Model documentation and write-up

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

   - I work as an ML Researcher building new ML features for the recipe sharing platform using recipe texts and image data. Before that I worked as a Data Scientist. I graduated from Moscow Institute of Physics and Technology with a master degree in applied physics and mathematics.

   - I'm Machine Learning Researcher specialised in Computer Vision and Deep Learning. I very much enjoy facing new challenges, so I spend part of my spare time participating in Machine Learning and Data Science competitions.

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

   Our approach is based on two layer pipeline, where the first layer is a CNN image classifier and the second one a GBM model adding extra features to the first layer predictions.
   The first layer is trained over a stratified 8 k-folds split squeme. We average the predictions of four slightly different models to improve the accuracy.
   The second layer add features from location, polygon characteristics and neighbors. Two models, catboost and lightgbm, are blended to improve final accuracy.

   First layer CNNs

   First layer models are trained based on 8 k-folds cross validation scheme. For each k-fold train, one testset prediction is made, so the final testset prediction is the average of 8 models.

   We selected 4 models for our final solution:

   - Model A06:
     - Basemodel: DualPathNet92
     - Pretrained: imagenet + 5k
     - Inputs: 224x224; polygon buffer = 1; mask polygon = True; draw actual polygon = False
     - Train Augmentations: RandomRotation, RandomAffine, RandomCrop, RandomHorizontalFlip, RandomVerticalFlip, ColorJitter, RandomErasing
     - Batch size: 48
     - Optimizer: Adam(lr=0.0001)
     - Scheduler: ReduceLROnPlateau(factor=0.1, patience=2)
     - Early stopper: patience = 6
     - Performance (LocalCV|PublicLB|PrivateLB): **0.4315 | 0.3867 | 0.4066**

   - Model A10:
     - Basemodel: SE-ResNeXt101_32x4d
     - Pretrained: imagenet
     - Inputs: 224x224; polygon buffer = 1; mask polygon = True; draw actual polygon = False

    - Train Augmentations: RandomRotation, RandomAffine, RandomCrop, RandomHorizontalFlip, RandomVerticalFlip, ColorJitter, RandomErasing
    - Batch size: 48
    - Optimizer: Adam(lr=0.0001)
    - Scheduler: StepLR(gamma=0.1, step_size=8)
    - Early stopper: patience = 6
    - Performance (LocalCV|PublicLB|PrivateLB): **0.4402 | 0.3845 | 0.3964**

- Model A11a:
    - Basemodel: InceptionV4
    - Pretrained: imagenet + background
    - Inputs: 299x299; polygon buffer = 1; mask polygon = False; draw actual polygon = True
    - Train Augmentations: RandomRotation, RandomAffine, RandomCrop, RandomHorizontalFlip, RandomVerticalFlip, ColorJitter
    - Batch size: 48
    - Optimizer: Adam(lr=0.0001)
    - Scheduler: StepLR(gamma=0.1, step_size=8)
    - Early stopper: patience = 6
    - Performance (LocalCV|PublicLB|PrivateLB): **0.4499 | ? | ?**

- Model C06g: Same as A06, with the following differences:
    - Mixup: Alpha = 2.0
    - Scheduler: ReduceLROnPlateau(factor=0.1, patience=3)
    - Performance (LocalCV|PublicLB|PrivateLB): **0.4284 | 0.3932 | 0.4189**

Second level models - Catboost and Lightgbm

These models are used to classify roofs based on 1st level models predictions and other roof features, including:

- City name (1 feature)
- Polygon area (1 feature)
- Number of corners in a polygon (1 feature)
- Minimum internal angle in a polygon (1 feature)
- Latitude and longitude of a roof centroid (2 features)
- Number of closest neighbor roofs(max 8) per class with distance coefficient - `(0.001 / dist) ** 2`, where `dist` is the distance between a roof centroid in question and centroids of neighbor roofs (5 features)

All the features for both models are absolutely the same, so the only difference between these 2 models is the framework which was used to train them.

After ensembling predictions from these two models the final solution performance is(LocalCV|PublicLB|PrivateLB): **0.3567 | 0.3329 | 0.3543**

Please, note that blending these two models doesn’t improve the final score much. So just using one of the models predictions achieves a quite similar result and still by far the best result on the LB (0.3559 public LB for the catboost model).

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

   3.1 Training overfits quickly on this dataset. We include strong input augmentation to avoid this and train the model for longer.

```
args.update({
    'transformations': {
        'train': transforms.Compose([
            transforms.RandomRotation((0, 360), expand=True),
            transforms.Lambda(lambda x: trim(x)),
            transforms.Lambda(lambda x: transforms.CenterCrop((max(x.size),
max(x.size)))(x)),
            transforms.Resize(SIZE),
            transforms.RandomAffine(0, scale=(0.6, 1.1)),
            transforms.RandomCrop(SIZE, pad_if_needed=True),
            transforms.RandomHorizontalFlip(p=0.5),
            transforms.RandomVerticalFlip(p=0.5),
            transforms.ColorJitter(brightness=0.25, contrast=0.25,
saturation=0.25, hue=0.1),
            transforms.ToTensor(),
            transforms.RandomErasing(p=0.5, scale=(1 / 8, 1 / 4), ratio=(0.5, 2.0),
value=0, inplace=False),
            transforms.ToPILImage(),
        ]),
        'valid': transforms.Compose([
            transforms.Lambda(lambda x: trim(x)),
            transforms.Lambda(lambda x: transforms.CenterCrop((max(x.size),
max(x.size)))(x)),
            transforms.Resize(SIZE),
        ]),
        'fold': transforms.Compose([
            transforms.Lambda(lambda x: trim(x)),
            transforms.Lambda(lambda x: transforms.CenterCrop((max(x.size),
max(x.size)))(x)),
            transforms.Resize(SIZE),
        ]),
        'test': transforms.Compose([
            transforms.Lambda(lambda x: trim(x)),
            transforms.Lambda(lambda x: transforms.CenterCrop((max(x.size),
max(x.size)))(x)),
            transforms.Resize(SIZE),
        ]),
    }
})
```

   3.2 One of the models (C06g) uses mixup to improve generalization. We do it in batch during training for Categorical Cross Entropy loss:

```
bs = inputs.shape[0]
lam = torch.from_numpy(np.random.beta(self.mixup_alpha, self.mixup_alpha,
size=bs)).float()
new_bs = np.arange(bs)
```

```
np.random.shuffle(new_bs)
inputs = lam.view((bs, 1, 1, 1)) * inputs + (1 - lam.view((bs, 1, 1, 1))) *
inputs[new_bs]
labels[lam < 0.5] = labels[new_bs][lam < 0.5]
```

3.3 A piece of code that for each roof in the test set data frame finds closest 9 roofs from the training set and sums per class count with distant coefficient. Adding these features gives around 0.03 improvement to the model.

```
def add_centroid_dist_feature(train_df, test_df, num_classes=5):
    nB = np.array(list(zip(train_df.geometry.centroid.x.values,
                           train_df.geometry.centroid.y.values)))
    btree = cKDTree(nB, compact_nodes=True, balanced_tree=True)
    for col in [f'neighbour_{col}' for col in list(range(0, num_classes))]:
        test_df[col] = 0.0

    for i, row in tqdm(test_df.iterrows()):
        scores = {x: 0 for x in [f'neighbour_{col}' for col in list(range(0, num_classes))]}
        pt = row['centroid']
        i_start = 0
        k = 9
        dists, idxs = btree.query(np.array([pt.x, pt.y]), k=k)
        for dist, idx in zip(dists[i_start:k], idxs[i_start:k]):
            roof_id = int(train_df.loc[train_df.index[idx], 'roof_type_id'])
            scores[f'neighbour_{roof_id}'] += (0.001 / dist) ** 2.0
       for col, score in scores.items():
            test_df.loc[i, col] = score
    return test_df.loc[:, [f'neighbour_{col}' for col in list(range(0, num_classes))]]
```

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

- Manual or automatic relabelling of training data
- Unverified data
- Balancing classes

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

   We plotted maps with bounding boxes(roofs) of different colours to get a sense of the distribution and quality of the data.

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

   We visually checked predicted roof labels which differed substantially from the true labels on the validation sets in our k-folds CV scenario. By doing so, we identified a lot of mislabeled training data samples.

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.)?

Using GPUs is essential to train all CNN models in a decent time frame. We used Nvidia Titan X (Pascal) with 12 GB of memory. Training models using GPU with less memory may need to change batch_size in src/models.

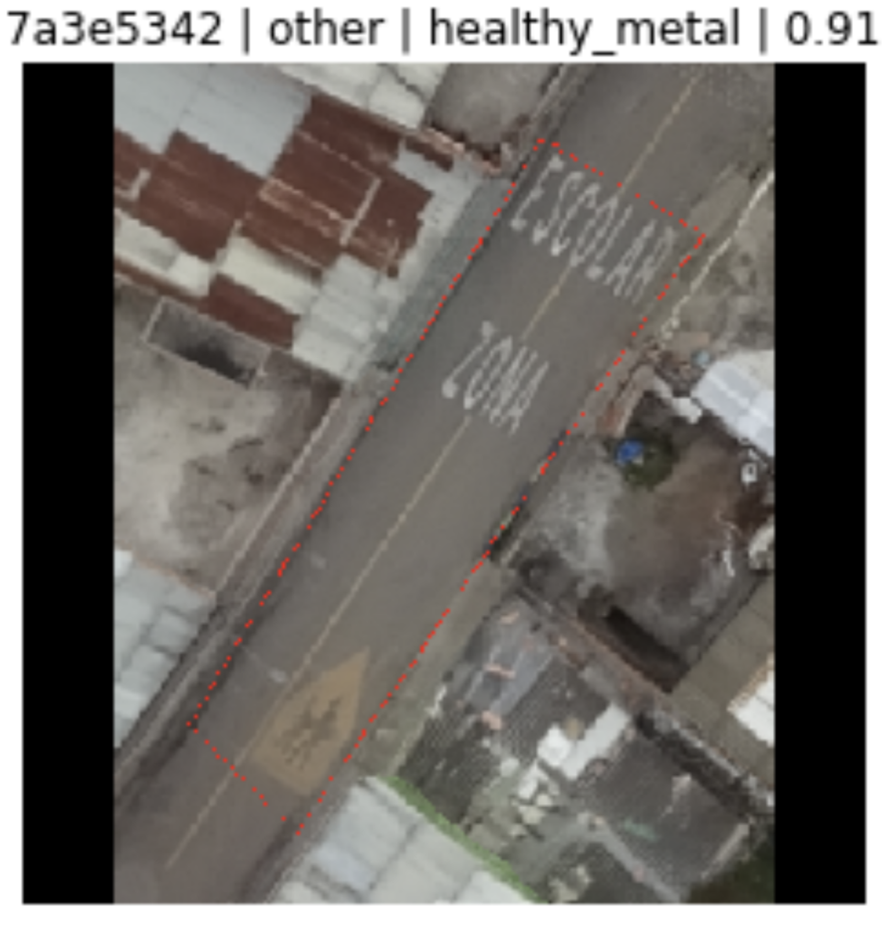
Because of memory overload during training, we run one script per fold for training, but removing '--folds' from the training command train all the folds in the same execution.

Some of the CNN models results could be unstable. For example A10 CNN model localCV/LB could fluctuate from 0.440/0.385 to 0.452/0.391. However, this instability doesn't influence our score significantly or our position on the LB.

It's also preferable to use a GPU to train catboost model, as it by default, to get the same result as we did.

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

   Image of a piece of a road which was labelled as a heavy metal roof.



9. 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?

   The biggest issue of this competition is data quality. Training data is heavily mislabelled. Namely, there are lots of roof labels in training data which are clearly incorrect.

The test data seemed to have the same problem, so we couldn’t apply any technique to deal with these noise labels successfully, as it would have a negative impact on the competition metrics (high confidence on mislabelled test samples strongly penalise the score).

So, the most impactful thing would be cleaning and correcting at least the test data.