# *Machine Learning A*

*2022-2023*

## Home Assignment 6

**Christian Igel & Sadegh Talebi**
Department of Computer Science
University of Copenhagen

The deadline for this assignment is **25 October 2022, 18:00**. You must submit your *individual* solution electronically via the Absalon home page.

A solution consists of:

- A PDF file with detailed answers to the questions, which may include graphs and tables if needed. Do *not* include your full source code in the PDF file, only selected lines if you are asked to do so.
- A .zip file with all your solution source code with comments about the major steps involved in each question (see below). Source code must be submitted in the original file format, not as PDF. The programming language of the course is Python.
- IMPORTANT: Do NOT zip the PDF file, since zipped files cannot be opened in speed grader. Zipped PDF submissions will not be graded.
- Your PDF report should be self-sufficient. I.e., it should be possible to grade it without opening the .zip file. We do not guarantee opening the .zip file when grading.
- Your code should be structured such that there is one main file (or one main file per question) that we can run to reproduce all the results presented in your report. This main file can, if you like, call other files with functions, classes, etc.
- Handwritten solutions will not be accepted, please use the provided latex template to write your report.

# 1 Convolutional Neural Networks (50 points)

## 1.1 Sobel filter (20 points)

In this assignment, you should implement a Sobel filter in PyTorch. The learning goals are a better understanding of basic image filtering operations, a deep understanding of how the PyTorch convolutional layers work, and better PyTorch programming skills.

The Sobel filter is a basic operator to highlight edges in images (e.g., see Wikipedia). It convolves the image with two $3 \times 3$ filters, the results of which are combined. Let the matrix $\boldsymbol{I}$ represent a (single-channel) 2D input image and $*$ the convolution operator. Then the Sobel filter computes in a first step

$$\boldsymbol{G}_x = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} * \boldsymbol{I} \qquad \text{and} \qquad \boldsymbol{G}_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} * \boldsymbol{I} \ , \tag{1}$$

which are then combined to a feature map

$$[\boldsymbol{G}]_{ij} = \sqrt{[\boldsymbol{G}]_{ij}^2 + [\boldsymbol{G}]_{ij}^2} \ , \tag{2}$$

where $[\cdot]_{ij}$ denotes the matrix element at position $ij$.

Start by going through the notebook `PyTorch 2D convolutions.ipynb`, which gives some examples on how to work with 2D convolutions in PyTorch and provides some basics on how to read and display images. The notebook requires the image file `diku.jpg`.

Implement the Sobel filter. Use a single `nn.Conv2d` call to generate the two feature maps $\boldsymbol{G}_x$ and $\boldsymbol{G}_y$. For squaring each element in a filter output you can use something like `torch.square`. For combining filter outputs as in (2) you can use `torch.sum`, but you have to set the `axis` argument properly.

Show the important parts of your code (i.e., the Sobel filter) in the report. Apply the Sobel filter to the same input image as in `PyTorch 2D convolutions.ipynb`. Show plots of the two feature maps and the final result (i.e., $\boldsymbol{G}_x$, $\boldsymbol{G}_y$, and $\boldsymbol{G}$).

In the report, discuss how you considered the difference between convolution and cross-correlation (note that the latter is what the "convolutional" layer is doing) in your implementation.

## 1.2 Convolutional neural networks (20 points)

The learning goal of this part of the assignment is to get more comfortable with implementing convolutional neural networks (CNNs) in PyTorch.


Figure 1: Examples from the traffic sign data set.

Consider the notebook `Torch_Traffic_Signs_Basic_Template.ipynb` and the file `GTSRBTrafficSigns.py`. The notebook is a template for training a convolutional neural network (CNN) for classifying traffic signs on the same data as the competition described in Stallkamp et al. (2012). The data is handled by the `Dataset` class in the file `GTSRBTrafficSigns.py`. Recognition of traffic signs is a challenging real-world problem of high industrial relevance. Traffic sign recognition can be viewed as a multi-class classification problem with unbalanced class frequencies, in which one has to cope with large variations in visual appearances due to illumination changes, partial occlusions, rotations, weather conditions, etc. However, humans are capable of recognizing the large variety of existing road signs with close to 100 % correctness – not only in real-world driving situations, which provides both context and multiple views of a single traffic sign, but also when looking at single images as those shown in Figure 1. Now the question is how good a computer can become at solving this problem.

The notebook is supposed to run with GPU support, for example, using *Google Colaboratory*. Even then, executing it will take some time.

The notebook misses the CNN definition. Define a network with a

- convolutional layer creating 64 feature maps using $5 \times 5$ kernels followed by a
- $2 \times 2$ max-pooling layer followed by a
- convolutional layer creating 64 feature maps using $5 \times 5$ kernels followed by a
- $2 \times 2$ max-pooling layer followed by a
- linear fully-connected layer with 43 outputs.

After each convolutional layer, the ELU (exponential linear unit, Clevert et al. (2016)) activation function should be applied. Show the model definition in the report.

You should get the following output when you print your model:

```
Net(
  (conv1): Conv2d(3, 64, kernel_size=(5, 5), stride=(1, 1))
  (pool1): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation
   =1, ceil_mode=False)
  (conv2): Conv2d(64, 64, kernel_size=(5, 5), stride=(1, 1))
  (pool2): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation
   =1, ceil_mode=False)
  (fc2): Linear(in_features=1024, out_features=43, bias=True)
)
```

### 1.3 Augmentation (10 points)

Convolutional neural networks are highly complex models. To reduce the risk of overfitting and to be able to learn difficult tasks with a high input variability, many training examples are needed. *Data augmentation* is a way to enlarge the training data set by artificial training points generated from the available data. Data augmentation refers to applying transformations to the input data–the images–that do not change their labels. If we know, for example, that the classifications of images do not change when the images are rotated, then showing rotated versions of the images during the training process helps to learn this invariance property. In the context of neural networks, this type of data augmentation can be traced back to Baird (1992). It has been used successfully for CNNs, for instance, already in the influential work by Krizhevsky et al. (2012) The learning goal of this part of the assignment is to get more experience in using data augmentation.

transformations: background change and light change

Inspect the notebook `Torch_Traffic_Signs_Basic_Template.ipynb`.

Answer the following questions briefly in your report: Which transformations are applied to the input images? Why is a transformation conditioned on the label?

Please add at least one additional (not completely nonsensical) transformation. Show the corresponding code in your report and briefly explain why you think that this may be a reasonable augmentation given the task.

the transformation in condictioned on the label, because if I rotate an image for example a 9. and I rotate it by 180 degrees, it becomes a 6, so the model should not label it as a 9 anymore, so in this case we know that for label 9 we can't apply that transformation, the same can happen with other examples and other images, so that's why it depends on the label.

## 2 Variable Stars (50 points)

A variable star changes its intensity, as observed by a telescope, over time. This can be caused extrinsically, for example by other objects temporarily occluding it, but also intrinsically, when the star changes its physical properties over time. Figure 2 shows an example. The graph of the varying intensity as a function of time is called the light curve. Variable stars can be further divided into many classes depending on other physical properties. The task we are trying to solve is

to predict the class of a variable star by its light curve. To achieve this, we train a classifier in a supervised setting using labeled data from the All Sky Automated Survey Catalog of Variable Stars (ACVS) (Pojmanski, 2000).

The data considered in the following is based on the study by Richards et al. (2012). We have a training and a test set, in the file `VSTrain.dt` and `VSTest.dt`, respectively, with 771 labeled samples each. Each sample encodes the astronomical properties of a variable star in a 61-dimensional feature vector. The features are listed in Table 2, for a detailed description of their meaning we refer to Dubath et al. (2011) and Richards et al. (2011). The labels indicate the class a variable star has been assigned to. In total there are 25 different classes, see Table 2.

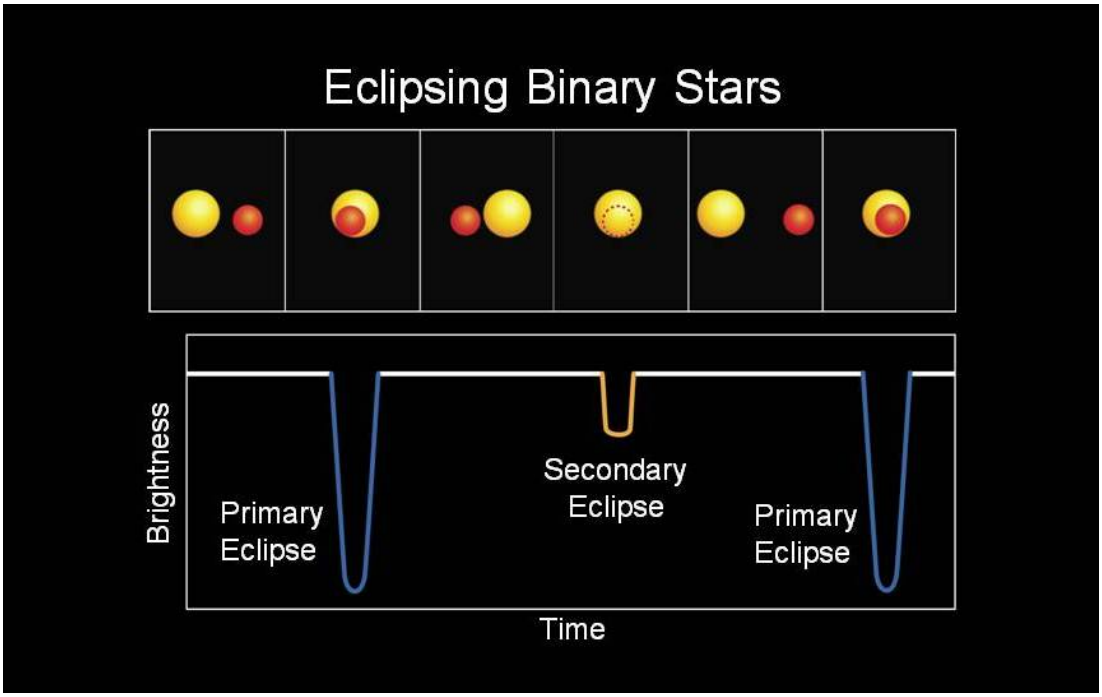


Figure 2: A variable star changes its intensity as observed from a telescope due to another smaller orbiting star. The image is taken from the NASA, `http://kepler.nasa.gov/news/nasakeplernews/index.cfm?FuseAction=ShowNews&NewsID=152`.

## 2.1 Data understanding and preprocessing (8 points)

Download the data in `VSTrain.dt` and `VSTest.dt`. Each line contains the input and as last value in a row the target label.

| # | Feature name | # | Feature name |
|---|---|---|---|
| 0 | amplitude | 31 | freq3_harmonics_rel_phase_2 |
| 1 | beyond1std | 32 | freq3_harmonics_rel_phase_3 |
| 2 | flux_percentile_ratio_mid20 | 33 | freq_amplitude_ratio_21 |
| 3 | flux_percentile_ratio_mid35 | 34 | freq_amplitude_ratio_31 |
| 4 | flux_percentile_ratio_mid50 | 35 | freq_frequency_ratio_21 |
| 5 | flux_percentile_ratio_mid65 | 36 | freq_frequency_ratio_31 |
| 6 | flux_percentile_ratio_mid80 | 37 | freq_signif |
| 7 | fold2P_slope_10percentile | 38 | freq_signif_ratio_21 |
| 8 | fold2P_slope_90percentile | 39 | freq_signif_ratio_31 |
| 9 | freq1_harmonics_amplitude_0 | 40 | freq_varrat |
| 10 | freq1_harmonics_amplitude_1 | 41 | freq_y_offset |
| 11 | freq1_harmonics_amplitude_2 | 42 | linear_trend |
| 12 | freq1_harmonics_amplitude_3 | 43 | max_slope |
| 13 | freq1_harmonics_freq_0 | 44 | median_absolute_deviation |
| 14 | freq1_harmonics_rel_phase_1 | 45 | median_buffer_range_percentage |
| 15 | freq1_harmonics_rel_phase_2 | 46 | medperc90_2p_p |
| 16 | freq1_harmonics_rel_phase_3 | 47 | p2p_scatter_2praw |
| 17 | freq2_harmonics_amplitude_0 | 48 | p2p_scatter_over_mad |
| 18 | freq2_harmonics_amplitude_1 | 49 | p2p_scatter_pfold_over_mad |
| 19 | freq2_harmonics_amplitude_2 | 50 | p2p_ssqr_diff_over_var |
| 20 | freq2_harmonics_amplitude_3 | 51 | percent_amplitude |
| 21 | freq2_harmonics_freq_0 | 52 | percent_difference_flux_percentile |
| 22 | freq2_harmonics_rel_phase_1 | 53 | QSO |
| 23 | freq2_harmonics_rel_phase_2 | 54 | non_QSO |
| 24 | freq2_harmonics_rel_phase_3 | 55 | scatter_res_raw |
| 25 | freq3_harmonics_amplitude_0 | 56 | skew |
| 26 | freq3_harmonics_amplitude_1 | 57 | small_kurtosis |
| 27 | freq3_harmonics_amplitude_2 | 58 | std |
| 28 | freq3_harmonics_amplitude_3 | 59 | stetson_j |
| 29 | freq3_harmonics_freq_0 | 60 | stetson_k |
| 30 | freq3_harmonics_rel_phase_1 | | |

Table 1: Different features are used to describe the light curve of a variable star.

Report the class frequencies; that is, for each class, report the number of data points belonging to that class divided by the total number of data points in the training data.

Then conduct two preprocessing steps.

1. Remove all data points belonging to classes with less than 65 training examples. Report which classes and how many training and test examples remain.

| Label | Class name | Label | Class name |
|---|---|---|---|
| 0 | Mira | 13 | Gamma Doradus |
| 1 | Semireg PV | 14 | Pulsating Be |
| 2 | RV Tauri | 15 | Per. Var. SG |
| 3 | Classical Cep | 16 | Chem. Peculia |
| 4 | Pop. II Cephe | 17 | Wolf-Rayet |
| 5 | Multi. Mode C | 18 | T Tauri |
| 6 | RR Lyrae, FM | 19 | Herbig AE/BE |
| 7 | RR Lyrae, FO | 20 | S Doradus |
| 8 | RR Lyrae, DM | 21 | Ellipsoidal |
| 9 | Delta Scuti | 22 | Beta Persei |
| 10 | Lambda Bootis | 23 | Beta Lyrae |
| 11 | Beta Cephei | 24 | W Ursae Maj |
| 12 | Slowly Puls. | | |

Table 2: The 25 different classes a variable star can be assigned to.

*Hint:* Assuming `import numpy as np`, the Python functions `np.unique(..., return_counts=True)`, `np.where(...)`, and `np.delete(...)` (with `axis=0` on the input data) may be useful.

2. Normalize the data to zero mean and unit variance on the training data set.

frequency of classes in original; the number of classes and number of training and test examples after removing small classes; code snippets for the normalization and data removal in the report

## 2.2 Principal component analysis (20 points)

Perform a principal component analysis (PCA) of the normalized training data.[1] Visualize the data by a scatter plot of the data projected on the first two principal components. Use different colors for the different classes in the plot so that the points belonging to different classes can be clearly distinguished.

scatter plot of the data projected on the first two principal components with different colors indicating the different classes

## 2.3 Clustering (22 points)

Perform clustering using `4-means` and `4-means++` of the training data.

[1]Note that PCA results including the eigenspectrum change due to the normalization.

After that, project the cluster centers to the first two principal components of the training data. Then visualize the clusters by adding the cluster centers to the plot from the previous exercise. pca.transform

Briefly discuss the results.

description of software used; one plot with cluster centers and data points; short discussion of results

# References


H. S. Baird. Document image defect models. In *Structured Document Image Analysis*, pages 546–556. Springer, 1992.

Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and accurate deep network learning by exponential linear units (ELUs). In Yoshua Bengio and Yann LeCun, editors, *International Conference on Learning Representations (ICLR)*, 2016.

Pierre Dubath et al. Random forest automated supervised classification of hipparcos periodic variable stars. *Monthly Notices of the Royal Astronomical Society*, 414(3):2602–2617, 2011.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2012.

Grzegorz Pojmanski. The all sky automated survey. Catalog of about 3800 variable stars. *Acta Astronomica*, 50:177–190, 2000.

Joseph W Richards, Dan L Starr, Nathaniel R Butler, Joshua S Bloom, John M Brewer, Arien Crellin-Quick, Justin Higgins, Rachel Kennedy, and Maxime Rischard. On machine-learned classification of variable stars with sparse and noisy time-series data. *The Astrophysical Journal*, 733(1):10, 2011.

Joseph W Richards, Dan L Starr, Henrik Brink, Adam A Miller, Joshua S Bloom, Nathaniel R Butler, J Berian James, James P Long, and John Rice. Active learning to overcome sample selection bias: Application to photometric variable star classification. *The Astrophysical Journal*, 744(2):192, 2012.

Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. Man vs. computer: Benchmarking machine learning algorithms for traffic sign recognition. *Neural Networks*, 32:323–332, 2012.