III. Model documentation and write-up

Information included in this section may be shared publicly with challenge results. You can respond to these questions in an e-mail or as an attached file. Please number your responses.

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.

   We are a group within Team Epoch, a student organization of the Technical University of Delft. https://teamepoch.ai/team. We take a gap year to participate in AI competitions and projects, and organize and attend events. The team completely cycles every academic year, so the current members have been here for half a year. Five of us worked on Kelp Wanted. See study background and LinkedIn in the winner's blog post.

2. What motivated you to compete in this challenge?

   We look for AI competitions that contribute to the UN SDGs, and have a timeframe of 2~3 months. At the time of selecting competitions, this was the most attractive in terms of sustainability, image segmentation being a new type of challenge for this team, and having a topic that would be easy to explain and visualize at events.

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

   See technical report on GitHub
   https://github.com/TeamEpochGithub/iv-q2-detect-kelp/blob/main/Detect_Kelp___Technical_Report.pdf

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

   See technical report on GitHub
   https://github.com/TeamEpochGithub/iv-q2-detect-kelp/blob/main/Detect_Kelp___Technical_Report.pdf

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

   We estimate the most impactful parts of our solution to be the choice of architectures, features, augmentations, and the concept of ensembling diverse models. This is all described in the report.

   As for specific code snippets that improved training, we can list the following:

Data Augmentation

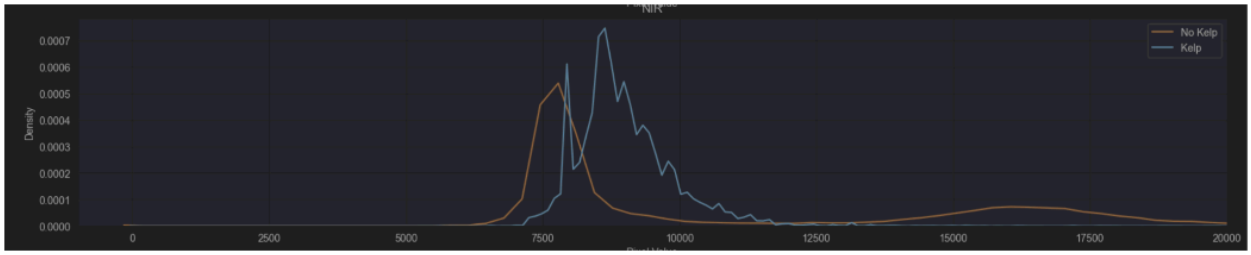
We applied various data augmentation techniques. On the right you are able to see our transformations applied to each batch for training all of our models. By applying manual hyperparameter tuning, we found that these were the optimal values for us.

Firstly, we always apply a random rotation by 90 degrees, followed by a random horizontal flip with 50% probability and a vertical flip with 50% probability. Then we apply with 20% probability either a mosaic augmentation[1], or a cutmix augmentation. These augmentations ensured robustness and reduced overfitting during training, improving our models significantly.

```yaml
# Augmentations
transformations:
  _target_: src.augmentations.transformations.Transformations
  alb:
    _target_: albumentations.Compose
    _args_:
      - _target_: albumentations.RandomRotate90
        p: 1
  korn:
    _target_: kornia.augmentation.AugmentationSequential
    _args_:
      - _target_: kornia.augmentation.RandomHorizontalFlip
        p: 0.5
      - _target_: kornia.augmentation.RandomVerticalFlip
        p: 0.5

      # Randomly apply one of the following mix augmentations
      - _target_: kornia.augmentation.AugmentationSequential
        random_apply: 1
        _args_:
          - _target_: kornia.augmentation.RandomMosaic
            p: 0.2
            output_size:
              - 350
              - 350
          - _target_: kornia.augmentation.RandomCutMixV2
            p: 0.2
          - _target_: kornia.augmentation.Resize # acts as a no-op
            p: 0.6
            size:
              - 350
              - 350
```

***Feature engineering**:*



We spend a lot of effort creating meaningful varying features from the 7 raw original channels of the data. From our exploratory data analysis located in `notebooks/eda-investigations.ipynb` *we encountered that* `NIR` *is a very meaningful feature looking at the distribution of values for kelp and no kelp. Therefore, we focussed on NIR related spectral features and added NDVI, NDWI, ONIR, ODVI, MNDWI, LandCloseness. Furthermore, we added convolutional features that also increased our score applied on our most important feature NDVI. Edge detection (Sobel), Sharpening, Contrast Enhancement, and Modal filters were also added to enhance different parts of the NDVI that also improved our results. All the features selected for a model are specified in our modular configuration pipeline for each model. These can be found under the section* `feature_pipeline:` *in our* `conf/model/*.yaml` *files.*

Test Time Augmentations

During inference we applied TT8 using rotations and flips. We made predictions on mirrored and rotated versions of input images, then re-aligned the predictions and averaged them. This removed any asymmetry from our models and gave us an improvement.

```
# forward pass
if self.self_ensemble:
    predictions = []
    for flip in [False, True]:
        for rotation in range(4):
            # Transform the batch
            X_batch_transformed = transform_batch(X_batch.clone(), rotation, flip=flip)

            # Get prediction
            y_pred_transformed = self.model(X_batch_transformed)

            # Reverse the transformation on prediction
            y_pred_reversed = reverse_transform(y_pred_transformed, rotation, flip=flip)

            # Collect the predictions
            predictions.append(y_pred_reversed)

    # Average the predictions
    y_pred = torch.mean(torch.stack(predictions), dim=0).cpu().numpy()
else:
    y_pred = self.model(X_batch).cpu().numpy()
```

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

    - *CPU: AMD Ryzen Threadripper Pro 3945WX 12-Core Processor / AMD Ryzen 9 7950X 16-Core Processor*
    - *GPU: NVIDIA RTX A5000 / NVIDIA RTX Quadro 6000 / NVIDIA RTX A6000*
    - *RAM: 96GB / 128GB*
    - *OS: Windows 10/11*
    - *Python: 3.10.13*
    - *Train duration: ~3-6h per model (with all data in memory)*
    - *Inference duration: 5-10m per model with 8x test time augmentations*

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.)?

    We fixed seeds for all our splits and model setups, this was to ensure consistency during training so that we could investigate if a change would make an improvement or not. Standard seed was set to 42.

During training, in the train.yaml config file cache_size is used to move the chosen number of data samples to memory. If it is set to -1 all data is loaded to memory (This can lead to using the disk as virtual memory and slow down the initial data loading stage due to concatenation of the features). The data that is not in memory will be read only when it is needed by the model and removed from memory afterwards. Not keeping the data in memory will result in slower training, but lower memory usage.

When retraining a model, if a model with the calculated model hash already exists in the 'tm*' folder, training will be skipped. There are also parameters in the model config that do not affect the model hash. These parameters are 'self_esnemble' (Boolean to enable TTA) and 'saved at' for the GBDT step (Changing the GBDT model path does not change the hash but has an impact on the model)*

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

   All EDA done for the competition can be found under the `notebooks` *folder in our GitHub repository. You can find the link to our repository here:* *https://github.com/TeamEpochGithub/iv-q2-detect-kelp.*

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

   Loss functions other than dice have been used, however the final evaluation was always done using dice score. During development we did 5-fold CV to validate our models in which we used dice score as our evaluation metric, as the exact same metric on DrivenData.

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

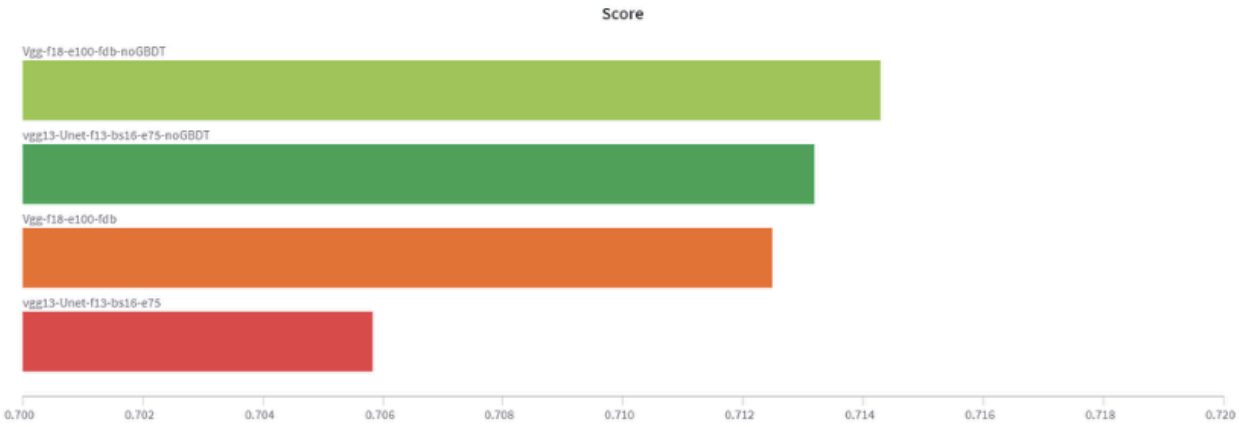
    For this question, please take a look at out technical report at the section: What didn't work *https://github.com/TeamEpochGithub/iv-q2-detect-kelp/blob/main/Detect_Kelp___Technical_Report.pdf*

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?

    Improve data quality: We have seen many images in the dataset that had very poor label quality. We found images that were mislabelled and images that were misaligned. Images where waves were labeled as kelp, kelp labels that were clearly offset from the real kelp patch. These can be quickly found by sorting on images where the model makes the most misclassifications according to the label. We have a notebook that analyzes model predictions to find on which images it has made the highest errors. This can be found in `notebooks/analyse_preds.ipynb`*.*

    Remove the GBDT model from model features: We have seen after the competition deadline was over that this feature actually makes the better models worse. Our VGG-f18-e100 model

was one of the best individual performing models included in our best ensemble, and when we applied a small ablation study we figured out that removing GBDT (XGBoost) as a feature gave us a significant local improvement which we would also expect on public and private. We would continue this ablation study if we have more time to find the optimal set of features.



Other approaches listed under 'What we haven't tried' in our technical report. This contains interesting angles that we think would give us an improvement if investigated further. See technical report on GitHub.
https://github.com/TeamEpochGithub/iv-q2-detect-kelp/blob/main/Detect_Kelp___Technical_Report.pdf

12. What simplifications could be made to run your solution faster without sacrificing significant accuracy?

    Using fewer models in an ensemble: We are most confident in our highest public score submission which includes 12 models (this submission scores higher than 2nd place in private score) but a submission with an ensemble of 3 models had the highest private score with a lower public score.

    Removing TTA: TTA results in a small improvement in the score at the cost of 8x inference times.

    Using DiceLoss instead of FocalDiceBoundary. Training times are around ~1.8x faster using normal DiceLoss instead of the more complex FocalDiceBoundary loss while only giving a minor improvement.