EE616 Computational Image Processing and Computer Vision: Final

Il Yong Chun
Department of Electrical Engineering, the University of Hawai'i, Mānoa

April 23, 2020

Instructions

- The submission is **due 11:59:59 PM on Sunday, May 10th, 2020** via Laulima.
- 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.

Python Environment

We are using Python 3.7 for the final project. You can find references for the Python standard library here. To make your life easier, I recommend installing Anaconda 5.2 for Python 3.7.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 Part 2 of final, 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.

As a deep learning library, PyTorch performs backpropagation automatically for you and trains your network faster. For this homework we have also provided some starter code with comments.

1 Fully-connected neural networks

Answer the following questions:

(a) **Derive** the result in (21) and that below (21) in EE616 lecture note §3. Show all details and logical reasonings. (10 pts) (**Note:** I don't expect you to copy and paste the answer in $\boxed{??}$ or related high-level derivations in the lecture notes, i.e., no score to such answers.)

(b) The matrix formulation summary in §3.4.4 contains all patterns as columns of a single matrix $\mathbf{X}$. This is ideal in terms of speed and economy of implementation; it is also well suited when training is done using mini-batches. However, there are applications in which the number of training pattern vectors is too large to hold in memory, and it becomes more practical to loop through each pattern using the vector formulation.
Compose a summary similar to the matrix formulation summary in §3.4.4, but using individual patterns $\mathbf{x}_k$, instead of matrix $\mathbf{X}$. (10 pts)

2 Convolutional neural networks for dental radiography classification

In Part 2, you will implement and train convolutional neural networks (CNNs) in **PyTorch** to classify dental radiographs. Unlike Part 1 above, back-propagation is automatically inferred by PyTorch (autograd), so you only need to write code for the forward pass.

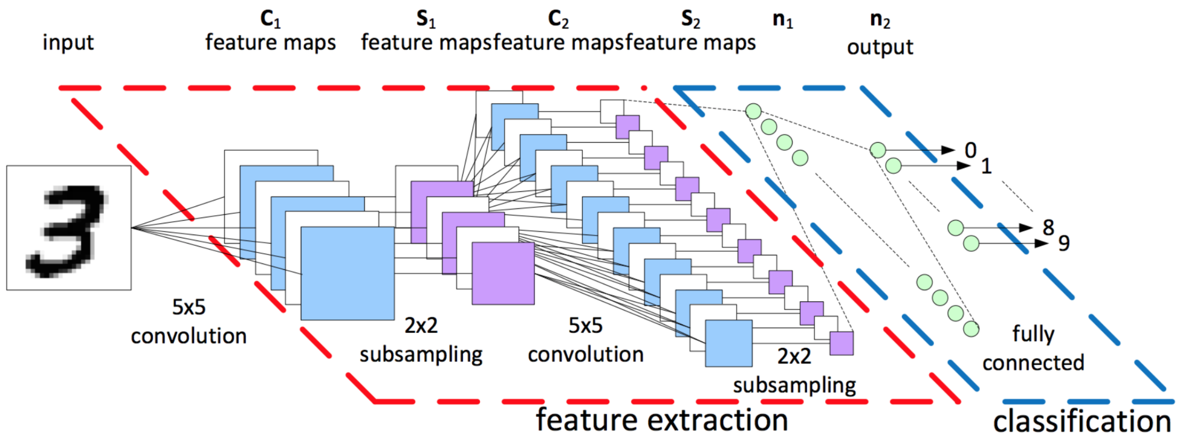


Figure 1: A CNN example for image classification. (Image is from this link.)

You will use the dental radiography dataset consisting of 120 748×512 dental images with two classes, "with dental diseases" and "without dental diseases," where the last 12 rows in contain the image information and must be removed in the preprocessing step.[1] There are three types of dental diseases in the dataset: *1)* enamel caries, *2)* dentinal caries, and *3)* pulpal caries. Then, you will split the dataset to training and testing sets containing the first 100 and the last 20 images, respectively. Because the provided dataset is small, you will apply *data augmentation*, e.g., rotating images by 90°, 180°, and 270° and increasing the number of images from 120 to 480.

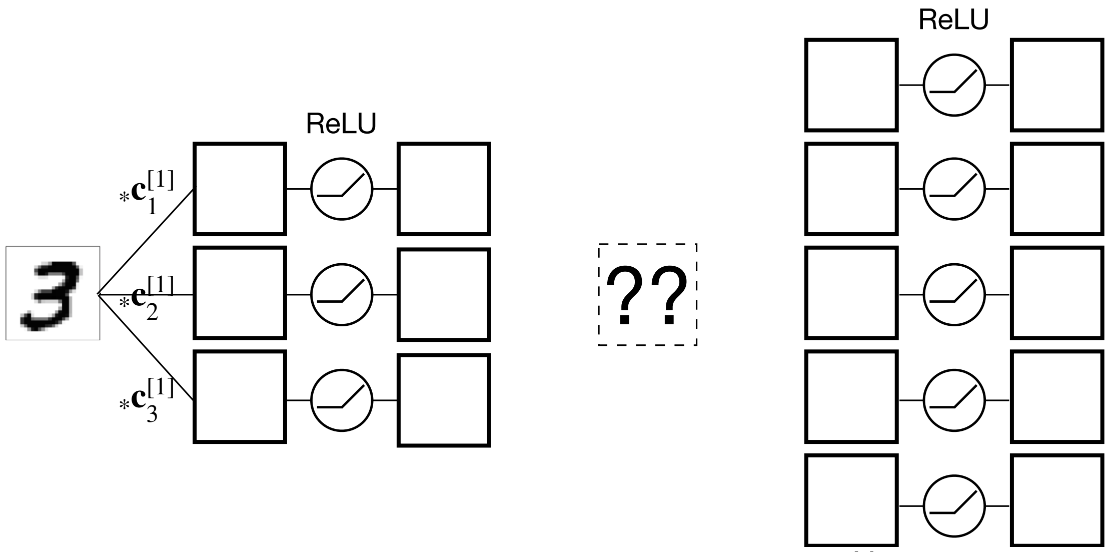
Using binary classification CNNs trained with training dataset provided by the procedure above, our aim is to determine if dental radiographies have dental diseases or not. I have provided some start code in eg.py where you need to modify and experiment with the following:

- The architecture of the network (define layers and implement forward pass).
- The training loss function. (*Default:* CrossEntropyLoss; see the Loss functions shortcut in this link)
- The optimizer (stochastic gradient descent, RMSProp, Adam, etc.) and its parameters. (weight_decay is the ℓ_2 regularization strength.)
- Training parameters (batch size, number of epochs, learning rate decay scheme, etc).

[1] Using it for any purpose other than the EE616's final project (at the University of Hawai'i–Mānoa) requires the instructor's permission. Consult with the instructor, Prof. Il Yong Chun (iychun@hawaii.edu), if you want to use it for any purpose other than EE616.

Complete the following:

(a) Suppose that you construct conventional two-layer CNNs with 5×5 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 $\{\mathbf{c}_k^{[1]} \in \mathbb{R}^{25} : k = 1, \ldots, 3\}$; 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 $\sum_?$. **Determine** the number of parameters based on your answer. (20 pts) (Note that this architecture will not apply to answering the questions below.)

(b) **Submit** a program which trains with your best combination of model architecture, training loss function, optimizer and training parameters, and evaluates on the test set to report an accuracy at the end. (20 pts)

(c) **Report** the detailed architecture of your best model. Include information on hyperparameters chosen for training and a plot showing both training and test loss across iterations. (20 pts) **Report** the accuracy of your best model on the test set. We expect you to achieve over 95%. (20 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.ReLU, which provides non-linearity between layers
- nn.MaxPool2d and nn.AvgPool2d, two types of pooling layer
- nn.Dropout, which helps reduce overfitting.

You will get partial credits for any accuracy over 85%, so do not worry too much and spend your time wisely.