Middle East Technical University | Department of Computer Engineering

CENG 483

Introduction to Computer Vision

Fall 2021-2022

Take Home Exam 3

Object Recognition

Due date: **30 January 2022, 23:55**

1 Objectives

The purpose of this assignment is to familiarize yourselves with convolutional neural networks (CNN) by solving a image colorization task. The learning outcomes of the assignment are getting familiar with CNN's, RGB color space, automatic image colorization, and some tricks to increase the training performance.

Keywords: *Image Colorization, Convolutional Neural Network, RGB, Hyperparameter Tuning, Batch Normalization, ReLU, tanh*

2 Specifications

In this assignment you are required to implement an image colorization (IC) system based on convolutional neural networks, and to evaluate it with the provided dataset. The text continues with detailed explanations of the methods and requirements.

2.1 Image Colorization

The task of coloring black & white images is predominantly carried out by artists. Since an artists has the common sense knowledge, she/he can easily predict the color of image regions. However, as the number of images to be colored increases, manual colorization becomes infeasible. At this point automatic image colorization systems come into play. The main purpose of the IC systems is to color provided gray scale images such that the result is a realistic color image of the same scene.

In this assignment you are required to construct a CNN with mean squared error as loss function(1). The goal is to learn a function, implemented as a neural network, to predict the missing color information from an input grayscale facial image. For this purpose, the network will be trained using the provided gray scale images of faces by using colored versions as ground truth ($\hat{y}$). Fig. **??** represents an example for image colorization.

$$L = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2 \tag{1}$$

Figure 1: Example for image colorization.


(a) Grayscale image.


(b) Color image (groud truth).


(c) Sample colorization.

2.2 Programming Tasks

You are required to implement the aforementioned IC system using a convolutional neural network. The following items summarize the details of the procedure.

- The network should take grayscale intensities (normalized to the range between -1 and +1) as input for an image in size 80x80.
- The network should output R,G and B channels for the color image in size 80x80, again in the range from -1 to +1 for each pixel color value.

The provided source already does the normalization of the input and target images to range [-1,+1].

2.2.1 Candidate Hyperparameters for the Baseline Architecture

Taking the explanations above into consideration, you have some candidate hyperparameters (regarding both the network structure and optimization) to experiment on. To achieve better understanding of these, you are expected to read the provided source code files carefully and fully understand them (read pytorch documentation where needed). The properties of your network should be:

- 1, 2, or 4 conv layers;
- all conv layers must have the same kernel size, which can be 3 or 5;
- all conv layers (except the last conv layer) must have the same number of kernels, which can be 2, 4, 8;
- each conv layer (except the last conv layer) must be followed by a ReLU activation (the last conv layer shall not have an activation function);
- learning rate can be between 0.0001 and 0.1;
- the number of epochs shall be less than 100.

Choose a "good-enough" combination of these hyper-parameters / options (based on the validation set). You may greedily choose a good architecture, or, by taking random combinations of all hyper-parameters, as discussed in the class, and explain the procedure you use in a detailed way in your report. (You are **not** expected to run all possible combinations!) Exceptionally:

- Your code must be automatically choosing the number of epochs, by testing the validation-set performance every few epochs.
- There is no restriction or requirement on the batch-size, as long as you choose a value that overall works well and fast (eg. 16.)

2.2.2 Further Experiments

Once you finalize your baseline architecture above, do the following experiments step-by-step, report your results and discuss what you observe:

- Try adding a batch-norm layer (torch.nn.BatchNorm2d) into each convolutional layer. How does it affect the results, and, why? Keep it if it is beneficial.
- Try adding a tanh activation function after the very last convolutional layer. How does it affect the results, and, why? Keep it if it is beneficial.
- Try setting the number of channels parameter to 16. How does it affect the results, and, why? Keep it if it is beneficial.

Note: while doing these experiments, you may keep the other details of the architecture and other hyper-parameters as constant. However, make sure that you still tune the number of epochs hyper-parameter in each case.

2.2.3 Report

Using the best model that you obtain, report the following(also given in the report template):

- the automatically chosen number of epochs.
- the plot of the training mean-squared error loss over epochs
- the plot of the validation 12-margin error over epochs (for details, see 2.3. Since you will be collecting points based on this error, it is better for you to tune your network based on this, rather than the MSE loss)
- at least 5 qualitative results on the validation set, showing the prediction and the target colored image.
- discuss the advantages and disadvantages of the model, based on your qualitative results, and, briefly discuss potential ways to improve the model.

Note that an epoch is completed when all of the training examples are used to update network parameters. Through the epoch one can compute the average training loss. After an epoch is finished, a full pass over validation set can be used to compute average validation loss. If computing the validation loss takes too much time in your computational resources, you may compute it a bit less frequently, for instance, every 5 epochs.

2.2.4 Hints

You may use Google Colab's free GPU to speed up the experiments. The provided code is (mostly) ready to run on the GPU. The following items are some hints and tricks that you can utilize while implementing your system.

- The provided source code (see `template.py` and `hw3utils.py`) gives you a good starting point, however, it does not provide all the functionality you need. You are expected to understand the given code and extend it as needed to complete the assignment.
- Saving the model weights and optimizer state for some epochs may be useful when you want to start from a pretrained point.
- Setting a seed for random number generators allows the experiments to be repeatable since whenever it is run, it starts from the same random parameters.

2.3 Evaluation

After training the network, there comes the evaluation part. In evaluation, your system should output a file named "estimations_test.npy" for test set that contains a serialized NumPy array holding colored images for each gray scale image (see numpy.save and PyTorch tutorial for the NumPy bridge). The array shape must be **(100, 80, 80, 3)**, where 100 is the number of test samples and the remaining is the shape of a single RGB image. Since test dataset contains 2000 images you need to select 100 of those and write the paths of those images in order to be able to compare them with the ground truth images (see inside of "hw3utils.py" to have an idea of how to save paths of images). The order in the input should be preserved. Note that values in the array must be in the range **[0,255]**.

To determine the accuracy of your system, you can use the provided "evaluate.py" script with the serialized outputs. It computes the ratio of correctly estimated pixels to all of them. In this context, correct means having estimated with an **error margin of 12**.

2.4 Dataset

The dataset consists of 5001 training, 2000 validation and 2000 test images of size 80x80. You should be using grayscale versions of the images as input to the system, where the colored versions should be used as ground truth value.

3 Restrictions and Tips

- Your implementation should be in Python 3.7.
- You are required to use PyTorch library (v1.1 or higher) to implement neural networks and train them. You are allowed to use all the functionality provided by PyTorch (and NumPy).
- It is a good idea to use Anaconda for PyTorch installation (see pytorch.org).
- Do not use any other Python repository files without (i) getting permission from us to use them, (ii) referring to them in your report.
- Don't forget that the code you are going to submit will also be subject to manual inspection.
- Please stick with the given report template.
- Dataloader normalizes images into [-1,+1] range, but you should map the images back into [0,255] range just before saving them.
- Please do not forget to take a look at "template.tex" file to see additional requirements and report format in general.
- You can form a group of 2 person to make collaboration in this assignment. Do not forget to include both Student IDs in the template.tex file in that case.

4 Submission

- **Late Submission:** As in the syllabus.
- Implement the task in a directory named **the3**. The implementation, estimations for the test set (which will be uploaded a day before the deadline) and the report should be uploaded on ODTÜCLASS before the specified deadline as a compressed archive file whose name is <**student_id**>_**the3.tar.gz**, e.g., 1234567_the3.tar.gz.

- The archive must contain the implementation directory, loss history plot, 12-margin history plot, estimations_test.npy for the test set and the README file if needed.
- Do not include the database and unmentioned files in the archive.

5 Regulations

1. **Cheating: We have zero tolerance policy for cheating**. People involved in cheating will be punished according to the university regulations.
2. **Newsgroup:** You must follow the course web page and ODTÜCLASS (odtuclass.metu.edu.tr) for discussions and possible updates on a daily basis.