Instructor: Vishnu Boddeti
CSE 891-001: Deep Learning
Programming Homework 2
October, 27

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 we will focus on applications of convolutional neural network for different image processing tasks. First, we will train a CNN 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. In the second part of the assignment we will fine-tune a pre-trained semantic segmentation model. Semantic segmentation seeks to cluster 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 CNN featuring dilated convolutions to segment flowers from the Oxford17 flower dataset[a].

[a] `http://www.robots.ox.ac.uk/~vgg/data/flowers/17/`

0 Software Setup and Data: The first step is to install Python, PyTorch, Pytorch Lightning, TorchMetrics, Tensorboard, SciPy, NumPy and Scikit-Learn. All the code in this assignment is based on PyTorch and Pytorch Lightning, so you are encouraged to look at the documentation.

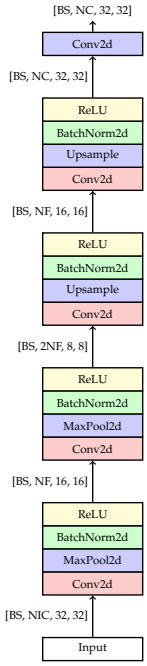
We will use the CIFAR-10 data set for the first part of the homework. It consists of images of size 32×32 pixels. For most of the questions we will use a subset of the dataset. The data loading script is included with the handout, and should download automatically the first time it is loaded. If you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

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

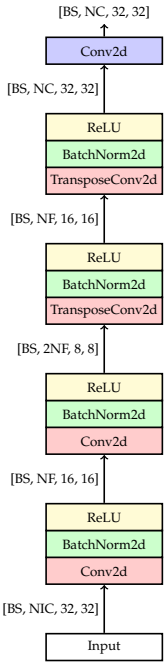
To make the problem easier, we will only use the "Horse" category from this data set. Now let's learn to color some horses!

We will use the Oxford17 Flower data set for the second part of the homework. It 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:

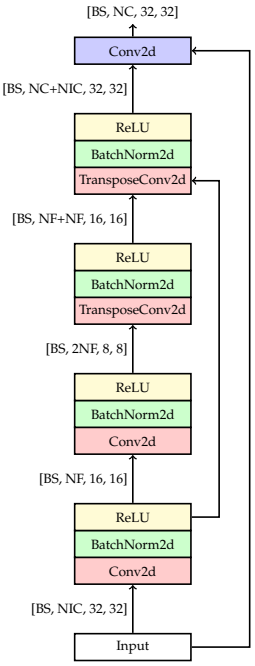
`http://www.robots.ox.ac.uk/~vgg/data/flowers/17/`



(a) PoolUpsampleNet



(b) ConvTransposeNet



(c) UNet

1 Colorization as Classification (6pts): We will perform image colorization using three convolutional neural networks. Given a grayscale image, we will predict the color of each pixel.

We provided a subset of 24 colors, 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. The colourization task will be framed as a pixel-wise classification problem, where we will label each pixel with one of the 24 colours.

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

1.1 Upsampling and Pooling (2pts):

1. Complete the model `PoolUpsampleNet` in `models/poolupsamplenet.py`, following the diagram above. 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. For `nn.Upsample` use `scaling_factor=2` and for `nn.MaxPool2d` use `kernel_size=2`. Grouping the layers according to the diagram using `nn.Sequential` will aid implementation of the `forward` function.

2. Run the main training script `colorization.py` by passing `PoolUpsampleNet` as the argument. This will train the CNN for a few epochs using the cross-entropy loss. It will also generate some images showing the output of the model. You can visualize the training and validation loss curves, and the images in tensorboard. You simply need to launch tensorboard as `tensorboard --logdir logs/`. Do the 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 dimention (width/height) is doubled. Report all 6 values.

1.2 Strided and Transposed Convolution (3pts): In this part, instead of using `nn.MaxPool2d` layers to reduce the dimensionality of the tensors, we will increase the stride 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*.

1. Complete the model `ConvTransposeNet` in `models/convtransposenet.py`, following the diagram above. Use the PyTorch layers `nn.Conv2d`, `nn.ReLU`, `nn.BatchNorm2d` and `nn.ConvTranspose2d`. Your model should be configurable by parameters `kernel`, `num_in_channels`, `num_filters`, and `num_colours`.
2. Train the model for at least 25 epochs using a batch size of 100 and a kernel size of 3. As before, run the main training script `colorization.py` by passing `ConvTransposeNet` as the argument. Plot the training and validation curves, and include them in your write-up. The plots should be available in Tensorboard.
3. How does the result compare to `PoolUpsampleNet`? Does the `ConvTransposeNet` model result in lower validation loss than `PoolUpsampleNet`? Why might this be the case?
4. How should the padding parameter in the first two `nn.Conv2d` layers, and the padding and output padding parameters in the `nn.ConvTranspose2d` layers, be modified if we use a kernel size of 4 or 5 (assuming we want to maintain the shapes of all tensors as before)?
5. Re-train the `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.

1.3 Skip Connections (1pts): 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 `ConvTransposeNet` 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 gray scale input as input. This type of skip-connection is introduced by [2], and is called a "UNet". Following the `ConvTransposeNet` 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 25 epochs using a batch size of 100 and a kernel size of 3. Plot the training and validation curves, and include them in your write-up. The plots should be available in Tensorboard.
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.

2 Image Segmentation as Classification (4pts) In the previous two parts, we worked on training models for image colorization. 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 predict the class label for each pixel. Fine-tuning is commonly employed when we have limited labeled data.

We will take a model that was already trained on the Microsoft COCO [3] dataset and fine-tune it for segmentation with classes it has never seen before using the Oxford17 [4] flower dataset.

We will consider binary (background and flower) semantic segmentation task, a simplified version of the full segmentation task. You will first load and fine tune the model by truncating the last layer of the network and replacing it with a randomly initialized convolutional layer. Note that we will only update the weights of the newly introduced layer.

1. **2.1 Fine-tune Semantic Segmentation Model with Cross Entropy Loss (2pts):**

   (a) For this assignment, we want to fine-tune only the last layer in our downloaded `deeplabv3`. For this purpose we should, (a) use `requires_grad()` to prevent back-prop through all the layers that should be frozen, and (b) 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. The classification layer we want to replace is called `classifier` and we want to replace the last layer in the classifier. Note that you can ignore the `aux_classifier` layer.

   Modify the `models/deeplabv3.py` file. Train the modified model. What is the best validation mIoU? Use a GPU if you have access to one, otherwise this training might be too slow.

   (b) Visualize the predictions from Tensorboard and the output saved in the `results/` folder.

2. **2.2 Fine-tune Semantic Segmentation Model with IoU Loss (2pts):** We will change the loss function from cross-entropy in the previous part to the (soft) IoU loss [5]. Complete the implementation of this loss in `losses/iou.py` file. 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).

$$\begin{aligned} I(X,Y) &= \sum_{ij} X(i,j) \cdot Y(i,j) \\ U(X,Y) &= \sum_{ij} X(i,j) + Y(i,j) - X(i,j) \cdot Y(i,j) \\ L_{IoU} &= 1 - IoU = 1 - \frac{I(X,Y)}{U(X,Y)} \end{aligned} \tag{1}$$

   (a) After implementing the soft IoU loss, train the model with this loss. What is the validation mIoU (mean IoU)? How does this compare with the mIoU when trained with cross entropy?

   (b) Visualize the predictions from Tensorboard and the output saved in the `results/` folder.

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.
    1. Q2 in 1.1 i.e., training/validation plots and comments on your results.
    2. Q3 in 1.1
    3. Q2 in 1.2 i.e., training/validation plots and comments on your results.
    4. Q3, Q4, Q5 in 1.2
    5. Q2 in 1.3 i.e., training/validation plots and comments on your results.
    6. Q3 in 1.3
    7. Q1 in 2.1 i.e., training/validation plots, best mIoU you achieved, and comments on your results.
    8. Q2 in 2.1 i.e., visualizations of input image, ground truth segmentation map and the predicted segmentation map
    9. Q1 in 2.2 i.e., training/validation plots, best mIoU you achieved, and comments on your results.
    10. Q2 in 2.2 i.e., visualizations of input image, ground truth segmentation map and the predicted segmentation map
  - Optional: What part of this homework was the most illuminating, and the part that was difficult?
- The following updated files with your code,
  1. `poolupsamplenet-msunetid.py`
  2. `convtransposenet-msunetid.py`
  3. `unet-msunetid.py`
  4. `deeplabv3-msunetid.py`
  5. `iou-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] 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*, 2014.

[4] Maria-Elena Nilsback and Andrew Zisserman. Automated flower classification over a large number of classes. In *Indian Conference on Computer Vision, Graphics & Image Processing*, 2008.

[5] Md Atiqur Rahman and Yang Wang. Optimizing intersection-over-union in deep neural networks for image segmentation. In *International Symposium on Visual Computing*, 2016.