Programming Assignment 2: Convolutional Neural Networks

Due Date: Wednesday, Feb. 26th, at 11:59pm
Based on an assignment by Lisa Zhang

Submission: You must submit 2 files through MarkUs[1]: a PDF file containing your writeup, titled `a2-writeup.pdf`, and your code file `a2-cnn.ipynb`. Your writeup must be typed.

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 think your computational results are incorrect, please say so; that may help you get partial credit.

Introduction

This assignment will focus on the applications of convolutional neural networks in various image processing tasks. The starter code is provided as a Python Notebook on Colab (`https://colab.research.google.com/drive/11sH_zVO8QvCAYrGDv83lI9-5mnmln3SV#scrollTo=JyzOT64xkqy6`). First, we will train a convolutional neural network for a task known as image colourization. Given a greyscale image, we will 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. In the second half of the assignment, we will perform fine-tuning on a pre-trained semantic segmentation model. Semantic segmentation attempts to clusters the areas of an image which belongs to the same object (label), and treats each pixel as a classification problem. We will fine-tune a pre-trained conv net featuring dilated convolution to segment flowers from the Oxford17 flower dataset[3].

Part A: Colourization as Classification (2 pts)

In this section, we will perform image colourization using a convolutional neural network. Given a grayscale image, we wish to predict the color of each pixel. We have provided a subset of 24 output colours, selected using k-means clustering[4]. The colourization task will be framed as a pixel-wise

[1]`https://markus.teach.cs.toronto.edu/csc413-2020-01`
[2]`https://csc413-2020.github.io/assets/misc/syllabus.pdf`
[3]`http://www.robots.ox.ac.uk/~vgg/data/flowers/17/`
[4]`https://en.wikipedia.org/wiki/K-means_clustering`

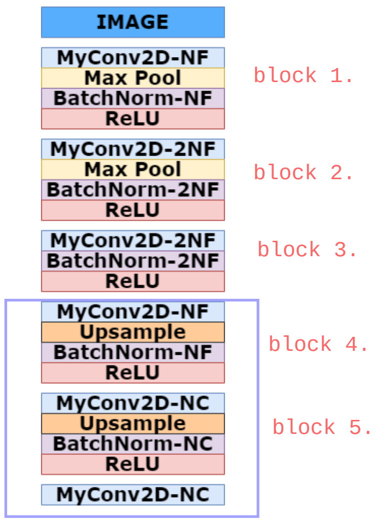
classification problem, where we will label each pixel with one of the 24 colours. For simplicity, we measure distance in RGB space. This is not ideal but reduces the software dependencies for this assignment.

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 notebooks, and should download automatically the first time it is loaded.

Helper code for Part A is provided in `a2-cnn.ipynb`, which will define the main training loop as well as utilities for data manipulation. Run the helper code to setup for this question and answer the following questions.

1. Complete the model `CNN`, following the diagram provided below. Use the PyTorch layers `nn.ReLU`, `nn.BatchNorm2d`, `nn.Upsample`, and `nn.MaxPool2d`, but do not use `nn.Conv2d`. Instead, use the convolutional layer `MyConv2d` included in the file to better understand its internals. Your CNN should be configurable by parameters `kernel`, `num_filters`, `num_colours`, and `num_in_channels`. In the diagram, `num_filters` and `num_colours` are denoted **NF** and **NC** respectively. Use the following parameterizations (if not specified, assume default parameters):

   - `MyConv2d`: Number of output filters to use shown after the hyphen. For example, `MyConv2D-2NF` has 2**NF** output filters. Set kernel size to parameter `kernel`. Set number of input filters for first `MyConv2d` to `num_in_channels`.
   - `nn.BatchNorm2d`: the number of features to use for a layer is shown after the hyphen.
   - `nn.Upsample`: use `scaling_factor = 2`
   - `nn.MaxPool2d`: use `kernel_size = 2`



Grouping layers according to the diagram (those not separated by white space) by using `nn.Sequential` containers will aid implementation of the `forward` method.

2. Run main training loop of `CNN`. This will train the CNN for a few epochs using the cross-entropy objective. It will generate some images showing the trained result at the end. Do these results look good to you? Why or why not?

3. Compute the number of weights, outputs, and connections in the model, as a function of **NF** and **NC**. Compute these values when each input dimension (width/height) is doubled. Report all 6 values.

4. Consider an pre-processing step where each input pixel is multiplied elementwise by scalar a, and is shifted by some scalar b. That is, where the original pixel value is denoted x, the new value is calculated $y = ax + b$. Assume this operation does not result in any overflows. How does this pre-processing step affect the output of the conv net from Question 1 and 2?

Part B: Skip Connections (2 pts)

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 to our previous model.

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 Ronneberger et al. [2015], and is called a "UNet". Following the `CNN` class that you have completed, complete the `__init__` and `forward` methods of the `UNet` class in Part B of the notebook.
   Hint: You will need to use the function `torch.cat`.

2. Train the model for at least 25 epochs and plot the training curve using a batch size of 100.

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.

4. Re-train a few more "UNet" models using different mini batch sizes with a fixed number of epochs. Describe the effect of batch sizes on the training/validation loss, and the final image output.

Part C: Fine-tune Semantic Segmentation Model (2 pts)

In the previous two parts, we worked on training models for image colourization. Now we will switch gears and perform semantic segmentation by fine-tuning a pre-trained model.

Semantic segmentation can be considered as a pixel-wise classification problem where we need to predict the class label for each pixel. Fine-tuning is often used when you only have limited labeled data.

Here, we take a pre-trained model on the Microsoft COCO [Lin et al., 2014] dataset and fine-tune it to perform segmentation with the classes it was never trained on. To be more specific, we use ***deeplabv3*** [Chen et al., 2017][5] pre-trained model and fine-tune it on the Oxford17 [Nilsback and Zisserman, 2008] flower dataset.

We simplify the task to be a binary semantic segmentation task (background and flower). In the following code, you will first see some examples from the Oxford17 dataset and load the finetune the model by truncating the last layer of the network and replacing it with a randomly initialized convolutional layer. Note that we only update the weights of the newly introduced layer.

1. For this assignment, we want to fine-tune only the last layer in our downloaded deeplabv3. We do this by keeping track of weights we want to update in `learned_parameters`.

   Use the PyTorch utility `Model.named_parameters()`[6], which returns an iterator over all the weight matrices of the model.
   The last layer weights have names prefix "classifier.4". We will select the corresponding weights then passing them to `learned_parameters`.

   Complete the 'train' function in Part C of the notebook by adding 2-3 lines of code where indicated.

2. For fine-tuning we also want to

   - use `Model.requires_grad_()` to prevent back-prop through all the layers that should be frozen.
   - replace the last layer with a new `nn.Conv2d` layer with appropriate input output channels and kernel sizes. Since we are performing binary segmentation for this assignment, this new layer should have 2 output channels.
     Complete the script in Question 2 of Part C by adding around 2 lines of code and train the model.

3. Visualize the predictions by running the helper code provided.

4. Consider a case of fine-tuning a pre-trained model with n number of layers. Each of the layers have a similar number of parameters, so the total number of parameters for the model is proportional to n. Describe the difference in memory complexity in terms of n between fine-tuning an entire pre-trained model versus fine-tuning only the last layer (freezing all the other layers). What about the computational complexity?

[5]deeplabv3 details: `https://pytorch.org/hub/pytorch_vision_deeplabv3_resnet101/`
[6]See examples at `https://pytorch.org/docs/stable/nn.html`

5. If we increase the height and the width of the input image by a factor of 2, how does this affect the memory complexity of fine-tuning? What about the number of parameters?

What you have to submit

For reference, here is everything you need to hand in. See the top of this handout for submission directions.

- A PDF file titled `a2-writeup.pdf` containing the following:
  - Answers to questions from Part A
  - Answers to questions from Part B
  - Answers to questions from Part C
- Your code file `a2-cnn.ipynb`

References

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.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

Liang-Chieh Chen, George Papandreou, Florian Schroff, and Hartwig Adam. Rethinking atrous convolution for semantic image segmentation. *arXiv preprint arXiv:1706.05587*, 2017.

Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *2008 Sixth Indian Conference on Computer Vision, Graphics & Image Processing*, pages 722–729. IEEE, 2008.