University of Michigan
EECS 442: Computer Vision
Fall 2023. Instructor: Andrew Owens.

Problem Set 5: Scene Recognition

Posted: Wednesday, October 4, 2023 **Due:** Wednesday, October 11, 2023
Submission instructions:

- Canvas:
  - Colab notebook (`.ipynb`) containing solutions and visualizations for 5.1(a-e). Before submitting, please make sure to rename the file to `<uniqname>_<umid>.ipynb`.
- Gradescope:
  - `.pdf` version of Colab notebook (conversion instructions here). If these instructions do not work, look into this additional method. For your convenience, we have included the PDF conversion script at the end of the notebook.

The starter code can be found at:
https://drive.google.com/file/d/1FRzeBgbh89DURjgwOVr2KlWZqL4qfEFx/view?usp=sharing

Please see **GPU Tips** at the end of this document to better understand how to efficiently use GPUs for this homework set (and all future homeworks that require GPUs). **tldr:** your code will run *very* slow if you run out of GPU hours early and you have to train your models on CPUs. Please follow the tips we provide to avoid this!

We recommend editing and running your code in Google Colab, although you are welcome to use your local machine instead.

Problem 5.1 *Scene Recognition*

In this problem set, you will train a CNN to solve the *scene recognition* problem, i.e., the problem of determining which scene category a picture depicts. You will train two neural networks, which we call `MiniVGG` and `MiniVGG-BN`. `MiniVGG` is a smaller, simplified version of the VGG [2] architecture, while `MiniVGG-BN` is identical to `MiniVGG` except that we added batch normalization layers after each convolution layer.

You will train the neural networks on the MiniPlaces dataset[1]. We'll use 90,000 images for training, 10,000 images for validation, and the remaining 10,000 images for testing. Sample images from MiniPlaces dataset (along with their categories) are shown in Figure 1. Below is an outline for your implementation. For more detailed instructions, please refer to the notebook.

(a) **(2 points)** We often train deep neural networks on very large datasets. Because it is

[1] https://github.com/CSAILVision/miniplaces

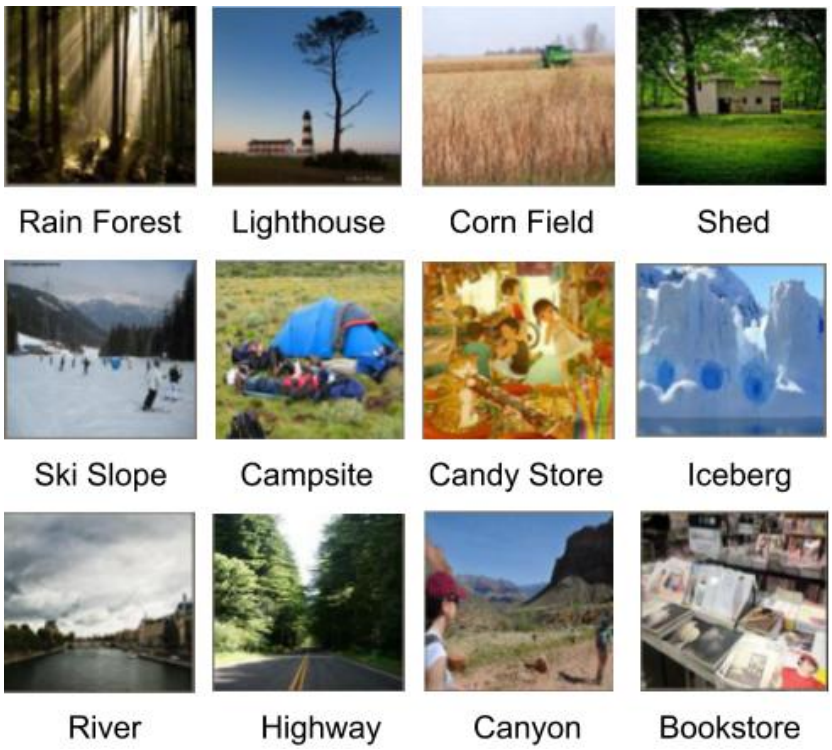

Figure 1: Sample images from MiniPlaces dataset.

impossible to fit the whole dataset into the RAM, loading the data one batch at a time is a common practice. We also often need to preprocess the data, which includes common preprocessing steps like normalizing the pixel values, resizing the images to have a consistent size, and converting `numpy` arrays to PyTorch tensors. As the first step of this problem set, please fill in the indicated part for building data loaders with the specified data preprocessing steps (for the specific preprocessing you need to perform, please see the comments in the provided notebook). You may also find the PyTorch tutorial[2] on data loading to be helpful.

(b) **(3 points)** Construct the `MiniVGG` and `MiniVGG-BN` models. Make sure the neural networks you build have the same architectures as the ones we give in the notebook before you start training them.

(c) **(3 points)** Implement the training and validation loops. For the training loop, you will need to implement the following steps: i) compute the outputs for each minibatch using the neural networks you build, ii) calculate the loss, iii) update the parameters using the SGD optimizer (with momentum). Please use PyTorch's built-in automatic differentiation, rather than implementing backprogagation yourself. The validation loop is almost identical to the training loop except that we do not perform gradient update to the model parameters; we'll simply report the loss and accuracy. Please see the notebook as well for further instructions.

(d) **(1 point)** Train the networks and visualize training and validation accuracy history for `MiniVGG` and `MiniVGG-BN` using the code we provide in the notebook. Comment on the effect of batch normalization. Please leave your comments in the text block we provide at the end of Step 4 section in the notebook. **Note: training each neural network will take about 25 minutes.**

[2] https://pytorch.org/tutorials/beginner/data_loading_tutorial.html

(e) **(1 point)** If the correct label is within the `top_k` predicted classes according to the network output scores, we count the prediction by the neural network as a correct prediction. Measure Top-1 and Top-5 accuracy on the test set (i.e. the probability that the true class appears as the most likely class, and within the Top-5 most likely respectively). To pass the test, both neural networks should have Top-5 accuracy above 55%.

Problem 5.2 *Residual Networks (Optional)*

We will now create a simplified version of the ResNet [1], which we will use to solve the scene recognition problem. To make the implementation easier, we will perform downsampling outside of the residual blocks, using strided convolutions. We have provided a description of the network in Fig. 2, and we have given you a partial implementation. Your job will be to complete it.

(a) (Optional, 0 points) Implement the *residual block*. Each block applies two convolutions and uses a residual connection. We will implement this as a class `ResidualBlock`, which is parameterized with the number of channels.

(b) (Optional, 0 points) Complete the implementation of the ResNet. Your task will be to add the initial layers of the network, and insert the residual blocks.

(c) (Optional, 0 points) Train the model. Your Top-5 accuracy should be at least 55% after 20 epochs of training (i.e., after 20 full passes through the training data).

(d) (Optional, 0 points) Add extra residual blocks to the network, and measure the improvement in accuracy.

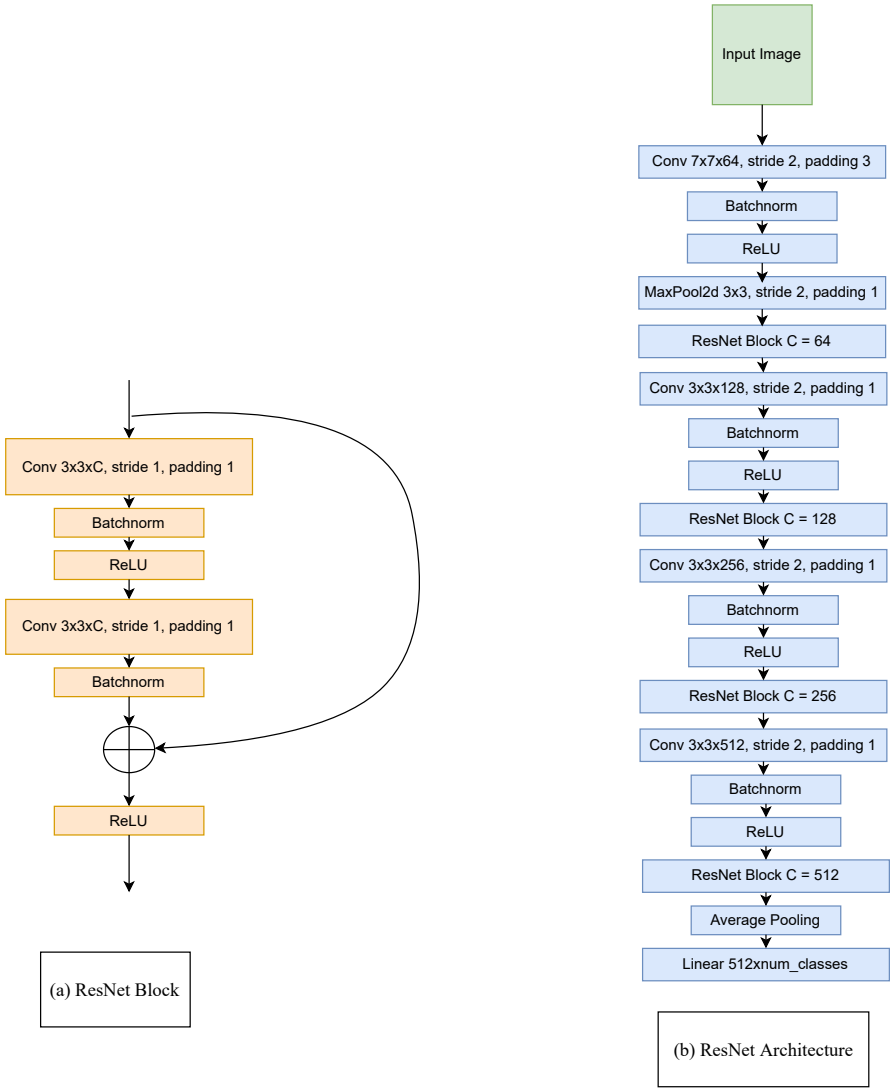

Figure 2: **Simplified ResNet.** You will implement a simplified ResNet. The network (right) reuses the residual block (left) with different numbers of channels, at each spatial scale.

1 GPU Tips

For the remainder of the class, we will be using GPUs with Google Colab. We use Colab, since we believe that it is the best possible option (given the fact that high quality GPUs are very expensive). However, there are a number of challenges due to the limits that Colab puts on GPU usage for free accounts. To help you get some information on this and make the most use of the free limits we have compiled a list of best practices to follow.

Colaboratory or simply Colab is part of the Google suite of products. Put simply, Colab is a free Jupyter notebook environment that runs entirely in the cloud. You can read more about it here: https://colab.research.google.com/. We use Colab with Python environment and to speed up model training we will be using a very important feature of Colab which is GPU access.

Colab usually provides K80 GPU for free accounts. Very rarely you'll get a T4, P100 or maybe even V100, which are much faster. Colab has time and usage limits for GPU access and the rules for these limits are vague so we have to be careful using them. The general observation is that you start off with a fixed (4 to 8hr, could be lower) bank. Once you exhaust that, you will not longer be able to use GPU for upto 24hrs. Once the counter resets, you get lower hours of usage each time. So how do we maximize our free GPU resources?

(1) **Use CPU for debugging**

The majority of the implementation in most of the assignments can be completed in CPU; you should only switch to GPU once you are confident about your code and ready to train your finals models. The notebook we give you might connect to a GPU runtime by default. You can change between CPU and GPU instances using Runtime → Change Runtime Type; this resets the notebook backend, so you'll need to rerun all cells after switching instance types. You will also need to change `device = torch.device('cuda')` to `device = torch.device('cpu')` Please make sure you don't have any `.cuda()` statements but rather `.to(device)`

(2) **Use other personal Gmail accounts**

Google provides Colab for all accounts and hence you can use your personal accounts to access Colab GPU too. Just share the notebook from you UMich drive with your personal drive and access it from there. This was you effectively have twice (or more) the free limits. Though please note that Google might have fraud detection if you make too many spam accounts.

Enable cell execution notifications Colab can notify you of completed executions even if you switch to another tab, window, or application. You can enable it via Tools → Settings → Site → Show desktop notifications (and allow browser notifications once prompted) to check it out. This is helpful when training models so you don't leave GPU connected and idle for long times.

(3) **Debugging tips for GPU**

```
RuntimeError: CUDA device-side assert triggered
```

Do not rely on the line where this error is raised. Even the error traceback says this:

RuntimeError: CUDA error: device-side assert triggered CUDA kernel errors might be asynchronously reported at some other API call, so the stacktrace below might be incorrect. For debugging consider passing `CUDA_LAUNCH_BLOCKING=1`

The stack trace might be incorrect. For better debugging, re-run your code as

```
CUDA_LAUNCH_BLOCKING=1 python script.py
```

or add

```
import os
os.environ["CUDA_LAUNCH_BLOCKING"] = 1
```

at the top of your code (temporarily) for debugging. The traceback you get now will point you to the actual line where the error occurs.

(4) **Non-Colab options**

We highly recommend using Colab. However, there are several other options.

One option is to use UM CAEN GPU compute. You can find instructions here: https://docs.google.com/document/d/1UuGfBfQ_yXq8EOgXad8OM6gJKUNexImQnJVeKlSnRPI/edit?usp=sharing. In our experience, this approach is also less reliable, so we recommend using Colab when possible. Also, since the problem sets are designed for Colab, running them here will require minor modifications, which we've described in the document. If you have a GPU in your local machine and can set it up to use within Jupyter notebooks, you can use it to run the notebooks. You can also connect Colab to a local runtime with GPU. Instructions here: https://research.google.com/colaboratory/local-runtimes.html. Please note that the instructors will not be able to help debug environment setup and code issues on non-Colab systems.

(5) **Colab Pro**

Google offers paid premium upgrades: Colab Pro and Pro Plus which are $10/month and $50/month respectively. These upgrades have access to faster GPUs and for longer. If you need more access (especially during final projects) you can consider these options. **Please note that buying premium Colab is not mandatory and the course will not offer any financial compensation for upgrading.** All assignments will be thoroughly tested to ensure that it can completed within the limits of free Colab resources. In fact, payments are blocked for UMich Google accounts. You can buy Pro for your personal accounts if needed. For students who would like to use this (optional) service, but are unable to afford it, we have been provided with a small amount of funding from the CSE DEI office. Please send the course staff a private message over Piazza or email if you would like to learn more about this option.

References

[1] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

[2] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *ICLR*, 2015.