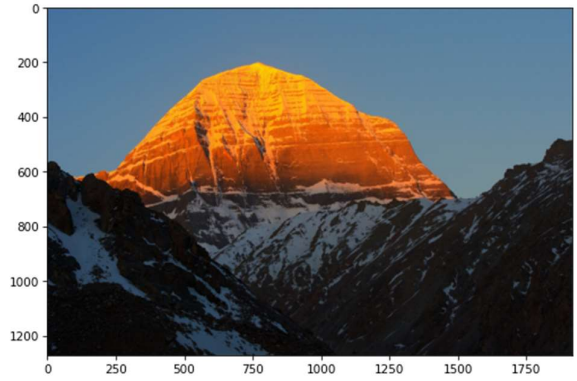
CS512 Assignment 1 – Report

Name: Sai Manohar Vemuri
CWID: A20514848

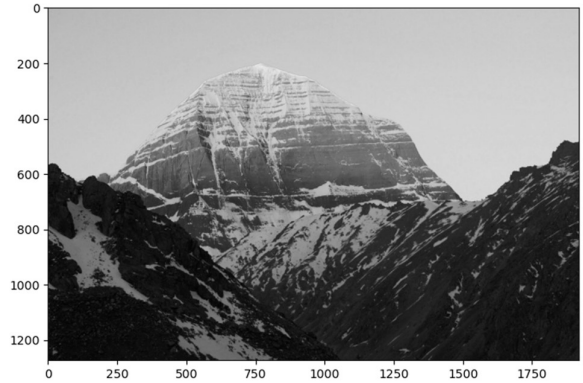
Open the Assignment2.ipynb in the jupyter notebook.

Programming Questions:

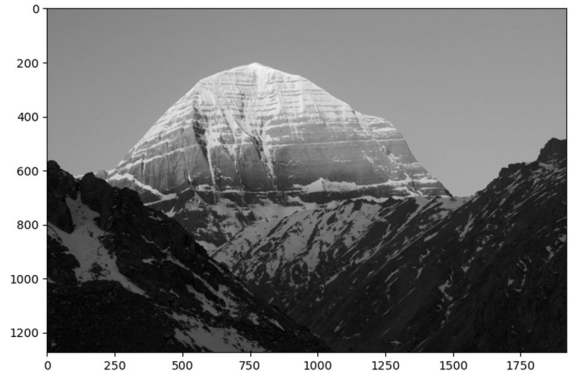
1. Read the image 'img.jpg' using the OpenCV library.
   You need to convert the image from BGR to RGB using the predefined method in OpenCV.
   Output:



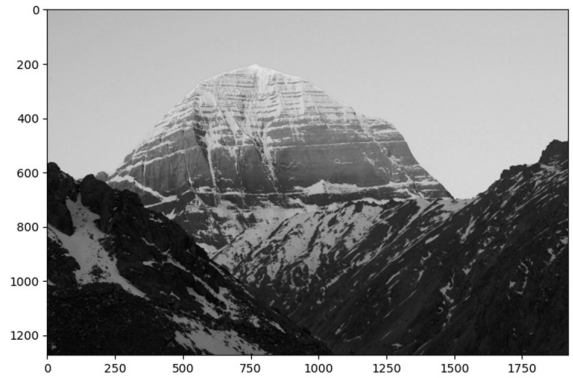
2. Convert the image to Grayscale using OpenCV function
   Output:
   Execution Time in Seconds: 0.0118



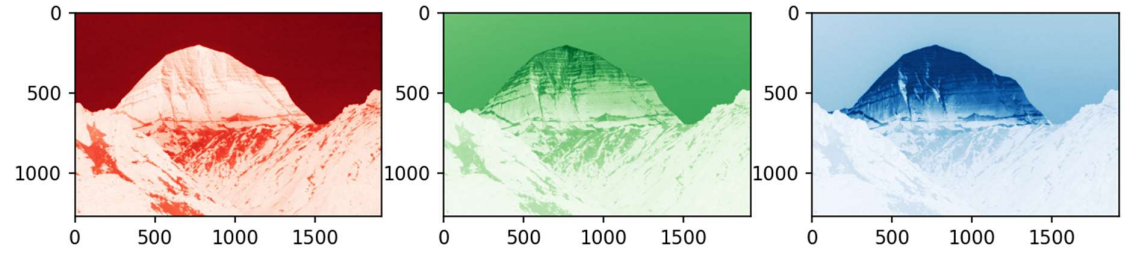
3. Convert the same image without using the OpenCV library
   You need to define the channel weights:
   [Red,Green,Blue] = [0.2989, 0.5870, 0.1140]
   Output:
   Execution Time in Seconds: 0.102



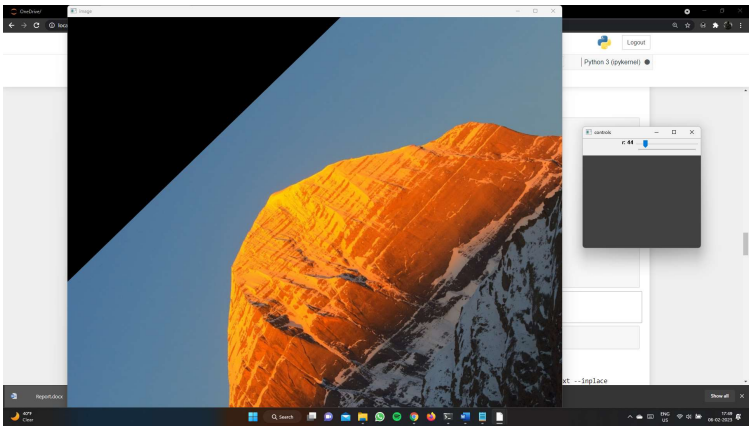
4. Converting the 3-channel image with weighted channels and 2 loops
   We used 2 loops to change each and every pixel to grayscale. We implemented using the Cython in order to increase the execution speed of the program. It took 2.19 seconds to execute the program.
   my.pyx file contains the necessary code for the conversion and the jupyter notebook cells contain the code which is used to execute the my.pyx file and get the desired output.
   We need to run the setup.py for configuring the Cython program. We need to run this setup.py file from command prompt but we can also run this code from the jupyter notebook itself using %%cmd statement.
   Output:
   Execution Time in Seconds: 2.19



5. Plot each channel in the image separately
   Output:



6. We used cv2.getRotationMatrix2D to get the rotation matrix for the given angle and cv2.warpAffine to rotate the image. implemented the trackbar to set the angle in real time.
   Output:

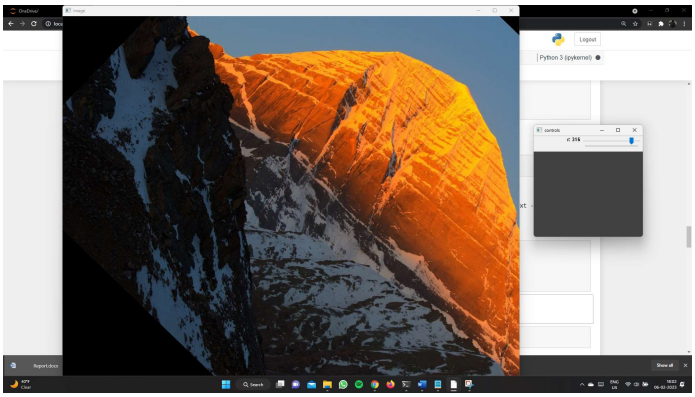


Trackbar implementation Output:

7. Here we need to implement Rotation Matrix without using the OpenCV library.
rotation_matrix = np.array([[np.cos(angle_rad), -np.sin(angle_rad), 0],[np.sin(angle_rad), np.cos(angle_rad), 0],[0, 0, 1]])

```
trans_matrix = np.array([[1, 0, center_x],
                         [0, 1, center_y],
                         [0, 0, 1]])
inv_trans_matrix = np.array([[1, 0, -center_x],
                         [0, 1, -center_y],
                         [0, 0, 1]])
```

We need to calculate the dot product of the above Rotation Matrix, Translation Matrix and Inverse Translation Matrix in order to get the Projection Matrix M. Then we need to pass this Projection Matrix M to the cv2.warpAffine function to rotate the image. We used Trackbar to get the angle of rotation.
Output:
my3.pyx contains the necessary code files and setup3.py contains the Cython configuration.



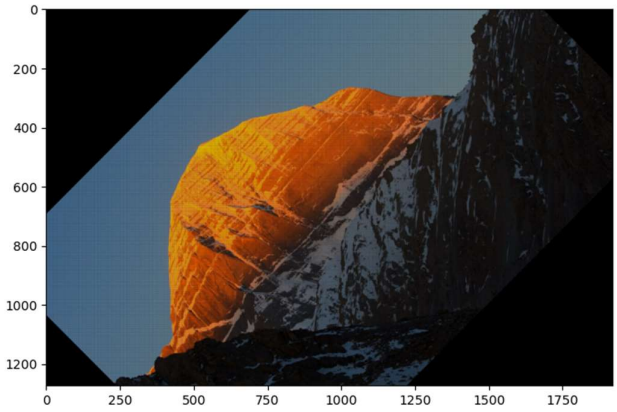
Trackbar Implementation:

8. We should find the rotation matrix and transform the image using the two for loops for transform each pixel in the image.

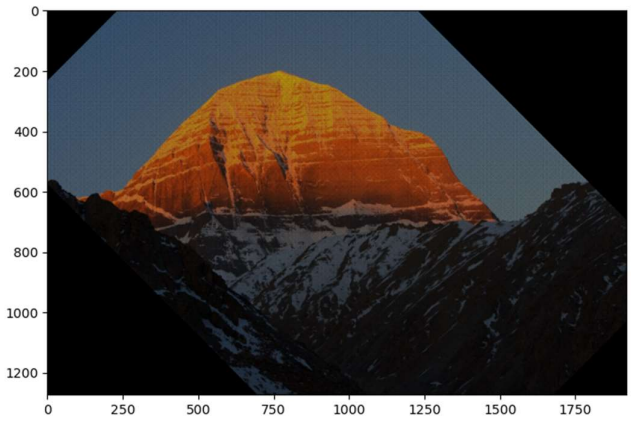
   We used the same projection matrix and used 2 loops to retrieve each pixel in the image. Next we transformed each pixel in the image using the projection matrix. If the transformed pixel is not within the original image coordinates then we map blank pixels. We used Cython implementation to speed up the computation.

   my2.pyx contains the necessary code and setup2.py contains the Cython configuration.

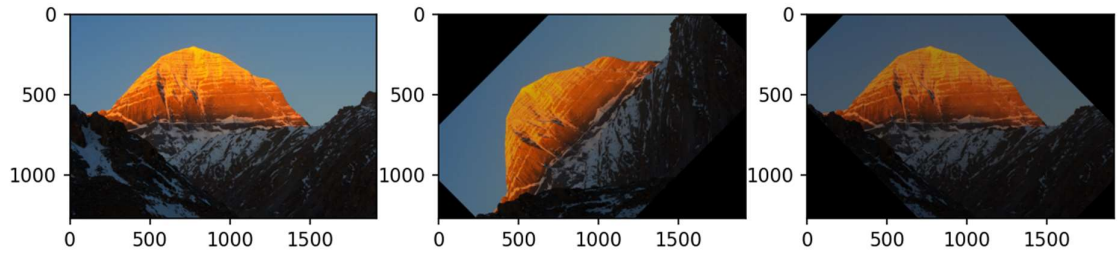
   Execution Time in Seconds is: 30.237
   Output:
   Rotation about 45 degrees.

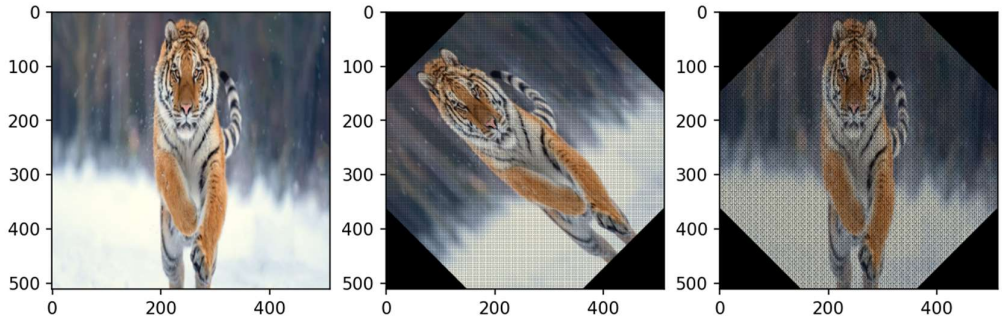


Next we use the inverse transform to rotate it back to original image.
Execution Time in Seconds: 55.021
Output:



Problems in this method:
When we are using custom defined rotation function we are loosing some pixels and hole are appearing in the image, but when we use OpenCV wrap function, bilinear interpolation or cubic interpolation are taken care by wrap function in OpenCV. But when we take high resolution image we don't see the difference but when we take the low resolution image we can see the difference.





The img_rotated.jpg contains the rotated image and img_original.jpg contains the original image after rotating it to back to original image using the inverse transformation matrix.

Results and Discussion:

If we transform an image using the custom defined warp function, it is taking too much time to execute and it is not taking care of the holes in the final image. In order to overcome this problem, we need to implement bilinear, intercubic or interarea which replicates the neighbouring pixels to fill the holes in the final image.

References:

https://stackoverflow.com/questions/23619269/calculating-translation-value-and-rotation-angle-of-a-rotated-2d-image