SOLUTION REPORT

for soil erosion detection

Task description

With one satellite image and masks corresponding to soil erosion, it is required to make a model capable of finding the location of those soil erosion (segmentation of image).

Data preprocessing

Preprocessing was done in data_preparation.py with usage of PIL, numpy and geolibraries: rasterio, geopandas and shapely.

Given data

There is Sentinel-2 “T36UXV_20200406T083559_TCI_10m.jp2”, which is 10980x10980 RGB aerial image with corresponding location data. Soil erosion masks are in “Masks_T36UXV_20190427.shp” and the rest of the related files. They contain database with geometry column. This column contain polygons which represent the locations (latitude and longtitude) of soil erosion. There are 945 non-zero rows with geomeries.

Preprocessing

Firstly, conversion of GeoData frame to raster CRS was done. 435 masks were outside of the aerial image. Therefore they were excluded from further analysis.
Secondly, a 10980x10980 mask image was created (corresponding to the aerial image).
Next, 10980x10980 images were chopped into 224x224 images.If there was a mask on that 224x224 clipping, the image and the corresponding mask were saved. Unfortunately, 10980 is not divisible by 224, so something had to be done with the scraps. If the mask was on such a snippet, the 224x224 window would slide to the border of the photo and such a photo would be cropped out.
Gathered data was stored in the data directory. Masks were in masks dir and images were in images dir. Additionally every gathered clipping is in “data.csv”.
In addition, the mean and standard deviation is calculated for each RGB channel.
Finally, dataset containing 290 images + masks was produced.

Modeling

I used PyTorch for modeling. For the model I have chosen UNet with ResNet-50, ResNet-101 and DenseNet-161 backbones.
There are 3 files corresponding to training: dataset.py, model.py and train.py. Dataset contains the Dataset class for PyTorch. Model.py contains one of the used models. Training loop is in train.py and this is the only file we run.

Dataset

In order to take advantage of PyTorch's capabilities, the *SoilErosionDataset* class was created. Its task is to return selected (indexed) images with masks. The "data.csv" is used here, which contains information about the indexes of the photos and masks. In addition, basic normalization is performed. Initially, the mean and standard deviation extracted during data processing were used for this. In the end, however, I decided to simply normalize from pixel values of 0-255 to a matrix with values from 0 to 1 (by simply dividing by 255). At the very end, the transformations given in the training loop are applied.

Models

I decided to go with UNet with ResNet-50, ResNet-101 and DenseNet-161 backbones. Initially I used UNet with ResNet-50 backbone from GitHub. Looking for more backbones I used UNet model from GitHub which contains most of VGG, ResNet and DenseNet as backbone for UNet. It can be installed with pip, so that model was not copied into this repo code but is in requirements.

Training

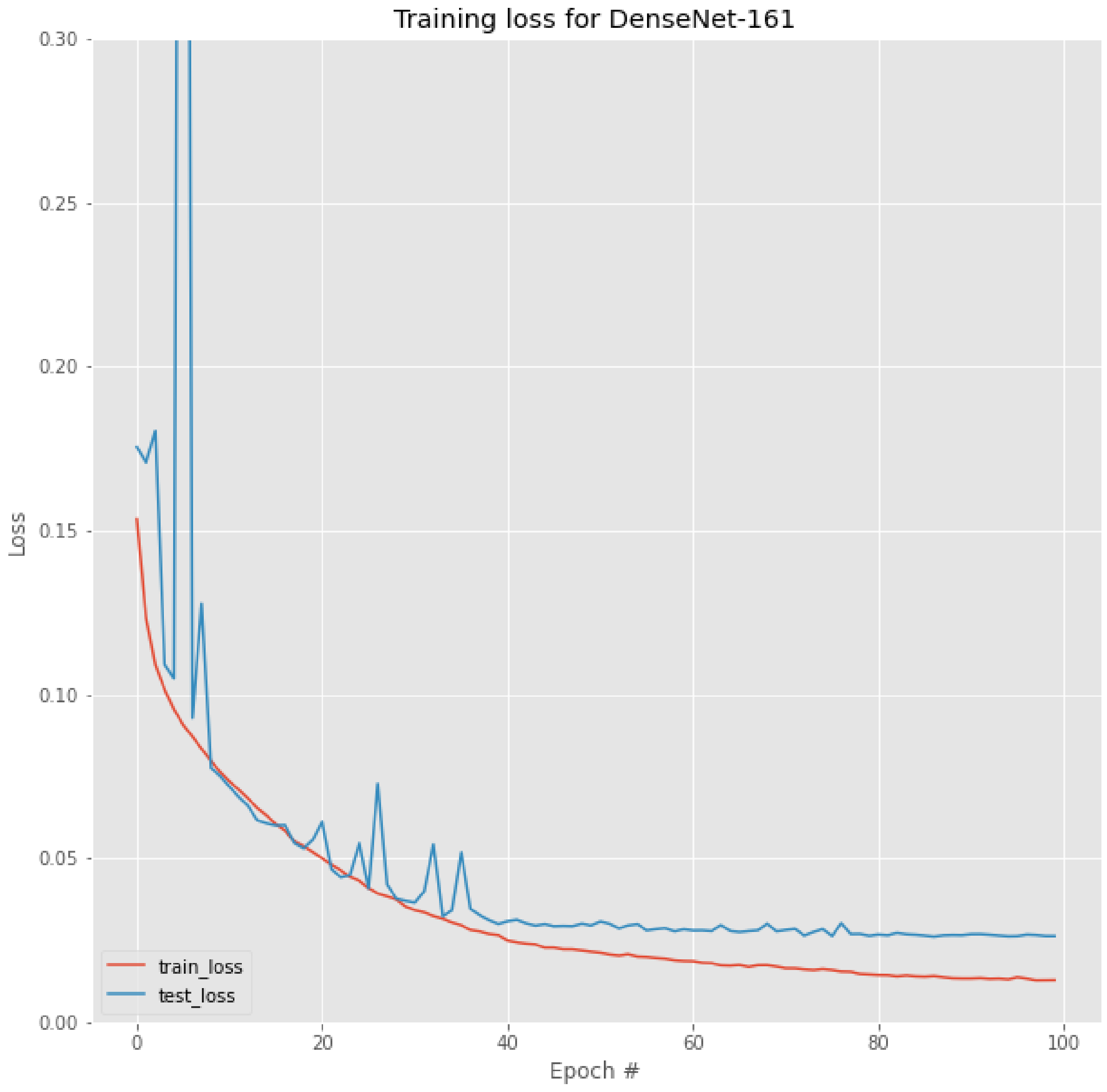
As mentioned earlier I used UNet with different backbones. In order to facilitate the reproduction of training seeds of torch, numpy and random were set to 1. Dataset was divided in 0.794:0.206 ratio, which translated into 230 images+masks for the training and 60 for testing.
For augmenting training images+masks *transforms* from *torchvision* were used. Transform was composed of random horizontal flip (with probability of 50%), random vertical flip (with probability of 50%) and random affine (scale from 0.85 to 1.0 and rotation from -90 to 90 degrees).
For the loss function I used *BCEWithLogitsLoss*. This loss combines a Sigmoid layer and the BCELoss (Binary Cross Entropy Loss) in one single class. I used it due to the fact that there was only one class to find - soil erosion.
For PyTorch’s optimizer (to update model’s parameters) I have chosen Adam with learning rate = 0.001. In addition to that I used StepLR scheduler to update the learning rate.

I trained models for 100 epochs with a batch size of 20 (or 10 depending on machine).

Results

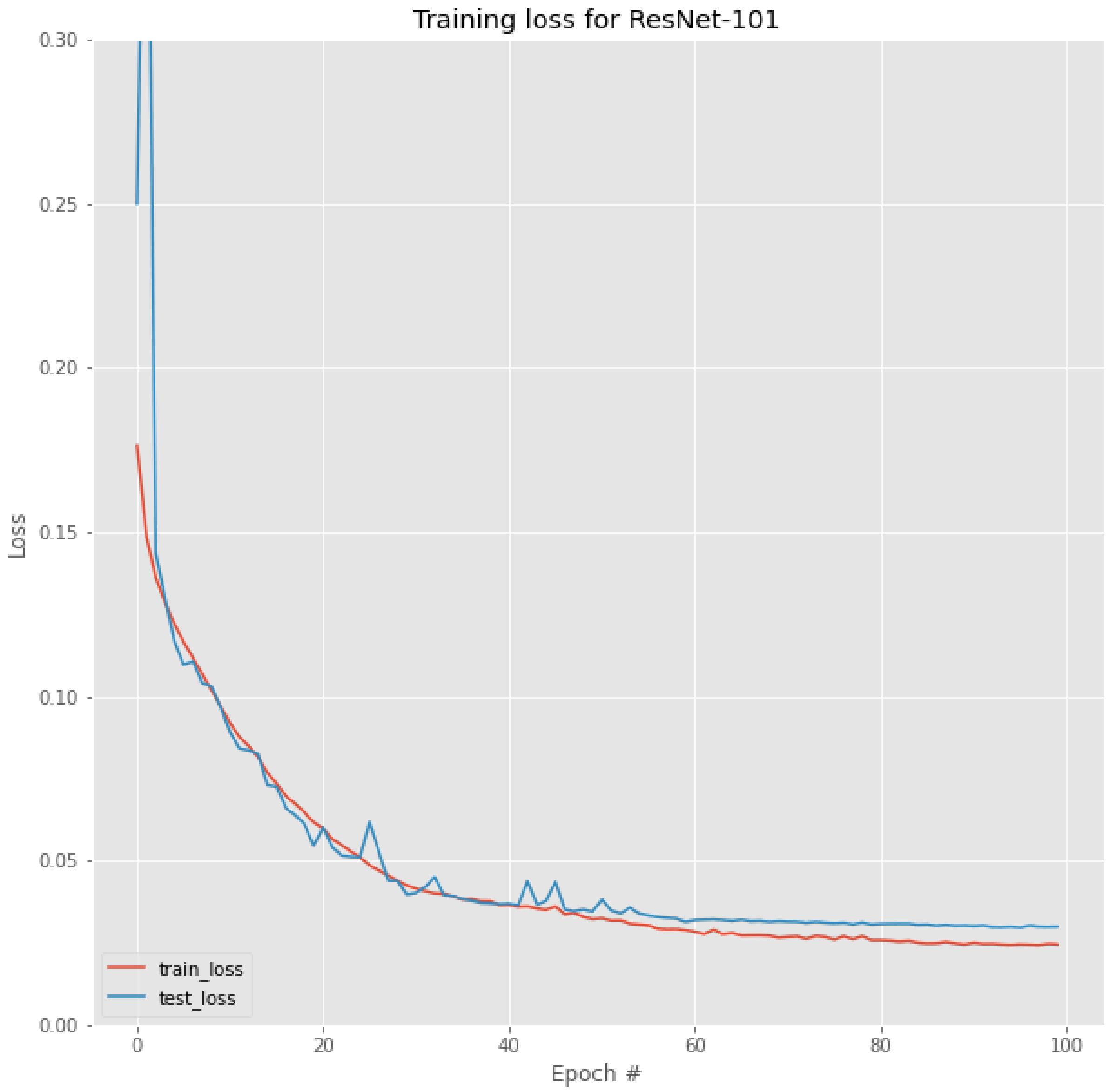
DenseNet-161

The best average loss of 0.02608 (with loss function BCEWithLogitsLoss) was achieved using DenseNet-161 as backbone. Below is a graph of train / test loss from epochs.



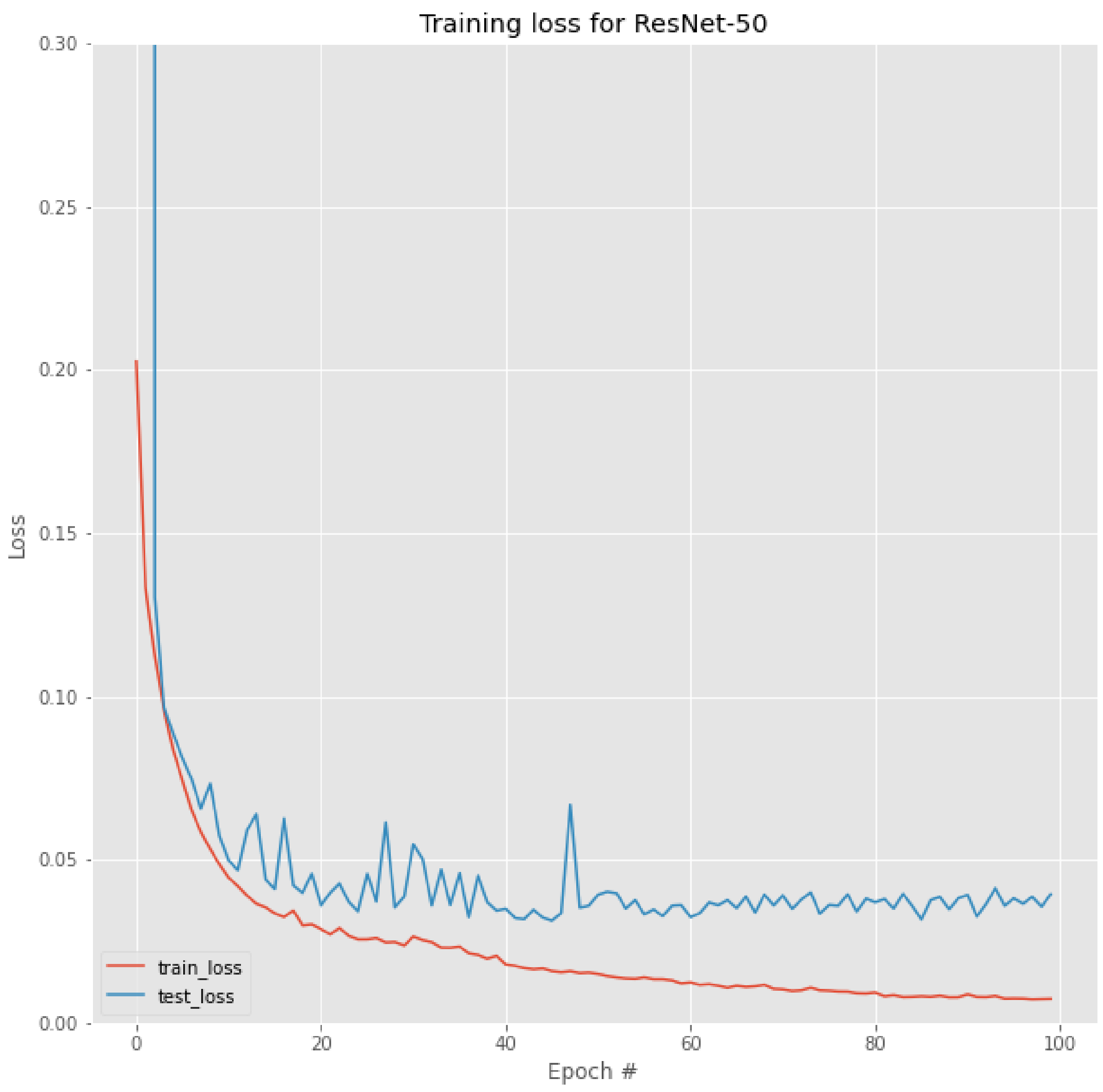
ResNet-101

Second best average loss of 0.02966741 was achieved using ResNet-101 as backbone. Below is a graph of train / test loss from epochs.



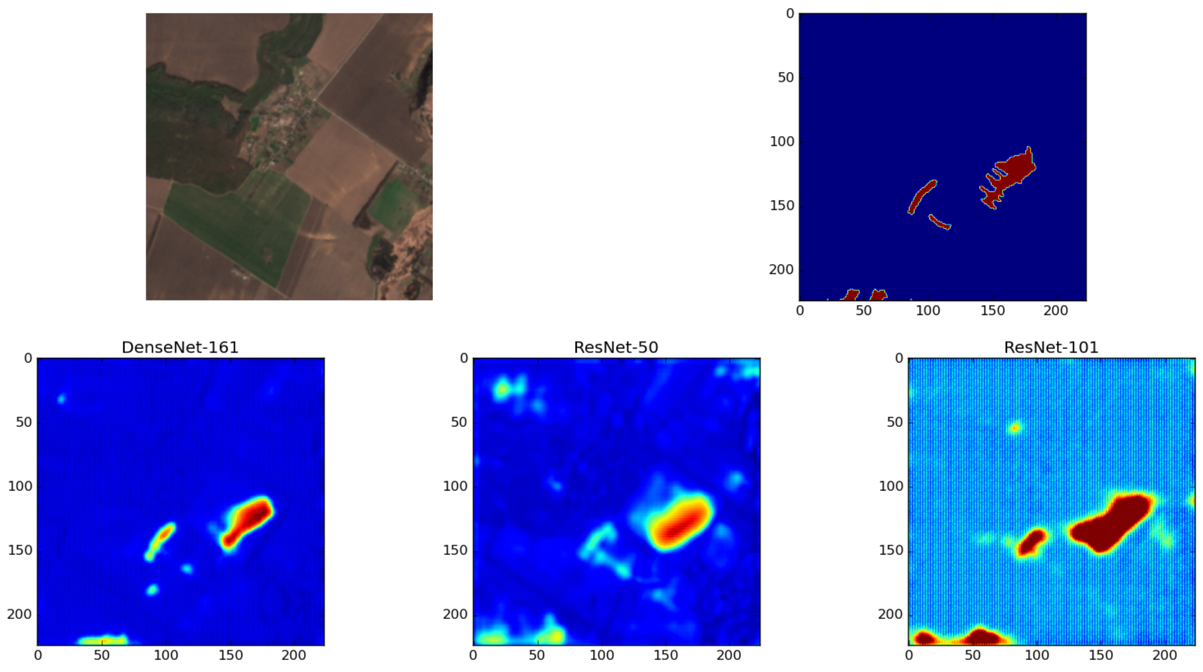
ResNet-50

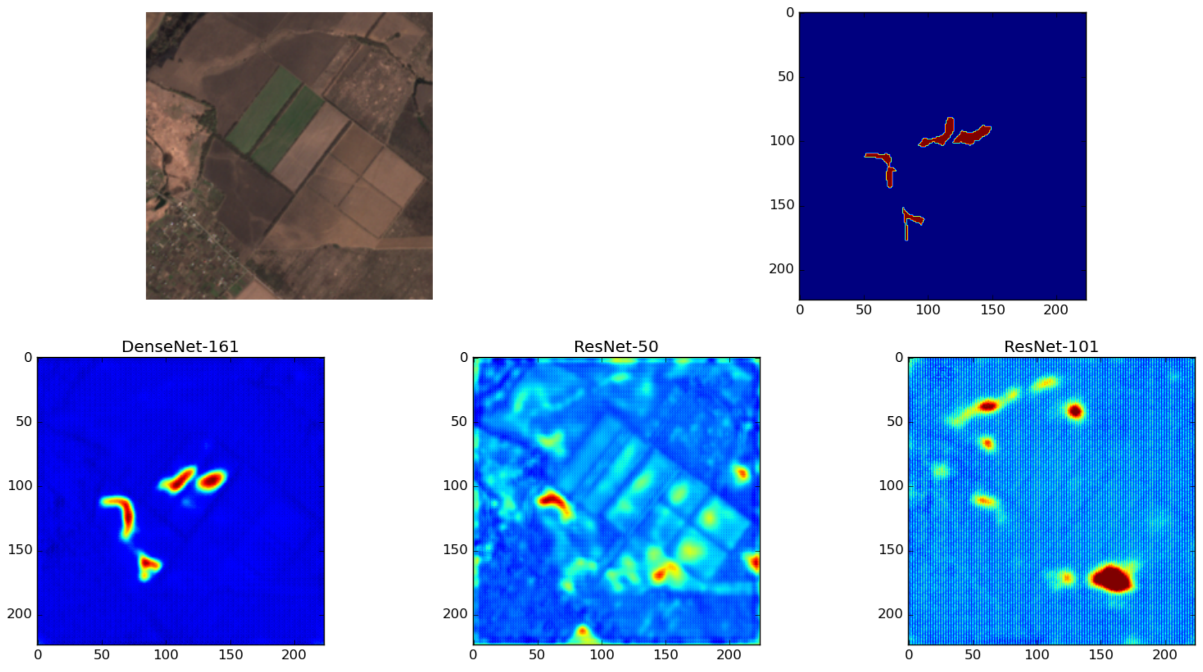
The worst average loss of 0.03128504 was achieved using ResNet-50 as backbone. Below is a graph of train / test loss from epochs. Test loss was converging fastest from all backbones but also jittered the most.



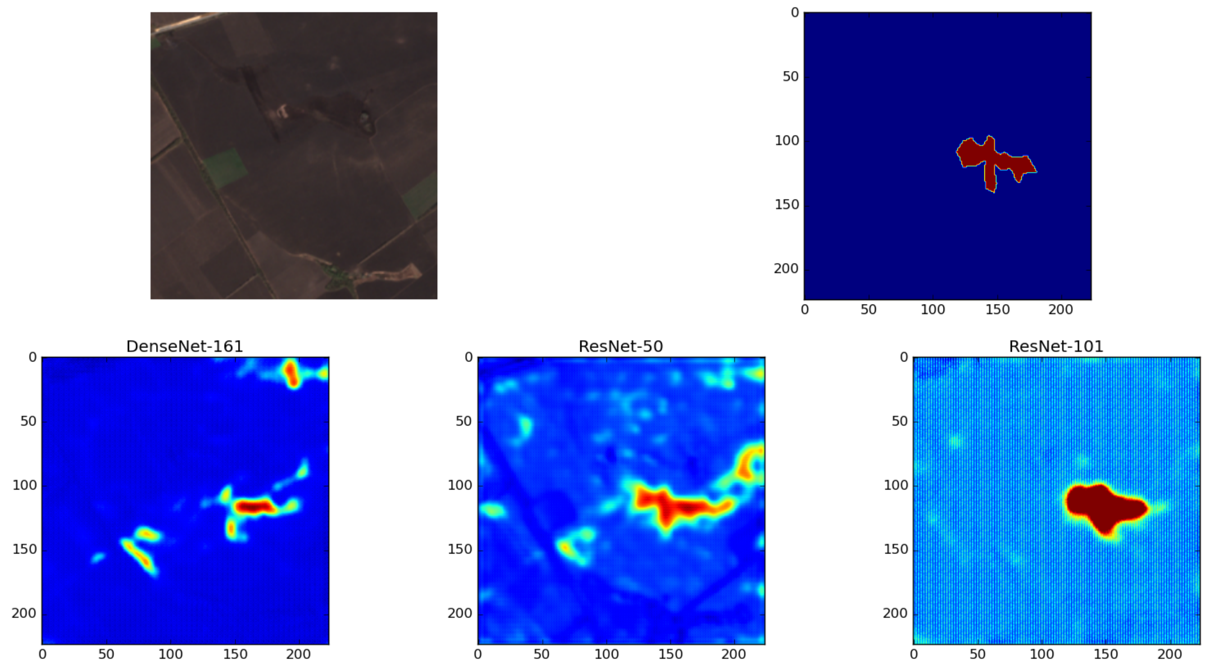
Analysis of images and predicted masks

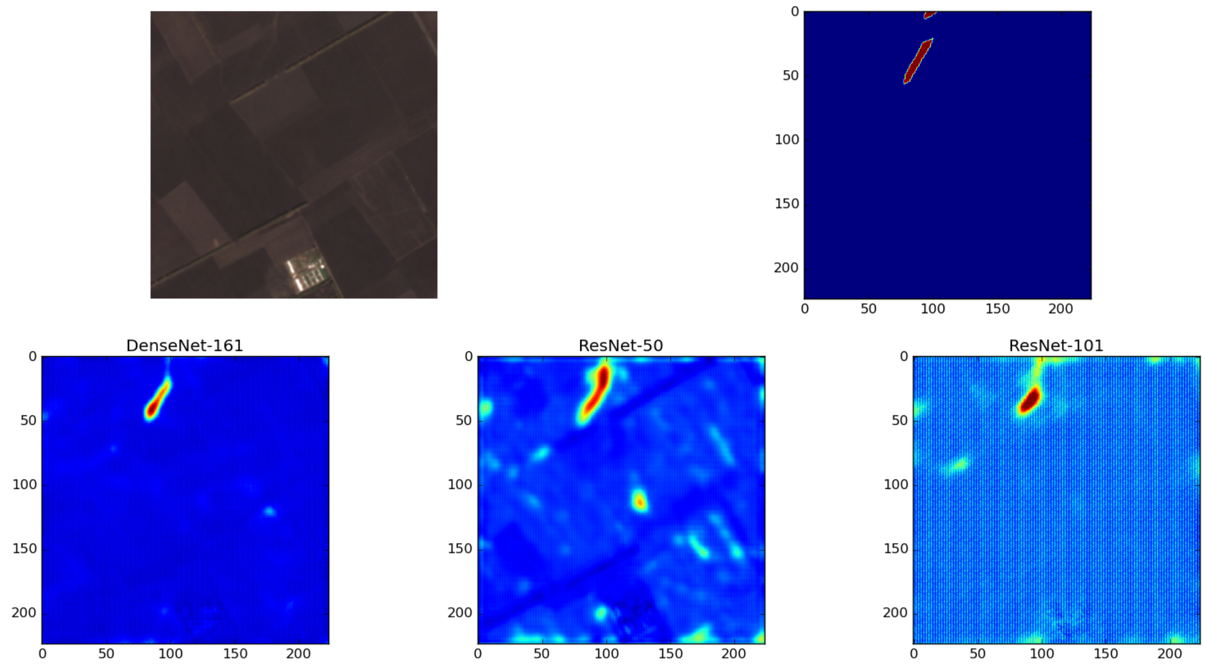
Below are some examples of images (top left corner) and expected mask (top right corner) with predictioned masks (from left: DenseNet-161, ResNet-50, ResNet-101). In the first picture we can see that all models detected the biggest piece of eroded earth. DenseNet-161 did the best with its shape. UNet with DenseNet-161 also detected one of the small lines next to the biggest piece of eroded earth and a point from the other line. Model with ResNet-50 as backbone also picked up the lines but wasn't so sure about them. Unet with ResNet-101 output one big blog from that area. All of UNets detected some of the left bottom corner mask. The least residue (Negative True pixels in mask) was produced by DenseNet-161.
In the second image only DenseNet-161 managed to detect all masks. ResNets detected only inverted "L" shape mask and mistakenly detected non-eroded soil.

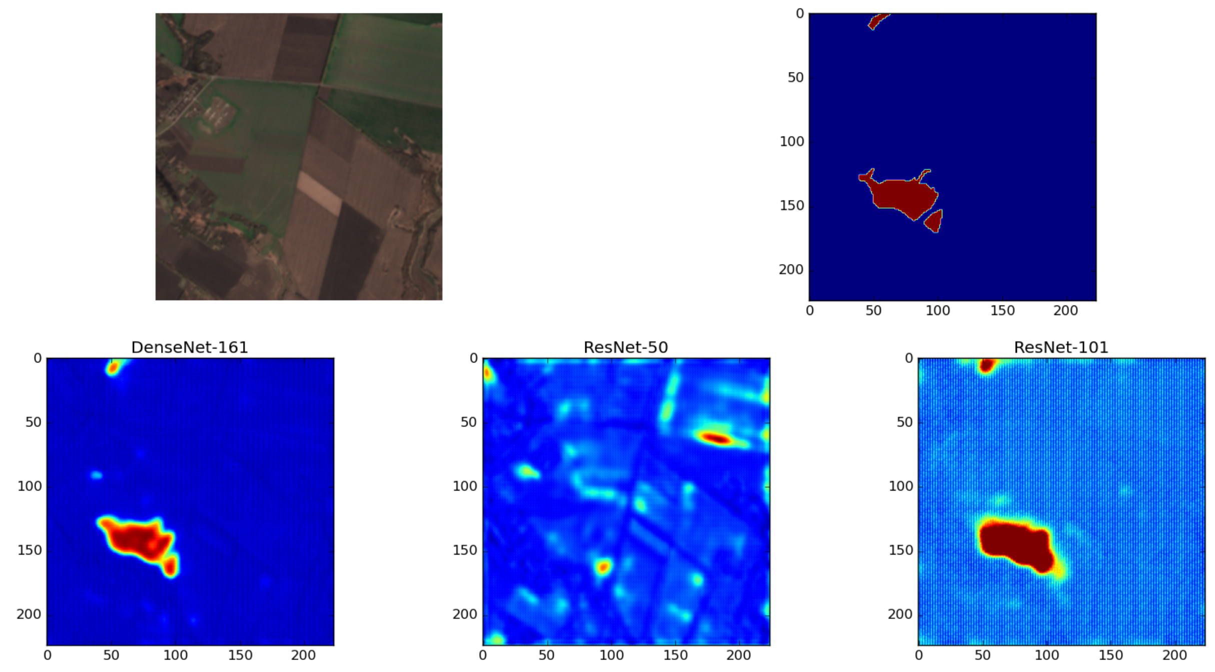

DenseNet-161
ResNet-50
ResNet-101


DenseNet-161
ResNet-50
ResNet-101

But DenseNet-161 isn’t always the best backbone. Third image shows that it can produce lots of False Positive pixels in the mask (although some of them look like eroded soil but are not marked in the mask database). ResNets were clear winners in this one.
Fourth and Fifth images’ situation is similar to first two.







Further development

Models

More Models with more backbones. For example some of SOTA models and backbones from https://github.com/sithu31296/semantic-segmentation:

- Supported Backbones:
    - ResNet+ (ArXiv 2021)
    - PVTv2 (CVMJ 2022)
    - PoolFormer (CVPR 2022)
    - ConvNeXt (CVPR 2022)
    - UniFormer (ArXiv 2022)
    - VAN (ArXiv 2022)
    - DaViT (ArXiv 2022)
- Supported Heads/Methods:
    - SegFormer (NeurIPS 2021)
    - FaPN (ICCV 2021)
    - CondNet (IEEE SPL 2021)
    - Light-Ham (ICLR 2021)
    - Lawin (ArXiv 2022)
    - TopFormer (CVPR 2022)

- Supported Standalone Models:
    - BiSeNetv2 (IJCV 2021)
    - DDRNet (ArXiv 2021)
- Supported Modules:
    - PPM (CVPR 2017)
    - PSA (ArXiv 2021)

The best of trained models can be aggregated and the average pixel from all of them can be calculated.

It is also worth trying transformers instead of CNNs. A Dilated Segmentation Network with the Morphological Correction Method in Farming Area Image Series is a paper in which they extracted global features with a transformer.

Data

As in most cases, more data equals better accuracy. To eliminate seasonal variances normalized difference vegetation index (NDVI) and normalized difference water index (NDWI) could be calculated and inserted into models input alongside RGB aerial image (either RGB image preprocessed or another layer to model).
Another technique in data processing can be cloud detection. In paper Seeing Through Clouds in Satellite Images they managed to recover pixels occluded by clouds in satellite images.