HOMEWORK ASSIGNMENT # 5
DUE: Tuesday, November 13, 2018, 5PM

CSCI677: Computer Vision, Prof. Nevatia

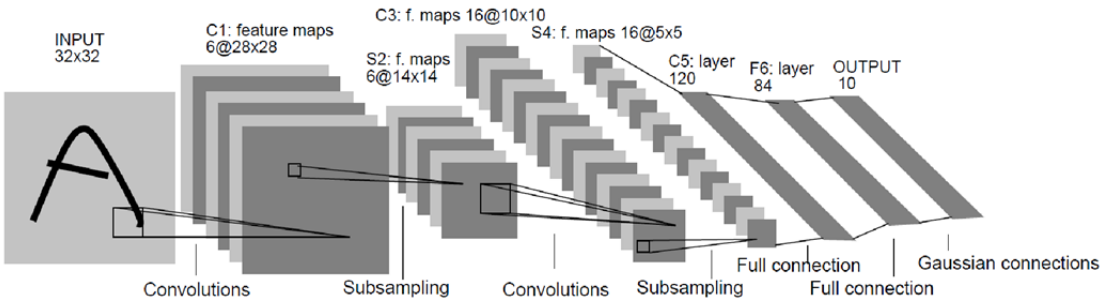
Fall Semester, 2018

This is a programming assignment to create, train and test a CNN for the task of object classification. To keep the task manageable, we will use a small dataset and small network.

a) **Dataset:**
We will use the CIFAR-100 dataset. It consists of 100 mutually exclusive classes with 50,000 training images and 10000 test images, evenly distributed across the 100 classes. Each image is a 32x32 RGB image. You can download the Python version of the dataset from https://www.cs.toronto.edu/~kriz/cifar.html. It contains four files. Please use 40000 images from ***train*** file for training, the remaining 10000 images for validation, and ***test*** file for testing. You can use pickle package to load the files. Each of the files contain a dictionary with several elements, below are the two elements we are going to use in this homework:

data -- a numpy array of uint8s. Each row of the array stores a 32x32 colour image
fine_labels -- a list of numbers in the range 0-99. The number at index *i* indicates the label of the *i*th image in the array **data**.

b) **Architecture**:
Construct a LeNet-5 style CNN network, using Tensorflow functions. LeNet-5 is shown graphically below. Additional details of LeNet-5 can be found in http://yann.lecun.com/exdb/publis/pdf/lecun-01a.pdf.



Our goal is to not exactly duplicate the architecture of the original LeNet-5 architecture. You may use the following hyper-parameters as a guide to start with.

First layer has six, 5x5 convolution filters, stride =1, each followed by a max pooling layer of 2x2 with stride = 2. Second convolution layer has 16, 5x5 convolution filters,

stride =1, each followed by 2x2 max pooling with stride = 2.

Next comes a fully connected layer of dimensions 120 followed by another fully connected layer of dimension 84.

A softmax layer of dimension 10 provides the classification probabilities (note that this is labeled "Gaussian connections" in the diagram but you can ignore this distinction).

All activation units should be ReLU.

c) **Training**:
Train the network using the given training data. We suggest using a mini-batch size of 64, learning rate of 0.001 and the ADAM optimizer, though you are free to experiment with other values. Note that you need not write your own backpropagation algorithm, it is implemented in Tensorflow. However, you will need to write your own sampling program for constructing mini-batches; Python provides a number of "Random" functions that can generate a number in [0, 1] or return a sequence of desired size.

Record the error after each step so you can monitor it and plot it to show results.
During training, you should test on the validation set at some regular intervals; say every 0.5 epochs, to check whether the model is overfitting.

d) **Preprocessing**:
You should subtract mean of the images in the test set for pre-processing. An easy way to do this is to compute a mean image (per channel) and subtract from each train and test image. This replaces earlier guidance to use "per_image_standardization" which is more complex.

Initialize network parameters by using the Xavier function.

e) **Augmentation**:
You should augment the training data. A possible way to augment data is: enlarge the image by 10%, crop the 90% of the image in left-top right-top, left-bottom, right-bottom and central, and then flip the image and do this again. You may use "resize" functions in OpenCV or in TensorFlow to enlarge the image.

f) **Test Results**:
Test the trained network on the test data to obtain classification and show the results in the form of confusion matrix and classification accuracy for each class and superclass (you can find the superclass information in the CIFAR100 website mentioned above). Additionally, also show accuracy at different ranks for both superclasses and fine-labeled classes. (say rank-1 and rank-5). You may use tf.reduce_mean, tf, tf.argmax and tf.equal for accuracy computation. You can find the usage of those functions in TensorFlow document here: https://www.tensorflow.org/api_docs/python/tf

g) **System Variations**:

Experiment with variations of the network with the aim of improving performance or ease of training the network without losing accuracy. Some suggestions are provided below but these are not requirements nor guaranteed to improve performance. Instead, you should be guided by the results of variations you obtain and also by study of other methods that will have been discussed in class. A suggested list follows.

i) Change the filter size (make them smaller) followed possibly by additional layers.

ii) Change the number of filters in the early layers.

iii) Change the size of the fully connected layers.

iv) Use batch normalization.

v) For inference, take multiple crops of expanded test image and average the scores.

As the variations can be combined, the number of options to try can be very large, it is not expected that you would try all combinations but, instead, choose a small number of combinations that you expect will result in improved performance.

h) **You may be able to find pre-trained models for LeNet and CIFAR-100 online. You may not use these and should train the network entirely from your own program.**

SUBMISSION:

You should turn in the following in softcopy:

1. A brief description of the programs you write, including the source listing.

2. Show evolution of loss function with multiple steps.

3. A summary and discussion of the results, including effects of parameter choices. Include visualization of results; show some examples of successful and some failure examples.