HW2 – submission 20.5.19, 23:55

Guide lines

1. Include all your personal details including name, id, and **e-mail address.**
2. You should submit all functions and script files written in **MATLAB or Python**. Your code should be well documented and clear. The code should run from **any** computer and include all path definitions (You should take care of this in the code).
3. Please divide the code by questions.
4. Final report – should include explanations on the implementation and the execution, answers to the questions, results, conclusions and visual results. Do elaborate on all parts of the algorithms/solution. The grades are highly depended upon the analysis depth of the report. **Please submit a PDF file and not a DOC file.**
5. Please post question regarding this HW on the QA forum in Moodle.
6. Eventually submit one zip file including the code + images PDF. HW can be submitted in pairs. In such case only one of the student should submit the solution. The zip file should be named HW#_id0_id1 (for example: HW1_123456789_012345678).
7. Please follow all the submission instruction as describe here.

Good luck!

Introduction to CNN

Dry section:

You should answer in short, 1-3 sentences for each section

1. Working with a convolution network –
   a. Write the output dimensions of every layer for a network with 3 layers:
      - Input: RGB image of size 128X128X3
      - Layer 1: convolution with 64 kernels of size 1X1X3:
      - Layer 2: max pooling of size 2x2:
      - Layer 3: convolution with 32 kernels of size 5X5X64 (no zero padding).
   b. Explain the calculation of a 2D convolution with a kernel of size 1X1X(?) (a simple example will suffice).
   c. Convolve the following sub-image with a normalized 3×3 filter of your choice (select and design a filter). Write which filter you chose. What is the convolution output? Show the results for two options of stride and padding.

$$\begin{matrix} 4 & 1 & 6 & 1 & 3 \\ 3 & 2 & 7 & 7 & 2 \\ 2 & 5 & 7 & 3 & 7 \\ 1 & 4 & 7 & 1 & 3 \\ 0 & 1 & 6 & 4 & 4 \end{matrix}$$

2. Select on architecture suggested for image classification (e.g., one of VGG16, VGG19, AlexNet) and write the input and output sizes of each layer. If you make assumptions, e.g., of the stride and padding, please explain. We recommend you will search online for details- there are many resources.
3. What is overfitting and how can it be recognized during training? Draw a schematic accuracy vs. iterations graph. Discuss the different cases.
4. How do the learned parameters get updated during training?
5. Explain the term batch normalization. Write down the formulation and give the definition for each variable.

Wet section

It might be that this is slightly easier to solve in MATLAB, yet most of the deep learning researchers and developers are working in pytorch\tesorflow, thus it is worth the effort.

For this section you will need to use MATLAB Neural Network Toolbox if you are using MATLAB or pytorch\tesorflow if you are using python. You will not need a GPU (basic one should work) for implementing this section. You will probably need to install the tools above first.

*Keep in mind that some tools are harder to learn and get started with than others. This section was designed so it could be run in MATLAB without previous knowledge of the Neural Network Toolbox.

-Matlab Deep Learning tool box: https://www.mathworks.com/products/deep-learning.html
Examples of MATLAB Neural Network Toolbox: www.mathworks.com/products/neural-network/videos.html
-For installing pytorch: http://pytorch.org/
-For installing tensorflow: www.tensorflow.org/install/

A. Training a simple CNN

In this section, you will train a small CNN on the MNIST dataset. The following tutorials are very useful and contains some instructions on how to set and train NN. **It is highly recommended to follow them.**

- Matlab: https://www.mathworks.com/help/deeplearning/examples/create-simple-deep-learning-network-for-classification.html
- Pytorch: https://pytorch.org/tutorials/beginner/deep_learning_60min_blitz.html
- Tensorflow: https://www.tensorflow.org/tutorials/keras/basic_classification

1. Load the MNIST dataset.
2. Set-up a networks of the following form:
   - Input- MNIST digit of size 28x28
   - Layer1:
     - convolution: 8 kernels of size 3x3 (stride 1, zero padding)
     - BatchNormalization
     - ReLU
   - Layer2:
     - maxPooling: size 2x2, stride 2
   - Layer3:
     - convolution: 16 kernels of size 3x3 (stride 1, zero padding)
     - BatchNormalization
     - ReLU
   - Layer4:
     - maxPooling: size 2x2, stride 2
   - Layer5:
     - convolution: 32 kernels of size 3x3 (stride 1, zero padding)
     - BatchNormalization
     - ReLU
   - Layer6:
     - Fully connected layer with output dimension of 10 (why?)
     - Softmax Layer
3. Define the training loss to be cross entropy.
4. Train the network with the SDGM optimizer (this method is equivalent to the SGD we discuss with the addition of momentum that is beyond the context of out course), with a fixed learning rate of 0.01, minibatch of size 128 and 8 epochs in total.
5. Plot the learning curve:
   a. Plot the loss value as function of the iterations for the training set and validation set.

   b. Plot the classification accuracy as function of the iterations for the training set and validation set.
6. Report the classification accuracy for the test set.
7. Repeat 4-5 with learning rate values of 0.1,0.0001. Discuss the differences.
   Repeat 3-5 with learning rate values of 0.01, this time use the L2 norm as a loss function. Discuss the results and the differences.
8. Repeat 2-5 with learning rate values of 0.01, this time use the following architecture:
   - Input- MNIST digit of size 28x28
   - Layer1:
     - convolution: **2** kernels of size 3x3 (stride 1, zero padding)
     - BatchNormalization
     - ReLU
   - Layer2:
     - maxPooling: size 2x2, stride 2
   - Layer3:
     - convolution: **4** kernels of size 3x3 (stride 1, zero padding)
     - BatchNormalization
     - ReLU
   - Layer4:
     - maxPooling: size 2x2, stride 2
   - Layer5:
     - convolution: **8** kernels of size 3x3 (stride 1, zero padding)
     - BatchNormalization
     - ReLU
   - Layer6:
     - Fully connected layer with output dimension of 10 (why?)
     - Softmax Layer

   Discuss the results and the differences.

B. Using a Pre-trained network

1. Download a pre-trained VGG16:
   For MATLAB you can use this example www.mathworks.com/matlabcentral/fileexchange/61733-neural-network-toolbox-model-for-vgg-16-network (try the code. Most likely you will get an error message. Try installing the network from the link you get in the error).
   For pyTorch you use this link: http://pytorch.org/docs/master/torchvision/models.html
   For tensorflow you can use the slim library (has pretrained VGG16):
   https://github.com/tensorflow/models/tree/master/research/slim
2. Load the images in the attached *birds* folder and display them. Transform the images so they would fit as input to the network and show the transformed images. Think about size, type, normalization and range. Use the images as input to the network. What are the outputs?
3. Find an image on the web, use it as input, and show the output of the network for it.
4. Use one geometric transformation, one color transformation and one filter on your image (the image from section 3). Show the resulting images. Then use them as input to the network and show the outputs. What has changed with respect to item 3?
5. For 2 filters in the first conv layer of VGG16 show the filters and show their response for the 3 images from section 4. Discuss the differences.
6. For every image in *dogs* and for every image in *cats* extract the feature vector of layer FC7. Think on a method to visualize the Vectors, if possible so you can see the differences between dogs and cats. You can be creative or use methods like PCA.
7. Take one image of a cat and one image of a dog from the internet and show them. Extract their layer FC7 feature vector and find the nearest neighbor [=closest vector] from the feature vectors of section 6 (all 20). Present the nearest neighbor image. You can select any distance measure, L1 for example, state it in your answer.
8. Take one image of a wolf and one image of a tiger from the internet and repeat section 7 (again compare to all 20 vectors) for these images (i.e., find the most similar cat or dog of each image and present them).

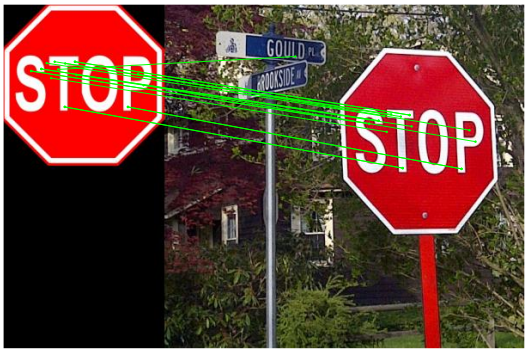
Image Stitching

In this exercise you will implement an image stitching algorithm, based on SIFT descriptors. You are provided with an implementation of SIFT for Matlab (Created by Andrea Vedaldi UCLA Vision Lab - Department of Computer Science, University of California). Due to some problems with SIFT implementation for pytorch, it is recommended to solve this with Matlab, and the instruction are for this specific Matlab implementation. However, you are welcome to use python.

In this exercise, the following Matlab function are **prohibited**:

`cp2tform, imtransform, tformarray, tformfwd, tforminv, maketform, imwarp` or any other matlab function which calculates a transformation or warping.

1. For each image from "Stop images" directory, extract its SIFT descriptor using the function `sift.m`
2. Find matching key-points: use the function `siftmatch` which gets two sets of SIFT descriptors (corresponding to two images) and return matching key points. Show all matching key-points using the function `plotmatches`, for the following image pairs: StopSign1-StopSign2, StopSign1-StopSign3, StopSign1-StopSign4. For example:

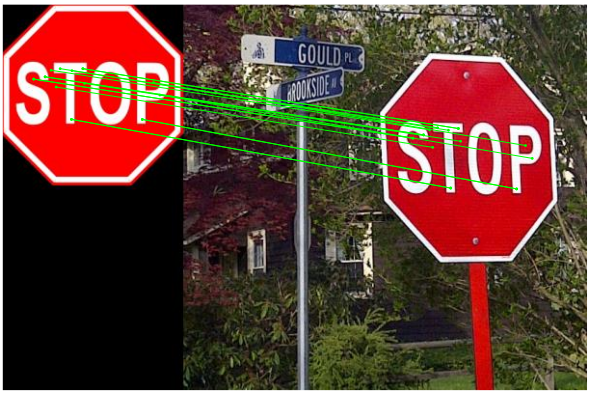


3. Implement a function that gets a set of matching points between two images and calculates the affine transformation between them. The transformation should be H_{3x3} homogenous matrix such that each key-point in the first image, p, will map to its corresponding key-points in the second image, $\tilde{p}$, according to $p = H\tilde{p}$. How many points are needed?
4. Find all the inliers: Implement a function that find the transformation according to all the inliers matches, using RANSAC algorithm. At each of the algorithm iteration:
   a. Randomly choose 6 pairs of matching key-points $\{p, \tilde{p}\}$ and calculate the affine transformation according to them, such that for each $i = 1,...,6, \quad p_i = H\tilde{p}_i$.
   b. Map **all** the coordinates from the first image to the second one. calculates the mapping error:

c. Save the number of inliers in the current iteration: the number of points that have a mapping error of less than 5. **If the number of inliers is maximal, recalculate *H* according to all inliers.**

Repeat for 1000 iterations. Return the transformation H corresponding to the maximal number of inliers.

Show all matching inliers key-points after RANSAC. For example:



useful functions: `imshow, hold on/of, plot, mldivide`

5. **Image warping:** Implement a function that gets an input image and an affine transformation and returns the projected image. Please note that after the projection there will be coordinates which won't be integers (e.g sub-pixels), therefor you will need to interpolate between neighboring pixels. For color images, project each color channel separately.
   Note: imwarp() is not allowed. You need to write your own implementation of warping.

Useful functions: `interp2`

6. **Stitching:** Implement a function that gets two images after alignment and returns a union of the two. The union should be simple overlay of one image on the other. Leave empty pixels painted black.

Show all the results for the pairs mentioned on (2). For one of the pairs, display and explain all the algorithm stages.