http://www.cs.ucf.edu/~bagci

[Programming Assignment] (2)

Computer Vision

Dr. Ulas Bagci • (Fall) 2017 • University of Central Florida (UCF)

Coding Standard and General Requirements

Code for all programming assignments should be **well documented**. A working program with no comments will receive **only partial credit**. Documentation entails writing a description of each function/method, class/structure, as well as comments throughout the code to explain the program flow. Programming language for the assignment is **Python**. You can use standard python built-in IDLE, or CANOPY for the working environment. Other commonly used IDLEs are the following: PyCharm Community Edition, PyScripter, CodeSculptor, Eric Python, Eclipse plus PyDev.

For deep learning experiment with TensorFlow, you may want to make sure that you install Python 3.5. Submit by **26th of October 2017**, 11.59pm.

Optical Flow [5 pts]

[2 pts] Implement Lucas-Kanade optical flow estimation, and test it for the two-frame data sets provided in the webcourses: basketball and grove.

[3 pts] Implement Lucas-Kanade optical flow estimation algorithm in a multi-resolution Gaussian pyramid framework. After experimentally optimize number of levels for Gaussian pyramid, local window size, and Gaussian width, use the same data sets (basketball, grove, and teddy) to find optical flows, visually compare your results with the previous step where you don't use Gaussian pyramid.

Evaluation: Please create some random colors to show velocity vectors on the original images, and save them as outcomes of your program. Do the same step for each data set. Visit `http://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_video/py_lucas_kanade/py_lucas_kanade.html` for a sample implementation of Lucas-Kanade using Python and OpenCV. Other Python sources are available too. You are allowed to use built-in functions such as Gaussian smoothing, convolution, gradient operations, corner detections, etc., but not the Lucas-Kanade implementation itself. For features: you are suggested to use OpenCV's **goodFeaturesToTrack**, or your own implementation of corner detection.

Convolutional Neural Network (CNN) for Classification [5 pts]

Implement ConvNET using **TensorFlow** for digit classification. Sample code files (two files) are given in the attachment. Fill the parts indicated clearly in the code. Output should be saved as **output.txt**. When you are asked to include convolutional layer, do not forget to include max pooling or average pooling layer(s) as well.

- STEP 1: Create a fully connected (FC) hidden layer (with 100 neurons) with sigmoid activation function. Train it with SGD with a learning rate of 0.1 (a total of 60 epoch), a mini-batch size of 10, and no regularization.

- STEP 2: Now insert two convolutional layers to the network built in STEP 1 (and put pooling layer too for each convolutional layer). Pool over 2x2 regions, 40 kernels, stride =1, with local receptive field of 5x5.
- STEP 3: For the network depicted in STEP 2, replace Sigmoid with ReLU, and train the model with new learning rate (=0.03). Re-train the system with this setting.
- STEP 4: Add another fully connected (FC) layer now (with 100 neurons) to the network built in STEP 3. Use L2 regularization on the parameters of the two FC layers. (remember that the first FC was put in STEP 1, here you are putting just another FC).
- STEP 5: Change the neurons numbers in FC layers into 1000. For regularization, use Dropout (with a rate of 0.5). Train the whole system using 40 epochs.

Please read TensorFlow tutorial from its own webpage and get familiar with it first. Make sure that you organize your code in folders properly. (Code and MNIST folders should be in the same level, the digit data will be under mnist/data).

The traces from running testCNN.py <mode> for each of the 5 steps should be saved in output.txt, as indicated above. Note that In each step you will train the corresponding architecture and report the accuracy on the test data. Each step is 1 point.