Understanding Cloud Organization

Liyuan Geng, Jinhong Xia, Jiayi Xu

Abstract

Clouds play a huge role in determining the Earth's climate. By classifying different cloud organizations accurately, people can have a better understanding of how clouds shape our future climates. Our project aimed at training a model that classifies different cloud organizations accurately, which may encounter difficulties in terms of the complex nature of clouds and the murky boundaries between different types. To solve this problem, we experimented with two models, Unet as the baseline model and PSPNet as a potentially improved model. We found out that the performance of PSPNet is slightly better than U-net, which is in our expectation.

1 Introduction

Clouds have a significant impact on the Earth's climate. Therefore, accurately classifying different types of cloud organizations can enhance our physical understanding of these clouds, which in turn will help us build better climate models to help reduce uncertainties in climate projections.

However, compared to the detection by human eyes, cloud organization is difficult to be detected by machines because of the murky boundary between different classes. Ideally, a model for such cloud organization detection should be able to identify the specific regions in satellite images that contain certain cloud formations.

In our project, we work on satellite images of clouds of 4 different labels, fish, flower, gravel, and sugar, for which we segment the regions in the satellite image. We train two models, vanilla U-net as a baseline model, and PSPNet for identifying the above four types of cloud formations on given satellite images and marking their specific locations.

2 Dataset

2.1 Dataset Description

Our dataset is provided by Max Planck Institute for Meteorology on Kaggle. It contains 5546 image instances that are each classified into four categories: Flower, Fish, Gravel, and Sugar. The first file we used is train.csv, which contains the run-length encoding (RLE in short) for all the images under the four categories. And the second file we used is a folder named trainimages, which contains the original images in jpg format for the instances present in the train.csv file.

To gain further insight into our dataset, we conducted a statistical analysis of three key metrics for each of the four categories. The metrics included the mask rate, mask size rate, and figure with object rate. The mask rate, as shown in Figure 1, represents the proportion of foreground pixels in the total number of pixels across all decoded masks for a given category. The mask size rate, as shown in Figure 2, represents the proportion of foreground pixels in the total number of pixels across all decoded masks for a given category, after removing all empty masks. The figure with object rate, as shown in Figure 3, represents the proportion of images that contain clouds of a specific category, relative to the total number of images in the dataset.

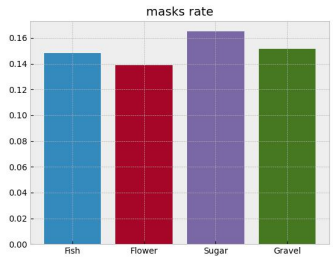

Figure 1: Mask Rate

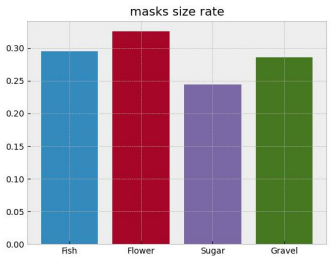

Figure 2: Mask Size Rate

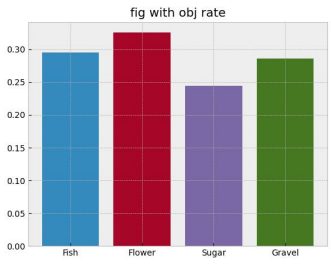

Figure 3: Figure With Object Rate

2.2 Data Preprocessing

2.2.1 Input & Output Value

In our model, we converted the original jpg format satellite picture with OpenCV package into a tensor of size 3×h×w, which is the input X into training models. 3 represents the three channels R, G, and B. Moreover, each image is resized from 1400×2100 pixels to 350×525 for h×w. In terms of the output value, ground-truth y is the mask tensor decoded from the RLE (run-length encoding) string for the four labels (Flower, Fish, Gravel, Sugar) of each image, which is of size 4×h×w, where 4 channels stand for 4 different labels.

2.2.2 Train Test Splitting

We splitted 1/5 of the dataset as the test set and used the remaining data as the training set and validation set. Specifically, we used 3993 instances for our training data, 444 for validation data, and 1109 for testing data.

2.2.3 RLE Encoding & Decoding

RLE (run-length encoding) is used to encode the location of foreground objects in segmentation. Instead of outputting a mask image, RLE gives a list of start pixels and how many pixels after each of those starts pixels are included in the mask. A simple example of RLE is shown in Figure 4.

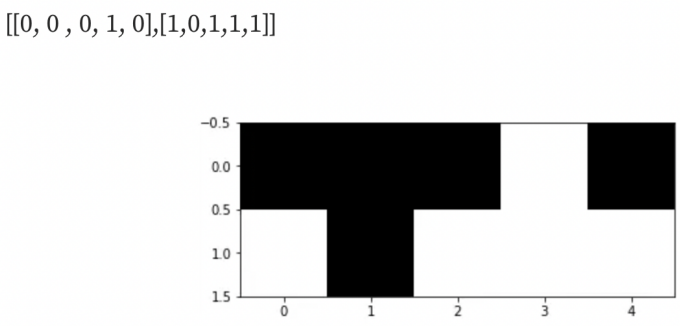

Figure 4: RLE Encoding

In the dataset train.csv, the ground truth for each image under the four labels is encoded using RLE. For each pixel of an image of size 1400×2100 pixels, RLE works by determining whether the pixel belongs to the foreground or the background for a certain class. For consecutive pixels belonging to the foreground, RLE records them as a sequence where the first element is the index (an integer from 0 to 1400×2100) of the first pixel among these consecutive pixels and the second element is the number of these consecutive pixels. For example, (264918 937) means that starting from the 264918th pixel, there are 937 sequential pixels belonging to the foreground.

2.2.4 Data Augmentation

To avoid overfitting and improve the generalization and robustness of the model, one commonly used method is to augment the data using some label-preserving transformations on both the training and validation sets[1].

In our model, we also employed data augmentation to generate the data set for training out a model with better generalization. For both the training and validation sets, we resized images inside to a size of 320 × 640, and their pixels in the three channels are normalized with a mean of 0.485, 0.456, 0.406, and a standard deviation of 0.229, 0.224, and 0.225 respectively. In particular, we augmented the data in our training set by flipping the images horizontally and performing random translation, scaling, rotation, and grid distortion. A simple example of augmentation is shown in Figure 5.

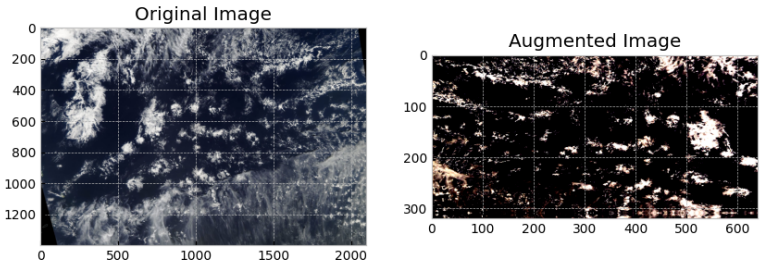

Figure 5: Original Image & Augmented image

3 Explanation of Methods used

3.1 Models

3.1.1 Baseline Model Vanilla U-net

We started with U-net as our baseline model. As is shown in Figure 6, the U-net model consists of a contracting path and an expansive path, forming a U-shape because of the symmetry between the two paths[2].

In our Vanilla Unet model, we used double convolutional layers in downsampling paths and bilinear interpolation as well as transposed convolution in the upsampling path. Specifically, in the double convolutional neural networks, we used batch normalization for faster convergence as well as more robustness and ReLU for activation function. And after the last upsampling which generates a feature map with the same size as the input image, a sigmoid function is performed on each pixel to get the probability that the pixel is a foreground pixel.

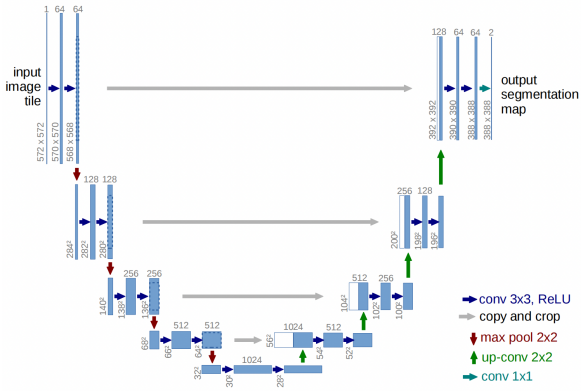

Figure 6: U-net Architecture

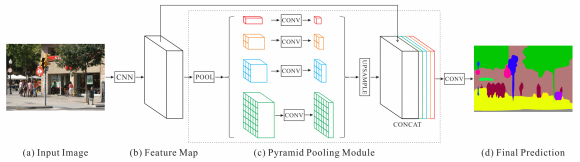

Figure 7: PSPNet Architecture

3.1.2 PSPNet

We then used Pyramid Scene Parsing Network (PSPNet) to seek potential improvements in the performance. PSPNet provides an effective global contextual prior for pixel-level scene parsing[3]. As is shown in Figure 7, the core component of the architecture is the pyramid pooling module (PPM in short), which fuses features under four different pyramid scales. The pyramid pooling module that we used is a four-level one with bin sizes of 1×1, 2×2, 3×3 and 6×6 respectively. For the pooling layer in the PPM, we used an adaptive average pooling strategy. The pyramid pooling module can collect levels of information, which is very representative [3]. Besides, we used a pre-trained ResNet model with the dilated network strategy to extract the feature map.

3.2 Evaluation Metrics

We used Dice Coefficient to evaluate the performance of our models in the validation and testing stage of our project.

$$DiceScore(y^i, y^i_{pred}) = \frac{2\sum_i y^i y^i_{pred} + \epsilon}{\sum_i y^i + \sum_i y^i_{pred} + \epsilon} \tag{1}$$

We added an ϵ to the denominator to prevent zero division error. Besides, this is a soft Dice Coefficient when it is used in the validation phase as the threshold is not fixed. In the test phase, it is a hard Dice Coefficient as

y_{pred} is a binary number based on the threshold which is decided in the validation phase. The larger the Dice Coefficient is, the better the model performs.

3.3 Loss Function

In terms of the loss function, we used BCEDice Loss, which combines both Binary Cross Entropy Loss and Dice Loss by adding them up.

$$loss(y, y_{pred}) = DiceLoss(y, y_{pred}) + BCELoss(y, y_{pred}) \tag{2}$$

Here, y_{pred} is the predicted probability on each pixel rather than a binary number as the threshold is unknown in the training phase.

1. **Dice Loss**: We used soft Dice loss due to its ability to handle imbalanced scenarios between multiple classes [4].

$$DiceLoss(y, y_{pred}) = 1 - DiceScore(y, y_{pred}) \tag{3}$$

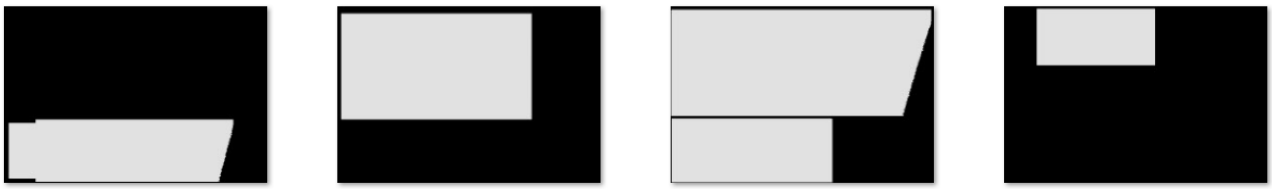

Figure 8: Masks of Different Classes of One Image

2. **BCELoss**: Cross-Entropy (CE) loss is the most popular option when dealing with natural images[4]. However, in this particular segmentation task, as is shown in Figure 8, some pixels of an image can be covered by masks of different classes. Therefore, we used Binary Cross Entropy (BCE) loss to measure the difference between predicted and ground-truth values.

$$BCELoss(y, y_{pred}) = -y \log(y_{pred}) - (1 - y) \log(1 - y_{pred}) \tag{4}$$

3.4 Optimizer

We used RAdam for optimization. Rectified Adam (RAdam in short), as a variant of the Adam optimization algorithm, has a better performance compared to vanilla Adam. The basic algorithm for RAdam is shown in Figure 9. Through the use of a variance correction term ρ_t and the adaptive rectification term r_t, it addresses several limitations of vanilla Adam, such as instability and slow convergence in the early stages of training[5].

Algorithm 2: Rectified Adam. All operations are element-wise.

Input: $\{\alpha_t\}_{t=1}^{T}$: step size, $\{\beta_1, \beta_2\}$: decay rate to calculate moving average and moving 2nd moment, θ_0: initial parameter, $f_t(\theta)$: stochastic objective function.

Output: θ_t: resulting parameters

$m_0, v_0 \leftarrow 0, 0$ (Initialize moving 1st and 2nd moment)

$\rho_\infty \leftarrow 2/(1-\beta_2) - 1$ (Compute the maximum length of the approximated SMA)

while $t = \{1, \cdots, T\}$ **do**

  $g_t \leftarrow \nabla_\theta f_t(\theta_{t-1})$ (Calculate gradients w.r.t. stochastic objective at timestep t)

  $v_t \leftarrow \beta_2 v_{t-1} + (1-\beta_2) g_t^2$ (Update exponential moving 2nd moment)

  $m_t \leftarrow \beta_1 m_{t-1} + (1-\beta_1) g_t$ (Update exponential moving 1st moment)

  $\widehat{m_t} \leftarrow m_t / (1-\beta_1^t)$ (Compute bias-corrected moving average)

  $\rho_t \leftarrow \rho_\infty - 2t\beta_2^t/(1-\beta_2^t)$(Compute the length of the approximated SMA)

  if *the variance is tractable, i.e.,* $\rho_t > 4$ **then**

    $l_t \leftarrow \sqrt{(1-\beta_2^t)/v_t}$ (Compute adaptive learning rate)

    $r_t \leftarrow \sqrt{\frac{(\rho_t-4)(\rho_t-2)\rho_\infty}{(\rho_\infty-4)(\rho_\infty-2)\rho_t}}$ (Compute the variance rectification term)

    $\theta_t \leftarrow \theta_{t-1} - \alpha_t r_t \widehat{m_t} l_t$ (Update parameters with adaptive momentum)

  else

    $\theta_t \leftarrow \theta_{t-1} - \alpha_t \widehat{m_t}$ (Update parameters with un-adapted momentum)

return θ_T

Figure 9: RAdam Algorithm

3.5 Hyperparameter Tuning

1. **Number of Epochs**: For both models, our training process takes 32 epochs. We tracked the training loss and validation loss to prevent overfitting and stop the training process when validation starts to increase.
2. **Threshold**: The outputs y_{pred} of both models are tensors of numbers between 0 and 1 as a result of the sigmoid activation function. In the validation phase, we found the best threshold for each class using grid search to judge whether the entry is 0 or 1, so that we could calculate the dice coefficient with thresholds in the testing phase. Then we could compare the dice coefficient with thresholds between the two models.
3. **Minimal Size**: Minimal size refers to the minimum size of a mask, as in the segmentation task, we should eliminate those masks with very small sizes.

We applied grid search to get the best threshold and minimal size of the mask. We tried to get the threshold and minimal size of the model which achieved the best dice score in the validation phase and applied the best threshold and minimal size of the model in the test phase.

4 Results and Discussion

4.1 Validation Results for U-net

The training loss and validation loss plot for U-net is shown in Figure 10. The dice score plot for U-net is shown in Figure 11.

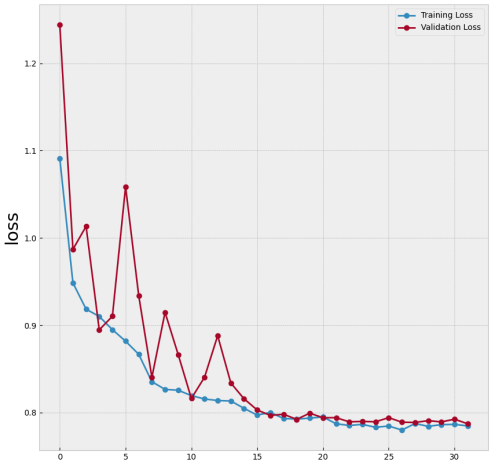

Figure 10: U-net Training Loss &Validation Loss

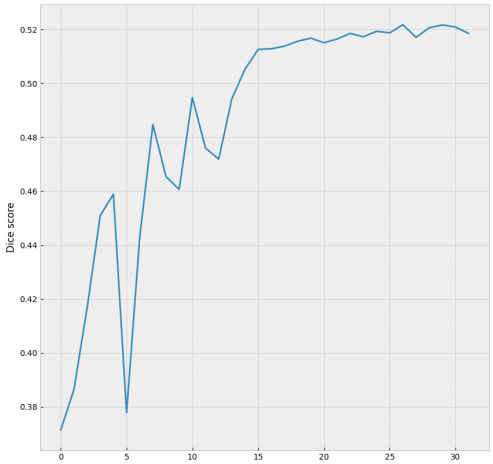

Figure 11: U-net Dice Score

As is shown in the plot, the validation loss still decreases during the last epoch of training, so the model we used is the model after 32 epochs of training.

Then we applied grid search to find the best threshold and minimal size, which can achieve the best dice coefficient.

The best threshold and minimal size of each class are shown in Table 1.

Class	Best Threshold	Best Minimal Size (Pixels)
0 (Fish)	0.55	30000
1 (Flower)	0.5	30000
2 (Gravel)	0.35	30000
3 (Sugar)	0.65	10000

Table 1: Best Threshold and Minimal Size

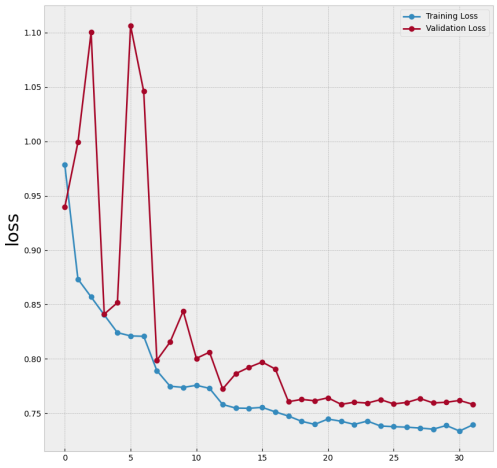

Figure 12: PSPNet Training Loss &Validation Loss

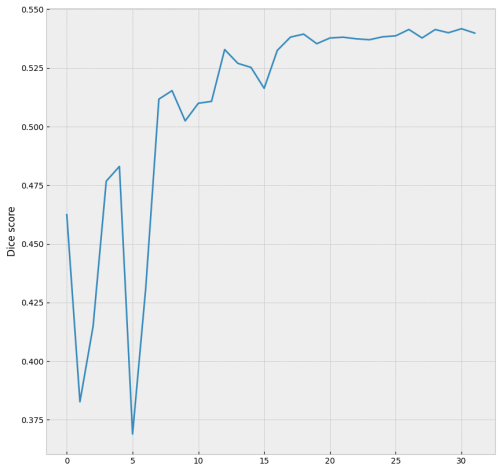

Figure 13: PSPNet Dice Score

4.2 Validation Results for PSPNet

The training loss and validation loss for PSPNet are shown in Figure 12. The dice score plot for U-net is shown in Figure 13.

As is shown in the plot, the validation loss stops decreasing till epoch 22, so the model we used is the model after 22 epochs of training. Here, we applied early stopping to prevent overfitting.

Then we applied grid search to find the best threshold and minimal size, which can achieve the best dice coefficient. The best threshold and minimal size of each class are shown in Table 2.

Class	Best Threshold	Best Minimal Size (Pixels)
0 (Fish)	0.3	30000
1 (Flower)	0.6	10000
2 (Gravel)	0.3	30000
3 (Sugar)	0.6	5000

Table 2: Best Threshold & Minimal Size

4.3 Comparison of Test Results of U-net & PSPNet

We loaded both the U-net model and PSPNet model that we derived in the validation phase, and we tested our models using the test set, which contains 1109 images and their corresponding masks. Table 3 shows the dice score of the two models, respectively.

Model	Dice Score	Test Loss
U-net	0.46761377073764	0.9649831320429697
PSPNet	0.49289291096971	0.9044053860819767

Table 3: Dice Score & Test Loss Comparison

From Table 3, we can see that PSPNet performs better than U-net in the segmentation task since the dice score of PSPNet is much larger than that of vanilla U-net. The result is in our expectation, which shows that PSPNet does improve the performance of the segmentation task.

From our perspectives, the potential reason for the improvement of the performance is that PSPNet has a pyramid pooling module, which can collect levels of information and is very representative. Compared to vanilla U-net, the encoder of PSPNet better captures the global context information, which may explain its better performance in this segmentation task.

4.4 Visualization of Results

Figure 14 shows a visualization of segmentation masks of a particular image, which is a training result of PSPNet. The first line shows the ground-truth masks of the image. The second line shows the predicted masks without thresholds using a color map. The third line shows the predicted masks with a threshold fixed during the validation phase.

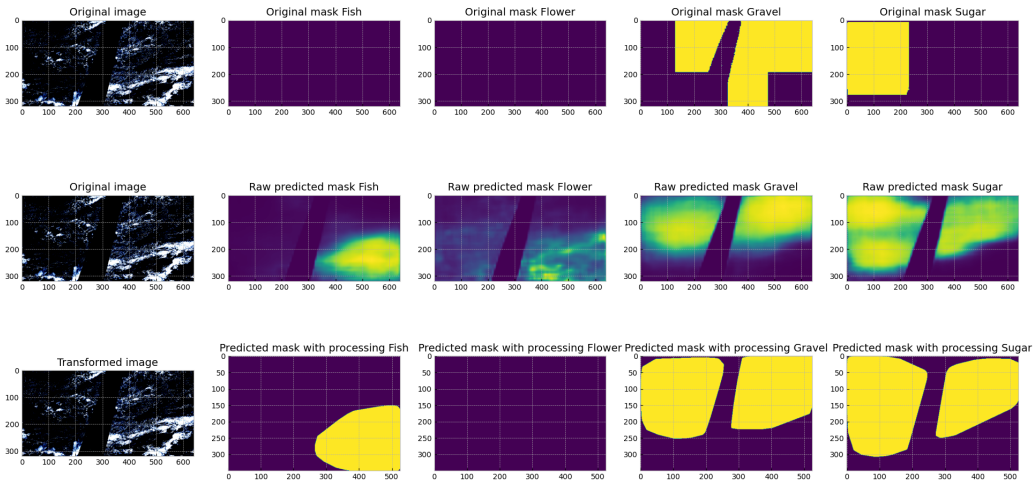

Figure 14: Segmentation Visualization

5 Conclusion & Future Works

This project focused on cloud organization segmentation using convolutional neural networks. Specifically, we trained two models, Unet and PSPNet, and compared their performance in terms of semantic segmentation on the cloud dataset with 5546 instances of four organization categories. The test results demonstrate that the PSPNet model outperforms UNet, which is in our expectation. We believe this is because PSPNet has better feature extraction capabilities than UNet, allowing it to obtain more accurate global context information and thus produce better prediction results.

In order to achieve better segmentation results, we still have several anticipations for our future work. For example, in the case of PSPNet, we can add an auxiliary loss to the loss function, which can potentially improve the training process. Adding an auxiliary loss helps optimize the learning process[3]. Besides, we can use some newly released models like transformers to do the segmentation task[6], which may potentially improve the testing results. Also, in terms of the loss function, we use a combination of BCELoss and DiceLoss whose weights are equivalent. We can set the relative weight λ of BCELoss and DiceLoss as a hyperparameter, and finetune this weight λ, which can potentially generate better loss function and testing results.

6 Acknowledgement

We would like to express our gratitude to our mentor, Professor Li Guo. Her valuable suggestions and patient mentorship were instrumental in shaping this project.

We also want to thank Haoming(Hammond) Liu, who provides us with invaluable guidance in using HPC, for his unwavering support, and suggestions throughout this project.

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," vol. 25, 2012. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/3065386

[2] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," *CoRR*, vol. abs/1505.04597, 2015. [Online]. Available: http://arxiv.org/abs/1505.04597

[3] H. Zhao, J. Shi, X. Qi, X. Wang, and J. Jia, "Pyramid scene parsing network," *CoRR*, vol. abs/1612.01105, 2016. [Online]. Available: http://arxiv.org/abs/1612.01105

[4] A. Galdran, G. Carneiro, and M. Ángel González Ballester, "On the optimal combination of cross-entropy and soft dice losses for lesion segmentation with out-of-distribution robustness," 2022.

[5] L. Liu, H. Jiang, P. He, W. Chen, X. Liu, J. Gao, and J. Han, "On the variance of the adaptive learning rate and beyond," 2021.

[6] R. Strudel, R. Garcia, I. Laptev, and C. Schmid, "Segmenter: Transformer for semantic segmentation," 2021.