Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

computer graphics laboratory

MATHEMATICAL FOUNDATIONS OF COMPUTER GRAPHICS AND VISION

EXERCISE 6 - NEURAL NETWORKS AND MACHINE LEARNING TECHNIQUES

Handout date: 7.05.2019
Submission deadline: 20.05.2019, 23:59
Demo date: 21.05.2019, at the exercise session

General Rules

Plagiarism note. Copying code (either from other students or from external sources) is strictly prohibited! We will be using automatic anti-plagiarism tools, and any violation of this rule will lead to expulsion from the class.

Late submissions will not be accepted, except in case of serious illness or emergency. In that case please notify the assistants and provide a relevant medical certificate.

Software. This exercise is to be implemented in Python programming language.

What to hand in. Upload a .zip file on moodle. The file must be called "MATHFOUND19-*-firstname-familyname.zip" (replace * with the assignment number) and the subject of the email must be "MATHFOUND19". The .zip file MUST contain a single folder called "MATHFOUND19-*-firstname-familyname" with the following data inside:

- A folder named "code" containing your MATLAB or Python code
- A README file (in pdf format) containing a description of what you've implemented and instructions for running it, as well as explanations/comments on your results.
- Screenshots of all your results with associated descriptions in the README file.

Grading. This homework is 8.3% of your final grade. Your submission will be graded according to the quality of the images produced by your program, and the conformance of your program to the expected behaviour of the assignment. The submitted code must produce exactly the same images included in your submission. Code that does not run will receive a null score.

To ensure fairness of your grade, you will be asked to briefly present your work to the teaching assistants. Each student will have 3-4 minutes to demo their submission and explain in some detail what has been implemented, report potential problems and how they tried to go about solving them, and point the assistants to the code locations where the various key points of the assignments have been implemented. See above for the scheduled demo date for this particular assignment.

Goal of this exercise

Introduction and settings

For this exercise you will be implementing a neural network to perform image inpainting. To do so you will be using the high-level API Keras, with the tensorflow backend. You will need the following packages:

- tensorflow-gpu (we recommend using at least 1.10)
- numpy
- scikit-image

If you have never used tensorflow, we recommend installing it using conda rather than pip. To do so, install miniconda (or Anaconda if you want the full download, but it is not necessary), and do the following:

- `conda create --name tf_gpu tensorflow-gpu`
- `conda activate tf_gpu`
- `conda install tensorflow-gpu`

This will create a virtual environnment with your conda installation of tensorflow, which will guarantee that it will not interfere with your other installs. Whenever you want to work with tensorflow:

- `conda activate tf_gpu`

All the installations you do while the virtual environment is active, will be done in this environment. When you are done, you can deactivate it:

- `conda deactivate`

Working on the Leonhard cluster. For this project, you will be granted access to the Leonhard cluster, which will give you access to powerful GPUs. We provide you here with a few useful commands, but we recommend you read the online documentation[1].

You can access it via SSH using your ETH login:

- `ssh username@login.leonhard.ethz.ch`

Leonhard should have a tensorflow install that you can load, so you don't need to install it:

- `module load python_gpu`

If however you face difficulty finding the right modules, you can always create a virtual environment with your own tensorflow installation (if you do so, please do it with a virtual environment).

You can load your code and data on leonhard and submit a gpu job:

- `bsub -R "rusage[ngpus_excl_p=1]" python path/to/your/code`

[1] https://scicomp.ethz.ch/wiki/Getting_started_with_clusters#Leonhard

We refer to the online documentation for more details on job submission and options, but the followin may be usefull to you (place them after bsub, and before your command):

- `-W hh:mm`: to ask for a job running for hh hours and mm minutes (default is 1 hour)
- `-n ${ncores}`: to ask for a job running on `${ncores}` cpu cores

Tensorflow and Keras. Tensorflow is a well known deep learning library, that allows you to train neural networks. Keras is a high level API that allows you to set up and train neural netwoks with tensorflow very easily. We recommend reading the keras guide from the tensorflow webpage[2] until the "Save and Restore" section.

1. Image inpainting using a neural network

Image inpainting is the task of filling missing regions in an image. It is a challenging problem, as it often requires to hallucinate what could be placed in these regions. This is why machine learning and especially neural networks are a promising research direction.

In this Homework, you will be implementing parts of *Context Encoders: Feature Learning by Inpainting* by Pathak *et al.* [1]. You will implement the graph from Fig 9.a. There will be different steps:

- Dataset processing
- Create the network
- Train it

Read [1] and follow the tasks to finish the implementation.

Provided code. We provide you with the framework to reimplement [1]. There are 4 scripts in the folder we provide you:

- *main.py*: the main code to run training / evaluation. You can change the data paths, as well as the number of epochs or the batch size.
- *dataset.py*: the Dataset class that allows you to load and preprocess the data.
- *model.py*: the Model class in which you will create the neural network graph as well as its loss.
- *trainer.py*: the Trainer class which contains the training operations.

The places where you need to code will be indicated with the following comment: `#TODO`.

HINT. Each bit of code may be difficult to evaluate independently, as they are intricated. We recommend starting with the implementation of 1.1, 2.1, and 2.2, and making sure that the code runs. If you have those three, should be able to do the first tasks of 3. Once those three run, it will be much easier to return to 1.2, 2.3.

[2] https://www.tensorflow.org/guide/keras

Data preprocessing

For this task, you will be working on the dataset.py code. The dataset we provide you only contains complete images of size 128 x 128. We now need to create our training data. To do so, we will crop out squares of size 64 x 64 in these images, and keep the crops as ground truth.

Task 1.1: Fixed location. The Dataset class has a member `data` which is a numpy array of size (num_data, 128, 128, 3) containing all the images in `data_path`. Implement the function `crop_center_img` such that it crops a square of size 64 x 64 located at the center of the images. In the paper, the authors specify that the ground truth is slightly larger than the cropped region, by 7 pixels in every direction: follow their example. The variable `data` should be a tuple with the degraded image in first position, and the cropped section in second position.

Task 1.2: Random location. It is possible to crop the square at random locations instead of using the fixed location, without needing to precompute these cropping. Thanks to the `tf.Dataset` class provided by tensorflow, these data modifications can be automated. A `tf.Dataset` is created in the `create_tfdataset`. It contains a `map` method that allows us to automate such operations. You need to pass it a function that takes as input an element of your dataset, and performs the augmentation. Implement the `tf.Dataset` to generate a similar tuple as in task 1.1. In the paper, the authors specify that the ground truth is slightly larger than the cropped region, by 7 pixels in every direction: follow their example.

HINT / REMARKS.

- `tf.Dataset.map` will work only if your function applies to tensors (not to numpy arrays...).
- The variable `use_random_crop` allows you to chose which method you want to use. Even if you cannot finish task 1.2, you can still procede through the exercise using fixed location crops.

Create the network

For this task you will be working with the model.py script

Task 2.1: Create the encoder_decoder graph. Use Keras to implement the encoder decoder graph and store it in its `encoder_decoder_graph` member. Do not forget kernel regularization.

Task 2.2: Create the reconstruction loss. Implement the l2 reconstruction loss described in the paper.

Task 2.3: Overlapping reconstruction loss. As specified in tasks 1.1 and 1.2, the ground truth is slightly larger than the missing region. The predicted area is therefore also slightly larger than the missing region. Implement a loss that penalizes 10 times more the regions where the prediction overlaps the non missing region.

Task 2.4: Question. What is the purpose of these different losses?

Train the networks

For this task, you will be mainly working with trainer.py, though you might need to change a few parameters in the other scripts (*e.g.* to select a loss...).

Task 3.1. Implement `create_full_images` to generate your images. They will be visualized in `show_prediction` and should look like 1

Task 3.2. Train the network with central cropping and the loss from task 2.2. Generate images. You can visualize the loss using tensorboard.

Task 3.3. Implement an evaluation function applies to the test dataset. Generate images.

Task 3.4. Train the network with random cropping and the loss from task 2.3. What should be the main difference in the results compared to task 3.2?

Task 3.5: Questions.

- What other method can you think of to prevent overfitting?
- What is the purpose of kernel regularization?
- What other form of preprocessing could we perform on the data to make training more stable?

Bonus: Adversarial loss

In the paper, the authors also use an adversarial loss. You can also implement it as a bonus exercise for this homework. To do so, in model.py change `self.use_adv` to `True`.

Task 4.1. Use Keras to implement the discriminator graph and store it in its `create_discriminator_graph` member.

Task 4.2. Implement the training of the adversarial lost in trainer.py, `train_adv_loss`.

References

[1] Deepak Pathak, Philipp Krähenbühl, Jeff Donahue, Trevor Darrell, and Alexei Efros. Context encoders: Feature learning by inpainting. 2016.

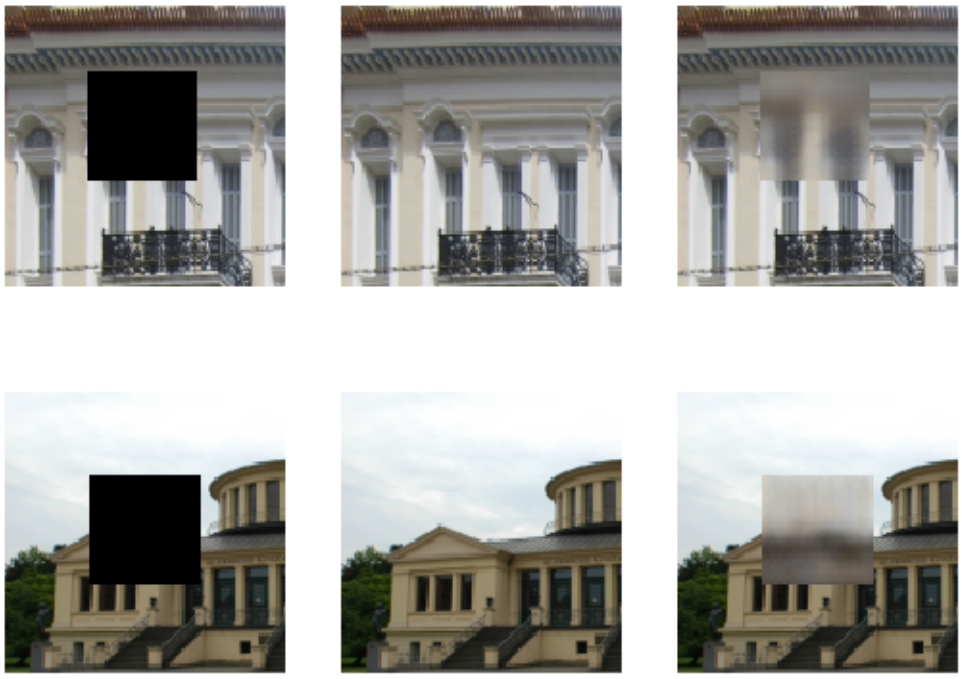

FIGURE 1. First row is an example on training set, second row on the validation set