IN5400 - Mandatory 1

Alex

February 23, 2021

Submission due 21st of March 23:59 = 11:59PM Oslo time.

On passing criteria

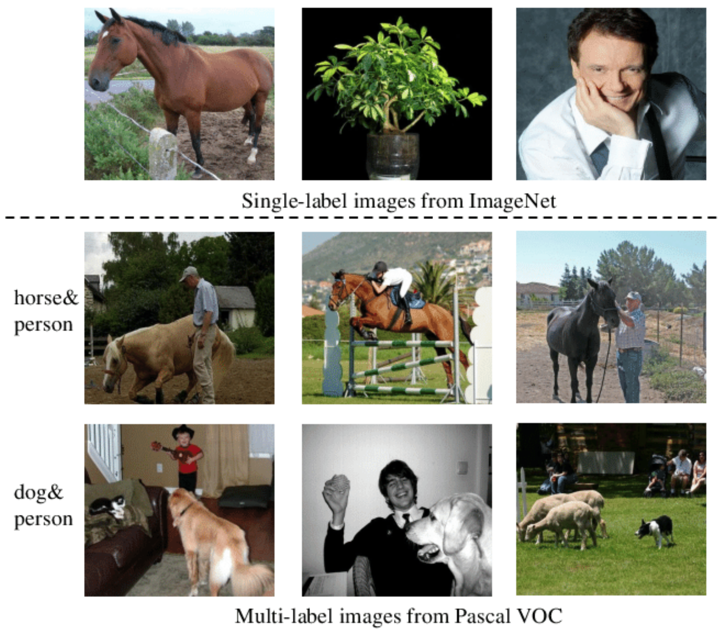
You are able to achieve passing grade if your solution for the part 2 task is wrong, but submit at least an attempt to solution 2 please. **A necessary passing criterion is to submit attempts for both part 1 and part 2.** In case of doubt check the difference between a necessary and a sufficient criterion. Satisfying necessary criteria does not guarantee passing, though.

1 The mandatory exercise 1, part 1: multi-label prediction (90% of evaluation weight)

Learning goals:

- writing a custom data loader
- Define and use a custom loss for a problem which is classification but not multi-class classification
- writing a simple GUI demonstrator in python
- 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: Pascal VOC 2012 (its way less data than MIT places or LSUN, thats why I gave this to you!)


Single-label images from ImageNet
horse&
person
dog&
person
Multi-label images from Pascal VOC

img credit: HCP: A Flexible CNN Framework for Multi-label Image Classification, Yunchao Wei et al.

There are other nice vision datasets: MS COCO, MIT Places, LSUN Challenge (small number of classes) – but they all have much more images, and thus are way slower to train.

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. aeroplane and bird (and a car) 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 20-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, read papers and github code.
- you need to think: what loss to use for training.

1.2 an overview over tasks

- write a dataloader for pascal VOC train and val datasets, you can use the multilabel extractor code `vocparseclslabels.py` provided to avoid dealing with the xml files. Important: one image now is present in sets of multiple classes, so looping over classes of `self.images[class].append(filename)` is a bad idea for a dataset class, because if you would do so, then an image will appear multiple times then in the dataset.

- as model use a resnet-18, but with simultaneous outputs for all 20 classes. Use Transfer Learning.
- come up 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 20 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 20 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 20, and the mean average precision over all 20 classes. (Consider for yourself why accuracy is not an informative measure (e.g. by comparing accuracies vs average precisions).). 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 20 classes.
- create a demonstrator which allows to browse in a GUI the precomputed top-ranked predictions for each class on the validation dataset
- for 5 random classes check visually the top-50 highest scoring images and the the top-50 lowest scoring images. To understand this: you have for every class a binary classifier. You can compute a score for every image for a given class. For a given class then you can sort images according to their scores for a given class.

  In the report show for 3 random classes the top-10 highest scored images, and the top-10 lowest scoring images (both for the same 5 classes). screenshots of the GUI suffice.
- 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 20 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 a plot of $Tailacc(t)$ averaged over all 20 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 how to deal with the errors of a DL system!

- 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, learning rate schemes, training procedures, **all relevant hyperparameters used in evaluation and the finally chosen hyperparameters** and so on. 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 20 classes.
- demo: a GUI for showing the predictions from the step before. It loads the saved predictions for each sample and each class. It supports the following functionality: user can select a class, then it shows the images sorted according to their prediction scores for this class and allows to browse those sorted images forward and backward.
- a brief pdf-report containing the following:
  - your name and your matriculation number
  - parameters you used for 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
  - 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) (see above)
    - writes any software package requirements for the GUI
    - novels longer than 10 pages will not be entertained. Brevity over pamphlets.

- put everything (including codes for the following task 2): 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
- GUI: do not try to run this on the cluster (no ui available there). Anyway it just loads saved predictions and displays them. Develop this on your own computer. For those who never used GUI programming, suggestion is: `https://pypi.org/project/PySimpleGUI/`, `https://pysimplegui.readthedocs.io/en/latest/`.

  If you use any other package: no kivy.
- **It should run on an ubuntu 20.04 for testing without any downgrading of standard packages.** If you use Windows or Mac, feel free to test all the code except the GUI on ml6@hpc.uio.no or ml7@hpc.uio.no . You cannot test the GUI code on ml6/ml7 because it has no window manager installed. But then, if we cant get the GUI to run, but you have submitted GUIcode that reads like it does the right thing, then this alone will not prevent you from passing.
- a requirements.txt for any packages used for the GUI.
- it should run using the following steps:
    - unpacking the zip files
    - installing package requirements for the GUI – the packages used must be explained by you. If installation requires more than pip, then also installation instructions
    - set **one single path** for the root of the pascal VOC dataset. This must be documented. Nothing else should be needed to set it up
- code should run without typing any 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:

- In your gui you need to know which scores belong to which image. Therefore, you may need to store also a file which establishes an association between the order of scores from your validation evaluation and the order of filenames (e.g. with a list of validation filenames or something similar).
- If you use a fixed order for this filelist (e.g. by sorting in the dataset class), then make sure that for evaluation shuffling is off in your dataloader.
- The filenames in this list need to be relative to the root directory. That cannot be absolute paths (see above).
- We do one thing wrong here!

  A proper training would need to estimate the best epoch to stop using cross-validation. 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.
- As for the GUI, if you use PySimpleGUI, then a good example code to start from is: `https://github.com/PySimpleGUI/PySimpleGUI/blob/master/DemoPrograms/Demo_Image_Viewer_Thumbnails.py`. You need to make only very view changes, namely: reading in a sorted list of images according to your predicted scores for a selected class (at program start this can be class $c = 0$), then also add some Gui element to switch selected classes and some gui element to state what class you are looking at.

How good are you compared to the state of the art?

- `http://host.robots.ox.ac.uk:8080/leaderboard/displaylb.php?challengeid=11&compid=2` reports state of the art scores on the pascal VOC test set - this is not the validation set. Thus is it not 100% comparable.
- important: if you consider to submit a result to the pascal VOC test server, note that you can do only 2 submissions per week. Please refrain from spamming it – if you really want to submit, do it if your average precision on val is above 90 or so, and please submit max 2, but hide it from the leaderboard. Imagine if hundreds of deep learning courses would submit ... ?!
- You are not expected to reach the top-scores on this dataset. Not all problems are easy like MNIST. Another point to see that it is easy to get an okay score (see demonstrator) but at the same time many harder problems need rounds of iterative improvement. This kind of iterative improvement is what I would want to teach you in this class if I would have more time for it.

2 The mandatory exercise 1, part 2: changing the network structure (10% of evaluation weight)

Here you will use a Resnet18 for it. The reason is that you need a neural network where each convolution is followed by a ReLU (it is still doable if not, but it gets then far more complicated. Off exercise: for a densenet with chains of type Convolution-ReLU-Batchnorm you can solve it using spatially inhomogeneous bias terms and shifted relus – that is too hard for an MSc course.).

2.1 Task overview

Learning goal here is:

- replace modules in a pretrained neural network by your own modules
- such that the neural network still is capable to predict and to be trained well.
- gain confidence in how to modify a pretrained neural net without breaking it.
- test this on imagenet validation data that it still predicts in the same way
- test this with 3 epochs of training with pascal voc using your code from the first task, that it is still suitable for training

We want you to be able to also modify the network moderately and not only to use it as some "dont touch me" black box.

What is the goal?

- Replace in your network `nn`.Conv2d by a module which does only the part of weight standardization which is about dividing by the standard deviation of weights.

If a usual vannilla convolution computes withhin an input window

$$z = W \star x$$

while using a weight parameter (see `https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html`) which has a shape

$$W.shape = (out\_channels, in\_channels, kernelsize[0], kernelsize[1])$$

then you are asked to implement

$$s[c] = std(W[c, :, :, :])$$
$$\hat{W}[c, :, :, :] = \frac{W[c, :, :, :]}{\sqrt{s[c]^2 + \epsilon}}$$
$$\hat{z} = \hat{W} \star x$$

- However simply replacing `nn`.Conv2d by this modified convolution will distort your loaded weights and result in poor performance. Good news: You can fix this shift by making adjustments in the following batchnorm layer. Details will be explained below.

2.2 steps to be taken for this

- implement the standardized convolution:

```
class wsconv2(nn.Conv2d):
  def __init__(self, in_channels, out_channels, kernel_size, stride,
                   padding, dilation = 1 , groups =1 , bias = None, eps=1e-5 ):
    super(wsconv2, self).__init__(in_channels, out_channels, kernel_size, stride, padding, dilation, groups, bias)

    self.eps=eps

  def forward(self,x):
    pass
    #torch.nn.functional.conv2d, torch.nn.Conv2d documentation tells about weight shapes

    #TODO
    #perform the standardization
    #perform the convolution using  torch.nn.functional.conv2d(...)
    # return the result
```

- think about the math of how to transform the following batchnorm layer.

  For notation purposes lets state that your original convolution-batchnorm sequence is using these constants:

$$z = W \star x$$
$$y(W, \alpha, \beta, \mu, \sigma) = \alpha \frac{z - \mu}{\sqrt{\sigma^2 + \epsilon_2}} + \beta$$

  where W is the convolution kernel, μ would be the running mean at eval time, and the batch-mean at training time, σ (not σ^2 which is a variance) is the running standard deviation at eval time, and the batch standard deviation at testing time. α and β are the trainable batchnorm parameters.

Assume that you are using instead now the standardized convolution

$$n_c = \sqrt{std^2(W[c,:]) + \epsilon}$$
$$\hat{W} = \frac{W}{(n_c)_{c=1}^{C_{out}}}$$
$$\hat{z} = \hat{W} \star x$$

This will require altered parameters in the Batchnorm. There are two ways to alter these parameters, way (A) and way (B)

– (A): change the non-trainable parameters μ, σ from their original values to $\hat{\mu}$ and $\hat{\sigma}$ such that $\hat{y} = y$, where y is computed using the original convolution-batchnorm sequence and $\hat{y}$ is the output of the modified batchnorm given below which takes the normalized $\hat{z}$ as input :

$$\hat{y}(\hat{W}, \alpha, \beta, \hat{\mu}, \hat{\sigma}) = \alpha \frac{\hat{z} - \hat{\mu}}{\sqrt{\hat{\sigma}^2 + \epsilon_2}} + \beta$$

Note further that $\hat{\mu}$ and $\hat{\sigma}$ will depend on the original values of the original batchnorm μ, σ and on n_c

– (B): change the trainable parameters α, β from their original values to $\hat{\alpha}$ and $\hat{\beta}$ such that $\hat{y} = y$.

$$\hat{y}(\hat{W}, \hat{\alpha}, \hat{\beta}, \mu, \sigma) = \hat{\alpha} \frac{\hat{z} - \mu}{\sqrt{\sigma^2 + \epsilon_2}} + \hat{\beta}$$

Note: Same as in (A) note that $\hat{\alpha}$ and $\hat{\beta}$ will depend on the original values of the original batchnorm α, β and on n_c

– the math thinking task here: How does $\hat{\mu}$ and $\hat{\sigma}$ look like in (A) ? How does $\hat{\alpha}$ and $\hat{\beta}$ look like in (B) ? **write down the formulas for them in your pdf report!**

In principle you have to take the modification done in $\hat{W}$, express it in terms of the original W and transport it onto μ, σ or α, β so that $y = \hat{y}$.

– spoiler lookahead: one of these two ways (A), (B) performs poorly during training, while both are fine if you use the network only for evaluation.

- complete/code missing parts in the function which transforms the network in def bntoWSconverter(model).
- Note that you have to replace a convolution by your custom class wsconv2 and to alter the parameters of the following batchnorm in above function.
- a helper: def setbyname2(targetmodel,name,value): is a convenience function for you which overwrites in the model targetmodel a module with name name by the new module value. It exploits that modulenames are separated by dots in pytorch.

```
def bntoWSconverter(model):

  #either you modify model in place
  #or you create a copy of it e.g. using copy.deepcopy(...)
  # https://discuss.pytorch.org/t/are-there-any-recommended-methods-to-clone-a-model/483/17

  lastwasconv2= False
  for nm,module in model.named_modules():
    #print(nm)

    if isinstance(module, nn.Conv2d):
      #replace, get std
      lastwasconv2= True

      usedeps= 1e-12 # use 1e-12 if you add it to a variance term, and 1e-6 if you add it to a standard deviation term

      #TODO
      # put in here your wsconv2, dont forget to copy convolution weight and, if exists, the convolution bias into your u

      setbyname2(model,nm,newconv)

    elif isinstance(module,nn.BatchNorm2d):

      if False == lastwasconv2:
        print('got disconnected batchnorm??')
        exit()


      print('got one', nm)

      #TODO
      # you will need here data computed from the preceding nn.Conv2d instance which came along your way

      #delete
      lastwasconv2= False

    else:
      lastwasconv2= False
```

- test at prediction time that you got the changes right for evaluation/prediction mode! for doing so, for your convenience we have provided: def test_WSconversion():
- **important: this routine assumes that** def bntoWSconverter(model) **modifies the neural network model in-place**. If instead your code modifies a copy and returns a copy, then replace the call to

  ```
  bntoWSconverter(model2)
  ```

  by a call like:

  ```
  model2= bntoWSconverter(model)
  ```

- note: getting an average MAE of orders $1e-3$ or $1e-2$ is okay
- test at training time that you got the changes right: train on pascal voc with the modified network for 3 epochs. if your mean average precision scores are comparable to the training which you have done before, then all is ok. if your mAP of epoch is below 35, then something is wrong.
- do it for mod (A) and mod (B). **write down in your pdf which mod breaks down in training**

- bonus task: think why the one mods breaks down during training.

2.3 Deliverables

- put your work on this task in a separate `.py` file. Put this file into the single zip mentioned above.
- fill all the missing code, in `wsconv2`, in `bntoWSconverter`.
- in main prepare two functions to run (one after the other):

  - `test_WSconversion()` – to show that test time statistics are okay (or are not okay if you had a hard time coding this part)
  - then a function for training and evaluation for 5 epochs with the modified network and the setting which does not break down (as you have done in part 1 already).

3 GPU resources and data

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 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.
- **load your environment:**

```
module load PyTorch-bundle/1.7.0
```

- 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*

3.1 I use my own GPUs, where to get the data?

in the folder /itf-fi-ml/shared/IN5400/dataforall/mandatory1/

- VOCtrainval_11-May-2012.tar contains the pascal VOC dataset. you can download it also from the pascal VOC servers

- for those who use ml6 or ml7: /itf-fi-ml/shared/IN5400/dataforall/mandatory1/VOCdevkit contains the unpacked content of it. its subfolders contain the images and the label files for it
- /itf-fi-ml/shared/IN5400/dataforall/mandatory1/imagenetval300imgs.zip contains the first 300 images from the imagenet validation set and their label xml files.
- for those who use ml6 or ml7: /itf-fi-ml/shared/IN5400/dataforall/mandatory1/imagenet300/ are the 300 images
- for those who use ml6 or ml7: /itf-fi-ml/shared/IN5400/dataforall/mandatory1/val/ are the 300 label xmls
- /itf-fi-ml/shared/IN5400/dataforall/mandatory1/students/ contains python code files for you.

Writing a mandatory exercise ... you can guess how long it took for that pdf, now it is your time :).