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

[Programming Assignment] (2)

Robot Vision

Dr. Ulas Bagci • (Spring) 2019 • 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 other IDLEs such as CANOPY, PyCharm Community Edition, PyScripter, CodeSculptor, Eric Python, Eclipse plus PyDev, etc.

Following libraries (or others if you can find in addition to those listed below) can be used throughout the course:

- PIL (The Python Imaging Library), Matplotlib, NumPy, SciPy, LibSVM, OpenCV, VLFeat, python-graph.

If you are asked to implement "Gaussian Filtering", you are not allowed to use a Gaussian function from a known library, you need to implement it from scratch.

Along with your well commented code, please submit a "brief" report as doc or pdf format.

Submit by **25th of March 2019**, 11.59pm.

Optical Flow [5 pts]

[2 pts] Implement Lucas-Kanade optical flow estimation, and test it for the two-frame data set provided in the webcourses: basketball. Also note that you are provided a sample code here: `https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_video/py_lucas_kanade/py_lucas_kanade.html`, you can use the similar format but you need to change the code to adapt into your case.

[3 pts] Implement Lucas-Kanade optical flow estimation algorithm in a multi-resolution Gaussian pyramid framework. You have to use your code that you developed above, and then you have to experimentally optimize number of levels for Gaussian pyramid, local window size, and Gaussian width, use the same data set (basketball) 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. If you have difficulty showing optical flows with velocity vectors, you can use some built-in functions (color) to illustrate optical flows. 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 [10 pts]

Implement ConvNET using **PyTorch** for digit classification. Sample code files (two files: CNN.py and main.py) 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. Note that while main.py is for training, CNN.py includes models.

- 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. Note that there will be an output layer after hidden layer to map hidden neurons to number of classes.
- 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 PyTorch tutorial from its own webpages (in addition to the TA's tutorial) and get familiar with PyTorch 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 main.py –mode <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 2 point.