ECE549 / CS543 Computer Vision: Assignment 4

Instructions

1. Assignment is due at **11:59:59PM on Wednesday April 29, 2020**.
2. Course policies: http://saurabhg.web.illinois.edu/teaching/ece549/sp2020/policies.html.
3. For starter code, and data for this assignment, please follow the first section below to setup on Colab.
4. Submission instructions:
   (a) A single report for both parts in PDF format, to be submitted to gradescope.
   (b) You also need to submit code for all questions in the form of a single .zip file. Submit a single zip file to Compass2g.
   (c) The LaTeXsource is available: http://saurabhg.web.illinois.edu/teaching/ece549/sp2020/mp/mp4-latex.zip.
   (d) We reserve the right to take off points for not following submission instructions.

Change log

v0	04/11/2020	Creation. pdf latex source

Google Colab and Dataset setup. In this assignment you will use PyTorch, which is currently one of the most popular deep learning frameworks and is very easy to pick up. It has a lot of tutorials and an active community answering questions on its discussion forums. You will be using Google Colab, a free environment to run your experiments. Here are instructions on how to get started:

1. Login with your Illinois Google account.
2. Open Q1 Starter code and Q2 Starter code, click on `File` in the top left corner and select `Save a copy in Drive`. This should create a new notebook, and then click on `Runtime` → `Change Runtime Type` → `Select GPU` as your hardware accelerator. Make sure you copied both Q1 (Fashion MNIST) and Q2 (Semantic Segmentation) to your Drive.
3. In your Google Drive, create a new folder titled `CS_543_MP4`. This is the folder that will be mounted to Colab. All outputs generated by Colab Notebook will be saved here.
4. Within the folder, create a subfolder called `data`. Copy all the files in this Google drive link to your data folder. Please do not change the file structures.
5. Follow the instructions in the notebook to finish the setup.
6. Keep in mind that you need to keep your browser window open while running Colab. Colab does not allow long-running jobs but it should be sufficient for the requirements of this assignment.

1. **Improving BaseNet on FashionMNIST [25 pts][1].** For this part of the assignment, you will be working with the FashionMNIST dataset (already loaded above). This dataset consists of 70K 28×28 color images from 10 classes. There are 60K training images and 10K test images. The images in FashionMNIST are of size $1 \times 28 \times 28$, i.e. single-channel gray images of 28×28 pixels. We have modified the standard dataset to create the FashionMNIST_CS543 dataset which consists of 60K training images, 5K validation images, and 5K test images.

   BaseNet. We created a BaseNet that you can run and get a baseline accuracy (87% on the validation set). The starter code for this is in the BaseNet class. It uses the following neural network layers:

   - Convolutional, i.e. `nn.Conv2d`
   - Pooling, e.g. `nn.MaxPool2d`
   - Fully-connected (linear), i.e. `nn.Linear`
   - Non-linear activations, e.g. `nn.ReLU`

   BaseNet consists of two convolutional modules (conv-relu-maxpool) and two linear layers. The precise architecture is defined below:

BaseNet Architecture

Layer No.	*Layer Type*	*Kernel Size*	*Input Dim*	*Output Dim*	*Input Channels*	*Output Channels*
1	conv2d	5	28	24	1	6
2	relu	-	24	24	6	6
3	maxpool2d	2	24	12	6	6
4	conv2d	5	12	8	6	16
5	relu	-	8	8	16	16
6	maxpool2d	2	8	4	16	16
7	linear	-	1	1	256	5
8	relu	-	1	1	5	5
9	linear	-	1	1	5	10

   Your goal is to edit the BaseNet class or make new classes for devising a more accurate deep net architecture. In your report, you will need to include a table similar to the one above to illustrate your final network.

   Before you design your own architecture, you should start by getting familiar with the BaseNet architecture already provided, the meaning of hyper-parameters and the function of each layer. This tutorial by PyTorch is helpful for gearing up on using deep nets. Also, see Andrej Karpathy's lectures on CNNs and neural network training.

   Improve your model. As stated above, your goal is to create an improved deep net by making judicious architecture and implementation choices. For improving the network, you should consider the following aspects. In addition, you can also try out your own ideas. Since Colab makes only limited computational resources available, we encourage you to rationally limit training time and model size. Remember to evaluate all these choices on the validation set, and run only your best model on the test set. Also, do not simply just copy over model architectures from the Internet.

   - **Data normalization.** Normalizing input data makes training easier and more robust. Similar to normalized epipolar geometry estimation, data in this case too could be made zero mean and fixed standard deviation ($\sigma = 1$ is the go-to choice). You may use `transforms.Normalize()` with the right parameters for this data normalization. After your edits, make sure that `test_transform` has the same data normalization parameters as `train_transform`.
   - **Data augmentation.** Augment the training data using random crops, horizontal flips, etc. You may find functions `transforms.RandomCrop()`, `transforms.RandomHorizontalFlip()` useful. Remember, you shouldn't augment data at test time. You may find the PyTorch tutorial on transforms useful.

[1] Adapted from L. Lazebnik.

- **Deeper network.** Experiment by adding more convolutional and fully connected layers. Add more `conv` layers with increasing output channels and also add more linear (fc) layers.
- **Normalization layers.** Normalization layers may help reduce overfitting and improve training of the model. Add normalization layers after `conv` layers (`nn.BatchNorm2d`). Add normalization layers after linear layers and experiment with inserting them before or after ReLU layers (`nn.BatchNorm1d`).

In your report include:

(a) **Your best model [10 pts].** Include final accuracy on test set, table defining your final architecture (similar to the BaseNet table above), training loss plot and validation accuracy plot for final model (auto-generated by the notebook). A reasonable submission with more than 90% accuracy will be given full credit for this part.

(b) **Factors that helped improve model performance [15 pts].** Explain each factor in 2–3 lines, and report a *control experiment* that measures the effectiveness of this factor: a comparison of two model variants, that only differ in the factor you are studying.

2. **Semantic Segmentation [35 pts].** In this part, you will build your own semantic segmentation model on the Stanford Background Dataset [1]. This task comprises of classifying image pixels into the following 9 categories: sky, tree, road, grass, water, building, mountain, foreground, misc. We will use the mean average precision (on the soft output) and mean intersection over union (of the hard output) to measure performance. We provide code for computing these metrics.

   Data. We have 571 images for training, 71 images for validation and 70 images for testing. Each image is 288×224. We provide a basic data loader that you can build upon. The dataset should already in the data folder (`data/sbd`). If not, go to the top of the PDF to see if you copied the data folder correctly to your drive.

   What you need to do:

   (a) **Implement training cycle:** Unlike Part 1 where we write the training cycle for you, you will implement it all by yourself this time. Make sure you evaluate metrics and loss on the validation set every so often to check for overfitting. You may also want to save well-performing models, for final testing on the test set.

   (b) **Develop your model [15 pts]:** Once you have a training cycle set up, you should design models for solving the task. You should experiment with different architectures, regularization, loss functions, data normalization and augmentation, etc. You can look at the following (or other) papers for inspiration [2, 3]. In your report, carefully document and report the major things you try, by noting the key architectural details, design rationale, and the impact on training plots and validation metrics. You should also include the relevant training plots and validation metrics.

   For reference, our very basic first implementation is able to do 1 training epoch in under 2 seconds, and achieves a mAP of 0.56 and a mIoU of 0.38 in under 20 minutes of training. At the very least your implementation should achieve as much accuracy on the validation set, but you may be able to do better with more training, and trying out alternate architectures.

   (c) **Build on top of ImageNet pre-trained Model [15 pts]:** Your next task is to build on top of a ResNet 18 [4] model that has been pre-trained on the ImageNet dataset [5] (via `models.resnet18(pretrained=True)`. These models are trained to predict the 1000 ImageNet object classes. To use this model for semantic segmentation, you will have to remove the classifier and global average pooling layers, and stack on additional layers for semantic segmentation. Once again, refer to papers pointed out above for inspiration for how you can build on top of such pre-existing models. For the purpose of this assignment, please keep the part of the model derived from the ImageNet pre-trained model fixed (you will have to figure out how to do so in PyTorch). Again, as above, carefully document the design choices you make in your report.

   For reference, our very basic first implementation is able to do 1 training epoch in 3.5s, and achieves a mAP of 0.59 and a mIoU of 0.43 in under 20 minutes of training. At the very least your implementation should achieve as much accuracy on the validation set, but you may be able to do better with more training, and trying out alternate architectural choices.

(d) **Report Performance on the Test Set [5 pts]:** You should use your best models from part (b) and (c) above, to report performance on the test dataset. In your report, include the PDF file that is produced by the evaluation code, and the architectural and training details for the two final models.

References

[1] Stephen Gould, Richard Fulton, and Daphne Koller. Decomposing a scene into geometric and semantically consistent regions. In *2009 IEEE 12th international conference on computer vision*, pages 1–8. IEEE, 2009.

[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] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[5] J Deng, W Dong, R Socher, LJ Li, K Li, and L Fei-Fei. A large-scale hierarchical image database. In *CVPR*, pages 248–255, 2019.