# *50.021 -AI*

*Alex*

*Week 06: 50.021 2017 project – prediction of benign vs malignant breast cancer types*

The goal is to implement the full lifecycle of a prediction task, on the BreakHis V1 dataset. **The best submission will be showcased in the commencement ceremony on 9th of September.**

The life cycle consists of:

- training the model, validating the model on the test data of a split
- but also: storing the model, and
- deploying it in a demo use case with a graphical user interface, where users can load cancer images and do a near-real-time prediction.

The dataset with predefined splits into train, val and test is given in an amazon AMI: *Oregon zone, AMI name 50.021project*, use the split 60-12-28. Breakhis v1 is publicly available:

Fabio Spanhol et al. *Breast Cancer Histopathological Image Classification using Convolutional Neural Networks*, IJCNN 2016

Fabio Spanhol et al. *A Dataset for Breast Cancer Histopathological Image Classification*, IEEE Transactions on Biomedical Engineering, 2016. If you use it for something, be so nice and cite people who took time-consuming efforts to create public datasets.

`http://ieeexplore.ieee.org/document/7727519/?reload=true`,
`http://www.inf.ufpr.br/lesoliveira/download/IJCNN2016-BC.pdf`, `http://www.inf.ufpr.br/vri/databases/BreaKHis_v1.tar.gz`
and downloaded from my dropbox
`https://www.dropbox.com/s/tf2e25zyybn2phg/breakhis.zip?dl=0`.
The splits used here can be separately downloaded in
`https://www.dropbox.com/s/mpqhiegvs5m7wkg/breakhissplits_v2.zip?dl=0`

The task can be described as: improve over their result on Image Level for your chosen image resolution, but be able to classify a single image in a GUI. You can use deeper neural nets with finetuning of the last layer, but please see the design choices below.

**SUBMISSION DEADLINE: 3rd of August.**

Upon student request I can give you time in the week 24th-28th July for coding on that, but 5 hours will not be enough!!!
I have validated that with a rather cheap neural network in my not-so-loved tensorflow and a patch-based approach with patches of a certain size one can get on split 1 85% accuracy. It took me in total maybe 9 hours of my life to get there. It is not a killer task.

## *Team up*

You can work in teams of up to 6. You can be a one man army, too. The goal is to improve over the reported results for your chosen image resolution for *Image Level* results in Table IV in `http://www.inf.ufpr.br/lesoliveira/download/IJCNN2016-BC.pdf`

## *Code and Model submission guidelines*

you should submit codes for 3 tasks:

1. code that prepares the dataset (if you use patches, or resize images or do any transformations which you do not do during the training)

2. code for training - the code should report a validation set accuracy at least once every epoch. epoch means here: when you are one time through all your training images.

3. code for A. predicting on all the test set images of at least one split without a GUI and B. reporting the accuracy on the test set images. note: test set, not validation set. You use the validation set during training.

4. code for running a GUI that allows a user to select one image of the test set, runs a prediction on it, and shows the result. `https://wiki.python.org/moin/GuiProgramming` – appJar seems to be the simplest, but use whatever you like, kivy seems to be a gui for games and cefpython to be the HTML5-est framework.

You should submit additionally a pdf document describing

- who is in your team

- on which resolution you worked on - 100X, 200X or 400X – chose one of them
- and on which train/val/test-splits the results are to be reported - there are for every resolution 5 splits available. It is allowed to take the data of one resolution and apply image resizing (shrinking) on the images, in order to improve performance of the prediction, that would still count under the original resolution (bcs if you take a look - then you will see that each resolution has its own field of view).
- a short report on training settings (what model, what parameters for data preparation and training), achieved accuracy on the test split (3 pages max, but you can go up to 6 pages if you embed lots of screenshots).

it is obvious that you will need to test the GUI outside of the amazon VM (unless you find a way to run a GUI on it).

the code should be runnable with only two settings to be made:

- one variable for: the path to the images, one variable for: the path for the code. all files (trained models,code,whatever) should be in subdirectories of those two paths. The code should be runnable when these two paths are set to wherever Yuqi will copy your subdirectory containing all your code and data. Unix supports relative paths starting with `./`. Python has: `os.path.join(basepath,yourrelativepath)`
- installation instructions for additional software packages if needed – which should be installed without sudo/root rights in those subdirectories. You can create a snapshot but note that every snapshot costs you roughly 0.1 USD per Gbyte and month.

  Message us if this is a problem!

  as for installation of software you should never need root rights (except ubuntu packages):

  pip installs without root rights

```
 pip install --user
```

  library paths (.so) can be set by

```
export LD_LIBRARY_PATH=<yourpath>:$LD_LIBRARY_PATH
```

  Same for binaries: executable paths can be set by

```
export PATH=<yourpath>:$PATH
```

Same for python packages: python packages can be locally installed and made visible by

```
export PYTHONPATH=<yourpath>:$PYTHONPATH
```

You can put those environment variables also directly in your $HOME/.bashrc . So that they are automatically loaded when booted (but just editing that file does not include them automatically, either you reboot, or you run a `source $HOME/.bashrc` ).

## *Grading*

The total weight of the project is that of one quiz - 5% of the total grade, the final exam will be weighted 25%. That reduces exam risks :) .

The grading of the project has the following components:

- 65% for working code and one trained model for at least one train-val-test split. The model must be trained only on the train part of the split. you can submit those on a USB stick, or put it in a dropbox.
- 10% absolute performance. The baseline are the results of *Image Level* in Table IV in `http://www.inf.ufpr.br/lesoliveira/download/IJCNN2016-BC.pdf`. You are considered to be on par with their results if you have same performance on the testset as them for the same chosen magnification factor - for their second worst result (example: 100X second worst is 81.0%).

  You get no deduction if you are 5% behind the reported results, and up to 10% deduction if you lag 5%-15% behind the reported result. You cannot lose more than 10%.
- 5% if you have models for all 5 train/val/test splits, and if the reported performance is an average on the test splits of all 5 splits. note: you must have a separate trained model for every split. Dont worry, you can see with neural nets quite easy if you use a model on a test split that used parts of the images of the test split during training. Doing so is considered cheating and will lead to deductions in the 65% part.
- 20% for the GUI that allows users to choose/load images from the test set (file chooser, preset to some folder), classifies them in near-real time, and displays the image and the prediction result.
- 5% bonus if you can train a second model that detects when an image is NOT a HE-stain (like natural photos), and when you can

deploy the second model in the GUI to reject non-HE-stains, before running a cancer classification!! do you want a model that always tried to predict even if the image is wrong (or has the wrong input shape ? or is black-white?) In the end: rejecting wrongly shaped images should be trivial, and training a non-HE reject model should be a simple task. Note: the rejection model can be simple and based on color statistics for example!! – as long as it is halfway reliable ... Note: such a model should not reject any of the images in the test split!!!

There are many image datasets out that can be used to serve as negative samples in a prediction task which predicts whether, you can use the `imagenet2500.tar` if you are totally uncreative.

- 5% bonus if your GUI can display an explanation which parts of the image contribute **this is no trivial task! – the other bonus task of a second model is way easier** to the prediction for benign vs malign cancers. You do not need to go as deep as deconvolution and LRP – an explanation can be done if you train a model that predicts on overlapping patches separately, the resolution would be the stride of the patch extraction.

## *Design choices*

You can train a model on images patches.

What do I mean by that? Well you can extract from every image in the training set patches of a certain size, and use these patches as input images for training.

Where to extract these patches? you can move a window with a stride, for example: extract the first patch having its upper left corner at $(h, w) = (0, 0)$, the second patch at $(h, w) = (0, 50)$, the third patch at $(h, w) = (0, 100)$, then $(h, w) = (0, 150)$, some patch later at $(h, w) = (50, 0), (50, 50), (50, 100)$ and so on. This would be extraction with a stride of 50 pixels.

At test time you take a test image, extract live patches of the same size and same stride, classify every patch, and average the classification scores over all patches from one test image (similar to the idea with center and edge crops).

This patch-based gives two independent design parameters: patch size and the stride of the grid on which you extract image patches (or patches at random positions). Usually one tries out strides that are related to the patch size. No one would try a stride of 5 for patches of size 150. it can be useful to try strides in $[patchsize *$

$0.25, ..., patchsize * 0.75]$ – so that one has an overlap.

You can learn an neural net on whole images (you can downsize the whole images by shrinking with a constant factor) – note that such a model will need lots of GPU mem and you may need to choose a batchsize at training time which is small enough so that training can be done on a GPU (fits into the lousy 12 Gbyte GPU mem ;) ).

learning rate can make a big difference. A too big learning rate can lead to NaNs. If it is big, it can make the validation accuracies oscillate wildly. You can experiment if you want to reduce the learning rate by a factor every $K$ epochs.

batchsize - affects accuracy and memory usage.

To see how much memory you really use:

```
config = tf.ConfigProto()
config.gpu_options.allow_growth = True
session = tf.Session(config=config, ...)
```

can help to see how much GPU memory you really use with your current settings.

If you use a multi-GPU machine ... in order not to grab all GPUs, use

```
CUDA_VISIBLE_DEVICES=0 python yourawesometrainingscript.py
```

for restricting tensorflow on GPU 0.

You can use any model you want. Googlenet/inception and resnets are likely good choices. I used for a quick test an alexnet (lazy but i wanted to use tensorflow, so i did code reuse).

With googlenets and inception models – you can make them usable to process images of almost any size (smaller or larger than the 224 or 299 defaults). convolutional filters do not care about the input image size, and you may need only to make the last pooling layer to be a global pooling layer (that is: no matter what 2-d resolutions $(h, w)$ your channels have before the last pooling layer – by global pooling in the last pooling layer $(c, h, w)$ will be pooled in to $(c, 1, 1)$ – which is a vector of length $c$. this vector of length $c$ can then be fed in into a fully connected layer [possibly a dropout before that] ).

it may help or may not help if at training time each batch has 50% positive and 50% negative images. I dont know.

## *Hints*

You are not required to use tensorflow - but then you must either provide an AMI snapshot (or the equivalent of Digital ocean, google cloud and so on) for testing your whatever framework, or clear and easy installation instructions.

You are not required to use deep learning fully. You can precompute features using a CPU and some neural net model with loaded weights and use whatever toolbox you love to classify the precomputed features. at test time, for the GUI you would need to extract the features live, and apply your classifier. You can get around using a GPU.

Technical note: you can create in amazon a snapshot to stored prepared image patches. it costs you roughly 0.1 USD per Gbyte and month, you can delete the snapshot once you have pretrained models (amazon saves only differential snapshots – so the effective storage usage might be less). For deployment - at test time - when computing predictions on a test image - the code must extract the image patches from the test image on the fly, and cannot rely on precomputed patches.

Try out at first with 50 iterations, whether model training,saving, loading and using the model on the test set works – before you train 100 models for hours on GPUs and realize that you cannot load the saved models.

Did I overlook or forget something?