Challenge Computer Vision for Remote Sensing TUB – WS23

Matthias Jean Théo Personnaz (Matrikelnummer 486656)

Note: All codes and figure sources not handed out are on my GitHub. Precise parameters are only provided in this report for the one handed-out model (§2.3).

1 Overall technical choices

Dataset-driven The dataset consists of 10 classes of satellite imagery that exhibit a uniform overall appearance. These aerial images primarily capture land features and are characterized by limited color variation, predominantly in darker tones, primarily falling within the green and blue regions of the color spectrum. In a way analogous to how medical image processing differs significantly from general-purpose image recognition AI, utilizing a ResNet model, as demonstrated in the example, might be unnecessary in this context. Given that all images share the same dimensions, possess low resolution, and are taken from a consistent distance, any attempt to apply a U-Net approach, aimed at accommodating vastly different pattern scales, would be impractical for these relatively small, homogenous images.

Python environment-driven Likewise, the provided `.yaml` environment does contain only general packages, for example, no `sklearn` for easier implementation of cross-validation, etc. With only the pretrained `torchvision` models at ouor disposal, that limited the possibilities of reusing already ridiculously large models anyway (for example, `torchvision.models.efficientnet_b0` following [TL19] starts at 5M+ parameters). Likewise, most of HuggingFace pretrained models are very big.

Because of these constraints and the arguably intuitive hypothesis that the complexity of the data set is very low (the meaning of complexity here is to be precised, but it could refer to the entropy of the latent space of these 10 classes), I therefore chose to build simple tailored networks in a regular train/validation-divided data set framework. Moreover, I only had access to a somewhat slow GTX 1650 mobile[1] with limited usable VRAM <4GB, filling quite fast during training and rapidly hitting the OOM killer `RunTimeError: CUDA out of memory` for networks reaching $\approx 10^6$ trainable parameters even for relatively small batches, because of torch's implementation overhead and additional information required to store the batches and the models.

Because of that, I've stuck to simpler, purpose-tailored models not exceeding the order of magnitude of a few 10^5 parameters in size.

2 Overview of tested approaches

In a few days, I tested mainly 3 (families of) approaches:

- A SimCLR contrastive learning framework following the work of [CKNH20];
- A non-sequential model leveraging the 2D Fourier spectrum or Sobel filtering of each image;
- A sequential convolution model with its variants (hyperparameters, pooling, normalization, etc) and a non-sequential convolution model with partial or total residual connections following the work of [HZRS15].

Every network stacks convolutions with the classical [RFB15] approach ordering Conv2d → BN → ReLU (→ MaxPool). Layers ranged from 4 to 8 with 16 to 64 convolution filters each, of size 6 to 15. Padding was introduced to allow for skip connections when necessary or to avoid resolution shrinking. A 80/20 train/validation random split was performed for each model of neural networks except SimCLR. No test subset was drawn, in order not to deplete the dataset too much, which was not huge (overall about 11,000 images). The dataset class remained that defined by the guidelines except for SimCLR, for which the dataset was subsequently mutated into a new dataset of tuples of augmented images.

2.1 [discarded] SimCLR framework to learn distambiguishing the classes from scratch to extract relevant features

The task of classifying images for few classes suggests that SimCLR is the perfect way to go. So I implemented it. Augmentation was performed only through rotations and color jittering / contrast / luminosity.

[1] About 10 times less theoretical FP32 TFLOPS than an A100, but the VRAM makes it more crippled anyway.

Since all images are from satellite imagery, with same scale in the dataset, trained to discriminate (simple multiclass classification) nor upscaling was used.

Recall SimCLR tries to minimize the following loss function for a batch B of N pairs of augmented images:

$$\mathcal{L} = -\frac{1}{N} \sum_{i,j \in B} \log \frac{\exp(\text{sim}(z_i, z_j)/\tau)}{\sum_{k=1}^{2N} \mathbb{1}_{[k \neq i]} \exp(\text{sim}(z_i, z_k)/\tau)} \tag{1}$$

After tinkering with the temperature and SGD learning rate hyperparameters, a point of fast and efficient convergence was reached (figure 1). Unfortunately, a t-SNE embedding of the encoder's latent representations shows that the separation is very poor except for a few couple of classes (figure 2). Later, a simple MLP classifier failed to converge sactisfactorily.

2.2 [discarded] Bifold neural network with Fourier resp. Sobel transform

Fourier transformation leveraging The intuition came from my knowledge of classical computer vision satellite images feature many regularities, typical frequencies, whether it's buildings or crops, quite universal in their appearance. Consequently, one can extract the energy spectrum of the images for leveraging regularities in those frequencies to help tell the classes apart. In figure 3, it stands obvious that those structures exist. The spectrum of water bodies has a characteristic noise to it, unlike crops that have strong harmonics often organized in a wind rose.

So I created multpiles variations of a network (figure 5) performing recognition from both the image and its Fourier *square root power spectrum* (i.e. $|\mathcal{F}(\mathbf{x})(\mathbf{k})|\sqrt{\mathbf{k}\cdot\mathbf{k}}\,\mathrm{d}\mathbf{k}$ like the signal processing equivalent) on its own small convnet in parallel with the image, and then linearly combining the features of both the to predict the class.

During my experiments, I found that **a relatively small 93 000 parameter network (including Fourier pattern recognition) could achieve the same validation accuracy as a 5× to 7× bigger residual convolutional network (of same depth but different resolution) without Fourier analysis**. This corroborates that the the spectrum carries useful information indeed in the context of satellite imagery.

Sobel transformation leveraging The same was done with the Sobel transform (which is equivalent to imposing a very small linear convolution at the very beginning of the network), although it does not seem that this technique improved the results any more.

2.3 [chosen] Traditional convolution ResNet

Getting back to simplicity, the final chosen model consists of a semi-sequential ResNet of only $\sim 213 \times 10^3$ parameters, totalling 6 layers of decreasing resolution from 64 to 16, alternating skip residual connections and simple max pooling. Its small size compared to other models I have trained overall makes it by far the **most efficient network found**, and thus, the chosen one to hand out. The number of layers and resolution (number of convolution filters) were set given the following:

- While the dataset features a relatively diverse extent of "canonical features", representing shades, patterns, edges, shapes;
- Only a few of them should suffice to identify the class *without much subsequent abstract post-processing.*

Therefore, not much abstraction is needed, cf. the first section of this report. Thus we do not need to overengineer the network by setting an outstanding depth, which only helps in case of extremely varied datasets featuring high-level features objects in size, lighting, 3D view, etc. But we still need *an uncompressible number of pattern filters* to have a sufficient grasp of the scenery, thus beginning by extracting 64 of them. One final linear layer classifies with the visual features. It achieves a 90% validation accuracy after less than 80 epochs of medium batches of size 50 (figure 4). The algorithm used was the `torch.optim.SGD` implementation of SGD with the default parameters: learning rate $lr = 10^{-3}$ and momentum $\mu = 0.9$. A dropout rate of 5% for the visual features was added during training before the linear layer. The chosen loss for such classification tasks has obviously been the cross-entropy for the logits.

Room for improvement To help nonlinear behaviours to emerge, adding one or more layers might help, as well as maybe th nnumber of filters to generalize more over the set of geographical configurations. In later revisions of the network, I would increasing the dropout rate and tweaking the SGD parameters for this task. I remain convinced that a SimCLR-like framework might very helpful to discriminate the classes, even though it is trickier and I was not able to find the proper hyperparameters. I think it might benefit from a larger dataset of images with finer resolution.

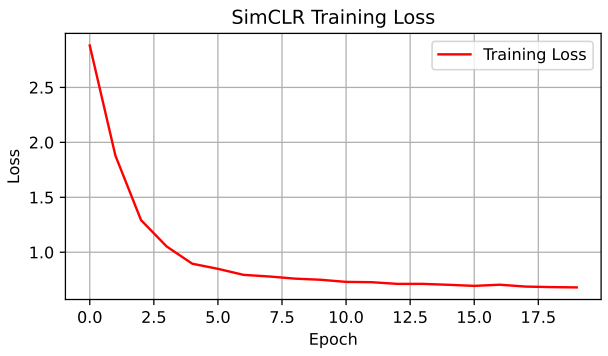

Figure 1: SimCLR loss (including encoder and projector head). The SGD algorithm converged well.

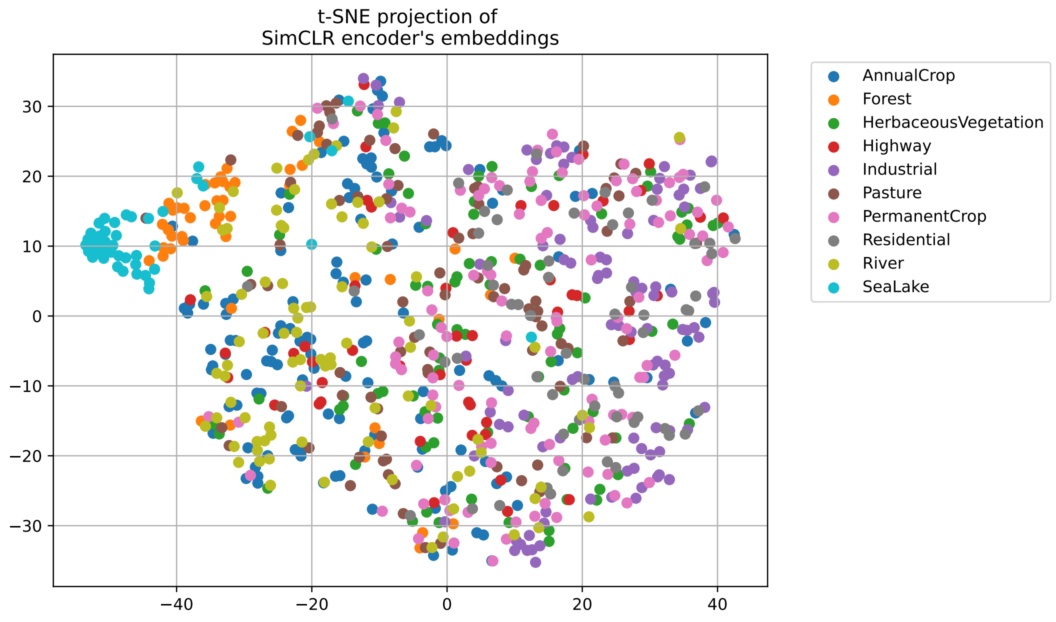

Figure 2: t-SNE projection of SimCLR's encoder in 32 dimensions. Although the embedding can definitely tell apart some categories (for exemple, Forest and Industrial do not overlap), the separation of the classes is still unusable to feed in a downstream classifier.

References

[CKNH20] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. February 2020.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. December 2015.

[LXT+17] Hao Li, Zheng Xu, Gavin Taylor, Christoph Studer, and Tom Goldstein. Visualizing the loss landscape of neural nets. December 2017.

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional networks for biomedical image segmentation. May 2015.

[TL19] Mingxing Tan and Quoc V Le. EfficientNet: Rethinking model scaling for convolutional neural networks. May 2019.

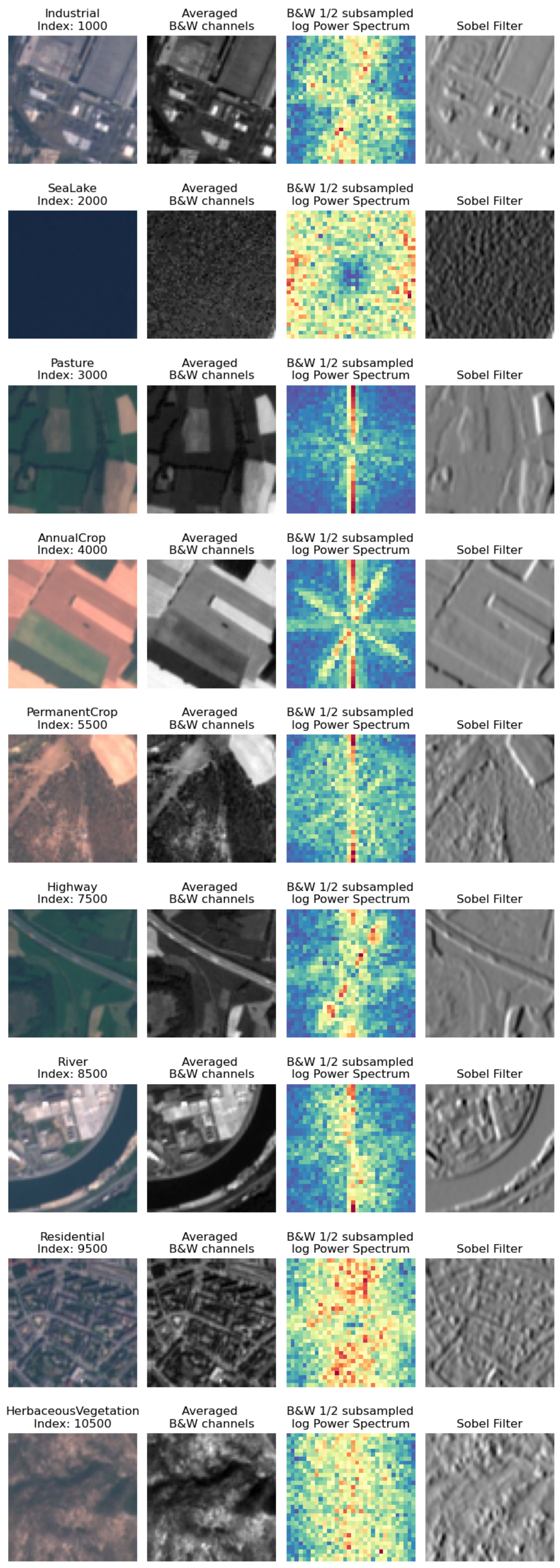

Figure 3: Original, B&W, half-subsampled Fourier logarithmic power spectrum and Sobel filters respectively applied to a few examples drawn from the dataset. Clearly, the Fourier transform does provide some insight about the landscape although it cannot discriminate every pair of classes entirely.

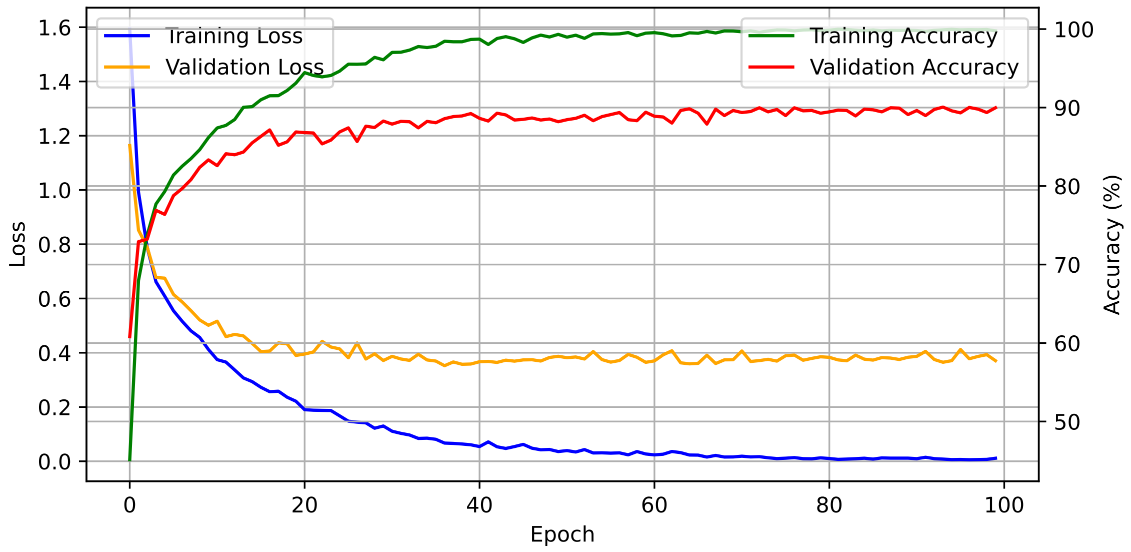

Figure 4: Accuracy and loss of the small semi-sequential ResNet. Convergence is fast and shows the simpler models are sometimes the best for simple tasks.

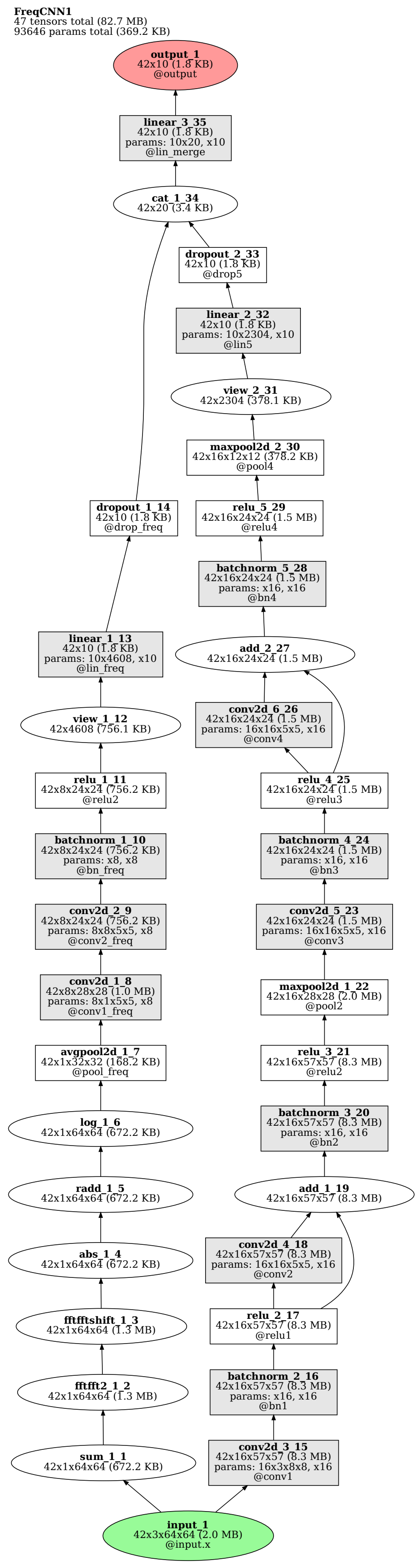
FreqCNN1
47 tensors total (82.7 MB)
93646 params total (369.2 KB)
output_1 42x10 (1.8 KB) @output
linear_3_35 42x10 (1.8 KB) params: 10x20, x10 @lin_merge
cat_1_34 42x20 (3.4 KB)
dropout_2_33 42x10 (1.8 KB) @drop5
linear_2_32 42x10 (1.8 KB) params: 10x2304, x10 @lin5
view_2_31 42x2304 (378.1 KB)
maxpool2d_2_30 42x16x12x12 (378.2 KB) @pool4
dropout_1_14 42x10 (1.8 KB) @drop_freq
relu_5_29 42x16x24x24 (1.5 MB) @relu4
batchnorm_5_28 42x16x24x24 (1.5 MB) params: x16, x16 @bn4
linear_1_13 42x10 (1.8 KB) params: 10x4608, x10 @lin_freq
add_2_27 42x16x24x24 (1.5 MB)
conv2d_6_26 42x16x24x24 (1.5 MB) params: 16x16x5x5, x16 @conv4
view_1_12 42x4608 (756.1 KB)
relu_1_11 42x8x24x24 (756.2 KB) @relu2
relu_4_25 42x16x24x24 (1.5 MB) @relu3
batchnorm_1_10 42x8x24x24 (756.2 KB) params: x8, x8 @bn_freq
batchnorm_4_24 42x16x24x24 (1.5 MB) params: x16, x16 @bn3
conv2d_2_9 42x8x24x24 (756.2 KB) params: 8x8x5x5, x8 @conv2_freq
conv2d_5_23 42x16x24x24 (1.5 MB) params: 16x16x5x5, x16 @conv3
conv2d_1_8 42x8x28x28 (1.0 MB) params: 8x1x5x5, x8 @conv1_freq
maxpool2d_1_22 42x16x28x28 (2.0 MB) @pool2
avgpool2d_1_7 42x1x32x32 (168.2 KB) @pool_freq
relu_3_21 42x16x57x57 (8.3 MB) @relu2
log_1_6 42x1x64x64 (672.2 KB)
batchnorm_3_20 42x16x57x57 (8.3 MB) params: x16, x16 @bn2
radd_1_5 42x1x64x64 (672.2 KB)
add_1_19 42x16x57x57 (8.3 MB)
abs_1_4 42x1x64x64 (672.2 KB)
conv2d_4_18 42x16x57x57 (8.3 MB) params: 16x16x5x5, x16 @conv2
fftfftshift_1_3 42x1x64x64 (1.3 MB)
relu_2_17 42x16x57x57 (8.3 MB) @relu1
fftfft2_1_2 42x1x64x64 (1.3 MB)
batchnorm_2_16 42x16x57x57 (8.3 MB) params: x16, x16 @bn1
sum_1_1 42x1x64x64 (672.2 KB)
conv2d_3_15 42x16x57x57 (8.3 MB) params: 16x3x8x8, x16 @conv1
input_1 42x3x64x64 (2.0 MB) @input.x

Figure 5: Graph of the non-sequential, partially residual model leveraging the Fourier transform (model in §2.2). The specific number of parameters and layers is irrelevant on the figure and bigger configurations were tested for that architecture. The other architectures tested for the Sobel transform and semi-sequential simple ResNets were either very similar, or only included the right branch of this network respectively. Thus they are not shown in this report to avoid redundancy.