CIS 6930/4930 Deep Learning for Computer Graphics
Dr. Corey Toler-Franklin

Part II. Deep Colorization with CNNs
DUE: November 9th, 11:59 pm

Overview | Details | Resources | Getting Help | Submitting | Extra Credit| Acknowledgements

Overview

Collaboration Rules: The entire course project may be completed in groups of one, two or three. For Part II, feel free to collaborate and discuss concepts between groups BUT each group must submit their own original code.

Course Project Part II: In this project, you will create a script that will train (and eventually test) a convolutional Neural Network (CNN) regressor for automatic colorization of an input black and white image. This project is inspired by a SIGGRAPH paper that can be found here. The assignment is adapted from a programming assignment from a course by Connelly Barnes at the University of Virginia.

Details

Getting Started

Set Up Your Code Base: For this assignment, you may use any deep learning framework you choose. We recommend you use PyTorch or other Torch variant. If you prefer to use your own system, you can install PyTorch locally if you have a Mac or Linux (or if you are on Windows, you can install a Linux VM). Everyone in the course will have access to NVIDIA GPUs and PyTorch which are pre-installed on HiPerGator. Other support libraries (like OpenCV for loading images) are available. See software list here. A member of the HiPerGator staff is scheduled to provide an in-class review of the policies for using the cluster. Please remember that your HiPerGator account and any data on it **will be removed without notice on December** 18th. You can find out more about the 72 million $ AI partnership between UF and NVIDIA and the new AI NVIDIA DGX A100 SuperPod here.

Figure 1: Result of the deep coloration algorithm on a roughly a hundred year old black-and-white photograph from the The US National Archives. Original Source: Deep Colorization [ISSI16]

Refer to the SIGGRAPH 2016 paper on deep colorization: [ISSI16] to become familiar with the topic and how colorization can be implemented using CNNs.

Examine the Dataset: The training dataset is posted on canvas (face_images.zip). The images are part of the Georgia Tech Face Database. The dataset contains images of 50 people taken in two or three sessions between 06/01/99 and 11/15/99 at the Center for Signal and Image Processing at the Georgia Institute of Technology. All people in the database are represented by 15 color JPEG images with cluttered backgrounds taken at a resolution of 640x480 pixels. The average size of the faces in these images are 150x150 pixels. The pictures show frontal and/or tilted faces with different facial expressions, lighting conditions and scale.

Load the Dataset: Load your images. You can use Python and OpenCV to load your images:

```
import cv2
import os
import glob
img_dir = "Enter Directory of all images"
files = glob.glob(data_path)
data = []
for f1 in files:
        img = cv2.imread(f1)
        data.append(img)
```

Fill in the directory path to your unzipped dataset. You should create a tensor of size nimages x channels x height x width, where nimages is the number of images in the folder, channels is 3 (for the RGB colors), and height and width are both 128. Be careful to read about the order in which OpenCV stores the red, green and blue color channels. The package- glob is the Unix style pathname pattern expansion for manipulating path strings. Load your data in a Tensor and randomly shuffle the data using torch.randperm. To reduce memory requirements, please set the default Torch datatype to 32-bit float with the following command at the top of your program (before calling the loader): torch.setdefaulttensortype('torch.FloatTensor').

Augment your dataset by a small factor such as 10 to reduce overfitting by using OpenCV (or other image package) to transform your original images. You can do this by creating a torch Tensor class that is 10x larger along the first dimension as the original image dataset, but has the same sizes for the other dimensions. You can use a Python *for loop* to populate this tensor with augmented copies of your original input images. Include in your dataset augmentation horizontal flips, random crops, and scalings of the input RGB values by a single scalar randomly chosen between $[0.6, 1.0]$ (the same scalar should be applied to each of the R, G, B channels). The crop operation can first crop the image, and then resize the image to be the same size as the input. It should be possible for all three "augmentation" operations to be applied to a single input image.

Convert your images to $L*a*b*$ color space. This color space is an alternative to the ordinary RGB color space. $L*a*b*$ color space separates luminance (in channel 1, $L*$) information from color or chrominance information (in channels 2 and 3, $a*$ $b*$). You can do this by creating a torch Tensor class that is the same shape as the previous augmented dataset. There are commands in the OpenCV package for color space conversion:

```
image = cv2.imread('example.jpg')
imageLAB = cv2.cvtColor(image, cv2.COLOR_BGR2LAB)
```

Build a simple regressor using convolutional layers, that predict the mean chrominance values for the entire input image. This regressor is given as input a grayscale image (only the $L*$ channel)

and it predicts the mean chrominance (take the mean across all pixels to obtain mean $a*$ and mean b*) values across all pixels of the image, ignoring pixel location (so it outputs only 2 scalars). You should be able to train this and find that it decreases in training error, and predicts some color information.

To get started, you may want to check out examples of training simple fully connected networks, as well as a more complicated example that includes CNNs. Note this example uses Torch which is built on Lua. If you are using PyTorch, you will have to use the Python equivalent functions.

You can create a network containing 7 modules, where each module consists of a SpatialConvolution layer followed by a ReLU activation function. For the SpatialConvolution layer set the padding and stride appropriately so that the image after convolution is exactly half the size of the input image. This way, the sizes of the images as they go through the CNN are decreasing powers of two: $128, 64, 32, 16, \cdots$. Use a small number of feature maps (3) in the hidden layers for now, since you only want to get the network working at this point. Additionally, scale the input $L*$ channel (which originally ranges from 0 to 100) to the range $[0, 1]$.

Once you have this working, make a copy of this code so that you can submit it later.

Colorize the image by including upsampling/deconvolution layers (such as SpatialFullConvolution or SpatialUpSamplingNearest). The image size output by your network should match the image size input to your network, except that it should have two color channels ($a*$ and $b*$) for the output versus only 1 color channel ($L*$) for the input. One reasonable architecture is to reduce the number of downsampling layers to N and then also use N upsampling/deconvolution layers. You can start with N = 5 and experiment with what gives the best results. You can set the initial spatial resolutions at different parts of the CNN as $128, 64, 32, 16, 8, 4, 8, 16, 32, 64, 128$.

Batch Normalization: The previous network will train and generalize better if it incorporates batch normalization. So incorporate batch normalization (e.g. using the SpatialBatchNormalization layer in Torch). This can be inserted directly after each SpatialConvolution layer. However, note that the SpatialBatchNormalization layer requires 4D tensor inputs, so you have to divide your training dataset into mini-batches of say 10 images each (so the inputs are size nbatchsize x 1 x height x width, and the outputs are size nbatchsize x 2 x height x width).

Training: Divide your dataset into two parts: 90% of the images can be in the training part, and 10% of the images can be in the testing part. Also, make sure that the testing images have not been subjected to data augmentation. Train your regressor on the training part, and then test on the testing part. To evaluate the test images, print a numerical mean square error value. Also, run the input luminance of the image through your network, then merge the a* and b* values predicted by your regressor with the input luminance, and convert back to RGB color space. This will let you view the colorized images. (You may have to use SFTP, SCP or other transfer program to transfer them back to a local machine if you are using remote computing). For example: if you have 750 images in the dataset in total, you could divide this into 90% training (675 images), and 10% testing (75 images): the training images are augmented and result in e.g. 6,750 images, whereas the test images are not augmented (so you only have 75 images).

To avoid running out of memory on large datasets and/or many feature maps, it is necessary to test in mini-batches (otherwise the hidden convolutional layers will allocate too much memory). Also, the batch normalization has to be put in a testing mode to get the best results for testing (this can be done by calling nn.Sequential method evaluate()).

GPU computing: Speed up your network by moving your CNN to the GPU, as explained in this

tutorial. You will need to add commands for CUDA and cuNN (Be sure to verify that you have CUDA-enabled, first). There is a fixed number of GPUs available for simultaneous use for our course. This will likely become congested near the assignment due date. You may update your submission in canvas with the CUDA component up to Wednesday November 11^{th}. Simply upload another zip file with your GPU implementation.

Explain your code for us: Make a 5-7 minute video screen tour walking us through your code, explaining what you did in your own words. We will suggest programs for doing this in canvas (see resources section of this doc). You may however use any program you like. These are only suggestions. **Please show us your implementation for each of bolded stages of the assignment**. Also run your code to show us the output of your training (which you may direct to an output text file) and colorization.

Grading: You will be evaluated based on the following components (1) Load the Dataset 10%, (2) Augment your dataset 10%, (3) Convert your images to $L*a*b*$ color space. 10% (4) Build a simple regressor 15%, (5) Colorize the image 15%, (6) Batch normalization 10%, (7) Training 10%, (8) GPU programming 10%, (9)Video tour and brief written report that explains your implementation 10%.

Extra Credit

You may do one or more of these for extra credit.

Optional Credit: In the last layer you can optionally use a Tanh unit instead of ReLU, similar to the SIGGRAPH paper. In this case you will have to rescale the a* and b* channels to be in the range -1 to 1 (and then scale them back during testing).

Optional Credit: Explore changing the number of feature maps for the interior CNNs to see if you can gain better test accuracy. Report in a brief writeup (it can be just a paragraph or two) what experiments you did for the architecture, your best architecture, and the best mean square error (in terms of a* and b*) you were able to achieve.

Optional Extra Credit: Review this paper, and write a short paragraph to explain how colorization using a classifier rather than a regressor might produce different results. Will a classification approach produce better results? Why or why not.

Resources

Supporting documents are posted to canvas:

Datasets
Face_images.zip

Linux virtual machine
UbuntuVirtualBox_readme.pdf

Videotour
Screencast Setup.pdf
Screen Cast Guide_v2

Everyone in the course will have access to NVIDIA GPUs and PyTorch on HiPerGator. Other support libraries (like OpenCV for loading images) are available. See available software list here. **Remember to back-up your work regularly!!!** Use version control to store your work. DO NOT PUBLISIZE SOLUTIONS.

Getting Help

Send me an email at ctoler@cise.ufl.edu for any issues or questions.

Office Hours are conducted in my personal meeting room in Zoom. A link to this meeting room and available hours are posted on canvas.

Collaborating You may collaborate and discuss this assignment other groups but each group should submit individual work. Remember to always credit outside sources you use in your code. University policies on academic integrity must be followed.

Submitting

Upload one zip file to Canvas. If your zip file is too large for a canvas upload, you must submit your written report to canvas, and send me an email link to a file share (like ONEDRIVE at UF) with your uploaded files by the due date and time. Your files must be uploaded to this file share by the due date and time. With the exception of the GPU programming component which I will accept up to 11:59pm November 11^{th} (see above), all submissions (Canvas or otherwise) must be completely uploaded by the due date and time to avoid late penalties. This assignment is subject to the late policy for course assignments.

Your submission should include everything needed to test, run and understand your code including:

- The complete source code including any third party libraries that are **NOT** available on HiPerGator. See software list here.
- The program that predicts mean chrominance.
- Programs to train and test your best resulting colorization model.
- Example image colorizations from both the training and test sets.
- Any other input needed to run your code that is not mentioned in this list. For example some programs need images icons or other support data to run out-of-the-box.
- A one page written report that includes:
  - An explanation of your implementation.
  - Evaluation results requested in the assignment (see bold text).
  - Instructions on how to run your programs. Your code will be tested on all the provided datasets using scripts.
  - Anything you would like us to know (bugs, difficulties).
- 5-7 minute video screen tour walking us through your code and explaining your implementation. It should be located in a folder in your zip called videotour. You should explain your code and step through the key parts of your code. You also need to run the system so that we can see the results being generated. Program options will be posted to canvas but you may use any program you prefer. We will not watch beyond the 7 minute time limit.

References

[ISSI16] IIZUKA S., SIMO-SERRA E., ISHIKAW H.: Let there be color!: Joint end-to-end learning of global and local image priors for automatic image colorization with simultaneous classification. *ACM Transactions on Graphics (Proc. of SIGGRAPH 2016)* (2016), 110:1–110:11.

Acknowledgements

Sources and Datasets:

Georgia Tech Face Database
Connelly Barnes, University of Virginia