CIS581: Computer Vision and Computational Photography
Project 4, Part B: Convolutional Neural Networks (CNNs)
Due: Dec.11, 2017 at 11:59 pm

Instructions

- CNNs is a **team** project. The maximum size of a team is **three** students. The same team will carry forward for the Part C as well. You are not permitted to do this individually. A team is not allowed to collaborate with another team. Only one individual from each team must submit the code for this part.
  Maximum late days allowed for this part of the project is the average of the late days of the two or three students in the team. The late days used will be subtracted from the individual tally of late days for each student.
- You must make **one submission** named `<Group_Number>_Project4B.zip` on Canvas. We recommend that you include a README.txt file in your submissions to help us execute your code correctly. The zip file should also contain a **report** with a .pdf extension and **3 folders** containing the Python code for each part. Note that you should include all the figures in your report.
- To help you implement the project efficiently, we provide you a documentation of the **PyNet** API, the package that will be used throughout the whole project. Please read it carefully prior to starting your implementation.
- This handout provides instructions for the code in Python. You are not permitted to use MATLAB for this project.
- Feel free to create your own functions as and when needed to modularize the code. For Python, add all functions in a ***helper.py*** file and import the file in all the required scripts.
- The *Third Problem*(Face Landmarks Detection) is **optional for extra credits**, but truly recommend to give a shot for that since it's related to the Part C.
- **Start early!** If you get stuck, please post your questions on Piazza or come to Office Hours!
- **Please follow the submission guidelines and the conventions strictly! The grading scripts will break if the guidelines aren't followed.**

1 Functional Layers Implementation

In this part, you should complete **four** functional layers in the file ***myLayers.py***, two non-linear activations (*Sigmoid* and *Relu*) and two loss functions (*l2_loss* and *Cross_entropy_loss*).
Note: Put all the helper functions (e.g. gradient computation) in the file ***p1_utils.py*** and other functions such as for 3D visualization methods should be in the file ***p1_main.py***.

As shown in Figure 1, you should implement a single neuron with an input array of size 1×1 and value 1. You should then use different activation and loss function combinations to achieve the following tasks:

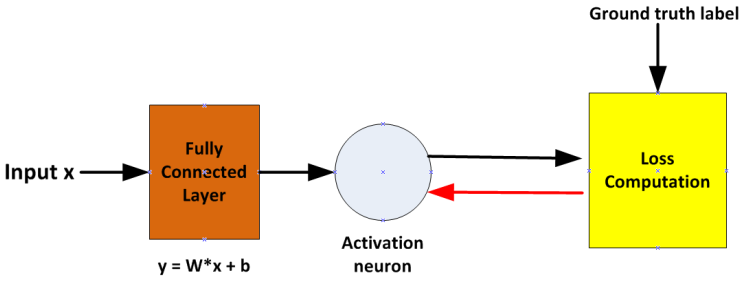


Figure 1: Network diagram for part 1

1. Complete the ***Sigmoid*** and ***Relu*** layers. Plot two 3D figures showing the relationship of the Sigmoid and Relu output with respect to the weight and bias. To be specific, x-axis is the weight (W), y-axis is the bias (b), and z-axis is the output of the function.
   Tips: Use the package *matplotlib* and the function *plot_surface* from *mpl_toolkits.mplot3d* to draw 3D figures.

2. Complete the ***l2_loss*** layer. Take the ground truth y equal to 0.5 (same type as the input x) and experiment with the forward and backward passes. Use the ***Sigmoid*** and ***Relu*** as activation neurons respectively.

   (a) Plot 3D figures showing the relationship of the l2 loss with the weight and bias. To be specific, x-axis is weight, y-axis is bias, and z-axis is the loss output.

   (b) Compute the backward gradient of the l2 loss with respect to the weight, $\frac{\partial l2\_loss}{\partial weight}$, and plot 3D figures showing the relationship of gradient with the weight and bias. To be specific, x-axis is the weight, y-axis is the bias, and z-axis is the gradient output.

3. Complete the ***Cross_entropy_loss*** layer. Take the ground truth y equal to 0.5 and experiment with the forward and backward passes. Use the ***Sigmoid*** and ***Relu*** as activation neurons respectively.

   (a) Plot 3D figures showing the relationship of the cross-entropy loss with the weight and bias. To be specific, x-axis is the weight, y-axis is the bias, and z-axis is the loss output.

   (b) Compute the backward gradient of Cross-entropy loss with respect to the weight, $\frac{\partial CE\_loss}{\partial weight}$, and plot 3D figures showing the relationship of the gradient with the weight and bias. To be specific, x-axis is the weight, y-axis is the bias, and z-axis is the gradient output.

4. Explain what you observed from the above plots. The explanations should include:

   (a) What are the differences between cross-entropy loss and L2 loss?

   (b) What are the differences between the gradients from cross-entropy loss and the l2 loss? Provide mathematical reasoning for the differences.

   (c) Predict how these differences will influence the efficiency of the learning and training process.

2 Simple Network Implementation

In this part, you will extend the theoretical analysis in the first part to perform quick experiments on a real dataset for the task of recognition. On the basis of the ***PyNet*** structure, complete the file ***p2_train.py***, put all help functions (e.g.data loader) in the file ***p2_utils.py***.
Note: if you have confidence in the layers you implemented in the part 1, feel free to make fully usage of them in the following questions.

2.1 Experiment with Fully Connected Network

Implement the network architecture in the Figure 2, by doing the experiment, you will experience how different activation and loss functions influence the learning (or convergence) of a simple neural network.

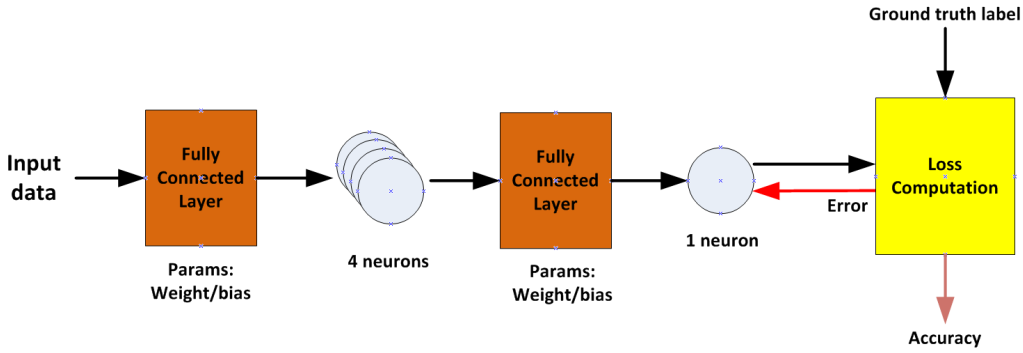


Figure 2: Network diagram for part 2.1

The input data is a random toy dataset containing 64 images, each of which is of resolution 4×4. The ground truth label is either 0 or 1. The data is stored as *'p21_random_imgs.npy'* and *'p21_random_labs.npy'*, the latter one represents the ground truth. For all training process, use simple *Gradient Descent* method with learning rate *0.1*, weight decay $5e^{-4}$ and momentum 0.99. Batch size in each training set is *64*.

Note: The nonlinear activation in the last neuron should use the *Sigmoid* layer, the max number of training epochs(iterations) is 1000.

1. Use ***Sigmoid*** function as neuron activation between two Fully Connected Layers, and ***l2_loss*** for the network. Plot two figures showing, (1) loss vs training iterations, (2) accuracy vs training iterations. Stop the training when accuracy reaches 100%.
2. Use ***Sigmoid*** function as neuron activation between two Fully Connected Layers, and ***Cross_entropy_loss*** for the network. Plot two figures showing, (1) loss vs training iterations, (2) accuracy vs training iterations. Stop the training when accuracy reaches 100%.
3. Use ***Relu*** function as neuron activation between two Fully Connected Layers, and ***l2_loss*** for the network. Plot two figures showing, (1) loss vs training iterations, (2) accuracy vs training iterations. Stop the training when accuracy reaches 100%.
4. Use ***Relu*** function as neuron activation between two Fully Connected Layers, and ***Cross_entropy_loss*** for the network. Plot two figures showing, (1) loss vs training iterations, (2) accuracy vs training iterations. Stop the training when accuracy reaches 100%.
5. Sort the settings (activation and loss function combination) according to the training iterations to reach 100% accuracy. Explain the reasons of different convergence rates.

2.2 Experiment with Convolutional Network

Implement the network architecture in the Figure 3, by doing the experiment, you will experience how the convolutional networks learn to interpret images and capture meaningful structure.

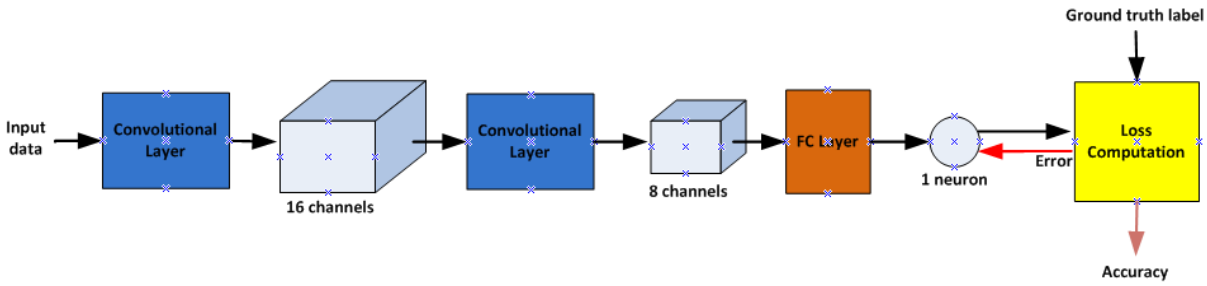


Figure 3: Network diagram for part 2.2

The Hyper-parameters of two Convolutional layers shown in the below table.

Layers	Hyper-parameters
Convolution 1	Kernel size = (7, 7), output channel = 16, stride = (1, 1), padding = 0, followed by ReLU.
Convolution 2	Kernel size = (7, 7), output channel = 8, stride = (1, 1), padding = 0, followed by ReLU.

The input data is a line dataset (samples in Figure 4) contains 64 images, each of which is of resolution 16×16. The ground truth label is either 0 or 1. The data for this part is stored as *'p22_line_imgs.npy'* and *'p22_line_labs.npy'*, the latter one represents the ground truth. For all training process, use simple *Gradient Descent* method with learning rate *0.01*, weight decay $5e^{-4}$ and momentum 0.99. Batch size in each training set *64*.

Note: The activation in the last neuron should use the *Sigmoid* layer, the loss function is the *Binary_cross_entropy_loss*, and the max number of training epochs(iterations) is 1000.

Plot two figures showing, (1) loss vs training iterations, (2) accuracy vs training iterations. Stop the training when accuracy reaches 100%.

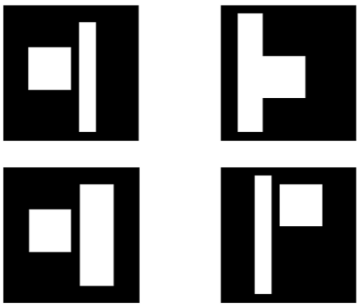


Figure 4: Four samples in the line dataset. The left column corresponds to the label 0, as the right column 1.

3 Face Landmarks Detection (Extra Credits)

In this section, you play around with a deep neural network to experiment with various images of faces. After training, the network should be able to detect a total of five landmarks in the face, namely **left eye**, **right eye**, **nose**, **left** and **right side of mouth**.
Complete the file ***'p3_train.py'*** and ***'p3_dataloader.py'***. Also include all related helper functions in the file ***p3_utils.py***.

Construct the network following the below architecture (Figure 5).

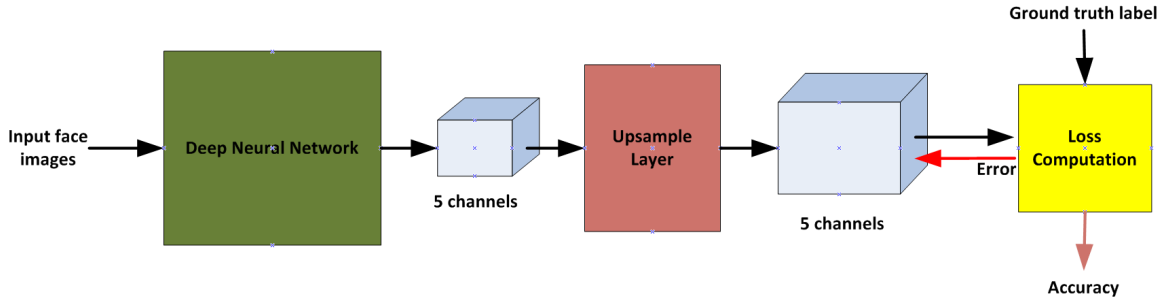


Figure 5: Network diagram for part 3

The below table shows detailed hyper-parameters of the layers in deep neural network.

Layers	Hyper-parameters
Convolution 1	Kernel size = (5, 5, 64), stride = (1, 1), padding = 2, followed by Batch normalization and ReLU.
Convolution 2	Kernel size = (5, 5, 64), stride = (1, 1), padding = 2, followed by Batch normalization and ReLU.
Pooling 1	Max operation. Kernel size = (2, 2), stride = (2, 2), padding = 0.
Convolution 3	Kernel size = (5, 5, 128), stride = (1, 1), padding = 2, followed by Batch normalization and ReLU.
Convolution 4	Kernel size = (5, 5, 128), stride = (1, 1), padding = 2, followed by Batch normalization and ReLU.
Pooling 2	Max operation. Kernel size = (2, 2), stride = (2, 2), padding = 0.
Convolution 4	Kernel size = (3, 3, 384), stride = (1, 1), padding = 1, followed by Batch normalization and ReLU.
Convolution 5	Kernel size = (3, 3, 384), stride = (1, 1), padding = 1, followed by Batch normalization and ReLU.
Convolution 6	Kernel size = (3, 3, 5), stride = (1, 1), padding = 1, followed by Batch normalization and Sigmoid.

Note: Please follow below tips to help your implementations.

- Download data via this link: http://mmlab.ie.cuhk.edu.hk/projects/TCDCN/data/MTFL.zip
- Choose the *Binary_cross_entropy_loss* for the loss computation.
- Use the *Stochastic Gradient Descent(SGD)* optimizer for training, with learning rate $1e^{-4}$, weight decay $5e^{-4}$ and momentum 0.99.
- Set the batch size of each step is **1**, around **45,000** steps in each epoch should be reasonable to get good results.
- Remember to *normalize* the raw input image via the following operations.

$$Channel_{blue} = (Channel_{blue} - 119.78)/255$$

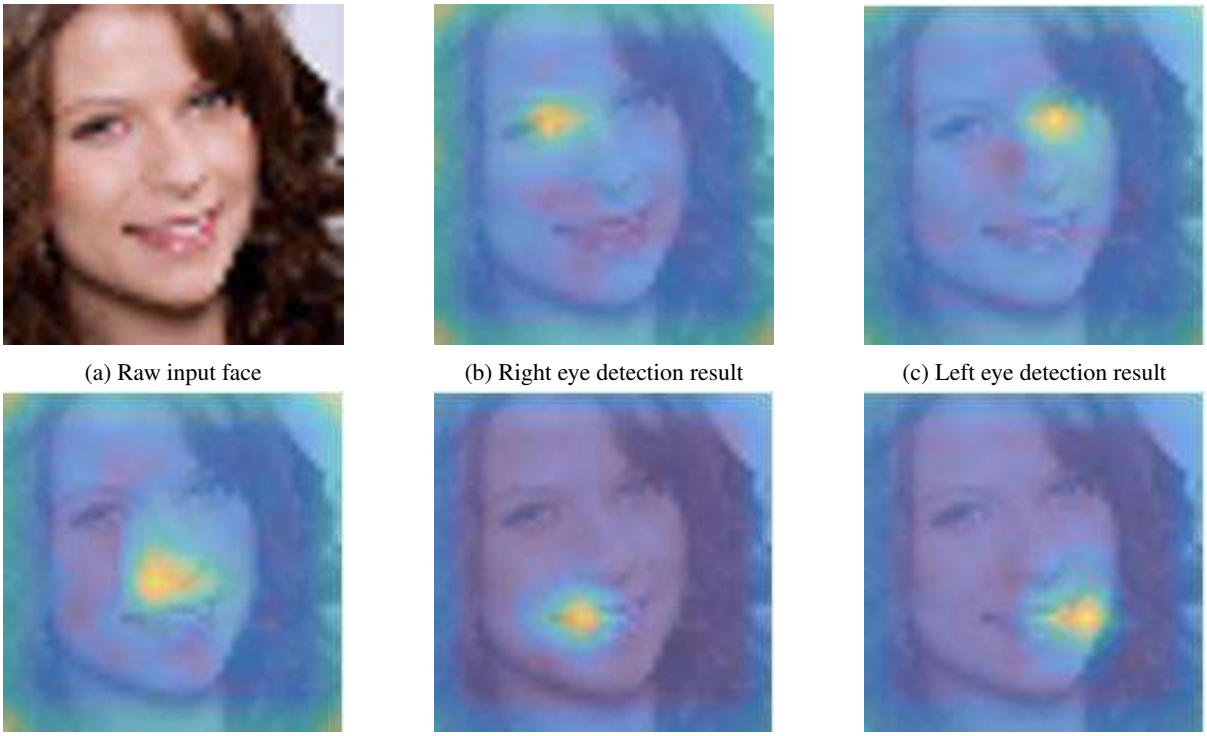
$$Channel_{green} = (Channel_{green} - 99.50)/255$$

$$Channel_{red} = (Channel_{red} - 89.93)/255$$

- Use the function ***get_gt_map()*** to pre-process ground truth label. In addition, since images have different size, use the function ***upsample2d()*** to make the input image and converted ground truth label into the same size (40×40 is recommended) for training.
- Train for 4 or 5 epochs should be enough.

Utilize the above instructions and complete total four tasks in this part.

1. Plot the figure showing loss vs training iterations.
2. Plot the figure showing accuracy vs training iterations.
3. Show at least two sets of feature maps after *Upsample* layer showing the prediction results of your network (See a sample plot in Figure 6).
4. Report the average test loss and accuracy based on your trained network.



(a) Raw input face (b) Right eye detection result (c) Left eye detection result

(d) Nose detection result (e) Right side month detection result (f) Left side month detection result

Figure 6: Sample face landmarks detection result (The bright mark represents the detection result)