Instructor: Vishnu Boddeti
CSE 891-001: Deep Learning
Programming Homework 2
October, 28

Instructions:

- **Submission:** Only homeworks uploaded to Google Classroom will be graded. Submit solutions in PDF format.
- **Integrity and Collaboration:** You are expected to work on the homeworks by yourself. You are not permitted to discuss them with anyone except the instructor. The homework that you hand in should be entirely your own work. You may be asked to demonstrate how you got any results that you report.
- **Clarifications:** If you have any question, please look at Google Classroom first. Other students may have encountered the same problem, and is solved already. If not, post your question there. We will respond as soon as possible.

Introduction: In this assignment you will train a convolutional neural network for a task known as image colorization [1]. That is, given a gray scale image, we wish to predict the color at each pixel. This is a difficult problem for many reasons, one of which being that it is ill-posed: for a single gray scale image, there can be multiple, equally valid colorings.

0 Software Setup and Data: The first step is to install Python, PyTorch, SciPy, NumPy and Scikit-Learn. After you install Python, see the instructions on this web page to install PyTorch (`https://pytorch.org`). PyTorch provides a tensor library just like NumPy, the code will be based on PyTorch, you are encouraged to look at the documentation.

We will use the CIFAR-10 data set, which consists of images of size 32×32 pixels. For most of the questions we will use a subset of the dataset. The data loading script is included with the handout, and should download automatically the first time it is loaded. If you have trouble downloading the file, you can also do so manually from:

`http://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz`

To make the problem easier, we will only use the "Horse" category from this data set. Now let's learn to color some horses!

1 Colorization as Regression (2pts): There are many ways to frame the problem of image colorization as a machine learning problem. A simple approach is to frame it as a regression problem, where we build a model to predict the RGB intensities at each pixel given the gray scale input. In this case, the outputs are continuous, and so squared error can be used to train the model.

A set of weights for such a model is included with the code. Read the code in `color_regression.py` and answer the following questions:

1. Describe the model RegressionCNN. How many convolution layers does it have? What are the filter sizes and number of filters at each layer? Construct a table or draw a diagram.
2. Run `color_regression.py`. This will load a set of trained weights and should generate some images showing validation outputs. Do the results look good to you? Why or why not?
3. A color space is a choice of mapping of colors into three-dimensional coordinates. Some colors could be close together in one color space, but further apart in others. The RGB color space is probably the most familiar to you, but most state of the art colorization models do not use RGB color space. The model used in `color_regression.py` computes squared error in RGB color space. How could using the RGB color space be problematic?
4. Most state of the art colorization models frame colorization as a classification problem instead of a regression problem. Why? (Hint: what does minimizing squared error encourage?)

2 Colorization as Classification (2pts): We will select a subset of 24 colors and frame colorization as a pixel-wise classification problem, where we label each pixel with one of 24 colors. The 24 colors are selected using k-means clustering[a] over colors, and selecting cluster centers. This was already done for you, and cluster centers are provided in `color/color_kmeans*.npy` files. For simplicity we still measure distance in RGB space. This is not ideal but reduces the software dependencies for this assignment.
Read the code in colorization.py and answer the following questions:

1. Complete the model `CNN`. This model should have the same layers and convolutional filters as the `RegressionCNN`, with the exception of the output layer. Continue to use PyTorch layers like `nn.ReLU`, `nn.BatchNorm2d` and `nn.MaxPool2d`, however we will not use `nn.Conv2d`. We will use our own convolution layer `MyConv2d` included in the file to better understand its internals.
2. Run the following command:

   ```
   python colorization.py --model CNN --checkpoint weights/cnn_k3_f32.pkl --valid
   ```

   This will load a set of trained weights for your CNN model, and should generate some images showing the trained result. How do the result compare to the previous model?

[a] `https://en.wikipedia.org/wiki/K-means_clustering`

3 Skip Connections (3pts): A skip connection in a neural network is a connection which skips one or more layer and connects to a later layer. We will introduce skip connections.

1. Add a skip connection from the first layer to the last, second layer to the second last, etc. That is, the final convolution should have both the output of the previous layer and the initial gray scale input as input. This type of skip-connection is introduced by [2], and is called a "UNet". Following the `CNN` class that you have completed, complete the `__init__` and `forward` methods of the `UNet` class.

   Hint: You will need to use the function `torch.cat`.

2. Train the model for at least 5 epochs and plot the training curve using a batch size of 10:

   ```
   python colorization.py --model UNet --checkpoint unet_k3_f32.pkl -b 10 -e 5
   ```

   This should take around 25 minutes on a CPU based desktop computer. If you train for 25 epochs, the results will be a lot better (but not necessary for this question). If you have an NVIDIA GPU (and associated drivers) your code will automatically use the GPU make training significantly faster.

3. How does the result compare to the previous model? Did skip connections improve the validation loss and accuracy? Did the skip connections improve the output qualitatively? How? Give at least two reasons why skip connections might improve the performance of our CNN models.

   Note: We recommend that you answer this question with the pre-trained weights provided, especially if you did not train for 25 epochs in the previous question. Clearly state if you choose to do so in your writeup. You can load the weights using the following command:

   ```
   python colorization.py --model UNet --checkpoint weights/unet_k3_f32.pkl --valid
   ```

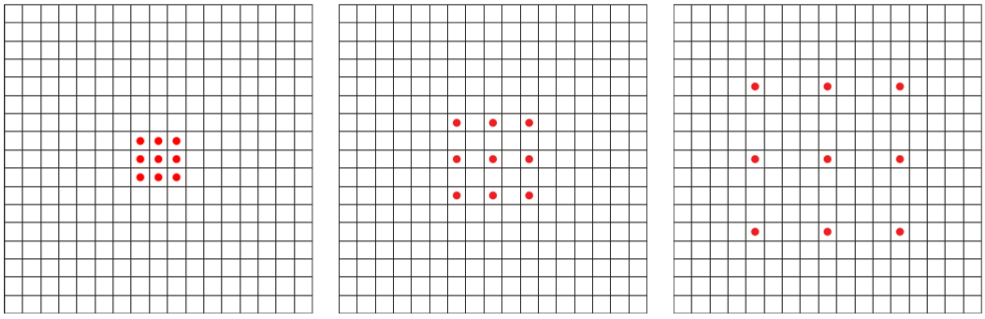

Figure 1: (a) no dilation, (b) dilation=1, (c) dilation=3. Note that the dilation value used in this assignment is slightly different from the original paper.

4 Dilated Convolution (1pt): In class, we discussed convolutional filters that are contiguous – that is, they act on input pixels that are next to one another. However, we can dilated the filters so that they have spaces between input pixels [3], as in Figure 1.

A dilated convolution is a way to increase the size of the receptive field without increasing the number of weights.

1. We have been using 3×3 filters for most of the assignment. Compare between:
   (a) 3×3 convolution
   (b) 5×5 convolution
   (c) 3×3 convolution with dilation 1

   How many weights (excluding biases) do they have? What is the size of the receptive field of each?
2. The `DilatedUNet` class replaces the middle convolution with a dilated convolution with dilation 1. Why we might choose to add dilation here, instead of another convolution? (Hint: think about impact on receptive field.)

5 Visualizing Intermediate Activations (1pt): We will visualize the intermediate activations for several inputs. The python script `activation.py` has already been written for you, and outputs.

1. Visualize the activations of the CNN for a few test examples. How are the activation in the first few layers different from the later layers? You do not need to attach the output images to your writeup, only descriptions of what you see.

   ```
   python activations.py <num> --model CNN --checkpoint weights/cnn_k3_f32.pkl
   ```
2. Visualize the activations of the UNet for a few test examples. How do the activations differ from the CNN activations?

   ```
   python activations.py <num> --model UNet --checkpoint weights/unet_k3_f32.pkl
   ```

6 Conceptual Questions (1pt):

1. Data augmentation can be helpful when the training set size is small. Which of these data augmentation methods do you think would have been helpful for our CNN models, and why?
   (a) Augmenting via flipping each image upside down
   (b) Augmenting via flipping each image left to right
   (c) Augmenting via shifting each image one pixel left / right
   (d) Augmenting via shifting each image one pixel up / down
   (e) Augmenting via using other of the CIFAR-10 classes
2. We also did not tune any hyperparameters for this assignment. What are some hyperparameters that could be tuned? List five.

7 Extra Points: Implementing Dilated Convolution (1pt) This is an optional portion of the assignment where we will implement the Dilated UNet.

1. Dilations are included as a parameter in PyTorch `nn.Conv2d` module and `F.conv2d` function. For the purpose of this assignment we will not use the native implementation. Instead, we will re-implement dilated convolution by directly manipulating the filters using PyTorch tensor manipulation, and ensuring that the implemented dilated convolution is consistent with Fig.1. The purpose is to (a) better understand PyTorch and (b) better understand what the filters look like. Implemented the `forward()` method of `MyDilatedConv2d` function, without using the dilation parameter of `F.conv2d`.

   Hint: You can do a lot with PyTorch tensors that you can do with NumPy arrays, like slicing `tensor[2:4]`, and assigning values to slices `tensor[2:4] = 0`. Bear in mind that PyTorch will need to backpropagate through whatever operations that you use to manipulate the tensors. You may also find the function `F.upsample` to be helpful.

   Note: If you cannot complete this problem, use the dilation parameter of `F.conv2d`. You will not receive credit for this section, but it will allow you to continue to the next section.

2. Using the pre-trained weights provided, use the following command to load the weights and run validation step:

   ```
   python colorization.py --model DUNet --checkpoint weights/dunet_k3_f32.pkl --valid
   ```

   How does the result compare to the previous model, quantitatively (loss and accuracy) and qualitatively? You may or may not see an improvement in this case. In what circumstances might dilation be more helpful?

Submission: Here is everything you need to submit.

- A PDF file titled `programming-assignment-2-msunetid.pdf` containing the following:
  - Answers to the conceptual questions and the requested outputs.
  - Optional: What part of this homework was the most illuminating, and the part that was difficult?
- Your code file `colorization-msunetid.py`

References

[1] Richard Zhang, Phillip Isola, and Alexei A Efros. Colorful image colorization. In *European Conference on Computer Vision*, pages 649–666. Springer, 2016.

[2] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 234–241. Springer, 2015.

[3] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.