The mandatory coding task

Prof. Alexander Binder

November 28, 2025

Submission due 5th of January 2026 11:59pm

Intro

Learning goals:

- data splitting
- writing a custom dataset class (seen in the exercises)
- training a neural net with fine-tuning (seen in the exercises)
- evaluating performance

In summary: Work with a custom vision dataset.

This is kind of the **minimum what one can expect from a graduate** who would start in industry.

Consider the perspective that you start a company and hire Bachelors/Masters. What would you wish that they are able to do ?

What to consider before starting to train

- team size: maximum three (3) students per team.
- you can use the uni leipzig GPUs provided for this project (see info in moodle), or your own GPU or your own CPU. If you use some slow surface notebook, it might get painful though.
- the data (dataset credit: [1]) is in:
  - internet: `https://hf.co/datasets/torchgeo/eurosat/resolve/1ce6f1bfb56db63fd91b6ecc466ea67f2509774c/`
  - cloud: `https://cloud.scadsai.uni-leipzig.de/index.php/s/PPbcaNYpfD5mko2`

pw:

`45::)onosato_hoshoryu_aonishiki`

–cluster: **see moodle** for the path

- labels are defined via the subdirectories

Task 1: data splitting (10%)

- Perform a train-val-test split of the data which depends on a seed parameter, and use a manual seed so that you and us can reproduce the same split your experiments. Use at least 2500 images for training, 1000 for validation and 2000 for testing.

  If you do not have a decent GPU, do not use 20k images for train.

  - split every class separately (stratified splitting). Lots of libraries have routines for unstratified data splitting. For example:

    `sklearn.model_selection.train_test_split`

    `https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html`
  - write code to verify that your obtained splits are disjoint.
  - the code should produce a list of files for splitting, not copying them around, no data file should be moved or deleted
  - If you are slow in coding, but have done the homeworks, that is at most 60 mins of effort to get this split and verification code done. All it needs is to obtained a list of files, read it into a list or numpy array, then to use above routine and to store your splits. It is important that you have practiced it.

    `os.path`

    has useful functions to concatenate and split file system paths, in particular `os.path.join`, `os.path.dirname`, `os.path.basename` are very useful. But there is also Pathlib. `os.walk` might be useful too.

    Note: that when you save filenames with paths in your split files – you should save the filenames starting from a dataset root path and not with their full absolute path (e.g. start paths with EuroSAT_MS). Reason: so that your splitting files remain usable when one moves the dataset to another location.

    Also, whatever files you write, they should end in a subdirectory starting at project root path which is given as a parameter in the code, so that the code can be moved to another location, and stuff still works

Task 2: classification with the RGB channels (50%)

- You can use either the jpg images in EUROSAT_RGB, or the channels with indices 3,2,1 (in this inverse order it is equal RGB in the MS part) in the .tif files in EUROSAT_MS. If you use the multispectral tif files read below in Section 3 how to read them.
- write a dataset class to be used for the train, val and test dataset parts.
- Choose a neural network according to your available resources, train using loading weights and finetuning of all layers.
  - use as seed for torch for training your matriculation number of one of the students in your team
    - reason: to avoid students using a seed of 1 and ending up having the same model submitted putting them under suspicion of plagiarism.

  - try out at least 2 different data augmentation settings for the training data - whatever you want to try. You can try also different strengths of augmentation parameters.

    A very mild data augmentation or just mirroring is also one. But do a bit more effort for the second augmentation.
  - Choose the best model according to its performance on the val part, save it. Predict then on the test set (which you can do with a saved model in a separate python file, see deliverables below).
  - Report the accuracy/ TPR per class on the validation set for each epoch. This should be one graph for each data augmentation-hyperparameter-model setting. Epochs are on the x-axis. You have as many graphs over epochs as you have different data augmentation and hyperparameter settings and models, so at least 2 graphs.
  - Report the accuracy/ TPR per class on the test set for the one finally selected model: this is only 1 result, and no graph, because you select only 1 model taken from 1 epoch (the one with the best validation performance)!
  - save the logits from the selected one model on the test data set.
  - for the finally selected model do a quick check using a ranking approach:
    - using the finally selected model, for 3 different classes find the top-5 and bottom-5 scoring images from your test set. Put these $(5 + 5) * 3$ images into the deliverable report. The top-5 and bottom-5 should look right

- write a reproduction routine: it predicts on the test set (for the one selected model), and then it compares the obtained logits of shape $(n_{data}, n_{classes})$ against the saved logits from the first initial call of the routine (saved above).
  - obviously this routine needs to have a switch to save logits instead of comparing them which is off by default

img credit: EuroSAT Dataset [1].
Example images from the six classes contained in the dataset.

Deliverables for Task 1 and 2

Provide to us as code and data for Task 1 and 2:

- code for data splitting and verification of disjointness of the splits
- code for training

- the saved finally selected trained model
- a py-file with **a reproduction routine:** the logit scores on test data using the saved model – which are computed newly when we run your code – should be compared against the logit scores which you saved when you ran your code
  - the logit scores of the predictions on the test data set, saved as pytorch tensor or tensors + a way to identify to which image the i-th tensor / i-th row in a tensor belongs to, e.g. a list of filenames in a separate file. You will need them for the reproduction routine.
- auxiliary files if needed. Eg. lists of files contained in the train-val-test split needed for your dataset classes
- a max 3 page pdf-file (if it is 1 page, it is fine, too) that states
  - the names of the team of students working on it
  - what parameters need to be set, so that we can run your code
  - what py-file to run to compute the prediction on the test data using the final model
  - what py-file to run the reproduction routine using the final model
  - it shows the validation performance graphs and the final performance number on test data using the saved selected model
  - the top-5 and bottom-5 images
  - any links to cloud storage for things (e.g. saved models) which are too large to upload in elearning.

Make your code so that it needs only two paths to be set in order to run:

- one path for the dataset
- one path for the project root path
  - all your files which are not the data set and which are to be written or to be read ... should be stored relative to your project root path. Consider that we will use linux or mac to check it, not windows.

Task 3: Classification with Multispectral images is not more difficult (40%)

`PIL.Image.open` will not work, opencv imread also wont. Never mind!

You can easily read the images in EuroSAT_MS using `skimage.io` (maybe needs the ubuntu package libtiff5-dev to be installed):

```
from skimage.io import imread

fn = './EuroSAT_MS/Industrial/Industrial_101.tif'

img = imread(fn)
print(img.shape)
```

The pip package name is scikit-image .

Alternatively you can check `https://github.com/cgohlke/tifffile` (which gets installed when one installs scikit-image).

- first things first: image values are now in $[0, 65535]$. You must normalize to floats in $[0, 1]$ yourself, and switch channel order rather than relying on `ToTensor`.
- do the same classification as in task 2, but with the following modification: use 6 channels from EuroSAT_MS such that you can still load weights for all layers except the last linear layer (the paper shows which channels contain much information and which do less).

  Do not use ensembling (= sum of predictions from two networks) for it. It should be one network and one final linear layer.

  - you can do late fusion before the last linear layer. The easiest.
  - you can duplicate the first K conv layers with loaded weights, then sum their outputs with a weight of 0.5 before the non-linearity – at a residual sum operation point, and continue with loaded layers on top. If one would use two sets of RGB channels, that would be an equivalent network.

    This will need to modify the forward pass of the network. Monkeypatching or derived classes may help.
  - you can try something smarter (slowly phased-in cross-attention or whatever)

  How to implement late fusion?

  - create a class `yournet` derived from nn.Module which
    - contains one nn.Linear layer
    - contains one or two neural nets initialized with loaded weights (use the `def __init__(self,...):` to define the neural net in class `yournet`), which will be used to extract features from groups of 3 bands
  - define a forward method for `yournet` which does the following:
    - process 3 channels by a network with a modified forward. This network with a modified forward is the one or two from "one or two neural nets initialized ..." mentioned above.
      - The modification ensures that the computation stops after the adaptive pooling, and results in a feature of shape (batch, numchannels) or (batch, numchannels,1,1) .
    - repeat, for the next 3 channels with the same neural net (or the second of the two neural nets) stored in the class instance of `yournet`
    - Fuse the features by using an addition or a concat of the features (or any other meaningful differentiable combination).

      If you use a concat, the resulting shape would be (batch, 2∗numchannels) = (batch, 6)
    - call the nn.Linear on the fused features to produce output logits
      - if you use one network, then you process each group of 3 bands using the same weights. If you use two networks, then each group of 3 bands gets its own network (you could extend it to any multiple of 3 channels ...)
  - you can use any 6 channels from EuroSAT_MS (or more if you want to) . The bands are:

```
bands = {
    'B01': 'Coastal Aerosol',
    'B02': 'Blue',
    'B03': 'Green',
    'B04': 'Red',
    'B05': 'Vegetation Red Edge 1',
    'B06': 'Vegetation Red Edge 2',
    'B07': 'Vegetation Red Edge 3',
    'B08': 'NIR 1',
    'B8A': 'NIR 2',
    'B09': 'Water Vapour', #AI startup default, if added gold sprinkles :)
    'B10': 'SWIR 1',
    'B11': 'SWIR 2',
    'B12': 'SWIR 3',
}
```

You may need to modify the forward of the neural net(s) stored in the class `yournet`, so that it stops computing at the Adaptive pooling layer. How to do that ?

- monkey-patching (ugly like mpox blisters, but works)
- derived class with modified forward (for those who drink coffee only if the beans are from Java)

Hint: you can use monkey patching to modify a function in a class. Below is an example for monkey-patching.

```
from types import MethodType

class C:
    def __init__(self):
        self.x= 3.0
    def f(self, a, b):
          print(f"Real f called with {self.x=}, {a=}, {b=}.")
# end of class C

#the patch. Note: it has a self variable (!!) and a different call signature
def f_monkey_patched(self,z):
    print(f"Patched f called with {self.x=} and {z=}.")

def run():
    my_c = C()  # init an instance of C
    my_c.f(a=6.0,b=7.0) # call the orig method

    # my_c.f = f_monkey_patched will NOT work!
    #monkey patching here, it does it for an instance of a class
    my_c.f = MethodType(f_monkey_patched, my_c)
    my_c.f(5.0)         # call again
```

Alternatively, you can write a derived class with a modified forward call (if you like more inheritance than monkeys). Just mind that you have to load weights into the feature computation networks (which can be transferred from one network to a compatible one using

```
sdict = anothermodel.state_dict()
model.load_state_dict( sdict )
```

`https://pytorch.org/tutorials/beginner/saving_loading_models.html`

Coding Guidelines

- path portability: all paths except the dataset (pretrained model, saved predictions) must be relative to the project root path

- no absolute paths used!
- reproducibility: set all involved seeds to fixed values (python, numpy, torch)
- one or several python files for the code
- it should run using the following steps:
  - install some required python packages
  - unpacking the zip files
  - set the path for the root of the dataset. set the project root path. In total **two paths** are to be set.

    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. No endless $--arg1--arg2$ parameters in the command line.
- Code must be python scripts. Why we do not accept Jupyter notebooks?

  Do you think that whoever will check your code, would like to click through 500 single steps ?? Maybe repeating those inside a for loop ? + Repeat this then for 50 students?? Do you think you can write code in jupyter notebooks for any projects which are beyond the very smallest ones?

  Jupyter notebooks are ok, if you just need to explore something (and you have a good coding style already!). We have to evaluate code that runs.

Pixabay image Peter Kusnetzov

Leipzig cluster accounts, data location etc

Please see the moodle.

References

[1] P. Helber, B. Bischke, A. Dengel, and D. Borth. Eurosat: A novel dataset and deep learning benchmark for land use and land cover classification. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 2019.

Additional caveats:

To track GPU memory consumption, you can use in linux:

```
nvidia-smi
```

to see how much your job uses and to get your job process id

```
pgrep python
```

or

```
ps -axu | grep yourusername | grep python
```