Assignment 1: Computer Vision

María Alejandra Bravo and Yassine Marrakchi

Due 04.05.2021

1 Introduction

NOTE: it is possible that the technical setup of the assignment (i.e. how to obtain/store the datasets) will change in the first days, as we are trying to optimize it. Therefore please make sure to check to always have the latest version of this document available from ILIAS.

The assignment is divided into 2 parts, both based on the uploaded code, which you will need to complete and run according to the instructions below. To complete the assignment you need to submit:

- A short (2-3 pages) report containing your results and analysis according to the instructions for each task.
- A copy of the (working) code you used to obtain those results.

In both experiments you will work with subsets of the COCO dataset[1], in particular with the segmentation annotations. Before starting, make sure to set up the working environment as described in the `README`.

2 Task 1. Self-supervised Learning and Binary Segmentation

The goal of the first task is to train a feature representation using the self-supervised learning method described in the lecture, and to use the learned representation to train a downstream network for binary semantic segmentation.

2.0.1 Self-supervised Pretraining

- Download and extract the dataset at:
  `https://lmb.informatik.uni-freiburg.de/lectures/dl_ss21/unlabelled_dataset.zip`
  which contains cropped instances of objects belonging to 5 different COCO

[1] http://cocodataset.org

classes. (Be aware of the "_" when copying the path) To extract the data use the password: "`dl_lab`".

- Download the weights initialization at:
  `https://lmb.informatik.uni-freiburg.de/lectures/dl_ss21/pretrain_weights_init.pth`.
  It contains a set of parameters that you can use to start your training from, so you don't have to run the networks for an excessively long time.

- Complete the `ImgRotation` class in `data/transforms.py`, which is responsible for producing the rotated images and corresponding labels.

- Complete the script `pretrain.py`. It contains the code for pre-training and testing the feature extractor using rotation prediction. The main model is a `ResNet18` network.

- Run the pre-training process starting from the weight initialization as indicated in the code. You shouldn't need to change hyperparameters from the default ones in the script arguments. Save your network snapshot after every epoch (or only the best ones), you'll need them later.

- Report the best validation results in terms of loss and classification accuracy. Include plots of the loss for both training and validation[2].

- Complete the script `nearest_neighbors.py` to run NNs analysis as discussed in the lecture. Do you think that the network has learned meaningful feature representations? Motivate by including qualitative examples in the report.

2.0.2 Binary Semantic Segmentation

- Download and extract the segmentation dataset at:
  `https://lmb.informatik.uni-freiburg.de/lectures/dl_ss21/segmentation_dataset.zip`.
  It contains images from the COCO dataset, and annotations for the segmentation masks.

- Complete the script `dt_binary_ss.py`. It contains the code for training and testing a network for binary semantic segmentation. The decoder network used is from the `DeepLabV3` architecture, it uses dilated convolutions to increase the decoder's receptive field. The positive class of the segmentation task is actually a meta-class for the classes: person, car, bicycle, cat, dog.

[2]NOTE: to log results and images you can use the `tensorboard` tool. You can install it as a python package in your virtual environment using `pip install tensorboard`, and use the class `SummaryWriter` from `torch.utils.tensorboard` to log the results. The logs can be visualized using a web browser, after having launched the tensorboard server with `python3 -m tensorboard.main --logdir=<path to log>`. More info at `https://pytorch.org/docs/stable/tensorboard.html`.

- Train the segmentation network starting from the best network snapshot from the pre-training task. Save your network snapshot after every epoch (or only the best ones), you'll need them later.
- Report your best validation results in terms of loss and mIoU. Include plots for both training and validation, as well as qualitative examples (images).

3 Task 2. Attention-based Semantic Segmentation

The objective of the second task is to compare an attention based method with a classical pixel-wise classification method for semantic segmentation. Here you will have to deal with attention mechanisms to get an intuition on how they work and where to apply them. To complete this task you will have to follow some steps. Record your loss and mIoU score for all training processes. If you like you can use tensorboard to plot the scores while training and to visualize examples.

3.0.1 Classical pixel-wise semantic segmentation

We will first train a classical method for semantic segmentation.

- Download the weights initialization at:
  `https://lmb.informatik.uni-freiburg.de/lectures/dl_ss21/binary_segmentation.pth`.
  It contains a set of parameters that you can use to start your training from, so you don't have to run the networks for an excessively long time.
- Write the script `dt_multiclass_ss.py` to train a multi-class semantic segmentation, base your code on the previous task. Make the modification in the output layer to obtain, instead of a binary segmentation, a segmentation for the 5 different semantic objects in the dataset provided. Remember that the network does not produce a background class by default. Start the weights from the best-performance model obtained in Task 1 (Section 2.0.2) for the binary segmentation and compare starting also from the downloaded weights.
- How does the network perform on the semantic segmentation task compared to the binary segmentation? Report the best validation results in terms of loss and mIoU. Include plots of the loss and mIoU for both training and validation.

3.0.2 Attention-based semantic segmentation

Now we will train a semantic segmentation method conditioned to the class using the attention mechanism.

- Implement the `General Dot-Product Based` attention mechanism by completing the corresponding function in `models/attention_layer.py`. Make sure it works by running the same script.

- Add the attention layer in your model so that when adding the class as a hot encoded vector your output segmentation corresponds to this class. For this, complete the script `models/att_segmentation.py`. Tips: place the attention layer at a point in the network where the semantic information is higher. Check that the dimensions correspond when calculating the attention.

- Train the model with the attention mechanism by running `dt_single_ss.py`. During training, save the loss and mIoU for both sets, training and validation, and plot the scores.

- Visualize the attention maps of the best model and compare to the predictions and labels. Draw conclusions about what the attention mechanism is doing.

- Compare the classical method of Section 3.0.1 with the Attention-based semantic segmentation method by running all the validation images in the dataset and calculating its mIoU. Compare quantitatively and qualitatively both models. Explain your results.