EECS 442 Computer Vision: Homework 4

Instructions

- This homework is **due at 11:59:59 p.m. on Sunday March 31th, 2019**.
- The submission includes two parts:
  1. **To Gradescope:** a `pdf` file as your write-up. Please reading the grading checklist for each part before you submit it.
     You might like to combine several files to make a submission. Here is an example online link for combining multiple PDF files: https://combinepdf.com/.
     Please mark where each question is on gradescope.
  2. **To Canvas:** a `zip` file including all your code.
- The write-up must be an electronic version. **No handwriting, including plotting questions.** LaTeX is recommended but not mandatory.

1 Image Classification [80 pts]

In part 1, you will implement Convolutional Neural Networks (ConvNets) from scratch to classify images. You will have to write code at two levels of abstraction: both specific layer implementations and your own Convolutional Neural Networks based on these layer functions. **You cannot use any deep learning libraries such as PyTorch in this part.**

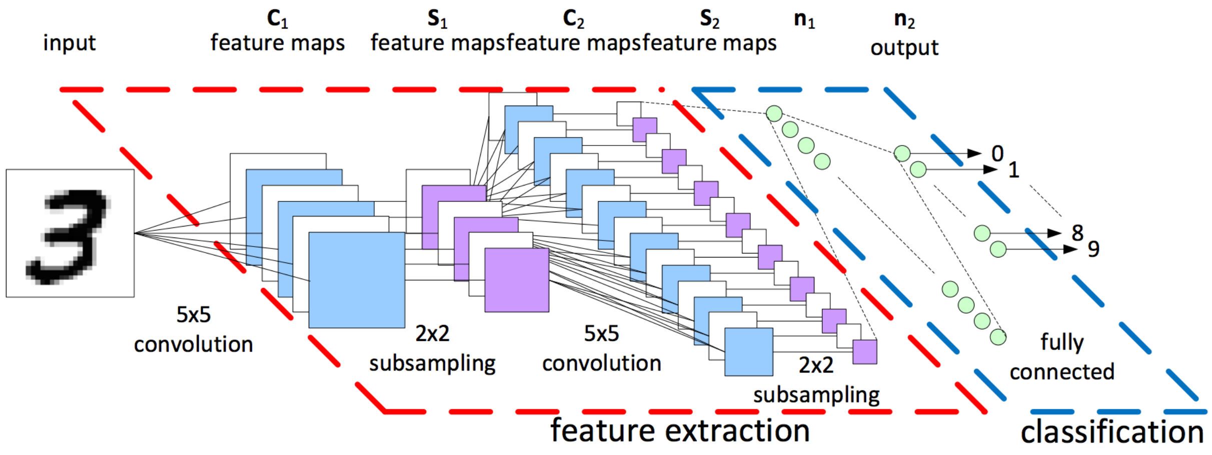


Figure 1: Convolutional Neural Networks for Image Classification[1]

[1] Image comes from https://medium.com/data-science-group-iitr/building-a-convolutional-neural-network-in-python-with-tensorflow-d251c3ca8117

In `layers.py`, you need fill in the implementation for a few layers. Each layer function has `forward` and `backward` methods. All layers functions are shown below:

- Linear layer.
$$y = Wx + b$$
- L2 loss.
$$L(x, \text{label}) = \frac{1}{N}\sum_{i=1}^{N}(x_i - \text{label}_i)^2$$
- ReLU layer.
$$y = \begin{cases} x, & x \geq 0, \\ 0, & \text{otherwise.} \end{cases}$$
- Softmax layer.
$$y_i = \frac{e^{x_i}}{\sum_{j=1}^{N} e^{x_j}}$$
- Cross-entropy loss.
$$L(x, \text{label}) = -\frac{1}{N}\sum_{i=1}^{N}\text{label}_i \cdot \log(x_i)$$
- Convlutional layer. Hints:
  - You can use `scipy.signal.correlate(x, kernel, mode='valid', method='fft')` to calculate 2d convolution. You can also implememnt the convolutional layer using nested loops.
  - The gradient of 2d convolutions can still be represented by 2d convolutions. Again, it's ok to implememnt the BP using nested loops.
- Maxpool layer.

In `network.py`, you need implement your network. You're only allowed to use layer functions you implement. However, you're encouraged to implement additional features in `layers.py` and use it here. Filling in all `TODO`s in skeleton codes should be sufficient to make it work.

You need test your network first on the given toy dataset. We provide you with `x.npy` and `y.npy`. It is basically a least-square problem. You can test your network on this toy dataset, this will verify if your network works. Report the lowest l2-loss you can achieve on this problem.

After making sure your network works, you need train it on Fashion MNIST dataset, which is available at https://github.com/zalandoresearch/fashion-mnist. Fashion MNIST has 10 classes, 60000 training images, and 10000 test images. You need split the validation set yourself. We have provided a python inferface to read the Fashion MNIST dataset in `train.py`, although you're free to modify it. We have also provided you the optimizer in `solver.py`. Hence, it is sufficient for you to just change hyperparameters in `train.py`.

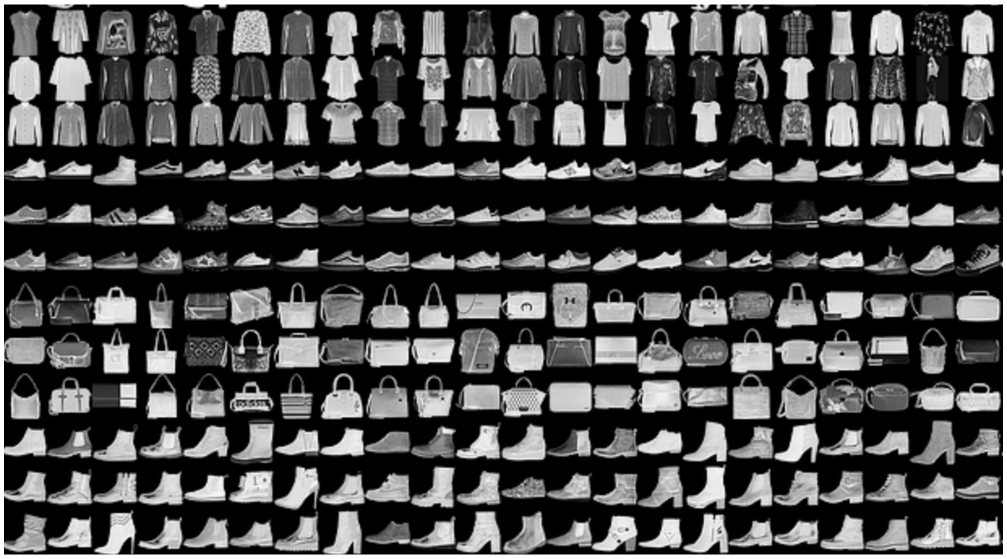


Figure 2: Example images from Fashion MNIST dataset [2]

This task is open-ended. However, here are some suggestions:

- When working on the backward function in `layer.py`, you need to calculate the gradients of the objective function. To check your implementation, you can take use of this:

$$\frac{\partial f(\cdots, x_i, \cdots)}{\partial x_i} = \lim_{\delta \to 0} \frac{f(\cdots, x_i + \delta, \cdots) - f(\cdots, x_i, \cdots)}{\delta}$$

  where $f : [x_1, \cdots, x_n] \in \mathbb{R}^n \mapsto \mathbb{R}$. So, you can derive some approximation for the gradients by setting some small value δ. You can compare the gradient your implementation gets with the numerical value of the approximation.

- Get a baseline model to work first. You can make it very simple. For example, a linear layer + l2 loss. You don't need to finish implementing all the layers before building a baseline model.

- After getting a baseline model working and implementing convolutional layers, you can start implementing a convolutional neural network. You can start by adding one or two convolutional layers into your network.

- Training convolutional neural networks can be hard and slow. It is expected to take several hours for the network to converge. Start early!

Grading checklist:

1. For all layers, we will have a series of tests that automatically test whether your functions are written correctly. You can run the provided test cases to verify your implementations yourself. You do not need to include any discussion of your code for part one unless you have decided to implement some extra features.

2. Report the lowest l2 loss you can achieve on the toy dataset.

3. Your report should detail the architecture you used to train on Fashion MNIST. Include information on hyper parameters chosen for training and a plot showing both training and validation loss across iterations.

4. Report the accuracy on the test set of Fashion MNIST. You should only evaluate your model on the test set once. All hyperparameter tuning should be done on the validation set. We expect you to achieve **90%** accuracy on the test set.

5. Include discussion of the parameter you chose for experimentation, and include your results and analysis. In terms of discussion, we expect to see plots and data instead of a general description.

2 Semantic Segmentation [20 pts]

Besides image classification, Convolutional Neural Networks can also generate dense predictions. A popular application is semantic segmentation. In part 2, you will implement your own Convolutional Neural Networks to perform semantic segmentation on the Wizarding Facade dataset. **You are required to use PyTorch in this part.**

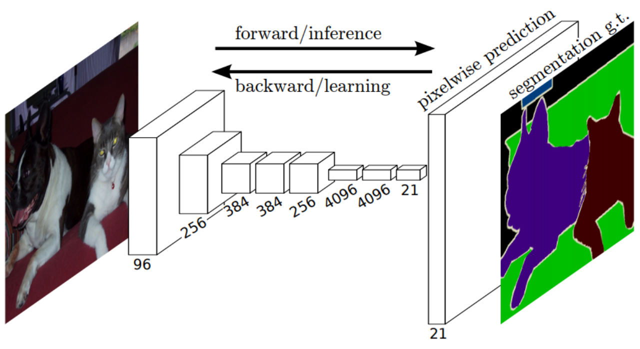
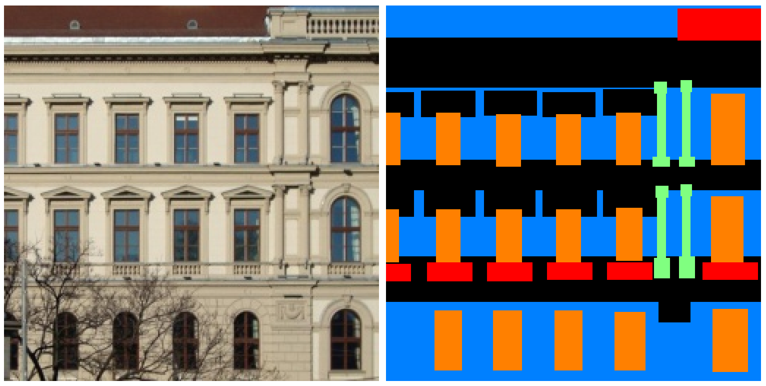


Figure 3: Fully convolutional networks can efficiently learn to make dense predictions for per-pixel tasks like semantic segmentation [1].

Wizarding Facade dataset consists of images of different cities around the world and diverse architectural styles (in `.jpg` format), shown as the image on the left. It also contains semantic segmentation labels (in `.png` format) in 5 different classes: `balcony, window, pillar, facade` and `others`. Your task is to train a network to convert image on the left to the labels on the right.



Note: The label images are plotted using 8-bit indexed color with pixel value listed in Table 1. We have provided you a visualization function using a "jet" color map.

Table 1: Label image consists of 5 classes, represented in [0, 4].

class	color	pixel value
others	black	0
facade	blue	1
pillar	green	2
window	orange	3
balcony	red	4

We have already given you a skeleton to run on this task. The skeleton code contains a dummy network, which is just a 1x1 convolution which convert 3 channels (RGB) to 5 channels, i.e. 5 heatmaps for each class. The loss function is Cross Entropy loss which is the same as that you implement in part 1. You can type `python train.py` and run it.

The metric we use to evaluate your model is average precision (AP). In general, the higher AP the better. We've already provided you the code to evaluate AP and you can directly use it. In case you're interested in how it works, an introduction to AP is available at https://scikit-learn.org/stable/modules/model_evaluation.html#precision-recall-f-measure-metrics.

Again, this task is open-ended. You're free to change the skeleton as you like. However, here are some suggestions:

- PyTorch has very nice documentation, which is available at https://pytorch.org/docs/stable/index.html. In terms of implementing the network, reading the documentation of a few important layers such as `Conv2d` of module `torch.nn` is essential (https://pytorch.org/docs/stable/nn.html).
- Training such a deep neural network can be hard and slow. Fortunately, modern GPUs can accelerate the training significantly. You're encouraged to use AWS to get a cloud GPU instance. Google Colab is also a good option. However, it can still take several hours for the network to converge. Start early!

Grading checklist:

1. Your report should detail the architecture you used to train on Wizarding Facade. Include information on hyper parameters chosen for training and a plot showing both training and validation loss across iterations.
2. Report the accuracy on the test set. You can use provided function to calculate AP on the test set. You should only evaluate your model on the test set once. All hyperparameter tuning should be done on the validation set. We expect you to to achieve **0.45** AP on the test set.
3. Include discussion of the architecture and parameters you chose for experimentation, and include your results and analysis. In terms of discussion, we expect to see plots and data instead of a general description.

3 Bonus points [10pts]

Extend your network in part 2 and try to get as good performance as you can, and submit it to the Kaggle leaderboard which will be available at least one week before the homework deadline. The bonus credits in this homework is more like a challenge, i.e. we'll give bonus credits according to your absolute performance and ranking of your performance in the class. To make sure the challenge is fair, the test set used on Kaggle is different from the test set we give you and you don't have ground truth. But they have similar distribution.

Again, here are some architectures you can try:

- Jonathan Long, Evan Shelhamer, Trevor Darrell. Fully Convolutional Networks for Semantic Segmentation. CVPR 2015.
- Olaf Ronneberger, Philipp Fischer, Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. MICCAI 2015.
- Alejandro Newell, Kaiyu Yang, Jia Deng. Stacked Hourglass Networks for Human Pose Estimation. ECCV 2016.

Python Environment We are using Python 3.7 for this course. You can find references for the Python standard library here: https://docs.python.org/3.7/library/index.html. To make your life easier, we **recommend** you to install the latest Anaconda for Python 3.7 (https://www.anaconda.com/download/). This is a Python package manager that includes most of the modules you need for this course.

We will make use of the following packages extensively in this course:

- Numpy (https://docs.scipy.org/doc/numpy-dev/user/quickstart.html).
- Matplotlib (http://matplotlib.org/users/pyplot_tutorial.html).
- Scipy (https://www.scipy.org/index.html). You may want to use `scipy.signal.convolve` or `scipy.signal.correlate` to implement 2d convolution.
- Scikit-learn (https://scikit-learn.org/stable/index.html). In this homework, we only use scikit-learn to read Fashion MNIST dataset. To install it, run `conda install scikit-learn`.
- PyTorch (https://pytorch.org/). In this homework, we're using the latest PyTorch in part 2 to implement the network. To install it, run `conda install pytorch torchvision -c pytorch`. More options such as GPU support are available on the website.
- OpenCV (https://opencv.org/). It's optional to use OpenCV. To install it, run `conda install -c menpo opencv`.

References

[1] Jonathan Long, Evan Shelhamer, Trevor Darrell. Fully Convolutional Networks for Semantic Segmentation. CVPR 2015.

[2] Han Xiao, Kashif Rasul, Roland Vollgraf. Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms. arXiv:1708.07747.

[3] Phillip Isola, Jun-Yan Zhu, Tinghui Zhou, Alexei A. Efros. Image-to-Image Translation with Conditional Adversarial Nets. CVPR 2017.

[4] Radim Tyleček and Radim Šára. Spatial Pattern Templates for Recognition of Objects with Regular Structure. GCPR 2013.

Acknowledgement

Part of the homework and the starter code are taken from previous EECS442 by Jia Deng and CS231n at Stanford University by Fei-Fei Li, Justin Johnson and Serena Yeung. The Wizarding Facade dataset are modified from CMP Facade Database by Radim Tyleček and Radim Šára. Please feel free to similarly re-use our problems while similarly crediting us.