III. Model documentation and write-up

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

Biochemist who switched to computational work when I heard about CUDA in 2007. It seemed like it would be an important technology. Spent a few years as a research professor at Vanderbilt University leading a group developing novel, GPU-accelerated ML techniques for computer-aided drug design. I currently serve as the VP of Data Science at Lose It!, and as a user advisor to the TensorFlow team at Google.

2. What motivated you to compete in this challenge?

I had the week off from work and decided to use that time to learn pytorch for segmentation. I came across the blog post for the competition, and it seemed like a great place to start. I have been a prize winner in a couple of other DrivenData competitions, and was already familiar with the great work they do.

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

My focus was really around building a generalizable ensemble. In my previous work in these types of competitions, I've found that my cross validation models have suffered from hold-out variance issues. I tried to identify chip ids that contributed to this variation and kept them in the training folds during all cv rounds. I figured if the other chips could not generalize to those "outliers", then it made sense to just keep them in the training sets and not try to use them for early stopping. I've also found that using the 5-fold cv models as an ensemble or jury typically results in better generalization versus trying to do a final fit using the hyperparameters discovered through the cross validation process. The other thing was to increase the types of augmentations during training.

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

No

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

A. Adding these transformations/augmentations improved the cv score and generalization

```
original_height = original_width = 512
training_transformations = A.Compose(
    [
#         A.RandomCrop(256, 256),
#         A.ShiftScaleRotate(p=.9),
        A.OneOf([
            A.RandomSizedCrop(min_max_height=(int(0.5 * original_height),int(0.9 *
original_height)), height=original_height, width=original_width, p=0.5),
            A.PadIfNeeded(min_height=original_height, min_width=original_width, p=0.5)
        ], p=1),
        A.Transpose(p=0.5),
        A.MaskDropout(p=0.5),
        A.VerticalFlip(p=0.5),
        A.HorizontalFlip(p=0.5),
```

```
        A.RandomRotate90(p=0.8),
        A.OneOf([
            A.MotionBlur(p=0.2),
            A.MedianBlur(blur_limit=3, p=0.1),
            A.Blur(blur_limit=3, p=0.1),
        ], p=0.2),
        A.OneOf([
            A.ElasticTransform(alpha=120, sigma=120 * 0.05, alpha_affine=120 * 0.03, p=0.5,
interpolation=cv2.INTER_NEAREST),
            A.GridDistortion(p=0.5, interpolation=cv2.INTER_NEAREST),
            A.RandomGridShuffle(p=0.5),
            A.OpticalDistortion(distort_limit=2, shift_limit=0.5, p=1,
interpolation=cv2.INTER_NEAREST)
            ], p=0.8),
        A.RandomBrightnessContrast(contrast_limit=0.1, brightness_by_max=False),
        A.Resize(256,256)
    ]

)
```

B. Changing the loss function to deal with pixel class imbalance

```
import torch.nn.functional as F

#PyTorch
ALPHA = 0.5
BETA = 0.5
GAMMA = 1

class FocalTverskyLoss(torch.nn.Module):
    def __init__(self, weight=None, size_average=True):
        super(FocalTverskyLoss, self).__init__()

    def forward(self, inputs, targets, smooth=1, alpha=ALPHA, beta=BETA,
gamma=GAMMA):

        #comment out if your model contains a sigmoid or equivalent activation layer
        inputs = torch.softmax(inputs, dim=1)[:, 1] #F.sigmoid(inputs)

        #flatten label and prediction tensors
#         inputs = inputs.view(-1)
#         targets = targets.view(-1)
        valid_pixel_mask = targets.ne(255)
        inputs = inputs.masked_select(valid_pixel_mask)
        targets = targets.masked_select(valid_pixel_mask)

        #True Positives, False Positives & False Negatives
        TP = (inputs * targets).sum()
        FP = ((1-targets) * inputs).sum()
        FN = (targets * (1-inputs)).sum()
```

```
    Tversky = (TP + smooth) / (TP + alpha*FP + beta*FN + smooth)
    FocalTversky = (1 - Tversky)**gamma

    return FocalTversky
```

C. Using mean/std scaling instead of min/max normalization improved cv scores

```
def std(self, arr, mean, std):
    return ( arr - mean) / std
```

6. Please provide the machine specs and time you used to run your model.
   - CPU (model): GCE c2-standard-30 instance type
   - GPU (model or N/A): A100
   - Memory (GB): 120GB
   - OS: deeplearning-platform-release vm image
   - Train duration: ~4hrs
   - Inference duration:

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.)?
   No

8. Did you use any tools for data preparation or exploratory data analysis that aren't listed in your code submission?
   No

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

10. What are some other things you tried that didn't necessarily make it into the final workflow (quick overview)?
    I was looking forward to trying pseudolabeling / semi-supervised learning using the test set but ran out of time. I was looking into ST++ for that.

    I also didn't do any hyperparameter optimization. I was planning to set up a bayesian optimization using the tool guild.ai, but didn't have time. It's a great tool for these types of tasks.

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?

    I didn't have time to look into where the misclassifications were occurring or what might be causing them. I think understanding that would have helped tremendously, but I had very little time to work on this problem unfortunately.

I was also trying to think of a way to better capture the importance of relative elevation differences specifically around areas of permanent water where flooding might be likely to occur. Didn't have time to flesh that out - not sure if that's an architecture/loss change or maybe feature engineering work :shrug: