Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

MATHEMATICAL FOUNDATIONS OF COMPUTER GRAPHICS AND VISION

EXERCISE 6 - NEURAL NETWORKS AND MACHINE LEARNING TECHNIQUES

Handout date: 12.05.2019
Submission deadline: 25.05.2019, 23:59

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 "MATHFOUND20-*-firstname-familyname.zip" (replace * with the assignment number) and the subject of the email must be "MATHFOUND20". The .zip file MUST contain a single folder called "MATHFOUND20-*-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.

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 (we recommend using at least 2.1.0)
- numpy
- scikit-image

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 the commands that you will need to use. We also recommend looking at the online documentation if you want to understand how Leonhard works[1].

You can access it via SSH using your ETH login:

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

You need to set your environment by loading the proper version of python, cuda and cudnn that you will need to run tensorflow 2.1.0.

- `module load python_gpu/3.6.4 cuda/10.1.243 cudnn/7.6.4`

To run your code on GPU, use the following command:

- `bsub -W hh:mm -n 4 -R "rusage[ngpus_excl_p=1]" python path/to/your/code`

where `hh:mm` is the number of hours (`hh`) and minutes (`mm`) during which your code will be running (you cannot have your code running indefinitely on the cluster).

NOTE. If you have access to a good GPU and prefer working with it, this is of course perfectly fine. Using leonhard is absolutely not mandatory for this homework.

Working with a virtual environment. We recommend using a virtual environmnent. If you have never used a virtual environment, `virtualenv` is a good choice. The principle is that it creates an isolated python environment, that come in the form of a folder where all your packages installed, without risking interfering with your system.

First time. You first need to download virtualenv. To do so, you first need to select python 3.6.4 on leonhard. Then you can use pip to download it.

- `module load python_gpu/3.6.4`
- `pip install --user virtualenv`

The next step is to create a virtual environment. You can create it in your Home folder.

- `virtualenv ~/hw6_env`

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

You should see a new folder appear in your Home, called `hw6_env`. You know need to activate it, before you can proceed.

- `source ~/hw6_env/bin/activate`

You can verify that the environment is active by typing the following command:

- `which python`

If the environment is active, then you should see the following output: `~/hw6_env/bin/python`. To deactivate it, you can simply enter the following command: `deactivate`.

Everytime you connect to leonhard. Next time you connect to leonhard and want to work on your homework, you need to set the leonhard environmnent and activate your python virtual environment.

- `module load python_gpu/3.6.4 cuda/10.1.243 cudnn/7.6.4`
- `source ~/hw6_env/bin/activate`

Python packages. Once you have activated your python environment, you can download the necessary packages.

- `pip install tensorflow scikit-image argparse`

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.

This homework is not a deep dive tensorflow tutorial. The only thing you need to know will be how to build a model using the keras Sequential class. For the rest. You can read the beginning of the tensorflow beginner's tutorial to see how to use `Sequential` to create a model[2].

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28, 28)),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Dense(10)
])
```

[2] `https://www.tensorflow.org/tutorials/quickstart/beginner`

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** (we reproduce it here in Figure 1.

Provided code. We provide you with the framework to reimplement [1]. There are multiple files, however you will only need to work on the `homework6.py`. To run the code, you will need to run the `main.py` script:

```
python main.py
```

Test your code. The `main.py` code is made to run once all the functions in `homework6.py` have been implemented. This is why we provide you with a jupyter notebook, `TestYourFunctions.ipynb`, that you can use to test your function and see if they behave as expected. You don't need to modify the notebook, if your functions are implemented correctly, the cells should run normally.

You do not need a GPU to test your functions, so you can perfectly run the notebook locally, on your own computer, even if you do not have a powerful GPU.

First network

Task 1: Fixed location. To begin with, you need to generate the dataset. Currently you have only access to complete pictures. Therefore you need to write a function that takes an image as input, and crop out a section. For this first task, you will crop at a fixed location, namely the center of the image.

- Modifiy the `crop_image` function to crop a 64×64 section at the center of the image. It should return the 128×128 input image with a missing region at its center, and a 64×64 image corresponding to the groundtruth cropped region.
- For this task, you should ignore the `random` parameter, this will be used in Task 6.

HINT. 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 (section 5.1 first paragraph). This means that the missing region in the input image is smaller than 64×64.

Task 2: Build the network. You now need to create the network that you will train. You need to create a `Sequential` model that reproduce Fig 9.a from the paper.

- Modify the `create_reconstruction_model` to create the network from Fig 9.a (reproduced here in Figure 1) without the adversarial discriminator, using tensorflow and Keras: it takes as input an RGB image of size 128 and outputs an RGB image of size 64.

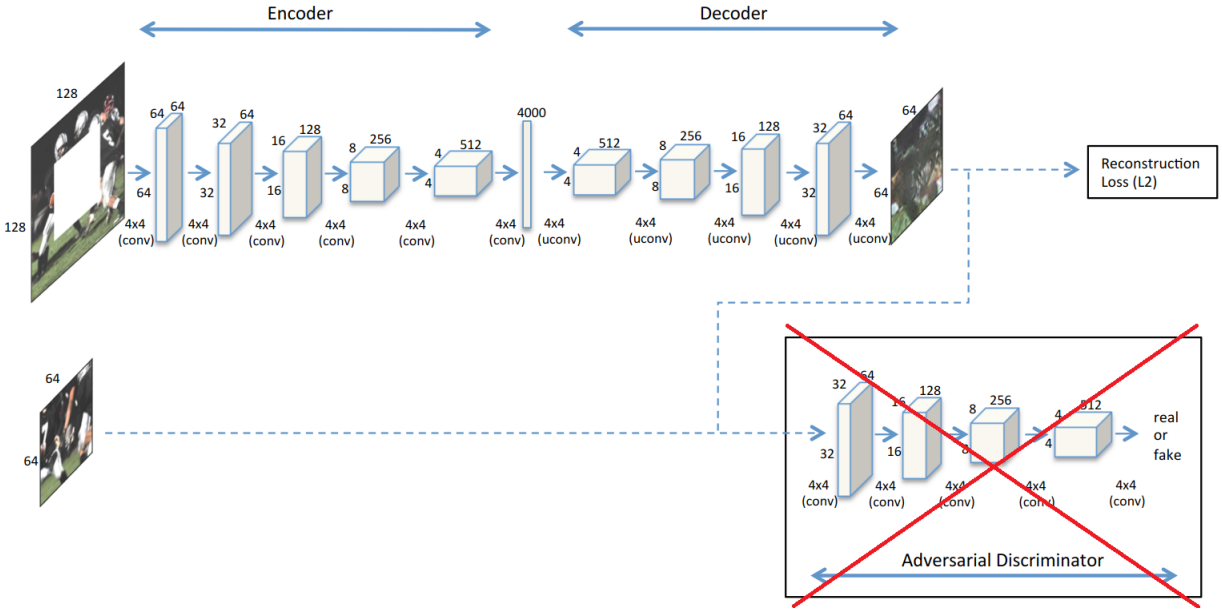


FIGURE 1. **The inpainting network that you will implement**. Figure 9.a. in the paper. For this homework, we ignore the adversarial discriminator.

HINT.

- In section 4 of the paper, the authors specify that they use strided convolutions instead of pooling layers.
- The supplementary section A gives you information about what activation functions are used.

Task 3: Loss function. You cannot train your network without a good loss function.

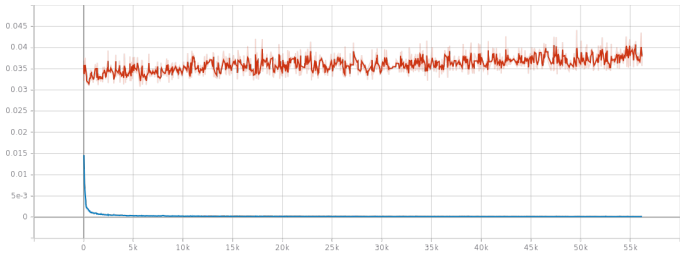
- Modify the `reconstruction_loss` function to compute the reconstruction loss between the predicted region and the groundtruth (**not the adversarial loss**). Do not forget to give more weight to the error made in the area that overlaps with the input image (section 5.1 first paragraph).

Task 4: Visualize the results. In order to have a good look at your result, create a function that combines your predicted region and the input.

- Modify the `reconstruct_input_image` to combine a predicted region with the input image.

Task 5: Train the model. To train the model, run the main script:

- `bsub -W hh:mm -n 4 -R "rusage[ngpus_excl_p=1]" python path/to/main.py --out_dir path/to/output/directory`



(A) **Reconstruction Loss** on training set (blue) and on validation set (red).

(B) **Reconstruction results** on example from the training and validation set.

FIGURE 2. Results when cropping the center of images.

The output directory will be created by the script. If you are familiar with tensorboard, you can use it to visualize the evolution of training. Enter the following command on leonhard to see the results.

```
tensorboard --logdir path/to/output/directory/logs
```

However, don't worry, if you are not familiar with it, or run into trouble using it on a remote server, the loss are saved in a csv file in `path/to/output/directory/logs`. You also have some reconstruction results saved in `logs/image` for visualization.

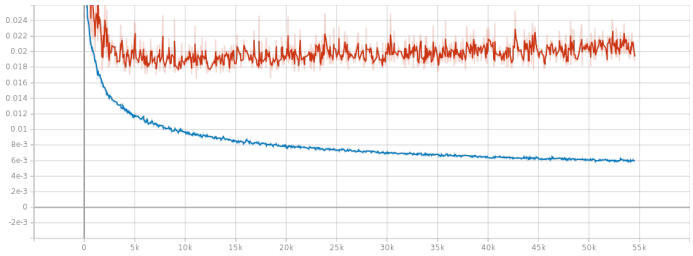
- Generate the loss graph and show some examples as in Figure 2. You can either use tensorboard, or the logs.csv file for the loss. Images from the training and validation set are automatically saved in output/directory/logs/image.
- When comparing the results on the training set and on the validation set, what problem do you see appearing? How is it called? What caused it?

HINT. You don't have to train until convergence, or for as many epochs as shown in the graph. We recognize that it can take time and that you have other duties to attend to. You can train your network for 2000 - 3000 epochs. What is important is to see your loss graphs display similar behaviour as those of Figure 2. You can change the number of epochs by adding the following argument to the command: `--num_epochs ${YOUR_NUMBER}`.

Task 6: Random crops.

- Modify the data generation method, so that the crops are randomly selected in the image if `random` is set to `True`.
- Retrain the model.
- Generate the loss graph and show some examples as in Figure 3. You can either use tensorboard, or the logs.csv file for the loss. Images from the training and validation set are automatically saved in `output/directory/logs/image`.
- Do you observe any improvements?

HINT. As for the previous question, it is OK not to train until convergence.



(A) **Reconstruction Loss** on training set (blue) and on validation set (red).

(B) **Reconstruction results** on example from the training and validation set.

FIGURE 3. Results when cropping random squares of images.

Task 7: Test the model. The final step is applying your trained model to the test set. This is the dataset that is usually used as a benchmark. The framework we provide you saves the trained weights and parameters regularly during training.

- Run the following command:
  ```
  bsub -W hh:mm -n 4 -R "rusage[ngpus_excl_p=1]" python path/to/main.py
    --out_dir path/to/output/directory --test
  ```
- Results are saved in `path/to/output/directory/test`. Report the average loss that is saved in `logs.txt`, as well as the saved examples that you obtain.

HINT. This task is here only for the sake of going through the complete training pipeline. Don't overthink it, simply run the command and report the results, there is nothing more to it. :)

Task 8: Questions.

- What is the purpose of the validation set?
- What methods can you think of to prevent overfiting?
- What kind of preprocessing could we perform on the data to make training more stable?
- In the paper, the authors also introduce an adversarial loss:
  - Explain in 2-3 sentences the principle of adversarial losses.
  - Why is it supposed to improve the training of the network?

REFERENCES

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