DSTL

https://www.kaggle.com/c/dstl-satellite-imagery-feature-detection/

Contents

Object types

Note that labels here are 1 less than in submission file:

- 0: Buildings - large building, residential, non-residential, fuel storage facility, fortified building
- 1: Misc. Manmade structures
- 2: Road
- 3: Track - poor/dirt/cart track, footpath/trail
- 4: Trees - woodland, hedgerows, groups of trees, standalone trees
- 5: Crops - contour ploughing/cropland, grain (wheat) crops, row (potatoes, turnips) crops
- 6: Waterway
- 7: Standing water
- 8: Vehicle Large - large vehicle (e.g. lorry, truck, bus), logistics vehicle
- 9: Vehicle Small - small vehicle (car, van), motorbike

General approach

UNet network with batch-normalization added, training with Adam optimizer with a loss that is a sum of 0.1 cross-entropy and 0.9 dice loss. Input for UNet was a 116 by 116 pixel patch, output was 64 by 64 pixels, so there were 16 additional pixels on each side that just provided context for the prediction. Batch size was 128, learning rate was set to 0.0001 (but loss was multiplied by the batch size). Learning rate was divided by 5 on the 25-th epoch and then again by 5 on the 50-th epoch, most models were trained for 70-100 epochs. Patches that formed a batch were selected completely randomly across all images. During one epoch, network saw patches that covered about one half of the whole training set area. Best results for individual classes were achieved when training on related classes, for example buildings and structures, roads and tracks, two kinds of vehicles.

Augmentations included small rotations for some classes (±10-25 degrees for houses, structures and both vehicle classes), full rotations and vertical/horizontal flips for other classes. Small amount of dropout (0.1) was used in some cases. Alignment between channels was fixed with the help of `cv2.findTransformECC`, and lower-resolution layers were upscaled to match RGB size. In most cases, 12 channels were used (RGB, P, M), while in some cases just RGB and P or all 20 channels made results slightly better.

Validation

Validation was very hard, especially for both water and both vehicle classes. In most cases, validation was performed on 5 images (6140_3_1, 6110_1_2, 6160_2_1, 6170_0_4, 6100_2_2), while other 20 were used for training. Re-training the model with the same parameters on all 25 images improved LB score.

Some details

- This setup provides good results for small-scale classes (houses, structures, small vehicles), reasonable results for most other classes and overfits quite badly on waterway.

- Man-made structures performed significantly better if training polygons were made bigger by 0.5 pixel before producing training masks.
- For some classes (e.g. vehicles), it helped a bit to make the first downscaling in UNet 4x instead of default 2x, and also made training 1.5x faster.
- Averaging of predictions (of one model) with small shifts (1/3 of the 64 pixel step) were used for some classes.
- Predictions on the edges of the input image (closer than 16 pixels to the border) were bad for some classes and were left empty in this case.
- All models were implemented in pytorch, training for 70 epochs took about 5 hours, submission generation took about 30 minutes without averaging, or about 5 hours with averaging.

Other things tried

A lot of things that either did not bring noticeable improvements, or made things worse:

- Losses: jaccard instead of dice, trying to predict distance to the border of the objects.
- Color augmentations.
- Oversampling of rare classes.
- Passing lower-resolution channels directly to lower-resolution layers in UNet.
- Varying UNet filter sizes, activations, number of layers and upscale/downscale steps, using deconvolutions instead of upsampling.
- Learning rate decay.
- Models: VGG-like modules for UNet, SegNet, DenseNet

Object types stats

Area by classs:

im_id	0	1	2	3	4	5	6	7	8
6010_1_2	0.0%	0.0653%	0.0%	1.3345%	4.5634%	0.0%	0.0%	0.0%	0.0%
6010_4_2	0.0%	0.0%	0.0%	1.9498%	12.3410%	0.0%	0.0%	0.0%	0.0%
6010_4_4	0.0%	0.0%	0.0%	0.0%	22.8556%	0.0%	0.0%	0.0%	0.0%
6040_1_0	0.0%	0.0%	0.0%	1.4446%	8.0062%	0.0%	0.0%	0.0%	0.0%
6040_1_3	0.0%	0.0%	0.0%	0.2019%	18.7376%	3.6610%	0.0%	0.0%	0.0%
6040_2_2	0.0%	0.0%	0.0%	0.9581%	18.7348%	0.0%	0.0%	0.0%	0.0%
6040_4_4	0.0%	0.0%	0.0%	1.8893%	2.9152%	0.0%	0.0%	0.0%	0.0%
6060_2_3	0.1389%	0.3037%	0.0%	3.0302%	8.4519%	93.5617%	0.0%	0.0%	0.0%
6070_2_3	1.5524%	0.3077%	0.8135%	0.0%	16.0439%	0.0%	10.6325%	0.0543%	0.0%
6090_2_0	0.0%	0.0343%	0.0%	0.4072%	10.1105%	28.2399%	0.0%	0.3130%	0.0%
6100_1_3	8.7666%	2.7289%	2.2145%	12.2506%	6.2015%	2.6901%	0.0%	0.6839%	0.011
6100_2_2	3.1801%	0.8188%	1.1903%	3.7222%	7.6089%	44.3148%	1.8823%	0.0512%	0.010
6100_2_3	8.2184%	1.4110%	1.2099%	9.5948%	7.5323%	0.0%	0.0%	0.0603%	0.014
6110_1_2	13.1314%	2.8616%	0.4192%	4.1817%	3.3154%	49.7792%	0.0%	0.1527%	0.0%
6110_3_1	4.5495%	1.2561%	3.6302%	2.8221%	5.4133%	57.6089%	0.0%	0.5531%	0.018
6110_4_0	2.4051%	0.5732%	1.8409%	2.8067%	5.7379%	80.7666%	0.0%	1.4210%	0.013
6120_2_0	1.7980%	0.7257%	0.8505%	4.4026%	5.6352%	79.5910%	0.0%	0.0%	0.013
6120_2_2	20.6570%	2.0389%	4.2547%	8.6533%	4.4347%	10.2929%	0.0%	0.2859%	0.007
6140_1_2	12.9211%	2.4488%	0.3538%	4.1461%	3.1027%	49.5910%	0.0%	0.1415%	0.0%
6140_3_1	5.2015%	1.4349%	3.4252%	2.5189%	5.8852%	57.3959%	0.0%	0.4664%	0.004

im_id	0	1	2	3	4	5	6	7	8
6150_2_3	0.0%	0.6055%	0.0%	3.0197%	13.5187%	80.6649%	0.0%	0.0%	0.0%
6160_2_1	0.0%	0.0%	0.0%	2.7986%	10.2713%	0.0%	0.0%	0.0%	0.0%
6170_0_4	0.0%	0.0016%	0.0%	0.1994%	24.8913%	0.0%	0.0%	0.0152%	0.0%
6170_2_4	0.0%	0.0011%	0.0%	2.5070%	7.7844%	49.5326%	0.0%	0.0089%	0.0%
6170_4_1	0.0%	0.0%	0.0%	0.1349%	20.2214%	0.0%	0.0%	0.0%	0.0%

Making a submission

Train a CNN (choose number of epochs and other hyper-params running without `--all`):

```
$ ./train.py checkpoint-folder --all --hps dice_loss=10,n_epochs=70
```

Make submission file (check hyperparameters doing a submission for the model trained with validation by running with `--validation *value*` and optionally `--valid-polygons`):

```
$ ./make_submission.py checkpoint-folder submission.csv.gz
```

Finally, use `./merge_submission.py` to produce the final submission.

This just gives a general idea, real submissions were generated with different hyperparameters for different classes, and all above commands have more options that are documented in the commands themselves (use `--help`, check the code if in doubt).