Middle East Technical University | Department of Computer Engineering

CENG 483

Introduction to Computer Vision

Spring 2017-2018

Take Home Exam 3

Image Colorization

Due date: **01.06.2018 - 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 and CIELab color spaces, automatic image colorization, and some tricks to increase the training performance.

Keywords: *Image Colorization, Convolutional Neural Network, RGB, CIELab*

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.

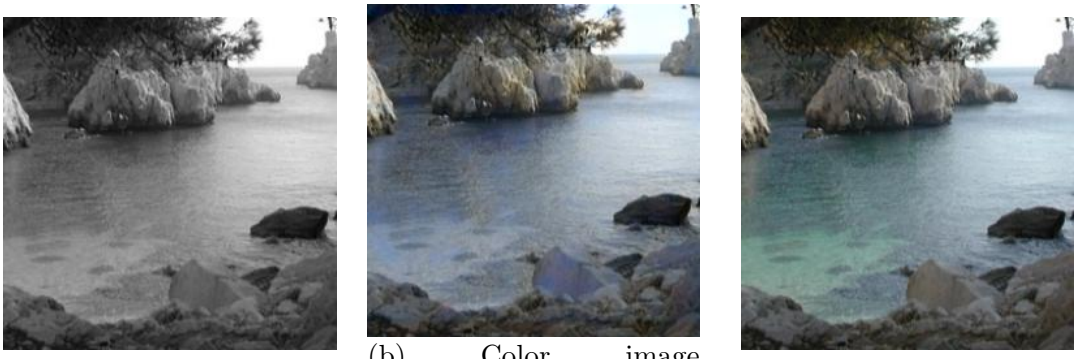
2.1 Image Colorization

The task of coloring black & white images is predominantly carried out by artists. Since an artists has the common sense knowledge, he/she 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 rectified linear unit (ReLU) as activation function, and train it using the provided gray scale images of beach scenes. Fig. 1 represents an example for image colorization. The images in 1a and 1b are the gray scale input and colorization output of the system, and the one in 1c is the actual color photograph of the same scene.

In this assignment, you are required to do estimations on CIELab color space. The details are explained in section 2.2. Basically, your system will input the L channel which encodes the lightness and output

Figure 1: Example for image colorization



(a) Gray scale image (b) Color image (groundtruth) (c) Expected colorization

a and b channels encoding the color information. That is, the goal is basically to learn a function that matches deep image features extracted from lightness information to color information. While training the network, you are required to use mean squared error loss function (1) to minimize the difference between actual color images and the estimated one.

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

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.

- In RGB color space, all channels hold values between 0 and 255. In CIELab space, L channel holds lightness values between 0 and 100, a and b channels hold green-to-red, blue-to-yellow color values between -128 and +127.
- The network should input L channel of the gray scale image in size 256x256. In order to read an image in RGB space and convert it to CIELab, you can utilize the functions provided in "*utils.py*".
- The network should output a and b channels for the color image in size 64x64.
- Since the final size is 64x64, you are required to do 4x4 down-sampling over the convolutional layers. One can achieve this by using pooling operations or arranging hyperparameters of convolutional layers.
- While training the network, you will use 64x64 sized outputs in the loss function. However, the final output of the system should be a 256x256 sized RGB image. After training, you are expected to find a way to create 256x256 sized CIELab image for the output and convert it to RGB space.

Taking the explanations above into consideration, we have fixed the network architecture. You are expected to implement the provided architecture and find good hyperparameters like regularization, number of epochs, and learning rate to achieve good results. While training you are expected to use RMSprop optimizer with L2 regularization.

The **FIXED** network architecture is the following:

1. Convolutional layer with padding of 2 pixels and 16 kernels where each kernel is of size (5,5).
2. Max pooling layer that will decrease the height and width to their half.
3. Batch normalization
4. Rectified linear unit as nonlinearity
5. Convolutional layer with padding of 1 pixel and 24 kernels where each kernel is of size (3,3).
6. Max pooling layer that will decrease the height and width to their half.
7. Batch normalization
8. Rectified linear unit as nonlinearity
9. Convolutional layer with padding of 1 pixel and 2 kernels where each kernel is of size (3,3).

The following items are some **HINTS** and **TRICKS** that you can utilize while implementing your system.

- Since L channel encodes lightness and does not include color information, it should be the same for gray scale and colored version of the same image.
- Normalizing the input of a network to [0,1] or [-1,+1] scale makes it easier to fit a better function to data.
- MSELoss works better with small numbers. Since it is squared loss it may give huge values when the target set consists of large numbers. However, it will give smaller values with target set in the range [-1,+1].
- Using Dataset and DataLoader facilities of PyTorch may ease your work on mini-batches and epochs.
- 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_validation.npy"* for validation set or *"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, 256, 256, 3)**, where 100 is the number of validation or test samples and the remaining is the shape of a single RGB image. 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**.

Along with the implementation and results, you are required to deliver a plot representing the training and validation losses for each epoch. 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.

2.4 Dataset

The dataset consists of 800 training, 100 validation and 100 test images of size 256x256. For each image, there are 256x256 gray scale, 256x256 color and 64x64 color versions. Gray scale versions will be used as input to the system. 64x64 sized ones will be used as ground truth value. The others are there for evaluating the system.

3 Restrictions and Tips

- Your implementation should be in Python 3.
- You are required to use PyTorch library (v0.4.0) to implement neural networks and train them.
- Do not use any available Python repository files without referring to them in a README file.
- Don't forget that the code you are going to submit will also be subject to manual inspection.

4 Submission

- **Late Submission:** As in the syllabus.
- Submission will be as a compressed archive file whose name is <student_id>_the3.tar.gz, e.g., 1234567_the3.tar.gz to ODTÜClass (odtuclass.metu.edu.tr). While creating the archive, do not compress with another format and change it with renaming to "*.tar.gz". Since it is not a tar-ball, it cannot be decompressed with "tar xvf *.tar.gz" command.
- Content of the archive file should be the followings;
  1. Implementation directory
  2. Loss history plot
  3. ***estimations_test.npy***
  4. README.txt if needed
- The archive must contain **no directories** on top of implementation directory, plot and the results.
- Do not include the database and unmentioned files in the archive.

5 Grading

Grading will be based on the accuracy of your results. The following intervals will be used while grading;

- **[0.55-1.00]: 100**
- **[0.50-0.55): 90**
- **[0.45-0.50): 80**
- **[0.40-0.45): 70**
- **[0.35-0.40): 60**

- **[0.30-0.35): 50**
- **[0.00-0.30): Manual inspection.**

Note that your final grade can be lower than the grade corresponding to the accuracy of your results. That is, if we detect any problems with your submission (such as the lack of a proper loss histogram plot), we will deduct points accordingly.

6 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.