IN5400/IN9400 - Mandatory 1

Alex

February 22, 2022

Submission due 9th of March 11:59pm

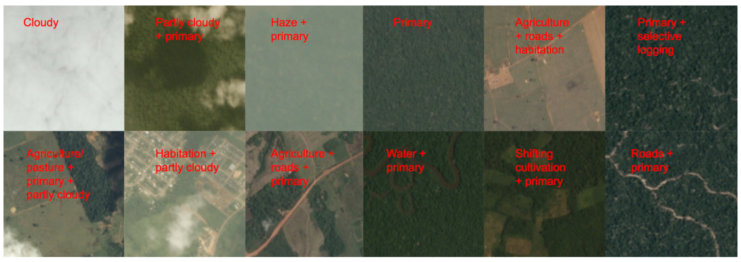
1 The mandatory exercise 1

Learning goals:

- writing a custom data loader
- changing the architecture of pre-trained networks
- Define and use a custom loss for a problem which is classification but not multi-class classification
- feel the importance of proper presentation of prediction results: the top-ranked images look well at the top and bottom even though your prediction accuracy over the whole dataset is so-so. The reason is that: thresholded subsets can have high accuracies even if the average performance over the whole dataset is not great.

In summary: Work with a custom loss on a bit more challenging vision dataset.

The dataset you will be using, collected by Planet, is images of the Amazon basin which includes Brazil, Peru, Uruguay, Colombia, Venezuela, Guyana, Bolivia, and Ecuador. The labels can broadly be broken into three groups: atmospheric conditions, common land cover/land use phenomena, and rare land cover/land use phenomena. Each image will have one and potentially more than one atmospheric label and zero or more common and rare labels. Images that are labeled as cloudy should have no other labels, but there may be labeling errors.


Cloudy
Partly cloudy + primary
Haze + primary
Primary
Agriculture + roads + habitation
Primary + selective logging
Agriculture/ pasture + primary + partly cloudy
Habitation + partly cloudy
Agriculture + roads + primary
Water + primary
Shifting cultivation + primary
Roads + primary

img credit: Planet. Example images contained in the dataset. In red are their corresponding labels.

1.1 What to consider before starting to train

It has one important property: it is not a multiclass dataset, and in that sense way more realistic than imagenet

- each image can have multiple groundtruth labels, e.g. primary rainforest and roads can be present as labels for the same image.
- prediction: you cannot use the argmax over classes of the logits or softmax predictions, because labels per image are not mutually exclusive.
- training: You cannot use just 17-class crossentropy loss, because labels per image are not mutually exclusive.
- you need to think: how to predict on a single image, what should the outputs be. If you have no idea after 15 minutes of thinking, look at examples of the training and validation labels and consider sources online.
- you need to think: what loss to use for training.
- the data is in:
  - images in /itf-fi-ml/shared/IN5400/2022_mandatory1/train-tif-v2/
  - labels in /itf-fi-ml/shared/IN5400/2022_mandatory1/train_v2.csv
  - all packed in /itf-fi-ml/shared/IN5400/2022_mandatory1/rainforest.tar (for copying off)
  - for test runs at home you might consider to copy just 1000 images from the image path above rather than copying off the whole tar file

1.2 an overview over tasks

- write a dataloader for Planet rainforest train and val datasets. In the scikit learn library is a useful function that can help you to binarise strings. Perform a test train split of the data, and use a manual seed so that you use the same train and test data in each of your experiments.

Task1 **First classification, only RGB parts:** using the `SingleNetwork` class, change the last linear layer of a pre-trained network (resnet-18), which can then be trained on using only the rgb channels of the images. The data has four channels, so use the class `ChannelSelect` in your transforms to keep only the channels you need. Get this task working first, and if you manage this move on to the following architectures below.

  - with a proper loss for minimizing and for training of the network. note that an image can have multiple labels present in it. **Thus cross-entropy-loss over 17 classes, or any other multi-class loss, is not the right way to do here and will result in a large penalty.** Use a loss which can minimize 17 separate binary classifiers. How to design that ? It is easy if you think about it for 15 minutes.

- Report the average precision measure (google for it, sci-kit learn has it too in a metrics subpackage, you can use that one) on the validation set – for every class of the 17, and the mean average precision over all 17 classes. The average precision is a measure for the quality of a ranking of predictions.

- note in particular: the average precision for one class is computed over the set of all predicted scores for this class. All predicted scores refers to scores for this one class obtained for all samples in the validation set and their labels for this particular class (which is binary, presence or absence). You need to compute average precision using an ordered set of prediction scores (for the whole validation set) and the corresponding binary labels (for the whole validation set). It is not computed in the same way as accuracy (one number for each sample, then averaged over all samples) would be. Therefore: to compute mean average precision, you must collect the set of predictions for all samples, and this for all 17 classes.

Task 2 **tailaccuracies and ranking:** Take the predictions from the model of Task 1. For a class with high AP sort the images according to descending prediction scores of that class. Take a look at the top-10 or top-20 images. You will observe that the top-ranked images look great. Why the top-50 highest scoring images for a given class looks so well when the ranking measure (average precision) is not perfect ? Compute for each of the 17 classes the accuracy of predictions in the upper tail, for 10 to 20 values of t from $t = 0$ if classification threshold is zero, or from $t = 0.5$ if classification threshold is 0.5, until $t = \max_x f(x)$.

$$\text{Tailacc}(t) = \frac{1}{\sum_{i=1}^{n} I[f(x_i) > t]} \sum_{i=1}^{n} I[f(x_i) = y_i] I[f(x_i) > t], t > 0$$

Show in your final report

- for the chosen class above the top-10 ranked images and the bottom-10 ranked images.
- a plot of $Tailacc(t)$ averaged over all 17 classes for 10 to 20 values of t as above. A reasonable choice for t would be such that each t separates a percentage of the validation data. Note how the Tailacc(t) accuracy increases as we look at the more top-ranked results (by increasing the value of t) – this is the explanation.

The point here is to show you, that it is a matter of HCI (human computer interaction) how to deal with the errors of a deep learning system!

- The next two tasks are easy once you solve task1. They use the same loss, dataloader and optimizer. Only the model is modified.

Task3 using the `TwoNetwork` class, use two pre-trained networks (again both resnet-18), one to handle the rgb and the other to handle the near infrared. Concatenate the high level features, and feed these features into a final linear layer.

Task4 using the `SingleNetwork` class, use the pre-trained weights from layers two and onwards. For the first layer, copy the three channel weights and initialise a fourth channel as "Kaiming-He-initialization". This means: one of the input channels (corresponding to the near infrared channel) to all output channels is initialized as per "Kaiming-He-initialization", the other 3 input channels to all output channels are initialized from the pretrained weights. Hint: you need to change for a resnet model named `model` the weights tensor of `model.conv1`

- write a report of 2 to 10 pages on what you did. Aim is so that others would be able to reproduce your results - it should contain what is needed to recode and reproduce your results (including seeds). Note down loss function, learning rate schemes, training procedures, **all relevant hyperparameters used in evaluation and the finally chosen hyperparameters** and so on. Show train and test loss and train and test mAP scores per epoch.

  It does not need to be lengthy, just be self-containing. Short sentences are okay, prettyness of language does not matter. It is not an English language essay contest.

1.3 Deliverables

- training phase: code for training on the dataset with transfer learning
- a pretrained model
- validation phase: code which uses the pretrained model to predict on the validation images and saves those predictions, and which computes the mean average precision over all 17 classes.
- **a reproduction routine: the scores from the pretrained model computed on the fly when we run your code should be compared against the scores which you saved when you ran your code**
- a brief pdf-report containing the following:
  - your name and your matriculation number
  - setting you used to define the last layer
  - what loss you used - in math formulas and the code you used for it. If you use latex, the package minted may help you for showing python code.
  - describes the experimental parameters of the training (learning rate batch size, seed values and anything else necessary to reproduce the training)
  - shows train test curves for the setting which you used to save the model,

  - the report for Tailacc(t), and top-10 and bottom-10 ranked images (see above)
  - novels longer than 10 pages will not be entertained. Brevity over pamphlets.

- put everything: codes, saved model, saved predictions, the pdf and everything else you want to add into one single zip file.

1.4 Coding Guidelines

- path portability: all paths (dataset, pretrained model, saved predictions) must be relative to the root path of the main .py-file
- no absolute paths
- reproducibility: set all involved seeds to fixed values (python, numpy, torch)
- a python file for the code or several files
- it should run using the following steps:
  - unpacking the zip files
  - set **one single path** for the root of the rainforest dataset. This must be documented. Nothing else should be needed to set it up
- code should run without typing tons/dozens/piles of parameters on the command line!! `python` blafile.py

- python scripts. What about jupyter notebooks?

Do you think that whoever will check your code, would like to follow up errors in a notebook ??

+ Repeat this then for 50 students?
Do you think you can write prototype code in jupyter notebooks for any projects which are beyond the very smallest ones?

1.5 Additional caveats:

- We do one thing wrong here!

  A proper training would need to estimate the best epoch to stop using cross-validation, and other hyperparameters. We do something not proper, namely deciding on the best epoch on ones test data. The validation data serves actually as test data here. Therefore we overfit on test data. The right thing would be to perform n-fold cross-validation on the training set. This would result in n classifiers. Then one would score on the test set using an average of those. Reason: keep the time of GPU usages low.

1.6 GPU resources

you have two options: use your own GPU, or use the university provided resources

- ml6.hpc.uio.no
- ml7.hpc.uio.no

How to use them ?

- log in using ssh and your ifi username:

```
ssh proffarnsworth@ml6.hpc.uio.no
ssh nibbler@ml7.hpc.uio.no
```

On windows the builtin Powershell, PuTTY or MobaXterm may help you. On mac you can use ssh as is. I do use windows, but for games :D.

- each of these nodes has 8 GPUs, each with 11 Gbyte GPU Ram. The critical resource will be GPU ram. if you go over the limit, your script will die with a mem allocation error.
- **keep the scripts at a training batchsize of 16 with using a resnet18 - in order to keep mem usage below 2Gbyte**. This does not apply if you use your own GPU, but then keep it below 5Gbyte (in case i got to check your code on my home GPU, alternatively i will reduce your batchsize manually).
- use nvidia-smi to see which on which GPUs scripts are running and how much memory is used on each GPU. Choose a GPU such which has still 2 Gbyte RAM unused.
- to start a script on a specific GPU with numerical number $x \in \{0, \ldots, 7\}$ use the following command below. However this will stop when you log out of ssh. Thus this makes sense only to debug your code.

```
CUDA_VISIBLE_DEVICES=x python yourscript.py
```

- to start a script which does not hang up on logout (on a specific GPU with numerical number $x \in \{0, \ldots, 7\}$), please use

```
CUDA_VISIBLE_DEVICES=x nohup python yourscript.py > out1.log 2> error1.log &
```

What does this do?

- nohup starts the command without hangup
- > out1.log redirects normal output onto out1.log
- 2 > error1.log redirects error messages onto error1.log
- & places the job in the background
- do not start a script when there are already 5 jobs running on it or when it is foreseaable that your 2Gbyte wont fit into this GPU RAM.
- how to kill your own process?

    ps -u proffarnsworth

    ↑ shows only the processes of the user proffarnsworth `https://en.wikipedia.org/wiki/Professor_Farnsworth`

    ps -u proffarnsworth | grep -i python

    ↑ shows only the processes of user proffarnsworth which are python. The -i makes a case sensitive grep search. If you see nothing, then you may have mistyped your command, or you are not using python, or your process has already finished.

    - both of these will show you process ids (PID)s

      kill -9 *PID*

      ↑ kills your process with pid *PID*