# DEEP LEARNING FOR VISION(CS5370) REPORT

**SAAHIL SIROWA**
**CS16BTECH11030**

Note: In the notebook file solution_123.ipynb go to Kernel-> Restart and Run All to verify the submitted solutions for questions 1, 2, 3. The notebook will take around 5 minutes to finish execution.

1) Code submitted in notebook(ipynb) file named solution_123.ipynb. Appropriate markdown cells are made in the notebook indicating which part is answered where. In output the coordinate corresponding to highest intensity(whitest spot) is printed. The function named correlate performs the operation

    a) Normalized cross-correlation = (Patch **.** Template)/sqrt(||Patch||$^2$*||Template||$^2$)
    By cauchy schwarz inequality
    (Patch.Template) <= sqrt(||Patch||$^2$*||Template||$^2$)
    The equality is achieved when,
    Patch = Some_Constant*Template
    In other words when template matches the patch

        Template Image: temp_s.jpg
        Base Image: image_s.jpg
        Output Image: output/1a.jpg

    b) It can be observed that template has white diagonal lines(high intensity). On cross correlation of the template with any patch we will get high responses where there are lines and will sharpen the diagonal and horizontal edges. It can be observed in the output that response corresponding to diagonal edges is higher than horizontal edges.
    Template Image: trailer.png
    Base Image: u2cuba.jpg
    Output Image: output/1b.jpg

    c) It can be observed that there is a scaling mismatch of template corresponding to the image. The responses are less sharp as compared to what we got in part (b). Therefore, we conclude that template must match the occurrence of the object in the corresponding image in terms of rotation as well as scaling.
    Template Image: trailerSlightlyBigger.png
    Base Image: u2cuba.jpg

Output Image: output/1c.jpg

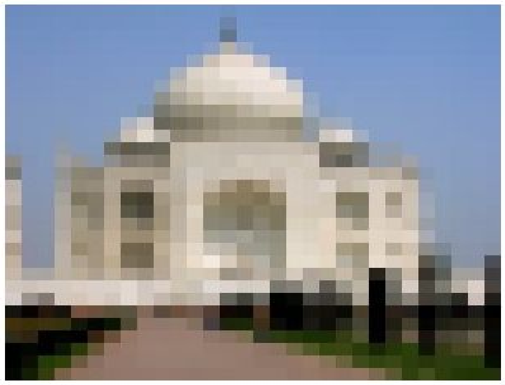
d) Assume that we want the size of output to be the same as that of input image. To get a single pixel value of output image requires $m^2$ operations. So, one template will take $O(n^2m^{2)})$ operations. Therefore, the answer is $O(N_sN_Rm^2n^2)$.

2) Code submitted in notebook(ipynb) file named solution_123.ipynb. Appropriate markdown cells are made in the notebook indicating which part is answered where. The function named convolve(image, template) is the answer to this question.

3) Code submitted in notebook(ipynb) file named solution_123.ipynb. Appropriate markdown cells are made in the notebook indicating which part is answered where.
   a) Laplacian Filter is more sensitive to noise than Sobel Filter. Reason being Laplacian is 2nd order filter while Sobel is first order filter.
      Base Image: clown.tif
      Output Image: output/3a_Laplacian.png, output/3a_Sobel.png
   b) There is not much difference observed between our implementation and library provided option. The only difference is the speed of operation. The library provided scipy.signal.convolve2d is faster when considered to our implementation. The possible reason being scipy uses FFT based algorithm to do convolution with computation complexity O(m*n*log(m*n)) which is far better than our implementation $O(m^2n^2)$.
      Base Image: clown.tif
      Output Image: output/3b_Laplacian.png, output/3b_Sobel.png

4) Code is submitted in the file named 4.py. If you look at the icon/thumbnail of the file 'output/4.png' in the file manager it looks like a dog. If you open the image it looks like a cat.
   Base Image: Data for Hybrid Images/dog.bmp, Data for Hybrid Images/cat.bmp
   Output Image: output/4.png

5) Following filters in GIMP were studied:
   i) Pixelize: The Pixelize filter renders the image using large color blocks. It is very similar to the effect seen on television when obscuring a criminal during trial.



It can be done by having a square mask, where all elements are zero other than the middle element of the mask. Let A be the matrix representing mask(0 indexed). If the side of the square is odd then A[floor(side/2)][floor(side/2)] = 1 and all other elements are zero. If it is even then the value at indices (side/2-1, side/2-1), (side/2, side/2), (side/2-1, side/2), (side/2, side/2-1) are equal to 0.25. All other values are zero. We will have parameter '**stride**' to determine how much pixelated image we want. Simple cross-correlation(averaging) operation will be done with this mask and a patch of the image. Stride will decide how much gap we want between the top-left end of two consecutive patches of original image vertically and horizontally. For example, consider an image of n*n and and template of k*k. Let the top-left coordinate of last image patch is (i, j). Then, the top-left coordinate of next image patch horizontally will be (i, j+stride) and vertically will be (i+stride, j).

Another possible way to achieve the effect is by using a non-linear filtering. In this approach we take take a mask of k*k and a stride  of k both vertically and horizontally. All the pixels of a patch(k*k) of image are overwritten to a single value(it may be median, mean, mode). This will give us the feel of square boxes in the image without any reduction in image size. This operation can be done inplace. For best results k can be chosen such that k is a divisor of n, where the size of the image is n*n. Basically the image is decomposed into ($n^2/k^2$) number of contiguous squares. All the values in the square are made the same to give the image a blocky look.

The first option mentioned reduces the size of the image, while the second keeps the size same.