EECS 442 Computer Vision: Homework 5

Instructions

- This homework is **due at 11:59:59 p.m. on Wednesday April 15th, 2020**.
- The submission includes two parts:
  1. **To Gradescope:** a `pdf` file as your write-up, including your plots and answers to all the questions and key choices you made.
     The write-up must be an electronic version. **No handwriting, including plotting questions.** LaTeX is recommended but not mandatory.
     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/.
  2. **To Canvas:** a `zip` file including a single directory named after your uniqname, containing all your code (in `.py` format) and files specified in questions with **Submit**. Remember to use the script `check_submission.py` (available at https://github.com/eecs442/utils) to check the format before you submit.

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 Anaconda(https://www.anaconda.com/distribution/). This is a Python package manager that includes most of the modules you need for this course.

In this homework, you are required to use PyTorch for building and training neural networks. Install PyTorch as `torch` and `torchvision` for datasets. You will also need `matplotlib.pyplot` to visualize results and `tqdm` to display a progress bar.

As a deep learning library, PyTorch performs backpropagation automatically for you and performs faster tensor operations. To familiarize yourself with PyTorch, we provide a notebook at https://github.com/eecs442/notebooks/blob/master/MNIST_pytorch.ipynb which contains a simple demo of image classification on MNIST dataset.

For visualization in Part 3, you will also need to install `pypng` and `colormap`.

1 Fashion-MNIST Classification [35 pts]

In this part, you will implement and train Convolutional Neural Networks (ConvNets) in **PyTorch** to classify images. Unlike HW4, backpropagation is automatically inferred by PyTorch, so you only need to write code for the forward pass.

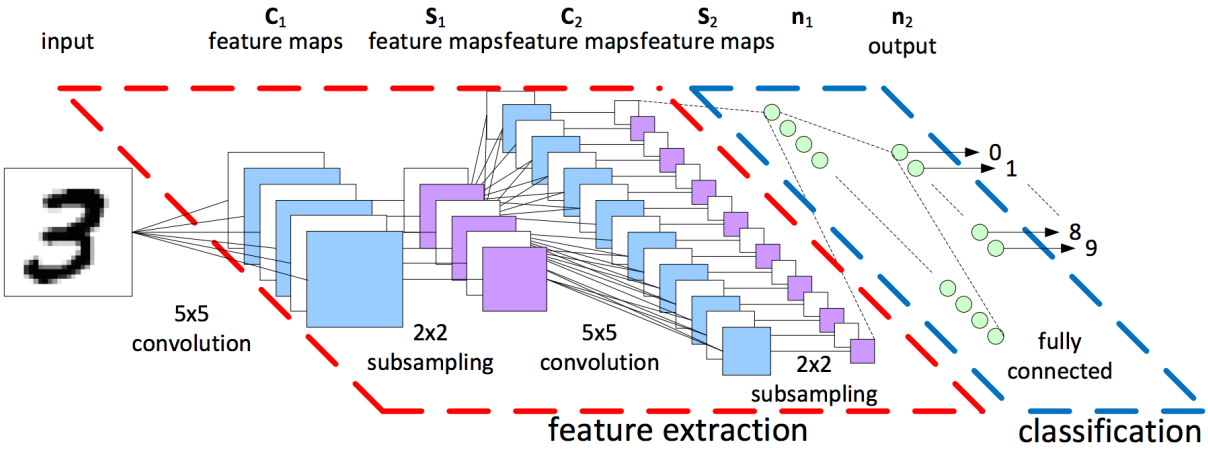


Figure 1: Convolutional Neural Networks for Image Classification[1]

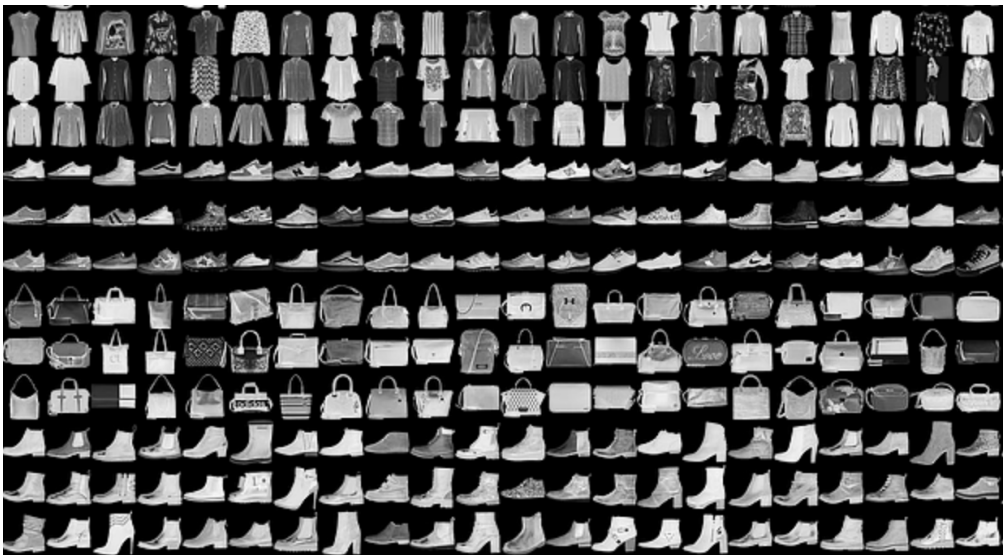


Figure 2: Example images from Fashion MNIST dataset [2]

The dataset we use is Fashion-MNIST dataset, which is available at https://github.com/zalandoresearch/fashion-mnist and in `torchvision.datasets`. Fashion-MNIST has 10 classes, 60000 training+validation images (we have splitted it to have 50000 training images and 10000 validation images, but you can change the numbers), and 10000 test images. We have provided some starter code in `part1.py` where you need to modify and experiment with the following:

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

- The architecture of the network (define layers and implement forward pass)
- The optimizer (SGD, RMSProp, Adam, etc.) and its parameters. (`weight_decay` is the L2 regularization strength)
- Training parameters (batch size and number of epochs)

You should train your network on training set and change those listed above based on evaluation on the validation set. You should run evalution on the test set **only once** at the end.

Complete the following:

1. **Submit** a program which trains with your best combination of model architecture, optimizer and training parameters, and evaluates on the test set to report an accuracy at the end. (15 pts)
2. **Report** the detailed architecture of your best model. Include information on hyperparameters chosen for training and a plot showing both training and validation loss across iterations. (10 pts)
3. **Report** the accuracy of your best model on the test set. We expect you to achieve over **90%**. (10 pts)

Hints: Read PyTorch documentation for `torch.nn` at https://pytorch.org/docs/stable/nn.html and pick layers for your network. Some common choices are

- `nn.Linear`
- `nn.Conv2d`, try different number of filters (`out_channels`) and size of filters (`kernel_size`)
- `nn.ReLU`, which provides non-linearity between layers
- `nn.MaxPool2d` and `nn.AvgPool2d`, two kinds of pooling layer
- `nn.Dropout`, which helps reduce overfitting

Your network does not need to be complicated. We achieved over 90% test accuracy with two convolutonal layers, and it took less than 15 mins to train on a laptop with CPU only. You will get partial credits for any accuracy over **50%**, so do not worry too much and spend your time wisely.

2 Activation Visualization [35 pts]

In this part, we will investigate a method to visualize the activation map of an image through a classification network and show intuitively how each part of the image contribute to the prediction. We have provided some starter code in `part2.py`.

To observe a meaningful pattern, we construct a custom dataset that localizes the Fashion-MNIST image with the help of MNIST images. Each image in this new dataset will be a 2×2 grid of one Fashion-MNIST image and three MNIST images. The Fashion-MNIST image is placed at a random grid, where the other three grids will be randomly-chosen MNIST images. This part is implemented for you in class `GridDataset`.

To apply the visualization method, our network needs to contain a global average pooling (GAP) layer followed by a linear layer at the end. When visualizing, we replace GAP and linear layers with a 1×1 convolution layer using weights from the linear layer. Instead of C class scores, the network output is now

C 2D arrays corresponding to each of the C classes. If we plot them as heatmaps as shown in the figure below, we should see that at ground truth class, activation is higher at the position of the Fashion-MNIST image in the input image, implying that our model has learned to "look at" only the Fashion-MNIST images for classification.

Notes on dimensions:

1. A global average pooling layer reduces each $H \times W$ channel to a single value by simply taking the average of all HW values.
2. Suppose the input to GAP layer in the original network has shape (F, H, W), it will become $(F, 1, 1)$ after GAP layer, so the linear layer has weight of shape (F, C). In the adapted network, the 1×1 conv layer has F input channels and C output channels, and therefore has weight of shape $(F, C, 1, 1)$. Since linear layer and 1×1 conv layer have weights of the same size, we can transfer weights from the former to the latter with a simple reshaping.
3. In PyTorch, the first dimension of the data tensor is for batch and the second is for channels. So each batch of input to the network (the input to `forward(self,x)` in code) has size (N, C, H, W), where N is the batch size and C is the number of channels.

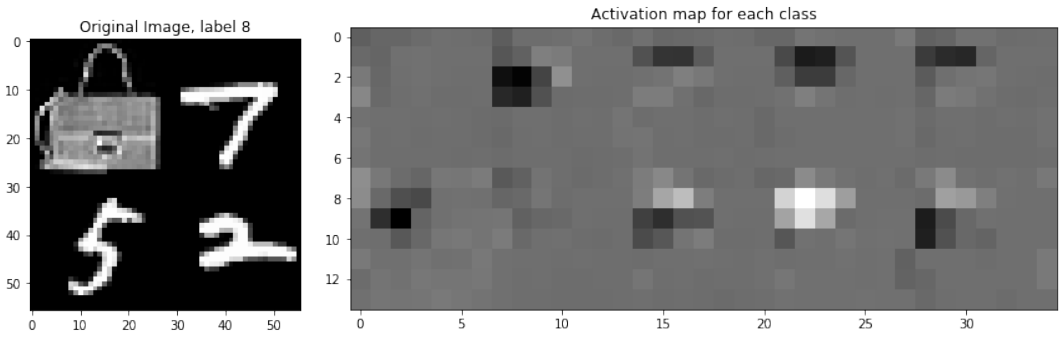


Figure 3: Example image from our custom dataset and its per-class activation maps

Complete the following:

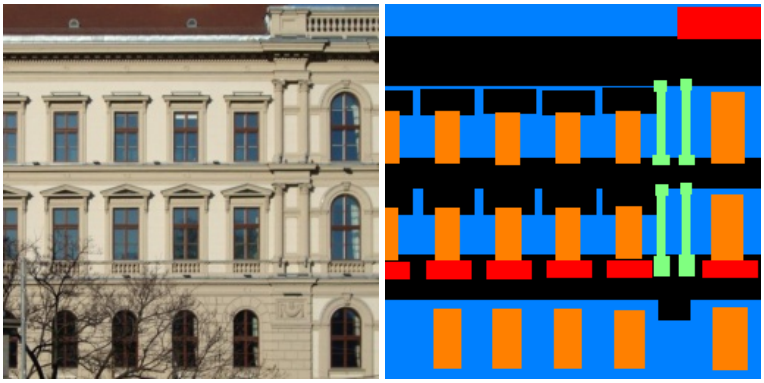
1. **Report** the detailed architecture of your `self.base` module. Include information on hyperparameters chosen for training, and the accuracy on the test set. To make the visualization look nice, you should achieve over 80% on the test set. (10 pts)
2. Choose a correctly classified image from the evaluation on test set. **Report** its index in the test set and include plots of both the image and the activation maps of all classes. (10 pts)
3. **Submit** a program which contains your best combination of `self.base` module, optimizer and training parameters, along with the code to select a correctly classified image and to visualize the results. (15 pts)

Hints: Apart from the last two layers, your network should look similar to the one in the previous part, and takes a similar amount of time to train. Again, any test accuracy over **50%** will receive partial credits.

3 Semantic Segmentation [30 pts]

Besides image classification, Convolutional Neural Networks can also generate dense predictions. A popular application is semantic segmentation. In this part, you will design and implement your Convolutional Neural Networks to perform semantic segmentation on the Mini Facade dataset.

Mini 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 provided some starter code in `part3/train.py` which contains a dummy network. It uses a 1x1 convolution to convert 3 channels (RGB) to 5 channels, i.e. 5 heatmaps for each class, with cross-entropy loss. Your network should give an output of the same shape, but contains more layers of different types.

We will evaluate your model on its average precision (AP) on the test set (higher the better). We have provided you the code to evaluate AP and you can directly use it. An introduction to AP is available at https://scikit-learn.org/stable/modules/model_evaluation.html#precision-recall-f-measure-metrics.

Complete the following:

1. **Report** the detailed architecture of your model. Include information on hyperparameters chosen for training and a plot showing both training and validation loss across iterations. (10 pts)

2. Report the average precision 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. (10 pts)

3. **Submit** a program which contains your best combination of `self.base` module, optimizer and training parameters. (5 pts)

4. Take a photo (or search for an image online) of a building in UM, preprocess it as you like and input it to your best trained model. Plot the output labels and comment qualitatively on why it works or doesn't work. Include both the original image and the label plot in your report. (5 pts)

Hints: You should still choose layers from classes under `torch.nn`. In addition to layers mentioned in the previous part, you might want to try `nn.Upsample` that upsamples the activation map by a simple interpolation, and `nn.ConvTranspose2d` that upsamples by performing transpose convolution. Any test AP over **0.25** will receive partial credits. Below are some architectures from papers that 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.

You will easily get a high test AP if you use one of these models, but they can take a long time to train (hours with a CPU). Consider doing this part with the help of a GPU on Google Colab.

Canvas Submission Checklist

In the `zip` file you submit to Canvas, the directory named after your uniqname should include the following files:

- Code files: `part1.py`, `part2.py` and `train.py` (from Part 3)

All plots and answer to questions should be included in your `pdf` report submitted to Gradescope.

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

The Mini 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.