dataset has n = 5546 images, and each image has dimension h = 1400, w = 2100. We have 46.6% of empty masks in the training data. The distribution of the classes is somewhat balanced. The ratio between training and validation data was taken to be $9:1$ and the validation data is a stratified split of the original training data such that both the training and validation data have about the same proportion of images with each cloud type. From the correlation matrix, we can observe that there is no correlation between any types of cloud.

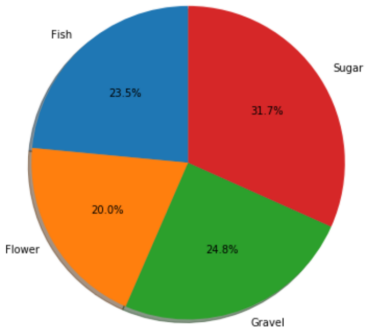
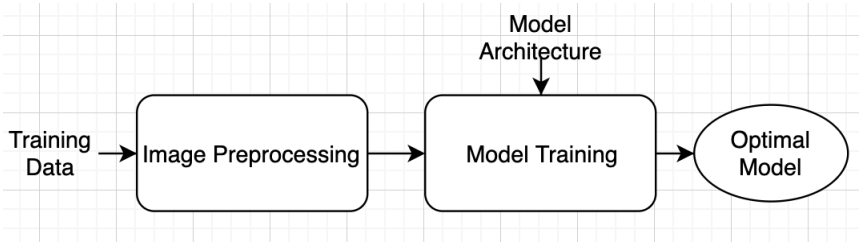


Figure 2: distribution per types of clouds

Figure 3: correlation matrix

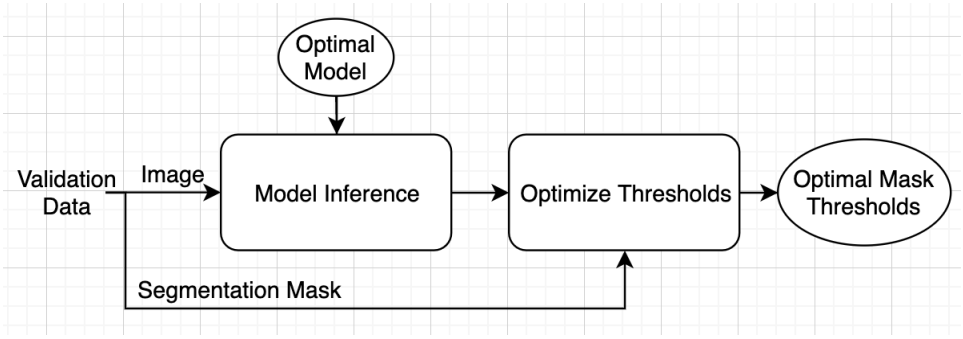
3 Methodology

Building the optimal segmentation model consists of three stages: Training, Inference, and Prediction. In the training phase the goal is to train an optimal segmentation model using an augmented training dataset. This first involves applying image preprocessing to augment our dataset, and then passing that augmented training dataset to our model's learning algorithm to train its optimal weights.

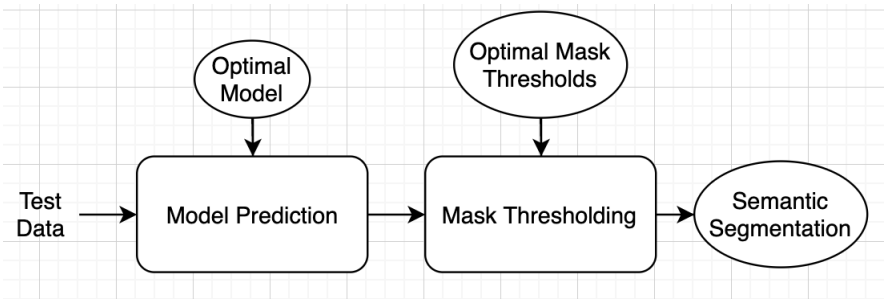


In the inference phase the goal is to learn the optimal model parameters. We use our optimal model learned in the training phase to output predictions for the image in our validation dataset. We then run

an algorithm that finds the optimal model parameters such that the discrepancy between our predicted and true outputs are minimized.



In the prediction phase the goal is to use both the optimal model weights learned from the training phase, as well as the optimal model parameters learned from the inference phase, to make the best possible prediction on the test data. The test data is first forwards passed through the optimal model, and then the resulting prediction is post-processed based on the optimal model parameters to return the final predicted output.



3.1 Image Pre-processing

There are two main purposes to preprocessing the training images. The first reason is that traditional convolutional networks are not equivariant to rotation. Pooling layers only help with invariance to small rotations, but in general rotation invariance is not addressed effectively by convolutional layers. In order to mitigate this issue we apply rotational augmentations to images and their segmentation masks. This allows the model to 'learn' the rotation invariance by training on rotated images of clouds. The other reason we preprocess the data is that the image dataset consists of images stitched from two separate satellites. This cases empty 'black bands' found in some images in the dataset. This black-banding greatly affects the ability of the model to learn accurate segmentations of clouds. To address this problem we apply an optical distortion that may 'squeeze' the black band in images and mitigate the affects it has on learning the optimal model.

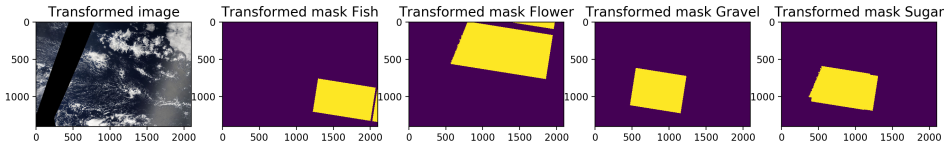


Figure 4: Rotations

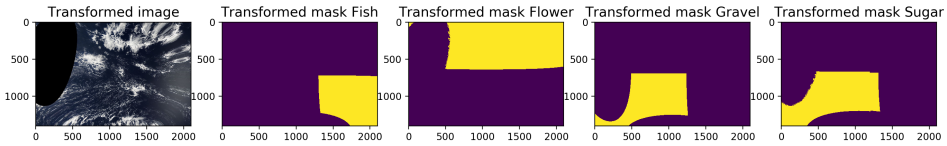


Figure 5: Optical Distortion

Both the rotational and optical image augmentations proved slightly helpful in learninig a more accurate segmentation model. The addition of rotation augmentations improved the final dice score by an average of +0.0012 for all trained models. The addition of optical augmentations only improved the final dice score by an average of +0.0004. Apart from rotational and optical augmentations we have also applied horizontal and vertical flips as well as shift and scale transforms.

3.2 Model Architecture

We explore the effectiveness of two segmentation models on the problem of classifying cloud structures. When training both types of models we use the same hyper-parameters and same training validation dataset so that any discrepancy in the results is purely due to the effectiveness of the models. The two models we compare are U-net (O. Ronneberger et al. 2015)[2] and FPN(T-Y. Lin et al. 2016)[3].

3.2.1 U-Net

U-Net is a special type of convolutional neural network that takes its name from the shape of the model, which when visualized, appears similar to the letter U. it is similar to an autoencoder architecture in that both are composed of encoder and decoder.

The encoder consists of the first half of the network, which is composed of several convolutional layers followed by a max-pool downsampling to encode the image into feature representations at different levels.

The second half of the network is the decoder, which is composed of upsampling and concatenation followed by regular convolution operations. The main contribution of U-net is that while upsampling in the network we are also concatenating the higher resolution feature maps from the encoder network with the upsampled features in order to better learn representations with following convolutions.

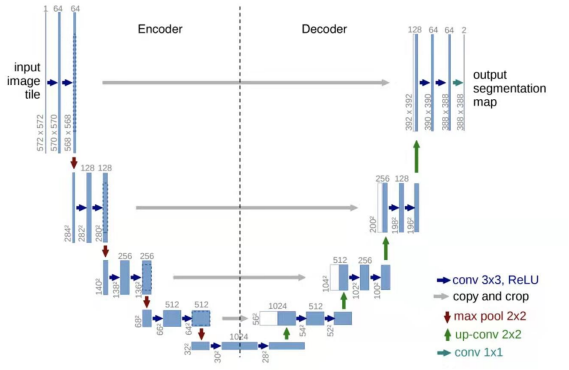


Fig. 1. U-net architecture (example for 32x32 pixels in the lowest resolution). Each blue box corresponds to a multi-channel feature map. The number of channels is denoted on top of the box. The x-y-size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations.

Figure 6: U-Net (O. Ronneberger et al. 2015)

3.2.2 Feature Pyramid Network FPN

FPN is composed of a bottom-up and top-down pathway, with different semantic layers of the pathway forming a pyramid of images. Similar to the U-Net, the FPN can also broadly be considered as an encoder and decoder network.

The bottom-up pathway consists of the feedforward computation of the backbone convolutional neural network. Like the U-Net this comprises of several convolution blocks followed by a 2x down-sampling block. The output of each forward-pass of the ConvNet is a layer in the pyramid that holds greater semantic value than the previous layer.

The top-down pathway combines low-resolution, semantically strong features with high-resolution, semantically weak features via a top-down pathway and lateral connections. More specifically the feature maps from the bottom-up pathway undergo 1x1 convolutions to reduce the channel dimensions. Then the resulting feature maps from the bottom-up and topdown pathway are merged by element-wise addition.

Segmentation masks at each feature level is generated by sliding a 5x5 MLP window over the feature maps to generate 14x14 segments. These segments are scaled and merged at different scales to form the final predicted segmentation mask.

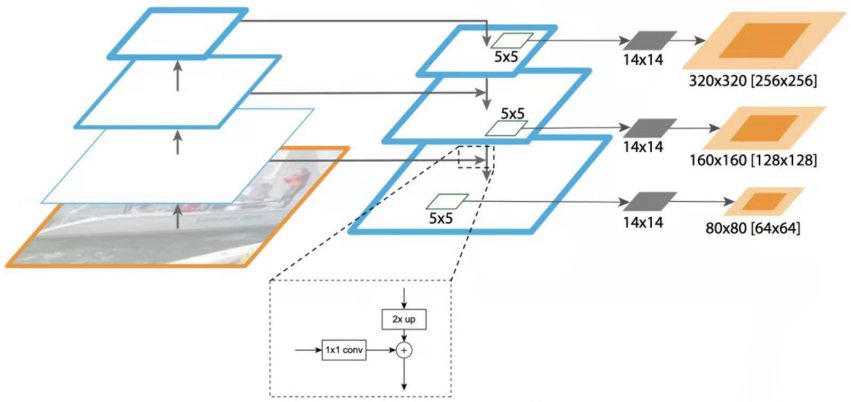


Figure 4. FPN for object segment proposals. The feature pyramid is constructed with identical structure as for object detection. We apply a small MLP on 5×5 windows to generate dense object segments with output dimension of 14×14.

Figure 7: FPN (T-Y. Lin et al. 2016)

3.2.3 U-Net vs FPN

The U-Net and FPN are similar in many ways. Both have an encoder consisting of forward passes through a convolutional network which contains a down-sampling layer. Both also utilize lateral connections between the encoder and decoder networks.

There are two main differences between the models. The first differences is that FPN generates multiple predictions, one for each up-sampling layer, whereas U-Net only generates one prediction. The idea behind generating predictions for each layer is that different layers have different semantic values, and therefore the predictions of layers with higher semantic values should be emphasized.

The second difference is in the way the lateral connections are combined. In U-Net the features from the encoder are simply appended to the features from lower semantic layers, whereas FPN applies a 1x1 convolution layer to reduce the dimension first and then apply element-wise addition to merge both feature maps.

3.2.4 Backbone

Although the encoder is specified to consist of convolution and down-sampling blocks, it does not mean the model is restricted to only have these blocks. Many different backbone architectures can be used as long as these basic building blocks are included.

3.3 Loss Function

As with any neural network, the loss function must be a measure of deviation between the predicted and true labels. For the case of a segmentation problem there are two types of loss functions we use: Binary Cross Entropy, and Dice Loss. The total loss that we then minimize is a combination of the BCE and Dice loss. L. Yang and C. You showed in their paper about Instance-U-Net and Watershed: Improved Segmentations for breast cancer cells[4] that BCE + DICE was the combination that produced the most optimal loss and so we define our total loss to be:

$L(y, \hat{y}) \; = \; BCE(y, \hat{y}) \; + \; D(y, \hat{y})$

3.3.1 Binary Cross Entropy

Binary Cross Entropy examines each pixel individually, comparing the pixel probability value of the class predictions to our one-hot encoded target vector. This measure of pixel-wise cross entropy is then totalled and averaged over all the pixels in the image. For our predicted segmentation masks the cross entropy can be calculated as:

$BCE(y, \hat{y}) \; = \; -\sum_{i \in pixels} y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)$

The corresponding gradient of the cross entropy loss is calculated to be:

$\frac{\partial BCE(y,\hat{y})}{\hat{y}_j} = -\frac{y_j}{\hat{y}_j} + \frac{1-y_j}{1-\hat{y}_j}$

3.3.2 Dice Loss

The Dice Loss is based on the Dice Coefficient, which is a measure of overlap between two samples. It ranges from 0 to 1 where 0 denotes zero overlap and 1 denotes complete overlap between two samples. For two sets of data the Dice Coefficient can be calculated as:

Dice Coefficient$= \frac{2|A \cap B|}{|A| + |B|}$

When dealing with matrices of pixel probabilities rather than sets, we can treat each pixel as binary and apply DeMorgan's Laws to approximate the intersection between two pixels as:

$x_i \cap x_j = x_i \cdot x_j$

Similarly, the cardinality of the set is taken as the sum of all elements in the set:

$|X| = \sum_{i,j} x_{ij}$

Finally the Dice Loss is defined to be 1 - Dice Coefficient to allow us to apply gradient descent

$$D(A, B) = \left(1 - \underbrace{\frac{2|A \cap B|}{|A| + |B|}}_{Dice\ Coefficient}\right) \Longrightarrow D(y, \hat{y}) = \left(1 - \frac{2 \sum\limits_{i \in pixels} y_i \cdot \hat{y_i}}{\sum\limits_{i \in pixels} y_i + \sum\limits_{i \in pixels} \hat{y_i}}\right)$$

The corresponding gradient of the Dice loss is calculated as:

$\frac{\partial D(y,\hat{y})}{\partial \hat{y_j}} = -2\left[\frac{y_j\left(\sum y_i + \sum \hat{y_i}\right) - \sum y_i \hat{y_i}}{\left(\sum y_i + \sum \hat{y_i}\right)^2}\right]$

3.4 Model Inference

The goal of model inference is to learn the optimal mask threshold value, and optimal minimum component size. The mask threshold value is a value that filters out predicted pixel probabilities less than the threshold. For instance, if the model predicted a pixel has a 0.53 probability of being in the segmentation mask, and the threshold for this class is 0.55, then this pixel would be filtered out and its

value would be set to 0. If a model predicted a pixel has a 0.61 probability of being in the mask, then it would not be filtered and instead is set to 1. The minimum component size denotes the minimum size that any predicted segmentation mask must contain. This means that predicted segmentations with fewer number of pixels than the minimum component size are filtered out. This is to prevent the prediction of extremely tiny segmentations. We implemented a simple iterative algorithm to systematically evaluate the effectiveness of both parameters, and determine the optimal parameters that would maximize the Dice score of the validation dataset. The basic idea is that for some class k, we iterate through a range of values for both the mask threshold value and minimum component size. At each iteration we apply the mask thresholding and component filtering for each predicted segmentation of class k, and then calculate the Dice score for that prediction. The Dice score is then averaged over all predictions of class k. The best parameters are the ones that produced the highest average Dice score on the validation dataset. Pseudocode for the algorithm is shown below

Algorithm 1: Optimize Model Parameters

```
Find_Optimal_Class_Parameters (class=k);
    params = ∅;
    for thresh, min_size ∈ [0, 1], [0, 10000] do
        scores = ∅;
        for predicted_mask, true_mask ∈ class k do
            Threshold_Mask (predicted_mask, thresh);
            Filter_Components (predicted_mask, min_size);
            dice_score = Dice (predicted_mask, true_mask);
            scores = scores ∪ dice_score;
        end
        average_score = Average(scores);
        params = params ∪ {average_score, thresh, min_size};
    end
    return Maximum(params, criteria=average_score);
```

Figure 8: Algorithm for learning optimal parameters

Example output of the algorithm, which returns the calculated optimal parameters

```
0
    threshold   size      dice
69       0.65  10000  0.595904
54       0.50  10000  0.592882
74       0.70  10000  0.590582
49       0.45  10000  0.589705
79       0.75  10000  0.589443
```

Figure 9: Optimal parameters for class 1

```
1
    threshold   size      dice
59       0.55  10000  0.769015
64       0.60  10000  0.767196
63       0.60   5000  0.764458
54       0.50  10000  0.758603
69       0.65  10000  0.758260
```

Figure 10: Optimal parameters for class 2

```
2
    threshold   size      dice
64       0.60  10000  0.626579
69       0.65  10000  0.622466
59       0.55  10000  0.620565
54       0.50  10000  0.619235
68       0.65   5000  0.617678
```

Figure 11: Optimal parameters for class 3

```
3
    threshold   size      dice
44       0.40  10000  0.631604
49       0.45  10000  0.627144
54       0.50  10000  0.622694
53       0.50   5000  0.616798
58       0.55   5000  0.616318
```

Figure 12: Optimal parameters for class 4

4 Training Results

Both U-Net and FPN architectures were used to train the cloud image database. The learning rate and augmentations used were the same and both models were trained for 25 epochs. The architectures were compared based on the type of backbone used, and so any difference in results is purely a result of the architecture (U-Net or FPN) used. The results are shown below:

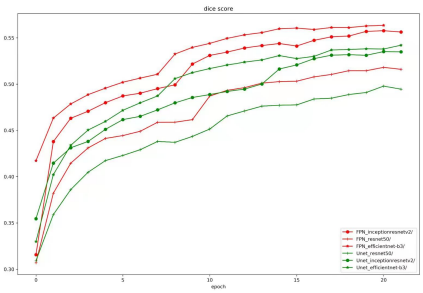


Figure 13: Dice score

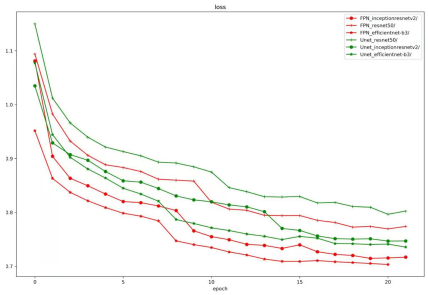


Figure 14: Loss

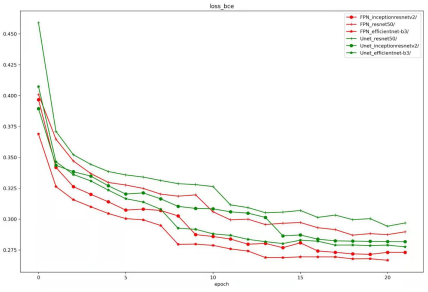


Figure 15: BCE loss

Table 1: Model Dice Scores

Backbone	Unet	FPN
resnet50	0.63438	0.63374
inceptionresnetv2	0.64993	0.65018
efficientnet-b3	0.65032	0.65637

Figure 16: comparison of architectures and backbones

Based on our final result our ranking on this Kaggle competition is 127^{th}.

The loss graphs suggest that FPN generally learns faster than U-Net. For most of the backbones used, the FPN has a steeper loss curve, meaning it reaches convergence at a faster rate. Furthermore FPN seems to perform slightly better than U-Net as well. The final Dice scores show that most backbones with FPN end up with a slightly higher Dice score than backbones with U-Net. While these results are far from conclusive, it does give us insight into the effectiveness of both models as a means of segmentation and classification.

Acknowledgments

This work was supported by Tsinghua University

References

[1] S. Rasp & H. Schulz & S. Bony & B. Stevens (2019) Combining crowd-sourcing and deep learning to explore the meso-scale organization of shallow convection

[2] T-Y. Lin & P. Dollar & R. Girshick (2016) Feature Pyramid Networks for Object Detection

[3] O. Ronneberger & P. Fischer & T. Brox (2015) U-Net: Convolutional Networks for Biomedical Image Segmentation

[4] L. Yang & C. You (2018) Instance-U-Net and Watershed: Improved Segmentations for breast cancer cells