Programming Assignment 2: Convolutional Neural Networks

Version: 1.0
Version Release Date: 2021-02-05
Due Date: Thursday, Feb. 25th, 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.

The teaching assistants for this assignment are Alexey Strokach and Yun-Chun Chen. Send your email with subject "*[CSC413] PA2 ...*" to csc413-2021-01-tas@cs.toronto.edu or post on Piazza with the tag `pa2`.

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/github/csc413-uoft/2021/blob/master/assets/assignments/a2_cnn.ipynb`). 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].

[1] `https://markus.teach.cs.toronto.edu/csc413-2021-01`
[2] `https://csc413-uoft.github.io/2021/assets/misc/syllabus.pdf`
[3] `http://www.robots.ox.ac.uk/~vgg/data/flowers/17/`

Image Colourization as Classification

In this section, we will perform image colourization using three convolutional neural networks (Figure 1). 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 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.

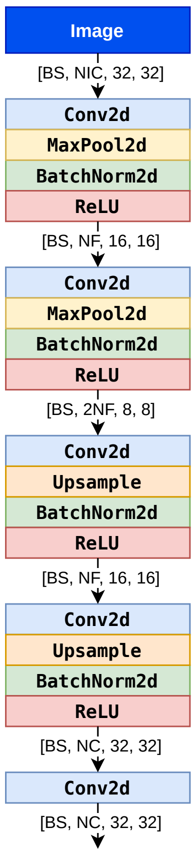
Part A: Pooling and Upsampling (2 pts)

1. Complete the model `PoolUpsampleNet`, following the diagram in Figure 1a. Use the PyTorch layers `nn.Conv2d`, `nn.ReLU`, `nn.BatchNorm2d`, `nn.Upsample`, and `nn.MaxPool2d`. Your CNN should be configurable by parameters `kernel`, `num_in_channels`, `num_filters`, and `num_colours`. In the diagram, `num_in_channels`, `num_filters` and `num_colours` are denoted **NIC**, **NF** and **NC** respectively. Use the following parameterizations (if not specified, assume default parameters):

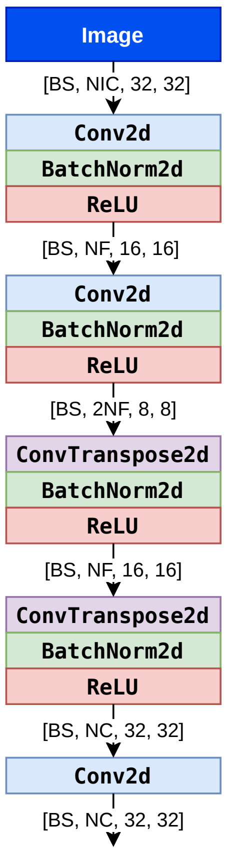
   - `nn.Conv2d`: The number of input filters should match the second dimension of the *input* tensor (e.g. the first `nn.Conv2d` layer has **NIC** input filters). The number of output filters should match the second dimension of the *output* tensor (e.g. the first `nn.Conv2d` layer has **NF** output filters). Set kernel size to parameter `kernel`. Set padding to the `padding` variable included in the starter code.
   - `nn.BatchNorm2d`: The number of features should match the second dimension of the output tensor (e.g. the first `nn.BatchNorm2d` layer has **NF** features).
   - `nn.Upsample`: Use `scaling_factor = 2`.
   - `nn.MaxPool2d`: Use `kernel_size = 2`.

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

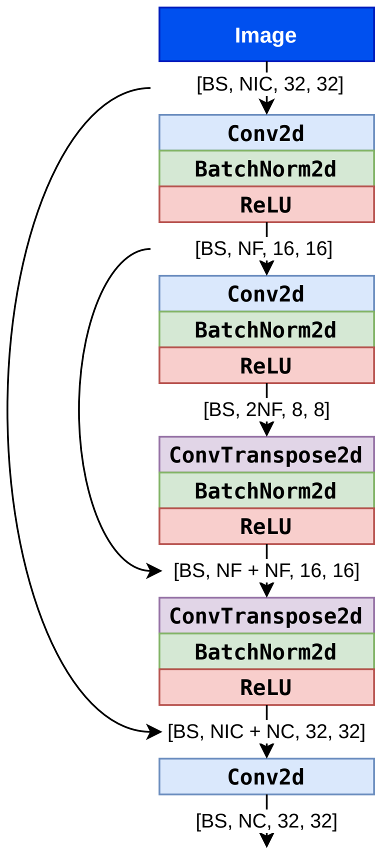
[4] `https://en.wikipedia.org/wiki/K-means_clustering`



(a) `PoolUpsampleNet`



(b) `ConvTransposeNet`



(c) `UNet`

Figure 1: Three network architectures that we will be using for image colourization. Numbers inside square brackets denote the shape of the tensor produced by each layer: **BS**: batch size, **NIC**: `num_in_channels`, **NF**: `num_filters`, **NC**: `num_colours`.

2. Run main training loop of `PoolUpsampleNet`. 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 **NIC**, **NF** and **NC**. Compute these values when each input dimension (width/height) is doubled. Report all 6 values.

Part B: Strided and Transposed Convolutions (3 pts)

For this part, instead of using `nn.MaxPool2d` layers to reduce the dimensionality of the tensors, we will increase the step size of the preceding `nn.Conv2d` layers, and instead of using `nn.Upsample` layers to increase the dimensionality of the tensors, we will use *transposed* convolutions. Transposed convolutions aim to apply the same operations as convolutions but in the opposite direction. For example, while increasing the stride from 1 to 2 in a convolution forces the filters to skip over every other position as they slide across the input tensor, increasing the stride from 1 to 2 in a transposed convolution adds "empty" space around each element of the input tensor, as if reversing the skipping over every other position done by the convolution. Excellent visualizations of convolutions and transposed convolutions have been developed by Dumoulin and Visin [2018] and can be found on their GitHub page[5].

1. Complete the model `ConvTransposeNet`, following the diagram in Figure 1b. Use the PyTorch layers `nn.Conv2d`, `nn.ReLU`, `nn.BatchNorm2d` and `nn.ConvTranspose2d`. As before, your CNN should be configurable by parameters `kernel`, `num_in_channels`, `num_filters`, and `num_colours`. Use the following parameterizations (if not specified, assume default parameters):

   - `nn.Conv2d`: The number of input and output filters, and the kernel size, should be set in the same way as Part A. For the first two `nn.Conv2d` layers, set `stride` to 2 and set `padding` to 1.
   - `nn.BatchNorm2d`: The number of features should be specified in the same way as for Part A.
   - `nn.ConvTranspose2d`: The number of input filters should match the second dimension of the *input* tensor. The number of output filters should match the second dimension of the *output* tensor. Set `kernel_size` to parameter `kernel`. Set `stride` to 2, and set both `padding` and `output_padding` to 1.

2. Train the model for at least 25 epochs using a batch size of 100 and a kernel size of 3. Plot the training curve, and include this plot in your write-up.

[5] `https://github.com/vdumoulin/conv_arithmetic`

3. How do the result compare to Part A? Does the `ConvTransposeNet` model result in lower validation loss than the `PoolUpsampleNet`? Why may this be the case?

4. How would the `padding` parameter passed to the first two `nn.Conv2d` layers, and the `padding` and `output_padding` parameters passed to the `nn.ConvTranspose2d` layers, need to be modified if we were to use a kernel size of 4 or 5 (assuming we want to maintain the shapes of all tensors shown in Figure 1b)?

   Note: PyTorch documentation for `nn.Conv2d`[6] and `nn.ConvTranspose2d`[7] includes equations that can be used to calculate the shape of the output tensors given the parameters.

5. Re-train a few more `ConvTransposeNet` models using different batch sizes (e.g., 32, 64, 128, 256, 512) with a fixed number of epochs. Describe the effect of batch sizes on the training/validation loss, and the final image output quality. You do *not* need to attach the final output images.

Part C: Skip Connections (1 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 the model we implemented in Part B.

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 (see Figure 1c). This type of skip-connection is introduced by Ronneberger et al. [2015], and is called a "UNet". Following the `ConvTransposeNet` class that you have completed, complete the `__init__` and `forward` methods of the `UNet` class in Part C of the notebook.

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

2. Train the model for at least 25 epochs using a batch size of 100 and a kernel size of 3. Plot the training curve, and include this plot in your write-up.

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.

[6] `https://pytorch.org/docs/stable/generated/torch.nn.Conv2d.html`

[7] `https://pytorch.org/docs/stable/generated/torch.nn.ConvTranspose2d.html`

Image Segmentation as Classification

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][8] 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.

Part D.1: Fine-tune Semantic Segmentation Model with Cross Entropy Loss (2 pts)

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()`[9], 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 pass them to `learned_parameters`.

   Complete the `train` function in Part D 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 D by adding around 2 lines of code and train the model. What is the best validation mIoU?

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

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

Part D.2: Fine-tune Semantic Segmentation Model with IoU Loss (2 pts)

1. We will change the loss function from cross entropy used in part D.1 to the (soft) IoU loss [Rahman and Wang, 2016][10]. Complete the `compute_IoU_loss` function in Part D.2 of the notebook by adding 2-3 lines of code each where indicated.

   Below are the equations for computing the intersection (I), the union (U), and the IoU loss (L_{IoU}) between the soft prediction (X) and the ground-truth (Y).

$$I(X, Y) = \sum_{i,j} X(i, j) \cdot Y(i, j), \tag{1}$$

$$U(X, Y) = \sum_{i,j} (X(i, j) + Y(i, j) - X(i, j) \cdot Y(i, j)), \tag{2}$$

$$L_{\text{IoU}} = 1 - \text{IoU} = 1 - \frac{I(X, Y)}{U(X, Y)} \tag{3}$$

   After you have implemented the function, train the model with the IoU loss. What is the validation mIoU (mean IoU)? How does this compare with the mIoU when training with the cross entropy?

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

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 only the following:
  - Answers to questions from Part A
    - Q1 code for model `PoolUpsampleNet` (screenshot or text)
    - Q2 visualizations and your commentary
    - Q3 answer (6 values as function of **NIC**, **NF**, **NC**)
  - Answers to questions from Part B
    - Q1 code for model `ConvTransposeNet` (screenshot or text)
    - Q2 answer: 1 plot figure (training/validation curves)

[10] See the definition of IoU loss in Eq (5) of `https://www.cs.umanitoba.ca/~ywang/papers/isvc16.pdf`

    - Q3 answer
    - Q4 answer
    - Q5 answer
  - Answers to questions from Part C
    - Q1 code for model `UNet` (screenshot or text)
    - Q2 answer: 1 plot figure (training/validation curves)
    - Q3 answer
  - Answers to questions from Part D.1
    - Q1 code for `train` (screenshot or text)
    - Q2 code for fine-tuning, and answer best validation mIOU
    - Q3 answer: visualization of predictions
  - Answers to questions from Part D.2
    - Q1 code for `compute_IoU_loss` and fine-tuning (screenshot or text), and answer best validation mIOU with comparison to D.1
    - Q2 answer: visualization of predictions
- Your code file `a2-cnn.ipynb`

References

Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning, 2018.

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.

Md Atiqur Rahman and Yang Wang. Optimizing intersection-over-union in deep neural networks for image segmentation. In *International symposium on visual computing*, pages 234–244. Springer, 2016.