ICE3056: Foundations of Modern Artificial Intelligence Final Project

Il Yong Chun
Associate Professor of EEE, AI, ECE, ADE, SCE, & DCE, SKKU
Associate Professor of CNIR, IBS

November 11, 2025

Instructions

- The submission is **due 11:59 PM on December 19 (Fri.), 2025** via i-Campus.
- A **single** 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.** LaTeXis 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/`.
- A zip file including all your codes, and files specified in questions with **Submit**, all under the same directory. You can submit Python code in either .py or .ipynb format.

1 Python Environment

We will use Python 3.7–3.9 for the final project. You can find references for the Python standard library here. To make your life easier, I recommend installing Anaconda 22.9.0 for Python 3.7.x–3.9.x. This is a Python package manager that includes most of the modules you need for this course.

You are expected to use the following packages extensively:

- `Numpy`
- `OpenCV`
- `PyTorch`
- `Matplotlib`

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

Additionally, I recommend setting up your environment with CUDA 11.8, PyTorch 2.1.0, and Python 3.9.x, which achieved the best performance.

2 Convolutional neural networks for semantic segmentation

Convolutional Neural Network (CNN) methods can generate dense predictions. A popular application is semantic segmentation.

You will design, implement, and train CNNs in **PyTorch** to perform on the Mini Facade dataset. Back-propagation is automatically inferred by PyTorch (`autograd`), so you only need to write code for the forward pass.

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

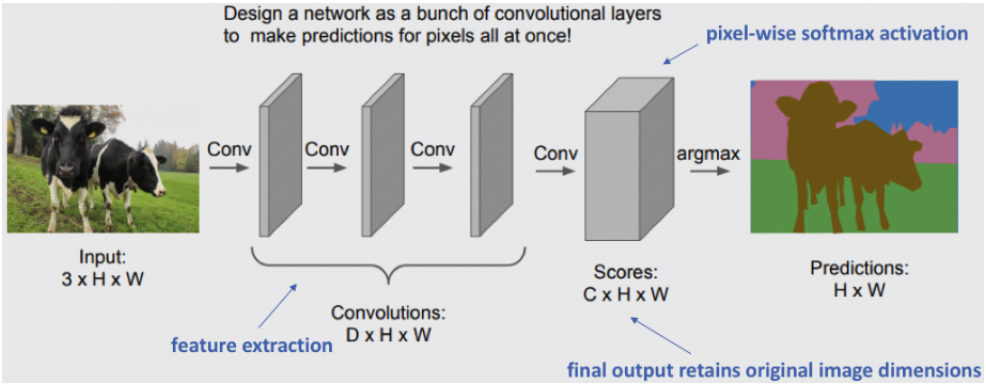


Figure 1: A CNN example for semantic segmentation. (Image is from this link.)

(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.

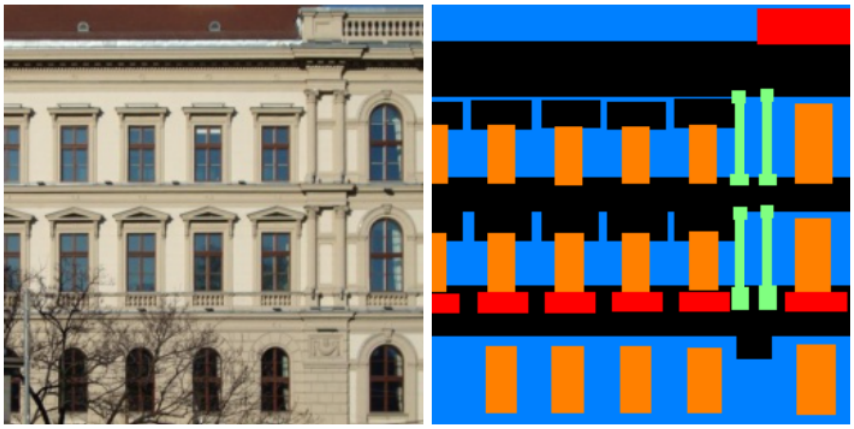


Figure 2: The label images are plotted using 8-bit indexed color with pixel value listed in Table 1. We have provided you a visualization function using a "jet" color map.

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

I have provided some starter code in `segmentation/train.py` which contains a dummy network. It uses a 1×1 convolution to convert 3 channels (RGB) to 5 channels, i.e. 5 heatmaps for each class, with cross-entropy loss. You need to modify and experiment with the following:

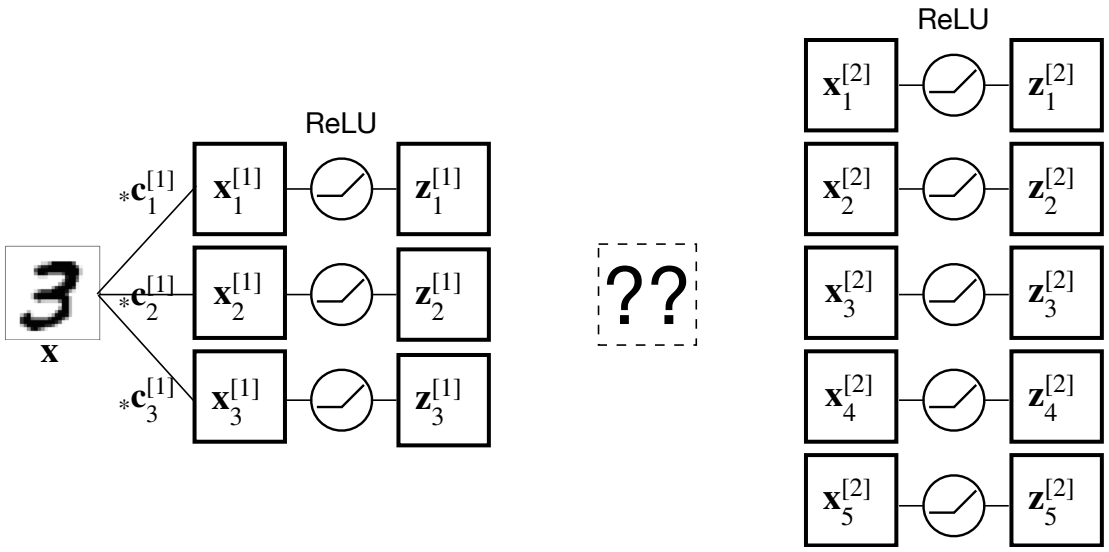
- The architecture of the network (define layers and implement forward pass).
- The optimizer (stochastic gradient descent, RMSProp, Adam (default), etc.) and its parameters. (`weight_decay` is the ℓ_2 regularization strength.)
- Training parameters (batch size, number of epochs, initial learning rate, learning rate decay scheme, etc).

(These are specified with "TODO" in `segmentation/train.py`.)

We will evaluate our model with average precision (AP) on the test set (higher the better). I've already provided you the code to evaluate AP and you can directly use it.

Complete the following:

(a) Suppose that you construct conventional two-layer CNNs with 9×9 kernels and 3 features in the first convolutional layer and 5 features in the second convolution layer. The first convolution layer is parameterized with the first layer convolutional kernels $\{\mathbf{c}_k^{[1]} \in \mathbb{R}^{81} : k = 1, \ldots, 3\}$; its operators can be specified as follows:



Fill the question-box above with the second layer convolution kernels $\{\mathbf{c}_?^{[2]}\}$, convolution operator $*$, and summation $\sum_?$ (ignore bias vectors). **Determine** the number of parameters for the *entire* network, based on your answer above (ignore bias vectors). (10 pts)

(b) Consider a semantic segmentation CNN in Figure 1. **Design** a convolutional network at the last layer. **Justify** your answer with a linear algebra perspective. (20 pts)

(c) **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. You may use five-fold cross validation, splitting the dataset in `starter_set/train` to training and validation sets containing 80% and 20% samples, respectively. (20 pts)

(d) **Submit** a program that contains your best combination of `self.base` module, optimizer and training parameters. (20 pts)

(e) **Report** the average precision on *both* validation set and 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. I expect you to achieve **0.5** AP on the test set. (20 pts)

(f) Take a photo of a building in SKKU, 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. **Submit** the image as `input.jpg` and the output labels as `output.png`. (10 pts)

Hints: Read the PyTorch documentation for `torch.nn` and pick layers for your network. Some common choices are

- `nn.Linear`
- `nn.Conv2d`. Try different number of features (`out_channels`) and size of filters (`kernel_size`). `nn.ConvTranspose2d` upsamples by performing transpose convolution.

- `nn.ReLU` provides non-linearity between layers.
- `nn.MaxPool2d` and `nn.AvgPool2d`, two types of pooling layer.
- `nn.Upsample` upsamples the activation map by a simple interpolation.
- `nn.Dropout` helps reduce overfitting.

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` and `nn.ConvTranspose2d`. Any test AP over 0.25 will receive partial credits. Below are some architectures from papers that you can try:

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