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?
   Kareem Eissa: *I am a Senior Research Engineer at Visual and AI Solutions (a Nile University spinoff) with experience in the fields of Natural Language Processing, Computer Vision, and Deep Learning. I have a master's degree in Informatics from Nile University (Egypt) during which I did a research internship at Siemens Healthineers in Princeton, USA. Over the course of my research, I contributed to three publications in top-tier venues and participated in several data science competitions.*

   Karim Amer: *I am a Co-founder and R&D Director at Visual and AI Solutions. I previously worked as a research assistant with the Ubiquitous and Visual Computing Group at Nile University, where I published several papers about satellite image analysis between 2016 and 2018. In 2019, I became a research intern at Siemens Healthineers Technology Center, NJ, USA where I worked on development of cutting edge segmentation models that can be used in multiple clinical applications.*

2. What motivated you to compete in this challenge?
   We enjoy participating in data science competitions. The problem of time series forecasting the Disturbance Storm Index is intriguing, practical, and challenging.

3. High level summary of your approach: what did you do and why?
   We preprocessed the "solar_wind" data by aggregating hourly features: mean, std, min, max, and median in addition to the first and last-minute features and the difference between them (gradient). We also added the daily ACE satellite positions to our list of features. We utilize the latest 96 hours (4 days) in our model to predict the following 2 hours.

   Our model is a 4-block deep convolutional neural network. Each block has two consecutive convolution layers residually connected. The output of the convolutions is passed through Leaky ReLU non-linear activation function then maxpooling to reduce the sequence length by a factor of 2. Finally, a fully-connected layer projects the convolutional features to 2 outputs.

   We utilize a custom loss function:
   $$Loss = log((y-p)^2)^p + |y-p|$$
   where the logarithm is raised to a power P. This allows us to control over and under shooting of our loss function. We built an ensemble of models using different power parameters P and different seeds and averaged their predictions.

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

```
class Conv(nn.Module):
    def __init__(self, ni, no, kernel, stride, pad):
        super().__init__()
        self.layer1 = nn.Sequential(nn.Conv1d(ni, no, kernel, 1, pad), nn.LeakyReLU())
        self.layer2 = nn.Sequential(nn.Conv1d(no, no, kernel, 1, pad), nn.LeakyReLU())

    def forward(self, x):
        x1 = self.layer1(x)
        x2 = self.layer2(x1)
        return x1+x2

class ConvNet(nn.Module):
    def __init__(self, in_ch, kernel, n_outputs, p=0.1):
        super(ConvNet, self).__init__()
        pad = kernel//2
        self.conv1 = Conv(in_ch, 64, kernel, 2, pad)
        self.conv2 = Conv(64, 128, kernel, 2, pad)
        self.conv3 = Conv(128, 256, kernel, 2, pad)
        self.conv4 = Conv(256, 384, kernel, 2, pad)
        self.max_pool = nn.MaxPool1d(2, 2)
        self.dropout = nn.Dropout(p=p)
        self.fc1 = nn.Sequential(nn.Linear(384 + in_ch, 64), nn.LeakyReLU())
        self.linear = nn.Linear(64, n_outputs)

    def forward(self, x):
        x_last = x[:,:,-1]
        x = self.conv1(x)
        x = self.max_pool(x)
        x = self.dropout(x)
        x = self.conv2(x)
        x = self.max_pool(x)
        x = self.dropout(x)
        x = self.conv3(x)
        x = self.max_pool(x)
        x = self.dropout(x)
        x = self.conv4(x)
        x = self.max_pool(x)
        x = self.dropout(x)
        x = x.mean(-1)
        x = torch.cat([x, x_last], dim = 1)
        x = self.dropout(x)
        x = self.fc1(x)
        x = self.linear(x)
        x[:,1] += x[:,0]
        return x
```

- ***Architecture:*** *We developed a convolutional neural network with residual connections and temporal max pooling.*

```
def make_criterion(p):
    def Criterion(inp, targ):
        return (F.l1_loss(inp, targ, reduction='none') + (torch.log2((inp - targ)**2 + 1)**p)).mean()
    return Criterion
```

- ***Loss Function:*** *We developed a loss function that approximates the Root Mean Squared Error by taking the Mean of the Logarithm of the Squared Error. This loss is parameterized by a power P that can over or under shoot with respect to the RMSE. We also predict the* *first hour t1 and the difference between t2 and t1. Given the model predictions y1 and y2: t1 = y1; t2 = y1 + y2*

```
for s in [2, 3, 5, 7, 11, 13, 17]:
    for pwr in [1.5, 2.4, 2.5]:
        set_seed(s)
        model = ConvNet((ds.data.shape[1]-2), 7, 2, 0.1)
        model = model.to(device)
        criterion = make_criterion(pwr)
        model_w_arr = train_model_snapshot(model, criterion, metric, 0.001, dataloaders, dataset_sizes,
                                           device, num_cycles=1, num_epochs_per_cycle=6)
        model.load_state_dict(model_w_arr[0])
        model = model.cpu()
        model.eval()
        torch.save(model.state_dict(), MODEL_PATH / 'conv_{:d}_{:.1f}'.format(s, pwr))
```

- ***Ensemble:*** *We built a combination of 21 models using 3 different power parameters in our loss function each with 7 different random weight initializations. We take the mean of their outputs as our final prediction.*

5. Please provide the machine specs and time you used to run your model.
    - CPU (model): ***AMD® Ryzen 7 3700***
    - GPU (model or N/A): ***RTX 2080 Ti***
    - Memory (GB): ***32G***
    - OS: ***Ubuntu 18.04***
    - Train duration: ***40 mins***
    - ~~Inference duration:~~

6. What are some other things you tried that didn't necessarily make it into the final workflow (quick overview)?
    - *XGBoost Model*
    - *Feature Engineering*
    - *Recurrent Neural Models*
    - *Models with different time steps*
    - *Blending*

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

8. How did you evaluate performance of the model other than the provided metric, if at all?
    Two-stage Time-split K-fold cross validation

9. 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

10. Do you have any useful charts, graphs, or visualizations from the process?
    No

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 could explore more Time series analysis to extract seasonality information, having access to a longer history than the provided 7-day window for inference may improve the forecasting accuracy.