Programming Assignment 2: Convolutional Neural Networks

Deadline: Feb. 28, 2018 at 11:59pm
TAs: Harris Chan (csc321staff@cs.toronto.edu)
Based on an assignment by Lisa Zhang

Submission: You must submit two files through MarkUs[1]: a PDF file containing your writeup, titled `a2-writeup.pdf`, and your code file `colourization.py`. Your writeup must be typeset using LaTeX.

The programming assignments are individual work. See the Course Information handout[2] for detailed policies.

You should attempt all questions for this assignment. Most of them can be answered at least partially even if you were unable to finish earlier questions. If you were unable to run the experiments, please discuss what outcomes you might hypothetically expect from the experiments. If you think your computational results are incorrect, please say so; that may help you get partial credit.

Colourization using Convolutional Neural Network

In this assignment, we will train a convolutional neural network for a task known as image colourization. That is, given a greyscale image, we wish to predict the colour at each pixel. This a difficult problem for many reasons, one of which being that it is ill-posed: for a single greyscale image, there can be multiple, equally valid colourings.

Setting Up

We recommend that you use the Teaching Labs machines for the assignment, as all the required libraries for the assignment are already installed there. Otherwise, if you are working on your own environment, you will need to install Python 2, PyTorch (`https://pytorch.org`), SciPy, NumPy and scikit-learn. Check out the websites of the course and relevant packages for more details.

Dataset

We will use the CIFAR-10 data set, which consists of images of size 32x32 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 manully 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 colour some horses!

[1] `https://markus.teach.cs.toronto.edu/csc321-2018-01`
[2] `http://www.cs.toronto.edu/~rgrosse/courses/csc321_2018/syllabus.pdf`

Questions

A. Colourization as Regression (2 points)

There are many ways to frame the problem of image colourization as a machine learning problem. One naïve approach is to frame it as a regression problem, where we build a model to predict the RGB intensities at each pixel given the greyscale input. In this case, the outputs are continuous, and so a L1 or L2 loss can be used to train the model.

A set of weights for such a model is included in the handout. Read the code in `colour_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 `colour_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 colour space [1] is a choice of mapping of colours into three-dimensional coordinates. Some colours could be close together in one colour space, but further apart in others. The RGB colour space is probably the most familiar to you, but most state of the art colourization models do not use RGB colour space. The model used in `colour_regression.py` computes L2 distance in RGB colour space. How could using the RGB colour space be problematic?

4. Most state of the art colourization models frame colourization as a classification problem instead of a regression problem. Why? (Hint: what does L2 distance encourage?)

B. Colourization as Classification (2 points)

We will select a subset of 24 colours and frame colourization as a pixel-wise classification problem, where we label each pixel with one of 24 colours. The 24 colours are selected using k-means clustering[3] over colours, and selecting cluster centers. This was already done for you, and cluster centers are provided in `colour/colour_kmeans*.npy` files. For simplicy, we still measure distance in RGB space. This is not ideal but reduces the software dependencies for this assignment.

Read the code in `colourization.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 colourization.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?

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

C. Skip Connections (3 points)

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 greyscale input as input. This type of skip-connection is introduced by [3], 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 colourization.py --model UNet --checkpoint unet_k3_f32.pkl -b 10 -e 5
   ```

   This should take around 25 minutes on the Teaching Labs computer. If you train for 25 epochs, the results will be a lot better (but not necessary for this question). If you have a gpu available you can use the `--gpu` flag to make training much, much 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 colourization.py --model UNet --checkpoint weights/unet_k3_f32.pkl --valid
   ```

D. Dilated Convolution (1 point)

In class, we have 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 [4], as follows:

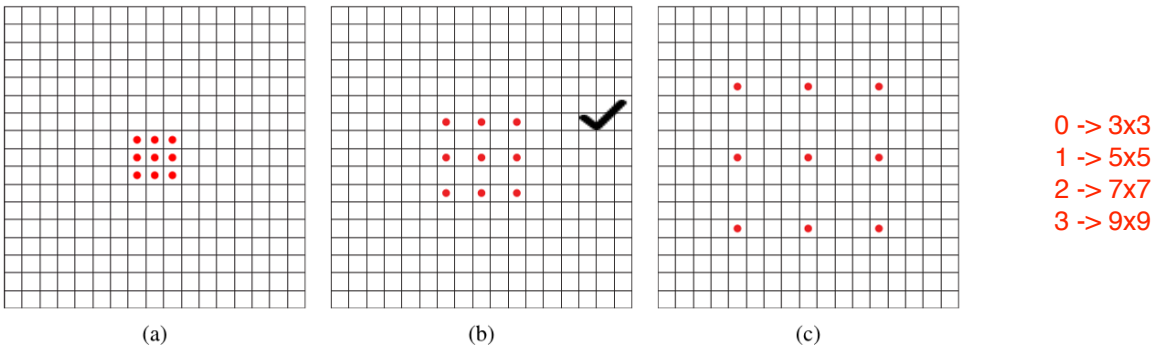

Figure 1: (a) no dilation, (b) dilation=1, (c) dilation=2

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

1. We have been using 3x3 filters for most of the assignment. Compare between:

   (a) 3x3 convolutions,

   (b) 5x5 convolution,

   (c) 3x3 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.)

E. Visualizing Intermediate Activations (1 point)

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_k2_f32.pkl
   ```

F. Conceptional Problems (1 point)

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.

G. (Bonus) Dilated Convolution Implementation (1 point)

This is an optional portion of the assignment where we will implement the Dilated UNet.

1. Dilations are included as a paramter in PyTorch `nn.Conv2d` module and `F.conv2d` funcion. 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. 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 colourization.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?

What To Submit

For reference, here is everything you need to hand in:

- A PDF file `a2-writeup.pdf`, typeset using LaTeX, containing your answers to the conceptual questions and requested outputs.
- Your implementation of `colourization.py`.

References

[1] `https://en.wikipedia.org/wiki/Color_space`

[2] Zhang, R., Isola, P., and Efros, A. A. (2016, October). Colorful image colorization. In European Conference on Computer Vision (pp. 649-666). Springer International Publishing.

[3] Ronneberger, O., Fischer, P., and Brox, T. (2015, October). U-net: Convolutional networks for biomedical image segmentation. In International Conference on Medical Image Computing and Computer-Assisted Intervention (pp. 234-241). Springer, Cham.

[4] Yu, F., and Koltun, V. (2015). Multi-scale context aggregation by dilated convolutions. arXiv preprint arXiv:1511.07122. Chicago

[5] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... and Rabinovich, A. (2015). Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1-9).