**Assignment 2 (Image Registration)**
**Worth 20% of total weight**

This assignment is on registering two images with homography (projective transformation) model. The assignment makes you familiar with:

(1) Matrix manifold-based computation for homography
(2) Multiresolution image registration algorithm
(3) Mutual information and differentiable mutual information
(4) Writing your own backpropagation rule in PyTorch
(5) Calculus of variations

Study the following algorithm and its implementation (SSD_Homography_Registration.ipynb). In particular pay attention to: (a) Matrix exponentiation to create a homography matrix, (b) Use of meshgrid and grid_sample functions, and (c) creation of multi-resolution, anti-aliased image pyramid.

Let us first define symbols. Consider a fixed (template) image *T* and a moving image *M*. Also consider a vector *v* of length 8 denoting parameters of the Homography matrix.

For projective transformation/Homography there are 8 basis matrices. Call them $B_1, ..., B_8$. By the notation *exp*( $\sum_{i=1}^{8} v_i B_i$ ) we denote matrix exponentiation operation that projects *v* to the homography matrix manifold. *exp*( $\sum_{i=1}^{8} v_i B_i$ ) is the homography matrix here. Computing the exponential of a matrix can be done by one of the following formulas:

$exp(B) = \lim_{n \to \infty} (I + \frac{1}{n} B)^n$, *I* is the identity matrix, or, $exp(B) = \sum_{n=0}^{\infty} \frac{B^n}{n!}$.
In the notebook (SSD_Homography_Registration.ipynb), `MatrixExp` function implements it. To know more about exponential maps, look at [1].

Warping the moving image *M* by homography matrix *exp*( $\sum_{i=1}^{8} v_i B_i$ ) is denoted by Warp(*M*, *exp*( $\sum_{i=1}^{8} v_i B_i$ ), *X*, *Y*) , where *X* and *Y* need to be generated by a meshgrid function (e.g., `torch.meshgrid`). Note that our notebook uses `grid_sample` to implement Warp function.

The image registration cost SSD (sum of squared difference) between *T* and warped moving image is denoted by SSD(*T*, Warp(*M*, *exp*( $\sum_{i=1}^{8} v_i B_i$ ),*X*,*Y*)). PyTorch function `mse_loss` implements SSD cost.

The multiresolution matrix manifold registration algorithm using SSD cost is as follows.

1. Create anti-aliased multiresolution pyramid $\{T_l, M_l\}_{l=1}^{L}$ for both *T* and *M*. *L* is the coarsest level of resolution.
2. Initialize parameter vector *v* to the *0* vector.
3. For *l* = *L* down to 1
    4. [*Y*, *X*] = meshgrid ([0, $h_l$-1], [0, $w_l$-1]), where ($h_l$, $w_l$) are the height and width of $T_l$.
    5. Normalize coordinates in the range [-1,1]: *Y* = 2**Y*/$(h_l - 1)$ - 1 and *X* = 2**X* /($w_l$-1) - 1
    6. For iteration 1, 2, …,*N*
        7. Compute gradient *g* of SSD($T_l$, Warp($M_l$, *exp*( $\sum_{i=1}^{8} v_i B_i$ ), *X*, *Y*)) with respect to *v*.
        8. Update parameter vector by gradient descent: $v = v - \gamma g$, where $\gamma$ is the learning rate.
9. Compute homography matrix *H* = *exp*( $\sum_{i=1}^{8} v_i B_i$ ).

This algorithm has been implemented in the notebook: SSD_Homography_Registration.ipynb. Using the sample image pair "fixed.bmp" and "moving.bmp" test it.

**Question 1 (60%)**. Modify our implementation in SSD_Homography_Registration.ipynb to use differentiable mutual information [2]. Algorithm is outlined for you here.

Let us use the notation MINE(I, J) to denote a neural network parameterized by $\theta$. It takes in two grayscale values and outputs a lower bound for mutual information as defined in [2]. The following algorithm uses MINE in mult-resolution image registration.

1. Create anti-aliased multiresolution pyramid $\{T_l, M_l\}_{l=1}^{L}$ for both *T* and *M*. *L* is the coarsest level of resolution.
2. Initialize parameter vector *v* to the *0* vector.
3. For *l* = *L* down to 1
    4. [*Y*, *X*] = meshgrid ([0, $h_l$-1], [0, $w_l$-1]), where ($h_l$, $w_l$) are the height and width of $T_l$.
    5. Normalize coordinates in the range [-1,1]: *Y* = 2**Y*/$(h_l - 1)$ - 1 and *X* = 2**X* /($w_l$-1) - 1
    6. For iteration 1, 2, …,*N*
        7. Compute gradients $g_1$, $g_2$ of MINE($T_l$, Warp($M_l$, *exp*( $\sum_{i=1}^{8} v_i B_i$ ), *X*, *Y*)) with respect to *v* and MINE parameter $\theta$, respectively.
        8. Update parameters: $v = v + \gamma_1 g_1$ and $\theta = \theta + \gamma_2 g_2$ where $\gamma_1, \gamma_2$ are the learning rates.
9. Compute transformation matrix *H* = *exp*( $\sum_{i=1}^{8} v_i B_i$ ).

**Question 2 (10%).** Write a matrix exponentiation function that uses the following derivative (backpropagation) rule:

$\frac{d}{dt}exp(tX) = Xexp(tX) = exp(tX)X$ (see equation (1) here).

This requires extension of a function in PyTorch (see how “`LinearFunction`”
Is extended here).

**Question 3 (5%).** Modify MINE network to accommodate color images, so that you can register color images using differentiable mutual information.

**Question 4 (25%).** Let us denote by *J*(*f*) a functional (a functional is a function of function(s)):

$$J(f) = \int f(x,z)p_{XZ}(x,z)dxdz - log(\int exp(f(x,z))p_X(x)p_Z(z)dxdz),$$

where $p_{XZ}$ , $p_X$ and $p_Z$ are joint and marginal densities respectively.

Using calculus of variations [3], show that when *J*(*f*) is maximized with respect to function *f*, we get the following condition:

$$p_{XZ}(x,z) = \frac{exp(f(x,z))p_X(x)p_Z(z)}{\int exp(f(x,z))p_X(x)p_Z(z)dxdz}$$

Now using this condition show that *J*(*f*) reaches its upper bound, mutual information MI defined as:

$$MI = \int log\frac{p_{XZ}(x,z)}{p_X(x)p_Z(z)}p_{XZ}(x,z)dxdz$$

[Hint: Consider using a perturbation function *g*(*x*,*z*), then take the limit $\lim_{\varepsilon\to 0}\frac{J(f(x,z)+\varepsilon g(x,z))-J(f(x,z))}{\varepsilon}$ ]

## References


[1] Lie groups for computer vision, http://ethaneade.com/lie_groups.pdf

[2] MINE, https://arxiv.org/abs/1801.04062

[3] Variational calculus, https://en.wikipedia.org/wiki/Calculus_of_variations