Model documentation and write-up

1. Who are you (mini-bio) and what do you do professionally? If you are on a team, please complete this block for each member of the team.

I (qqggg, Qixun Qu in real name) am a vision algorithem developer and focus on image and signal analysis.

I (Hongwei Fan) am PhD student affiliated with the Data Science Institute,Imperial College London. Before my PhD, I completed an Master degree in biomedical engineering at Tsinghua University and worked as an algorithm engineer at SenseTime.

2. What motivated you to compete in this challenge?

As our team name **Just4Fun**, we are excited to solve more interesting problem than what we did at work to have fun.

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

We defined this task as a temporal-spatial regression problem.Thus, we applied a 3D neural network to make regression.

(1) S1 and S2 features and AGBM labels were carefully preprocessed according to statistics of training data. Outliers were replaced by the lower or upper limitations.

(2) Training data was splited into 5 folds for cross validation.

(3) Processed S1 and S2 features were concatenated to 3D tensor in shape [B, 15, 12, 256, 256] as input, targets were AGBM labels in shape [B, 1, 256, 256].

(4) Some operations, including horizontal flipping, vertical flipping and random rotation in 90 degrees, were used as data augmentation on 3D features [12, 256, 256] and 2D labels [256, 256].

(4) We applied Swin UNETR with the attention from Swin Transformer V2 as the regression model. Swin UNETR was adapted from the implementation by MONAI project.

(5) In training steps, Swin UNETR was optimized by the sum of weighted MAE and SSIM Loss. RMSE of validation data was used to select the best model.

(6) We trained Swin UNETR using 5 folds, and got 5 models.

(7) For each testing sample, the average of 5 predictions was the final result.

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

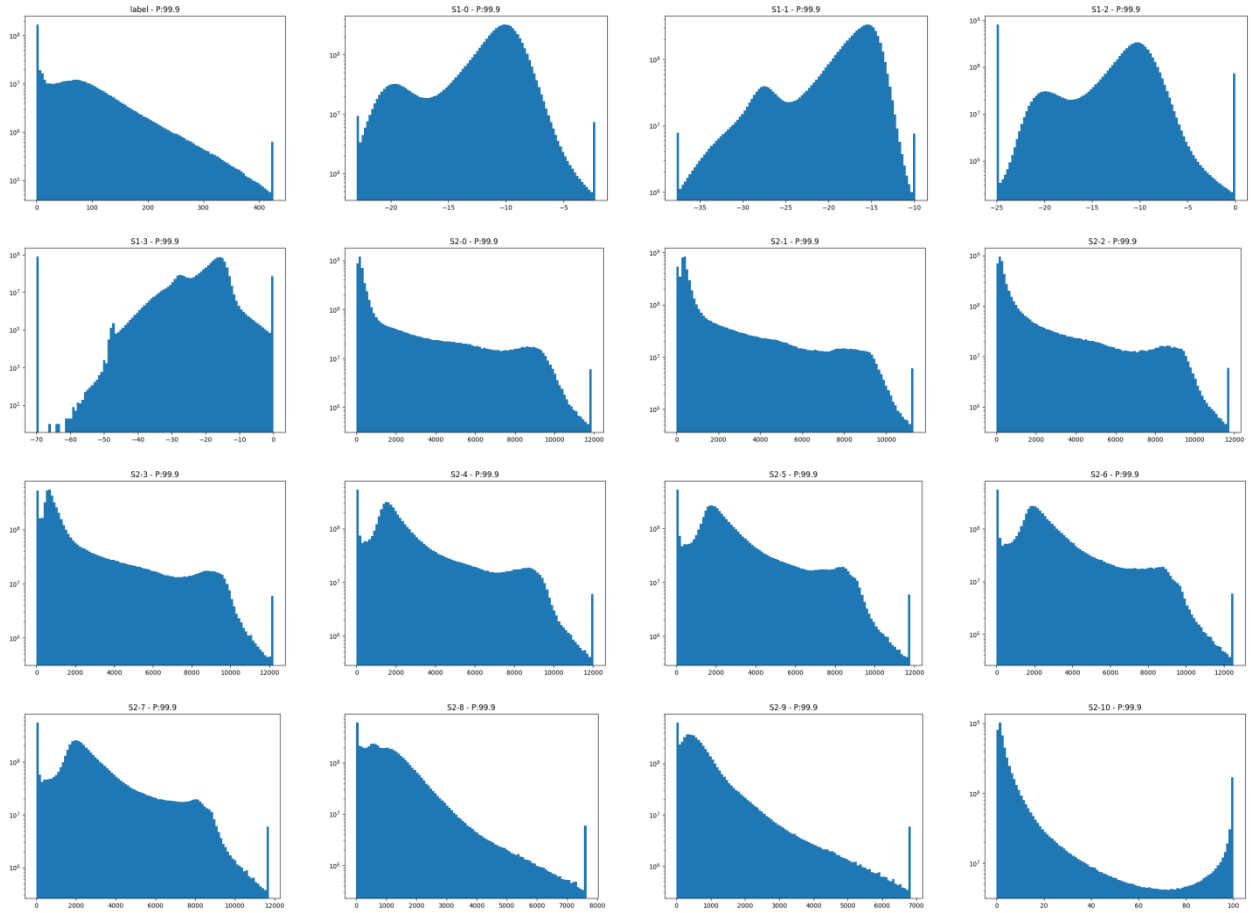

Figure 1. Distribution of Preprocessed Feartures and Label. In preprocessing step, we manually set reasonable range for each kind of data. Outliers were replaced by the lower or upper limitation. Then, we calculated statistics, such as mean, std, min, max etc. of every feature and label for normalization. Thus, training process coubld be more stable.

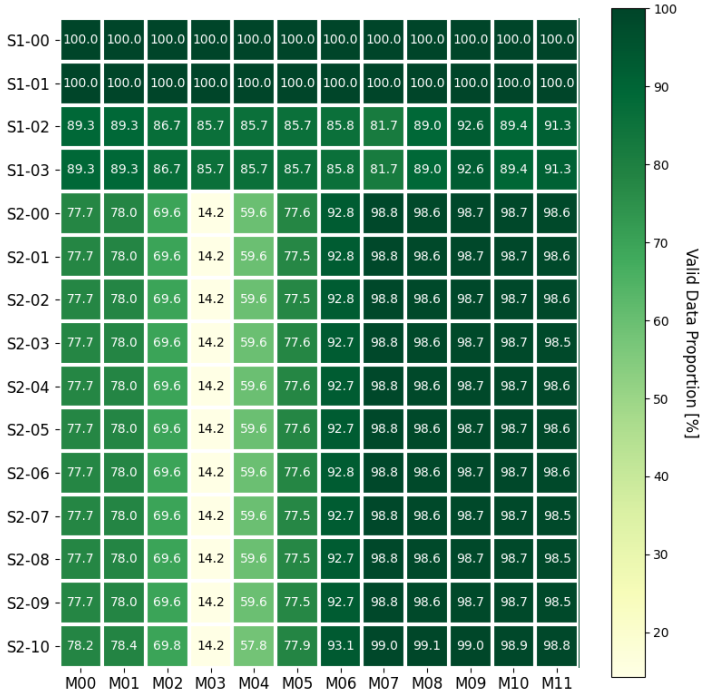

Figure 2. Valida Data Proportion of Each Feature and Label at Each Month. After removing outliers from preprocessed dataset, there were still some invalid data, sunch as NaN or missing data. We took a look at the proportion of valid values of each feature and label at each month, to see if some features or some months could be more important than others.

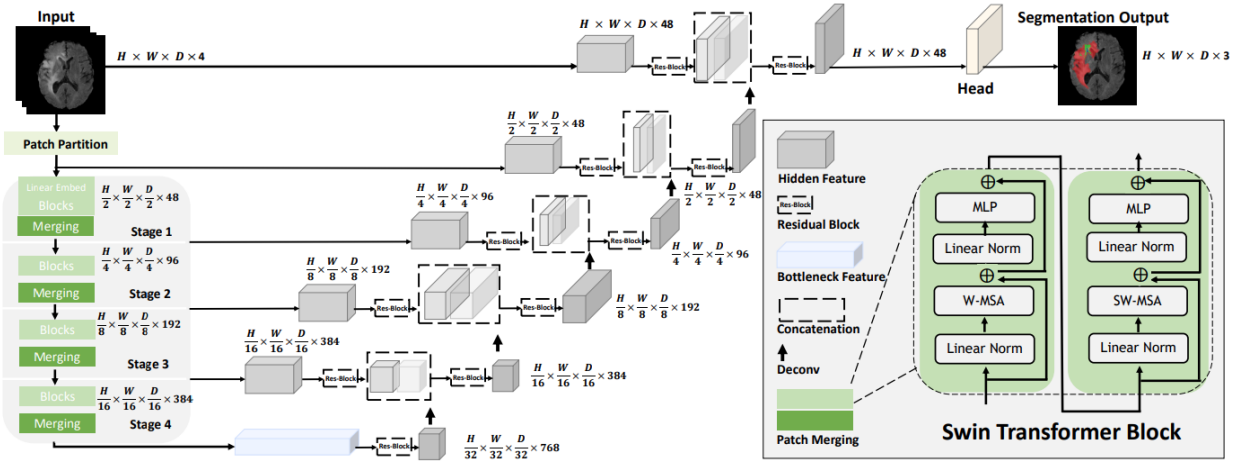

Figure 3. The Structure of Swin UNETR. This figure was directly copied from the paper of Swin UNETR (Figure 1 in https://arxiv.org/abs/2201.01266). Swin UNETR is a UNet-like model with encoder-decoder for semantic segmentation. The encoder of original Swin UNETR is Swin Transformer V1. We replaced encoder with Swin Transformer V2. We also modified the Head layer to output 2D result without depth channel and with only one color channel.

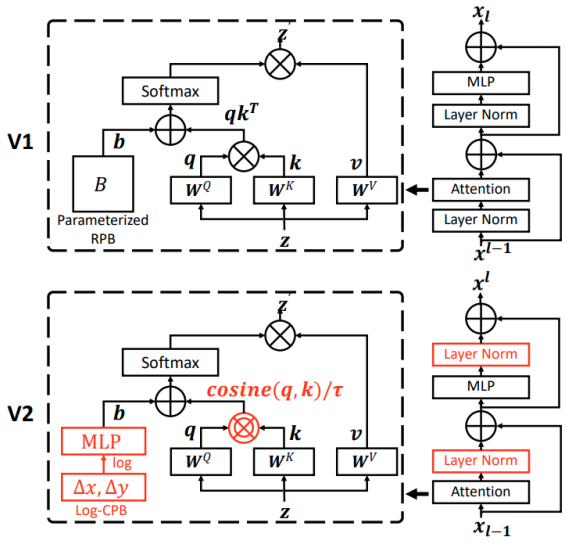

Figure 4. Difference between V1 attentoin and V2 attention in Swin Transformer. This figure was copied from the paper of Swin Transformer V2 (https://arxiv.org/abs/2111.09883). Please read the paper for more details.

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

Preprocessing on Features and Labels

```
INLIERS_PLAIN = {
    'S1': {
        0: {'min': -25.0, 'max': 30.0},
        1: {'min': -63.0, 'max': 29.0},
        2: {'min': -25.0, 'max': 32.0},
        3: {'min': -70.0, 'max': 23.0},
    },
    'S2': {
        10: {'min': 0.0, 'max': 100.0},
    }
}


def remove_outliers_by_plain(data, data_name, data_index=None):

    if data_name == 'label':
        data = np.where(data < 0.0, 0.0, data)

    elif data_name == 'S1':
        inlier_dict = INLIERS_PLAIN['S1'][data_index]
        min_ = inlier_dict['min']
        max_ = inlier_dict['max']
        data = np.where(data < min_, min_, data)
        data = np.where(data > max_, max_, data)

    elif data_name == 'S2':
        if data_index == 10:
            inlier_dict = INLIERS_PLAIN['S2'][10]
            min_ = inlier_dict['min']
            max_ = inlier_dict['max']
            data = np.where(data < min_, min_, data)
            data = np.where(data > max_, min_, data)

    return data
```

Code 1. Preprocessing on Features and Label. In preprocessing step, we manually set reasonable range for each kind of data. Outliers were replaced by the lower or upper limitation.

Data Augmentation on Features and Labels

```python
class BMDataset(Dataset):

    def __init__(self, augment=True, ...):
        super(BMDataset, self).__init__()

        ...

        self.augment = augment

        if self.augment:
            self.transform = V.Compose([
                V.Flip(1, p=0.1),
                V.Flip(2, p=0.1),
                V.RandomRotate90((1, 2), p=0.1)
            ], p=1.0)

        ...

    def __getitem__(self, index):

        subject_path = self.data_list[index]
        label, feature = self._load_data(subject_path)
        # label:   (1, 256, 256, 1)
        # feature: (M, 256, 256, F)

        if self.augment:
            data = {'image': feature, 'mask': label}
            aug_data = self.transform(**data)
            feature, label = aug_data['image'], aug_data['mask']
            if label.shape[0] > 1:
                label = label[:1]

        feature = feature.transpose(3, 0, 1, 2).astype(np.float32)
        # feature: (F, M, 256, 256)
        label = label[0].transpose(2, 0, 1).astype(np.float32)
        # label: (1, 256, 256)

        return feature, label
```

Code 2. Data Augmentation on Features and Label. Some operations, including horizontal flipping, vertical flipping and random rotation in 90 degrees, were used as data augmentation on 3D features [12, 256, 256] and 2D labels [256, 256].

Structure Similarity Loss

```python
from piqa import SSIM

class SimLoss(nn.Module):

    def __init__(self, mode='ssim', weight=1.0):
        super().__init__()

        self.weight = weight

        if mode == 'ssim':
            self.sim_func = SSIM(window_size=11, sigma=1.5, n_channels=1)
        else:
            raise ValueError('unknown sim loss mode')

    def forward(self, pred, label):
        similarity = self.sim_func(pred, label)
        sim_loss = 1.0 - similarity
        return sim_loss * self.weight
```

Code 3. Structure Similarity Loss. The sum of weighted MAE and structure similarity loss was used to optimize Swin UNETR.

6. Please provide the machine specs and time you used to run your model.
   - CPU (model): Intel® Xeon® E-2378G Processor
   - GPU (model or N/A): A100 with 40GB
   - Memory (GB): 128
   - OS: Ubuntu 20.04
   - Train duration: 7 to 8 days using single A100
   - Inference duration: 30 minutes using single A100 on public testing dataset

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.)?

The preprocessing step will take huge RAM usage about 80Gb. You don't have to run the script for prerpocessing again if you don't have a large RAM. All outputs of preprocessing step can be found in code submission.

If you have 5 GPUs, you can run each fold training on each GPU, and it will take less than 2 days.

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

   No, we didn't.

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

   We only used RMSE as the evaluation metric.

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

(1) We tried to add focal frequency loss (https://arxiv.org/abs/2012.12821) as a regression constraint to optimize Swin UNETR. But it didn't improve the evaluation metric.

(2) We tried to train VT-Unet (https://arxiv.org/abs/2111.13300) but got much worse results on local cv. VT-Unet is also a 3D UNet-like semantice segmentation model, whose both enconder and decoder are Swin Transformer.

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?

First of all, we will reimplement the statistics calculation using a two-pass method, to extreamly reduce the memory usage.

Our current approach has the potential to improve if we tuned hyper-parameters more carefully.

We will try some other 3D transformer based regression models, and try to apply some self-superwised techs to get pre-trained models.