III. Model documentation and write-up

1. Who are you (mini-bio) and what do you do professionally?

We are a team from Wuhan University, China. Our members are Xuechao Zhou, Yang Pan, and Yueming Sun, who are current graduate students in Photogrammetry and Remote Sensing, Gui Cheng, who is a current PhD student in Photogrammetry and Remote Sensing, and Professor Zhenfeng Shao, who is mainly engaged in teaching and research in the direction of smart cities and urban remote sensing.

2. What motivated you to compete in this challenge?

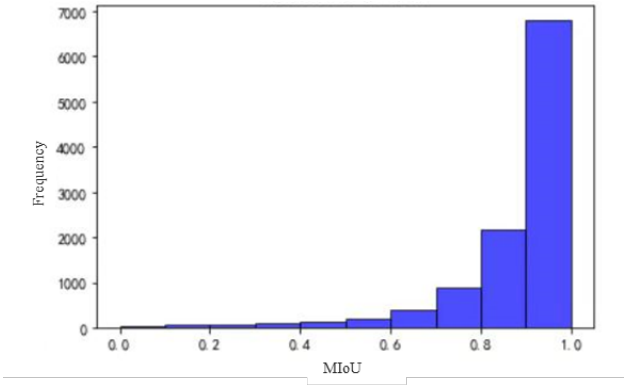
Our majors are both in remote sensing, and we have recently been conducting research in the direction of cloud detection of remote sensing images. We just saw this challenge and we were both very interested in it, and we thought it was a good opportunity to put our research knowledge into practice.

3. High level summary of your approach: what did you do and why?

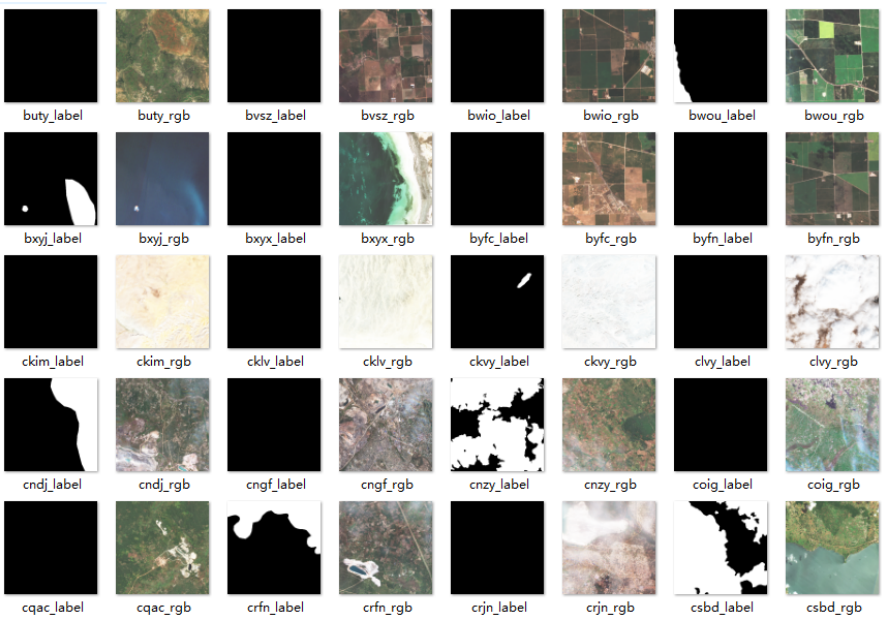
As there are many error labels in the original dataset, we first clean the dataset, and according to our research and study on cloud detection, we choose the UNet++ model as our base model to train and predict the original dataset. Then we visualize the labels of the images whose predicted MIoU is lower than 70% and remove the incorrect labels from the manual assisted screening. After that, the cleaned dataset is randomly divided into training and validation sets, and the operation is repeated to obtain five training and validation sets with different distributions for training, and the best-performing model and its dataset division are selected for integration based on the validation results. Finally, according to our experimental results, Test-Time Augmentation (TTA) is performed on the UNet++ model with efficientnetv2_rw_s as encoder during testing to increase the generalizability of the model.

4. Do you have any useful charts, graphs, or visualizations from the process?

When cleaning the dataset, we first counted the distribution of the prediction accuracy to determine the division of the exclusion threshold.



According to the above figure, the labels of the images with prediction accuracy lower than 70% were selected for visualization and the filtered wrong labels were removed. The visualization results of some of the erroneous labels are shown in the following figure.



5. Copy and paste the 3 most impactful parts of your code and explain what each does and how it helped your model.

① This code performs Test-Time Augmentation (TTA) on the Unet++ model with efficientnetv2_rw_s as encoder to give the model the best chance of correctly classifying the given image.

```
predict_1 = model(data)[:, 0, :, :] # [b,c,h,w]->[b,1,h,w]->[b,h,w]
predict_2 = model(torch.flip(data, [-1]))[:, 0, :, :] # [b,c,h,w]->[b,1,h,w]->[b,h,w]
predict_2 = torch.flip(predict_2, [-1])
predict_3 = model(torch.flip(data, [-2]))[:, 0, :, :] # [b,c,h,w]->[b,1,h,w]->[b,h,w]
predict_3 = torch.flip(predict_3, [-2])
output = (torch.sigmoid(predict_1) + torch.sigmoid(predict_2)
          + torch.sigmoid(predict_3)).cpu().numpy().astype('float32') / 3
```

②This code calculates the predictions of our best eight models and integrates the predictions of the above eight models and takes their average values to get the final output, which gives us a significant improvement in test accuracy.

```
batch_out_all = []
for idx, model in enumerate(models):
    model.eval() # [b,c,h,w]
    if idx < 6: # 0 1 2 3 4 5
        output = model(data)[:, 0, :, :] # [b,c,h,w]->[b,1,h,w]->[b,h,w]
        output = torch.sigmoid(output).cpu().numpy().astype('float32')
    else:
        predict_1 = model(data)[:, 0, :, :] # [b,c,h,w]->[b,1,h,w]->[b,h,w]
        predict_2 = model(torch.flip(data, [-1]))[:, 0, :, :] # [b,c,h,w]->[b,1,h,w]->[b,h,w]
        predict_2 = torch.flip(predict_2, [-1])
        predict_3 = model(torch.flip(data, [-2]))[:, 0, :, :] # [b,c,h,w]->[b,1,h,w]->[b,h,w]
        predict_3 = torch.flip(predict_3, [-2])
        output = (torch.sigmoid(predict_1) + torch.sigmoid(predict_2)
                  + torch.sigmoid(predict_3)).cpu().numpy().astype('float32') / 3
    batch_out_all.append(output)
output_mean = np.mean(batch_out_all, 0) #[num_models,b,512,512]->[b,512,512],mean
batch_pred = np.where(output_mean > 0.55, 1, 0).astype('uint8')#[b,512,512]
```

③This code feeds the test set into the network in batches to enable multi-threaded operation and thus shorten the inference time.

```
class LoadTifDataset(torch.utils.data.Dataset):
    def __init__(
        self,
        img_dir,
        chip_ids,
    ):

        self.img_dir = img_dir
        self.chip_ids = chip_ids
    def __len__(self):
        return len(self.chip_ids)
    def __getitem__(self, idx):
        chip_id=self.chip_ids[idx]
        sample = {}
        # Load in image
        arr_x = tifffile.imread([os.path.join(self.img_dir, chip_id+'/'+f"B0{k}.tif") for k in [2, 3, 4, 8]])
        arr_x = (arr_x / 2 ** 16).astype(np.float32)
        sample["chip"] = arr_x
        sample["chip_id"] = chip_id
        return sample
```

6. Please provide the machine specs and time you used to run your model.

●CPU (model): Intel(R) Xeon(R) CPU E5-2683 v3 @ 2.00GHz
●GPU (model or N/A): Nvidia Titan Xp*4
●Memory (GB):12GB*4
●OS: Linux ubantu18.0
●Train duration: training one Unet++ model takes ~ 6-9 hours, one Deeplabv3+ model takes ~ hours10.
●Inference duration: ~2.6 hours.

7. Anything we should watch out for or be aware of in using your model (e.g. code quirks, memory requirements, numerical stability issues, etc.)?

If you encounter memory overflow problems when training the model, you can set the batch size to be smaller, and nothing else needs attention.

8. Did you use any tools for data preparation or exploratory data analysis that aren't listed in your code submission?

None.

9. How did you evaluate performance of the model other than the provided metric, if at all?

We tested the models on validation sets.

10. What are some other things you tried that didn't necessarily make it into the final workflow (quick overview)?

We have tried to train the model using different data set divisions, using different loss functions (boundary loss, ohem loss, lovasze loss), and using different models for the ensemble, but the results

are not as good as the current combined approach.

11. If you were to continue working on this problem for the next year, what methods or techniques might you try in order to build on your work so far? Are there other fields or features you felt would have been very helpful to have?

We may try to use pseudo-labels in the dataset to replace the previously filtered mislabels. The model trained on the labeled data is used to make predictions on the unlabeled data, and the unlabeled samples are filtered based on the prediction results and fed into the model again for training to fill the gap caused by the previous removal of mislabeled data and to enhance the generalization of the model.