CS 6476: Computer Vision, Spring 2024
PS4

Instructor: Judy Hoffman

Due: Thursday, April 4th, 2024 11:59 pm ET

Setup

Note that we will be using a new conda environment for this project! If you run into import module errors, try `pip install -e .` again, and if that still doesn't work, you may have to create a fresh environment.

1. Install Miniconda. It doesn't matter whether you use Python 2 or 3 because we will create our own environment that uses 3 anyways.
2. Open the terminal
   (a) On Windows: open the installed **Conda prompt** to run the command.
   (b) On MacOS: open a terminal window to run the command
   (c) On Linux: open a terminal window to run the command
3. Navigate to the folder where you have the project
4. Create the conda environment for this project.
   (a) On Windows: `conda env create -f proj4_env_win.yml`
   (b) On MacOS: `conda env create -f proj4_env_mac.yml`
   (c) On Linux: `conda env create -f proj4_env_linux.yml`
5. Activate the newly created environment
   (a) On Windows: use the command `conda activate cs6476_proj4`
   (b) On MacOS: use the command `source activate cs6476_proj4`
   (c) On Linux: use the command `source activate cs6476_proj4`
6. Install the project files as a module in this conda environment using `pip install -e .` (Do not forget the `.`).
7. If you run into issues with Pytorch on Windows, please check if you have the 64 bit version installed.

Run the notebook using `jupyter notebook ./proj4_code/proj4.ipynb`.

At this point, you should see the jupyter notebook in your web browser. Follow all the instructions in the notebook for both the code + report portions of this project.

Assignment Overview

The goal of this project is to get familiar with convolutional neural networks to classify the scenes into different categories and generating pixel level segmented images. Part 1-5 and EC1 focus on classification task, and Part 6 and EC2 works towards the semantic segmentation task. The basic learning objectives of this project are:

- Construct the fundamental pipeline for performing deep learning using PyTorch
- Understand the concepts behind different layers, optimizers
- Experiment with different models and observe the performance

In order to train a model in Pytorch, following four components:

- Dataset: an object which can load the data and labels given an index (Part 1)
- Model - an object that contains the network architecture definition (Part 2)
- Loss function - a function that measures how far the network output is from the ground truth label (Part 3)
- Optimizer - an object that optimizes the network parameters to reduce the loss value (Part 4)

Dataset Details for Classification

The dataset used in this project is the same as PS4. The dataset is 15 scene dataset, with natural scenes from 15 different scenarios. It was first introduced by Lazebnik et al, 2006. Typically the image size is 256 by 256 pixels. The data folder will have two subfolders: train and test. We want our model to predict the class of the image from the RGB image. A sample collection of images are presented below:

Figure 1: Dataset

1 Dataset Loaders and Data preprocessing (Classification)

In this part we will setup the workflow to load the dataset and pre-processing. In DL we generally tend to perform some preprocessing like normalization and transformation. The transformations will aid to make the inputs compatible to the model input dimensions or help in data-augmentation to improve performance.

1.1 Compute mean and std. deviation of the dataset

- Implement in compute_mean_and_std() in proj4_code/classification/stats_helper.py
- Convert the image to grayscale and scale to [0,1] before computing mean and standard deviation
- Use StandardScaler function in the sklearn.preprocessing library

1.2 Data Loaders

We will be using the Pytorch's ImageFolder dataloader here. The images in data folder are organized such that each class name is the folder with the images belonging to that class inside the folder. We'll use the ImageFolder to make two loaders, for train and test split respectively.
Note: You don't have to implement anything in this subpart, go through the documentation of ImageFolder for better understanding.

1.3 Data Transforms

Implement the function get_fundamental_transforms() in proj4_code/classification/data_transforms.py. In this part we will be constructing some fundamental transforms to convert the RGB image to torch tensors:

- Resize: Resize the image in accordance with input dimensions of the architecture
- Grayscale: Convert the loaded image to grayscale (Although the colors of the image are gray, the ImageFolder will read the images as RGB)
- Convert it to tensor
- Normalize: Normalize the images based on mean and std. deviation computed in part 1.1
- Use torchvision's transforms

2 Model Architecture and Forward Pass (Classification)

In this part of the project, you will be working on implementing a simple 2-layer deep learning architecture in Pytorch.

2.1 SimpleNet Model

You need to add 2 convolutional layers, with supporting utilities like ReLU, Max Pooling, and Fully Connected layers with proper parameters(kernel size, padding etc), to make the output compatible with input for next layer. You can use the following image for a sample network architecture. In the image, the layers in use are written at the bottom with the kernel size, and the blocks show the shape of features after each layer.

To do: Implement the following in the proj4_code/classification/simple_net.py

- Initialize self.conv_layers

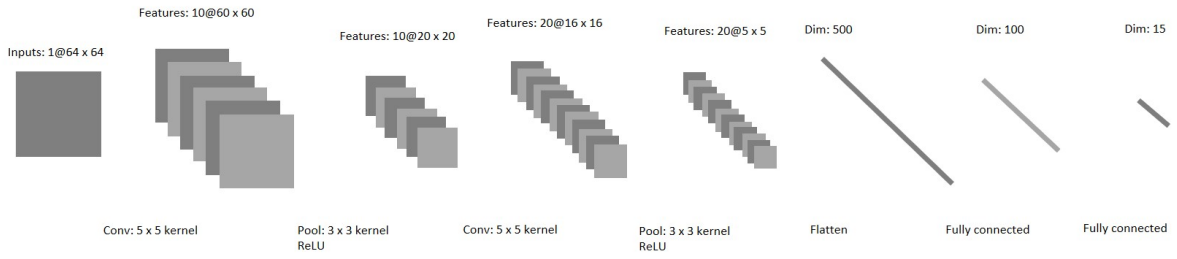

Figure 2: SimpleNet Architecture

- Initialize self.fc_layers
- Write the forward function

2.2 Output Prediction

In this section, you will be looking at how the model generates output and how is it related to classification. The model has been initialized with random weights; hence the output will not reflect the model performance. We will be using a data-point from the ImageFolder object defined in Part1.2. Passing the gray-scale image through the SimpleNet model, a 15-dimensional tensor is returned as output. This 15-dimensional tensor can be converted to probability of the image belonging to each of the class with Softmax activation. To save computation, you will be directly taking the index where the value is maximum in the 15-dimensional tensor returned by the model. Think about how the argmax operation on the 15 dimenasional tensor is same as the argmax operation on the softmax output.

- Implement the function predict_labels() in proj4_code/classification/dl_utils.py

3 Loss Function (Classification)

In the previous section, you saw how the output from the model relates to classification. Since the model was initialized randomly the output does not depict the model performance. Technically should produce a probability of 1 (or close to 1) for the target sample_label and 0 for the other classes. In order to train the model to do so, you will be implementing a loss function to penalize the deviation between the desired probability and the model's predicted output. You will be using KL divergence or cross-entropy loss. Refer to this stackexchange post for a complete explanation.
Implement the following:

- Assign a loss function to self.loss_criterion in proj4_code/classification/simple_net.py. Refer to Pytorch loss documentation.
- Compute the loss in compute_loss() in proj4_code/classification/dl_utils.py to use the model's loss criterion.

4 Optimizer (Classification)

In this section, you will be working on minimizing the loss to optimize the weights of the network.

4.1 Manual gradient descent using Pytorch's autograd

Pytorch uses the autograd feature to use gradient optimization and escape manual computations.

- Complete compute_quadratic_loss() in proj4_code/classification/optimizer.py : Define objective function L(w)

$$L(w) = w^2 - 10w + 25 \tag{1}$$

- Implement a single step of gradient descent in function gradient_descent_step() in proj4_code/classification/optimizer.py

4.2 Optimization with Pytorch's gradient descent optimizer

You will be using torch.optim. You can experiment with different features of the vanilla gradient descent.

4.3 Setting up the optimizer for SimpleNet

- Initialize and adjust the values of learning rate and weight decay to improve the model performance.
- Implement the get_optimizer() function in optimizer.py

5 Training SimpleNet

In this section, you will be training the SimpleNet architecture. Pass in the model architecture, optimizer and the transforms to the Trainer function.

- Adjust the value of your parameters, to obtain 45% validation accuracy to get full credit.

6 Semantic Segmentation with Deep Learning

6.1 Implementation

We'll start with a very simple baseline, a pretrained ResNet-50, without the final averagepool/fc layer, and a single 1x1 conv as a final classifier, converting the (2048,7,7) feature map to scores over 11 classes, a (11,7,7) tensor. Note that our output is just 7x7, which is very low resolution.

- Implement upsampling to the original height and width, and compute the loss and predicted class per pixel in proj_4_code/segmentation/simple_segmentation_net.py .

For this part, the majority of the details will be provided into two separate jupyter notebooks, The first, `proj4.ipynb` includes unit tests to help guide you with local implementation (Sec 6 Semantic Segmentation with Deep Learning). After finishing that, upload `proj4_code/proj4_colab.ipynb` to Colab. Next, zip up the files for Colab with our script `zip_for_colab.py`, and upload these to your Colab environment.
Train the model and report the mIoU score in the report. Record the results and upload the predictions, `grayscale_predictions.zip` to the Autograder. **Attempt to achieve an mIoU $>$ 40% for full credit.**

6.2 Dataset

The dataset to be used in this assignment is the Camvid dataset, a small dataset of 701 images for self-driving perception. It was first introduced in 2008 by researched at the University of Cambridge. You can read more about it at the original dataset paper or in the paper describing it. The images have a typical size of around 720 by 960 pixels. We'll down-sample them for training though since even at 240 x 320 px, most of the scene detail is still recognizable.
Today there are much larger semantic segmentation datasets for self-driving, like Cityscapes, WildDash V2, Audi A2D2, but they are too large to work with for a homework assignment.
The original Camvid dataset has 32 ground truth semantic categories, but most evaluate on just an 11-class subset, so we'll do the same. These 11 classes are 'Buliding', 'Tree', 'Sky', 'Car', 'SignSymbol', 'Road', 'Pedestrian', 'Fence', 'Column Pole', 'Sidewalk', 'Bicyclist'. A sample collection of the camvid images can be found here.

(a) Image A, RGB (b) Image A, Ground Truth (c) Image B, RGB (d) Image B, Ground Truth

Figure 3: Example scenes from the Camvid dataset. The RGB image is shown on the left, and the corresponding ground truth "label map" is shown on the right.

6.3 Setup Information

- Run the notebook using `jupyter notebook ./proj4_code/proj4.ipynb`.
- After implementing all the functions, ensure that all sanity checks are passing by running `pytest` inside the repository folder.
- After passing the unit tests, run `python zip_for_colab.py` which will create `cv_proj4_colab.zip`
- Upload `proj4_code/proj4_colab.ipynb` to Colab
- Upload `cv_proj4_colab.zip` to session storage
- Click Run All inside Runtime which will train your model
- Record the results and download `grayscale_predictions.zip`
- Generate the zip folder for the code portion of your submission once you've finished the project using `python zip_submission.py -gt_username <your_gt_username>`

7 EC1: Solving Overfitting (Classification)

Observing the train and validation losses, you will observe that the training accuracy is over 90% while the validation accuracy does not go past 45%. This is an indication of overfitting; the model fits too well with the training data, but the generalization capabilities of the model are limited.

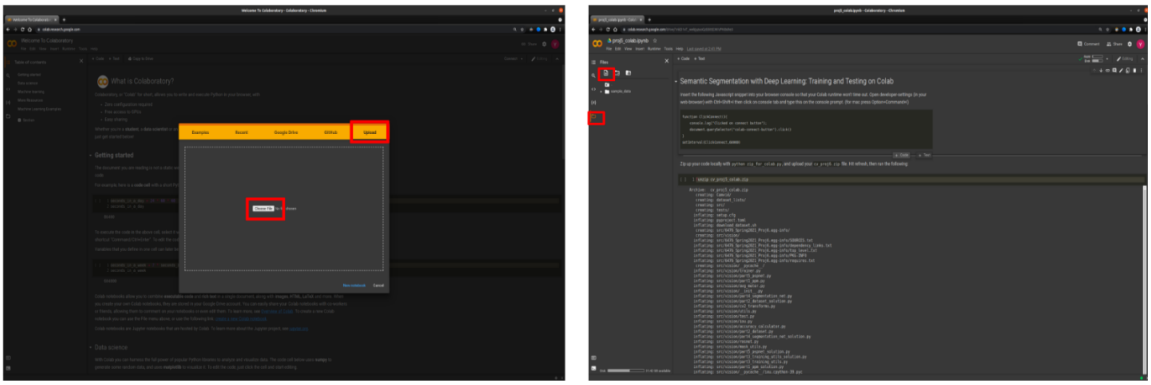

Figure 4: Figure describing setup steps

7.1 Jitter, Random Flip, and Normalization

Since we don't have a huge amount of data, we will be adding augmentations to the data, so that the model sees "different" images during every iteration.

- finish the get_data_augmentation_transforms() function in proj4_code/classification/data_transforms.py

7.2 Dropout

Dropout tries to solve the problem of co-adaption. It randomly turns off the connection betweehn neurons in order to make the neurons independent of each other.

- finish the proj4_code/classification/simple_net_dropout.py with your previous SimpleNet model, plus the dropout layer.
- Achieve 52% validation accuracy for full credits for this part.

8 EC2: PSPNet and ResNet50 (Semantic Segmentation)

We'll be implementing PSPNet for this project, which uses a ResNet-50 backbone. First, we will provide some background information to familiarize yourself with the network and architecture.

8.1 Implementation

For this part, the majority of the details will be provided into two separate jupyter notebooks, The first, `proj4.ipynb` includes unit tests to help guide you with local implementation (Sec EC2 PSPNet and ResNet-50). After finishing that, upload `proj4_colab.ipynb` to Colab. Next, zip up the files for Colab with our script `zip_for_colab.py`, and upload these to your Colab environment.
We will be implementing the PSPNet architecture. This network uses a ResNet backbone but uses dilation to increase teh receptive field, and aggregates context over different portions of the image with a "Pyramid Pooling Module" (PPM).

You can read more about dilated convolution in the Dilated Residual Network here, which PSPNet takes some ideas from. Also you can a helpful animation about dilated convolution here.

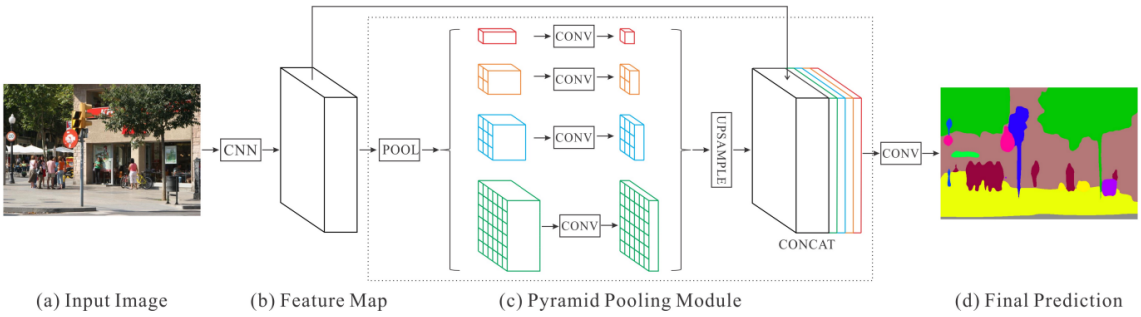

Figure 5: PSPNet Architecture. The Pyrmid pooling module (PPM) splitsthe H X W featue map into KxK grids. Here, 1 x 1, 2x 2, 3 x 3 and 6 x 6 grids are formed, and features are average-pooled within each grid cell. Afterwards, the 1 x 1, 2x 2, 3 x 3 and 6 x 6 grids are upsampled back to the original H x W feature map resolution, and are stacked together along the channel dimension.

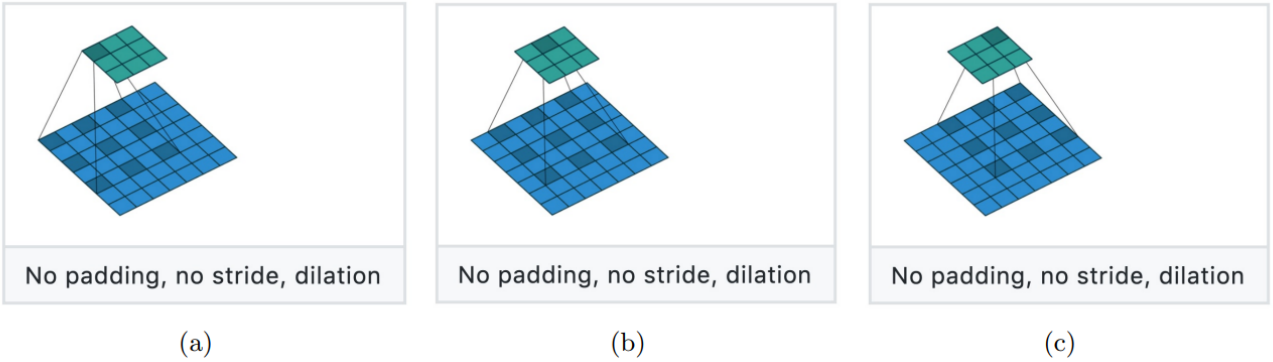

Figure 6: Dilated convolution. Figure Sources : https://github.com/vdumoulin/conv_arithmeticdilated-convolution-animations

8.2 ToDo:

1. Initialize the Resnet-50 backbone in proj4_code/segmentation/pspnet.py , specifically in the __init__ function.
2. Initialize the PPM Module in proj4_code/segmentation/pspnet.py . The PPM module has already been implemented here (proj4_code/segmentation/ppm.py).
3. Complete the rest of the functions for Net Surgery in proj4_code/segmentation/pspnet.py:
   - __replace_conv_with_dilated_conv()
   - __create_classifier()
   - forward()
4. Train the model and **achieve > 60% mIoU**, and report the mIoU score in the report.

9 Tips and tricks

- The shape of the features is of the form (N, C, H, W); N: batch size, C: number of channels, H,1: Height and width of the feature
- While selecting the loss function, keep in mind there is no softmax layer at the end of the model
- In this project, we will be using the test set as the validation set (i.e. using it to guide our decisions about models and hyperparamters while training). In actual practise, you would not interact with the test set until reporting the final results.
- Your CPU should be sufficient to handle the training process for the classification networks in this project, but you will need GPU (via e.g. colab) for the segmentation networks. You may also want to decrease the value for num_epochs and quickly experiment with your parameters. The default value of 30 is good enough to get you around the threshold for Part 1, and you are free to increase it a bit and adjust other parameters.
- If the loss decreases very slowly, try increasing the value of the lr (learning rate).
- If the loss does not decrease, this may be because that the learning is too low or the weight decay is too high so the model is taking too long to converge.
- Try to first adjust lr in multiples of 3 initially. When you are close to reasonable performance, do a more granular adjustment. Do try to keep lr values less than 0.1.
- If you want to increase the validation accuracy by a little bit, try increasing the weight_decay to prevent overfitting. Do not use tricks from Section 6 just yet.
- Try jittering the colors, and flipping the image horizontally. These two operations wont change the scene contents. You can also try resizing and cropping.

Submission Instructions

You will have 2 submission files for this project:

- Submit the code and the grayscale predictions as zip (python zip_submission.py –gt_username <username>) on **Gradescope**.
- Submit the report as <your_gt_username>.pdf on **Gradescope**

There is no submission to be done on Canvas.

Rubric

This problem set has 100 points of mandatory credits and 20 extra credits.

Code: The score for each part is provided below. Please refer to the submission results on Gradescope for a detailed breakdown.

Part 1: Dataset	10
Part 2: Model Architecture and Forward Pass	25
Part 3: Loss function	10
Part 4: Optimization	10
Part 5: Training SimpleNet	10
Part 6: Simple Segmentation Net	20
EC 1.1: Data Augmentation	3
EC 1.2: SimpleNetDropout	3
EC 2: PSPNet	6
Total	*85 (+12)*

Report: The report is worth 20 points. Please refer to the pptx template where we have detailed the points associated with each question.

Part 1: Dataset	3
Part 3: Loss function	3
Part 5: Training SimpleNet	2
Part 6: Simple Segmentation Net	2
Conclusion	1
Theory	4
EC 1: Overfitting	4
EC 2: PSPNet	4
Total	*15 (+8)*

Deliverables

The following code deliverables will be uploaded as a zip file on Gradescope.

1. `proj4_code/classification/stats_helper.py`
   (a) `compute_mean_and_std()`
2. `proj4_code/classification/data_transforms.py`
   (a) `get_fundamental_transforms()`
   (b) `get_data_augmentation_transforms()` - (Extra Credit)
3. `proj4_code/classification/simple_net.py`
   (a) `__init__()`
   (b) `forward()`
4. `proj4_code/classification/dl_utils.py`
   (a) `predict_labels()`
5. `proj4_code/classification/dl_utils.py`
   (a) `predict_labels()`
   (b) `compute_loss()`

6. `proj4_code/classification/optimizer.py`
   (a) `compute_quadratic_loss()`
   (b) `gradient_descent_step()`
   (c) `get_optimizer()`
7. `proj4_code/classification/simple_net_dropout.py` - (Extra Credit)
   (a) `__init__()`
   (b) `forward()`
8. `proj4_code/segmentation/simple_segmentation_net.py`
   (a) `forward()`
9. `grayscale_predictions.zip`
10. `proj4_code/segmentation/pspnet.py` - (Extra Credit)
    (a) `__init__()`
    (b) `__replace_conv_with_dilated_conv()`
    (c) `__create_classifier()`
    (d) `forward()`

Do not create this zip manually. You are supposed to use the command `python zip_submission.py -gt_username <username>` for this.

The second thing to upload is the PDF export of the report on gradescope.

This iteration of the assignment is developed by Deepanshi, George Stoica, Vivek Vijaykumar and Judy Hoffman.

This assignment was developed and maintained by Ayush Baid, Cusuh Ham, Jonathan Balloch, Haoxin Ma, Jing Wu, Shenhao Jiang, Frank Dellaert, James Hays, Judy Hoffman, Deepanshi, George Stoica, and Vivek Vijaykumar.