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.
   [redacted]
   I teach computer programming, data science and software engineering courses. My research activities and publications are currently focused on solving computer vision and data science problems in marine biology and other applied/industry projects.

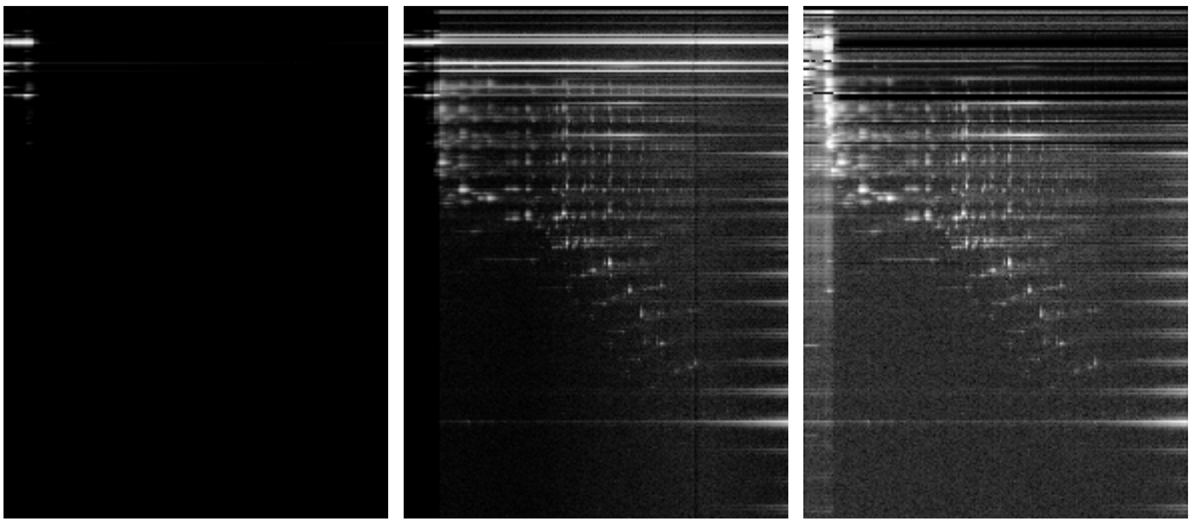
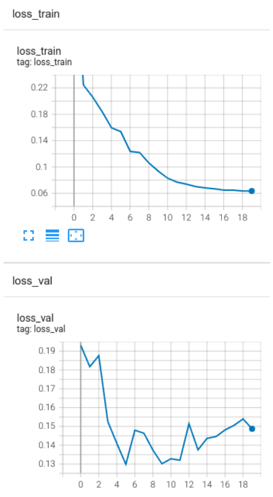
2. What motivated you to compete in this challenge? I compete regularly in various competitions and am currently a Kaggle competitions expert (https://www.kaggle.com/dmitrykonovalov). However, this is my first prize-winning finish on DrivenData. I mainly select competitions which are directly relevant to my research activities to learn and practice state-of-the-art solutions. In the predecessor Mars-1 competition, the first-place solution was entirely computer vision based and very elegant, which, together with the interesting dataset, inspired me to join this challenge.

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

   - I followed the first-place solution of the previous Mars challenge by converting the mass spectrometry data into 2D images. Various CNN models and data-processing configurations were ensembled for the final 2nd-place winning submission.
   - I experimented with different conversion configurations and found that the first 256 mass values (y-axis) were sufficient. For the time axis (x-axis), 192 time slots were selected as a reasonable baseline value, where larger values slowed down the CNN training.
   - In this competition, time was only a proxy for temperature. Therefore, I only explored ideas where exact dependence on time values was not required. That led to two key ideas, which I think pushed my solution to the winning range.
   - First, when training a CNN, the time dimension was randomly batch-wise resized on GPU within the 128-256 range of values. Then, at the inference phase, the corresponding TTA (test-time-augmentation) was done by averaging time sizes (5 steps of 32, centred at 192).
   - The second win-contributing idea, I think, was creating a different classifier head, where only the time dimension of the CNN backbone 2D features was averaged (rather than both the mas and time dimensions) before the last fully connected liner layer.
   - An extensive search of timm pre-trained models (https://github.com/rwightman/pytorch-image-models) found HRNet-w64 to be particularly accurate for the considered 2D representation of data.
   - Small but consistent improvement was gained by encoding the “derivatized” column as a 2-channel image and adding a trainable conversion layer before a CNN backbone.
   - I was not able to achieve any consistent validation loss improvement by adding noise or smoothing/preprocessing original data at the training and/or inference stages. Hence, all training and inference were done with the original data converted to 2D images.
   - For ensembling, averaging logits (inverted clipped sigmoids) rather than probabilities improved both validation and private LB results.

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

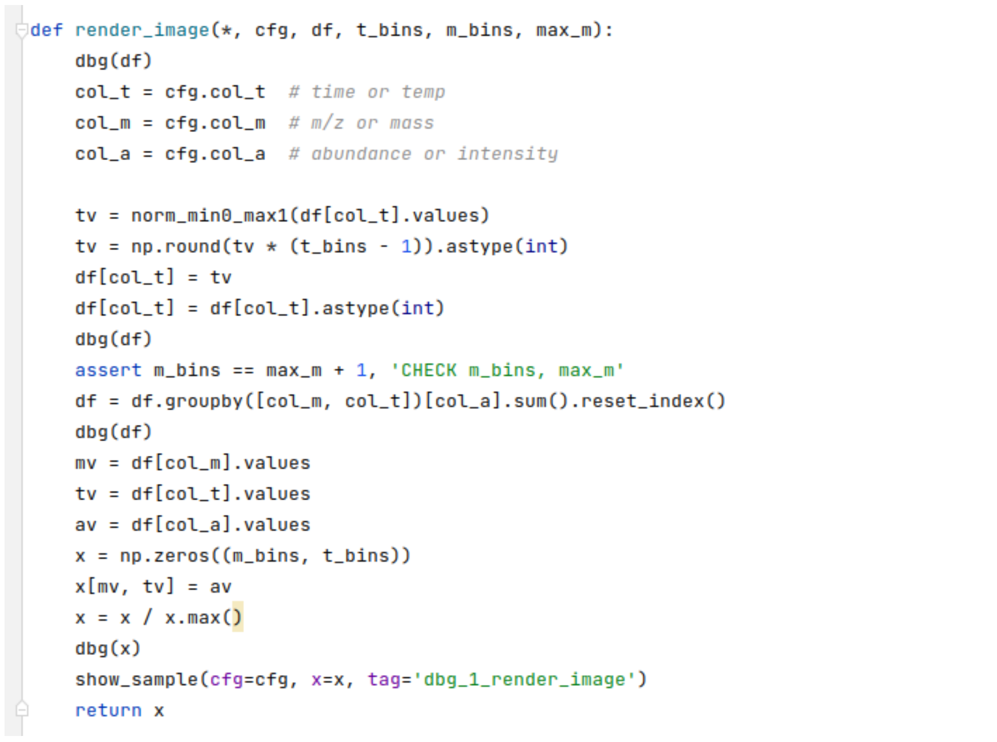
   - The conversion of a mass spectrogram to a 2D image is not unique and could be done in a variety of ways. Experimenting with possible conversions, the following three configurations appeared to be complementary. The CNN models trained on each of the three 2D representations improved oof (out-of-fold) predictions when ensembled together.
   - They were: a) logarithmic scaling; b) division by maximum row values (time-normalization) followed by the log-scaling c) division by maximum column values (mass-normalization) followed by the log-scaling. The following three images are the corresponding 2D (256x192) conversions for sample ‘S0592’:

- 
- All models were trained for 20 epochs with a cosine learning rate schedule and linear 2-epoch warmup.
- In the majority of cases, every training run (over 20 epochs) overfitted a CNN model, where the validation loss started to grow (second subfigure below) while the training loss was still decreasing (first subfigure below).
- 

- Therefore, only the best-val-loss model was retained from each run and used for TTA inference and subsequent ensembling.

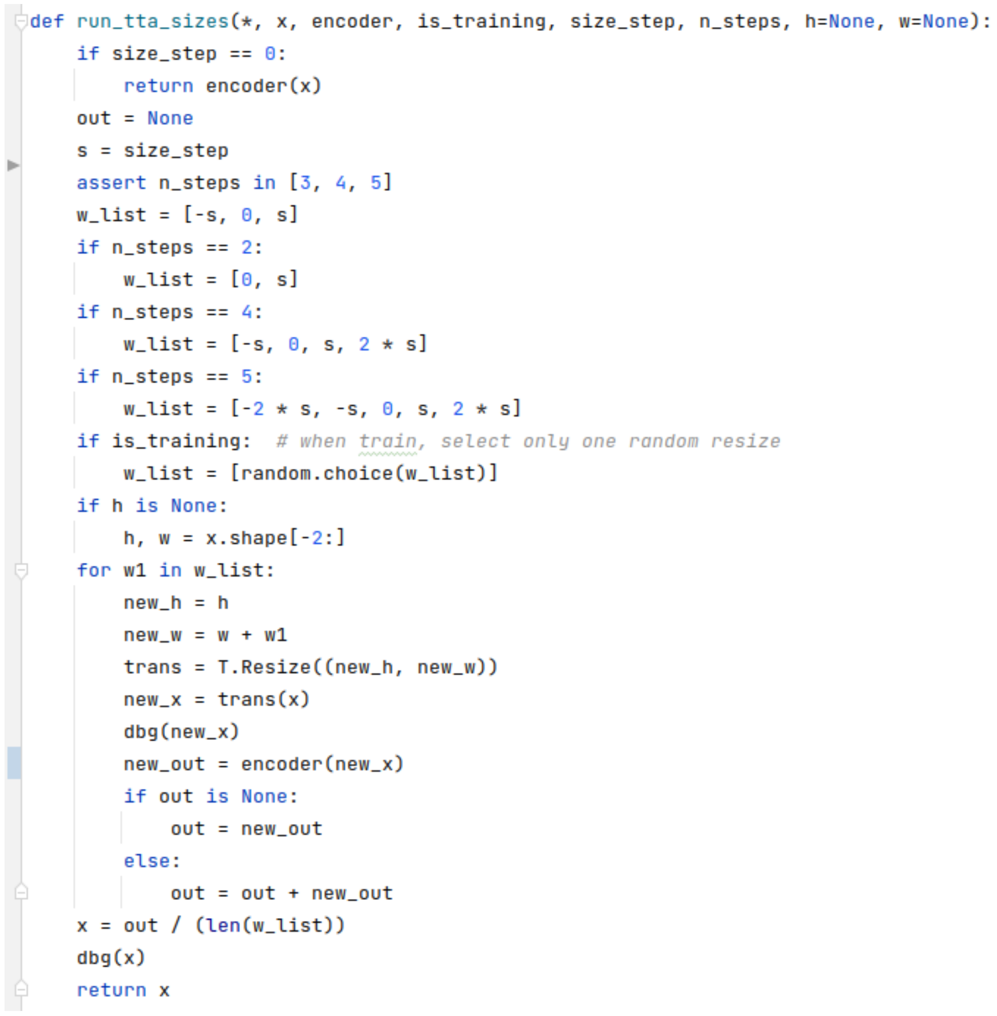
5. Copy and paste the 3 most impactful parts of your code and explain what each does and how it helped your model.

    - Below is my conversion to a 2D image. It works equally well for small or large numbers of time bins. If the number of time bins is small, the pandas' sum() will add multiple intensity values into the same mass-time-pair location.



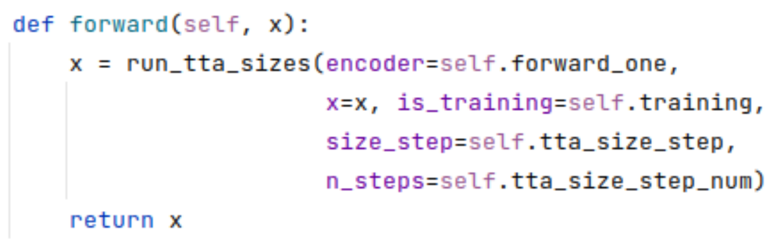
```python
def render_image(*, cfg, df, t_bins, m_bins, max_m):
    dbg(df)
    col_t = cfg.col_t  # time or temp
    col_m = cfg.col_m  # m/z or mass
    col_a = cfg.col_a  # abundance or intensity

    tv = norm_min0_max1(df[col_t].values)
    tv = np.round(tv * (t_bins - 1)).astype(int)
    df[col_t] = tv
    df[col_t] = df[col_t].astype(int)
    dbg(df)
    assert m_bins == max_m + 1, 'CHECK m_bins, max_m'
    df = df.groupby([col_m, col_t])[col_a].sum().reset_index()
    dbg(df)
    mv = df[col_m].values
    tv = df[col_t].values
    av = df[col_a].values
    x = np.zeros((m_bins, t_bins))
    x[mv, tv] = av
    x = x / x.max()
    dbg(x)
    show_sample(cfg=cfg, x=x, tag='dbg_1_render_image')
    return x
```

-
- The next code section is my TTA, where size_step=32 was used.



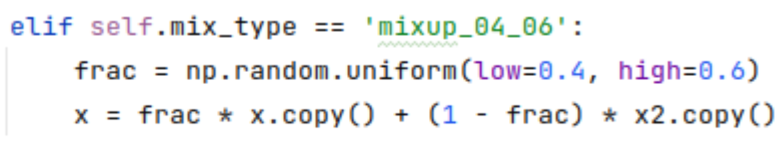
```python
def run_tta_sizes(*, x, encoder, is_training, size_step, n_steps, h=None, w=None):
    if size_step == 0:
        return encoder(x)
    out = None
    s = size_step
    assert n_steps in [3, 4, 5]
    w_list = [-s, 0, s]
    if n_steps == 2:
        w_list = [0, s]
    if n_steps == 4:
        w_list = [-s, 0, s, 2 * s]
    if n_steps == 5:
        w_list = [-2 * s, -s, 0, s, 2 * s]
    if is_training:  # when train, select only one random resize
        w_list = [random.choice(w_list)]
    if h is None:
        h, w = x.shape[-2:]
    for w1 in w_list:
        new_h = h
        new_w = w + w1
        trans = T.Resize((new_h, new_w))
        new_x = trans(x)
        dbg(new_x)
        new_out = encoder(new_x)
        if out is None:
            out = new_out
        else:
            out = out + new_out
    x = out / (len(w_list))
    dbg(x)
    return x
```

-
- It could be easily added to any torch Module/Model making it TTA-able, where the model's original 'forward' method needs to be renamed to e.g. 'forward_one', see below



```python
def forward(self, x):
    x = run_tta_sizes(encoder=self.forward_one,
                      x=x, is_training=self.training,
                      size_step=self.tta_size_step,
                      n_steps=self.tta_size_step_num)
    return x
```

-

- The third code section is my mixup, where two samples were mixed with a fraction randomly varied between 0.4 and 0.6. Surprisingly, but consistent with the Mars1-1st-place-solution, the mixup probability (mix_aug_prob) needed to be relatively small or it degraded the oof performance, where in some run configurations mix_aug_prob=0.1 was used.
- 

```
elif self.mix_type == 'mixup_04_06':
    frac = np.random.uniform(low=0.4, high=0.6)
    x = frac * x.copy() + (1 - frac) * x2.copy()
```

6. Please provide the machine specs and time you used to run your model.
   - CPU (model):
     Intel(R) Core(TM) i7-8700 CPU @ 3.20GHz;
     Intel(R) Core(TM) i9-7980XE CPU @ 2.60GHz;
     Intel(R) Core(TM) i7-9700K CPU @ 3.60GHz
   - GPU (model or N/A):
     2 x NVIDIA GeForce GTX 1080 Ti;
     2 x NVIDIA GeForce RTX 2080
   - Memory (GB): 32-64GB
   - OS: Ubuntu 20.04
   - Train duration: About 30 minutes per model-fold. To re-train all 100 model folds takes about 2 days on one GPU.
   - Inference duration:
     Less than one minute, as all TTA predictions are calculated at the end of each model-fold training.

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?
   I tried to monitor and save models with the best oof-accuracy rather than the best oof-loss. Those models did not improve oof-loss during ensembling.

10. What are some other things you tried that didn't necessarily make it into the final workflow (quick overview)?
    Tried transformers, LSTM, adding noise, model and data dropouts, and pseudo-labelling.

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?
    It seems that models' diversity is the most important contributor to improving the competition metric. Therefore, the table-based methods (lightgbm etc), LSTM-like and 1D-transformers should be added.