ELEC564 – Spring 2023
Homework 3

Date: February 28th, 2023

Student's Name: Hsuan-You (Shaun) Lin / Net ID: hl116

Link to Colab notebook: https://colab.research.google.com/drive/1C4l64HKk3CPvxZgukmcODQ8XqgNtcmRf?usp=sharing

1.0 Optical Flow

In this problem, you will implement both the Lucas-Kanade and Horn-Schunck algorithms. Your implementations should use a Gaussian pyramid to properly account for large displacements. You can use your pyramid code from Homework 2, or you may simply use Opencv's pyrDown function to perform the blur + downsampling. You may also use Opencv's Sobel filter to obtain spatial (x,y) gradients of an image.

1.1 Lucas-Kanade (5 points)

Implement the Lucas-Kanade algorithm, and demonstrate tracking points on this video.

1. **Select corners from the first frame using the Harris corner detector. You can use this command:** corners = cv.cornerHarris(gray_img,2,3,0.04)**.**

Figure 1. corners of cars video's first frame using Harris corner detector

2. **Track the points through the entire video by applying Lucas-Kanade between each pair of successive frames. This will yield one 'trajectory' per point, with length equal to the number of video frames.**
3. **Create a gif showing the tracked points overlaid as circles on the original frames. You can draw a circle on an image using cv.circle. You can save a gif with this code:**

   import imageio

   imageio.mimsave('tracking.gif', im_list, fps=10)

   where im_list is a list of your output images. You can open this gif in your web browser to play it as a video and visualize your results. Show a few frames of the gif in your report, and save the gif in your Google Drive, and place the link to it in your report. Make sure to allow view access to the file!

Link to the gif file: https://drive.google.com/file/d/1-NCadZ4XbevODbJpJbYgSmKm-pebWbUh/view?usp=sharing

Figure 2. Tracking points of cars video's gif using Lucas-Kanade algorithm

4. **Answer the following questions:**
   a. **Do you notice any inaccuracies in the point tracking? Where and why?**

      My Answer:

      I observed that the tracking points become inaccurate when there are changes in the shadows in the video. Specifically, when the truck in the upper left of the video passes by the pillars, the tracking points that were initially marked on the pillars disappear due to the appearance of the shadows.

   b. **b. How does the tracking change when you change the local window size used in Lucas-Kanade?**

      My Answer:

      When the window size is small, the Lucas-Kanade algorithm can track small-scale features accurately but may fail to track larger motions or when there is significant occlusion. On the other hand, when the window size is large, the algorithm can handle larger motions and occlusions but may suffer from blurring or smearing of features and may miss fine details.

1.2 Horn-Schunck (5 points)

Implement the Horn-Schunck algorithm. Display the flow fields for the 'Army,' 'Backyard,' and 'Mequon' test cases from the Middlebury dataset, located here. Consider 'frame10.png' as the first frame, and 'frame11.png' as the second frame for all cases.

Use this code to display each predicted flow field as a colored image:

```
hsv = np.zeros(im.shape, dtype=np.uint8)
hsv[..., 1] = 255
mag, ang = cv.cartToPolar(flow_x, flow_y)
hsv[..., 0] = ang * 180 / np.pi / 2
hsv[..., 2] = cv.normalize(mag, None, 0, 255, cv.NORM_MINMAX)
out = cv.cvtColor(hsv, cv.COLOR_HSV2RGB)
```

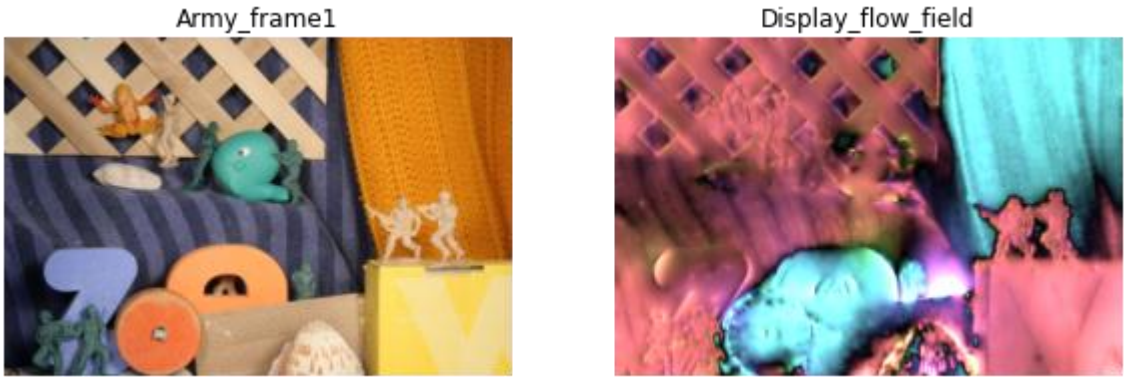

Figure 3. Army photo and flow field using Horn-Schunck algorithm

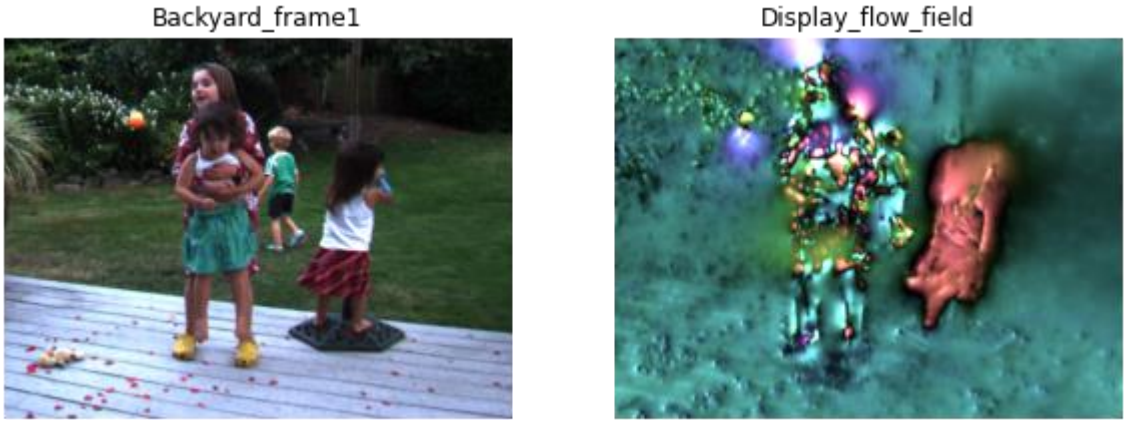

Figure 4. Backyard photo and flow field using Horn-Schunck algorithm

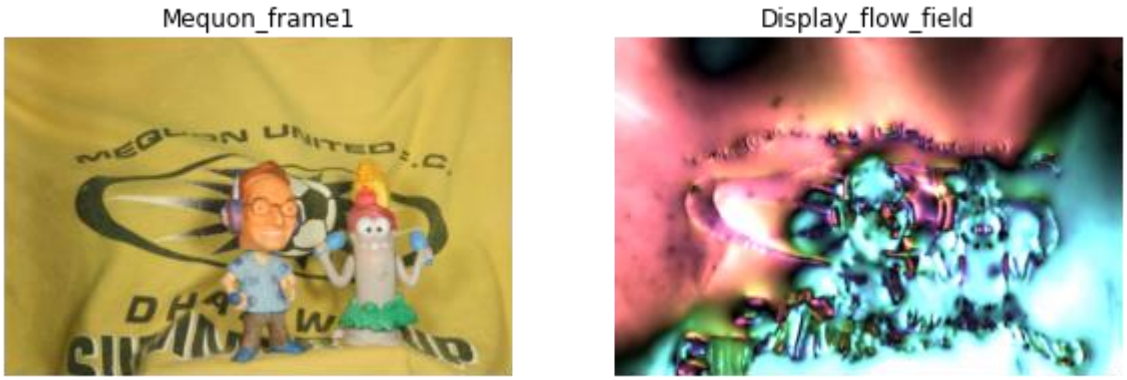

Figure 5. Mequon photo and flow field using Horn-Schunck algorithm

1.3 ELEC/COMP 546 Only: Improving Horn-Schunck with superpixels (5 points)

Recall superpixels discussed in lecture and described further in this paper. How might you use superpixels to improve the performance of Horn-Schunck? Can you incorporate your idea into the smoothness + brightness constancy objection function? Define any notation you wish to use in the equation. You don't have to implement your idea in code for this question.

My answer:

In this paper, the authors propose a novel algorithm, Simple Linear Iterative Clustering (SLIC), that clusters pixels in the five-dimensional color and image plane space to generate nearly uniform superpixels. The algorithm is simple to use and efficient, with a lone parameter specifying the number of superpixels.

Experimental results show that SLIC produces high-quality superpixels with lower computational cost compared to four state-of-the-art methods.

I think superpixels can improve the performance of the Horn-Schunck algorithm by reducing the influence of small image details that can cause noise in the flow estimates.

To incorporate superpixels into the Horn-Schunck algorithm, we can modify the smoothness and brightness constancy objective functions to operate on superpixels instead of individual pixels. Let X be the set of superpixels in the image, and let I(x,t) be the intensity of the image at pixel x and time t. Let u(x,t) and v(x,t) be the horizontal and vertical components of the flow vector at pixel x and time t.

The original HS objective function for smoothness is:

$$\int (\nabla u(x,t)^2 + \nabla v(x,t)^2)\, dx$$

where ∇u and ∇v are the horizontal and vertical gradients of u and v.

To incorporate superpixels, we can replace the integral over pixels with a sum over superpixels:

$$\sum (S \in X) \int S(\nabla u(x,t)^2 + \nabla v(x,t)^2) dx$$

where S is a superpixel in X.

Similarly, the original Horn-Schunck objective function for brightness constancy is:

$$\int (\mathrm{I}(x,t) - \mathrm{I}(x + u(x,t), t+1) + \mathrm{I}(x,y) - \mathrm{I}(x + v(x,t), y+1))^2\, dx$$

To incorporate superpixels, we can replace the integral over pixels with a sum over superpixels:

$$\sum (S \in X) \int S(\mathrm{I}(x,t) - \mathrm{I}(x + u(x,t), t+1) + \mathrm{I}(x,y) - \mathrm{I}(x + v(x,t), y+1))^2\, dx$$

where S is a superpixel in X.

By using superpixels instead of pixels in the Horn-Schunck algorithm, we can reduce the influence of small image details that can cause noise in the flow estimates. Superpixels also provide a more meaningful representation of the image, which can help to improve the accuracy of the flow estimates.

In summary, incorporating superpixels into the Horn-Schunck algorithm can improve the accuracy of the flow estimates by reducing the influence of small image details and providing a more meaningful representation of the image. Modified smoothness and luminance constant objective functions can be used to manipulate superpixels rather than individual pixels. Although this approach requires additional computational resources, the benefits in terms of improved accuracy are significant for applications requiring high-quality optical flow estimation.

2.0 Image Compression with PCA

In this problem, you will use PCA to compress images, by encoding small patches in low-dimensional subspaces. Download these two images:

Test Image 1
Test Image 2

Do the following steps for each image *separately*.

2.1 Use PCA to model patches (5 points)

Randomly sample at least 1,000 16 x 16 patches from the image. Flatten those patches into vectors (should be of size 16 x 16 x 3). Run PCA on these patches to obtain a set of principal components. Please write your own code to perform PCA. You may use numpy.linalg.eigh**, or** numpy.linalg.svd **to obtain eigenvectors.**
Display the first 36 principal components as 16 x 16 images, arranged in a 6 x 6 grid (Note: remember to sort your eigenvalues and eigenvectors by decreasing eigenvalue magnitude!). Also report the % of variance captured by all principal components (not just the first 36) in a plot, with the x-axis being the component number, and y-axis being the % of variance explained by that component.

Figure 6. Test image 1 – station image

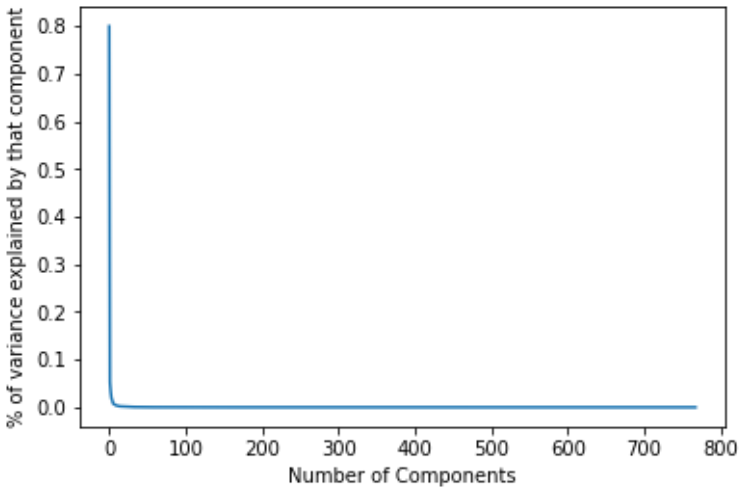

Figure 7. PCA % of variance captured by all principal components.

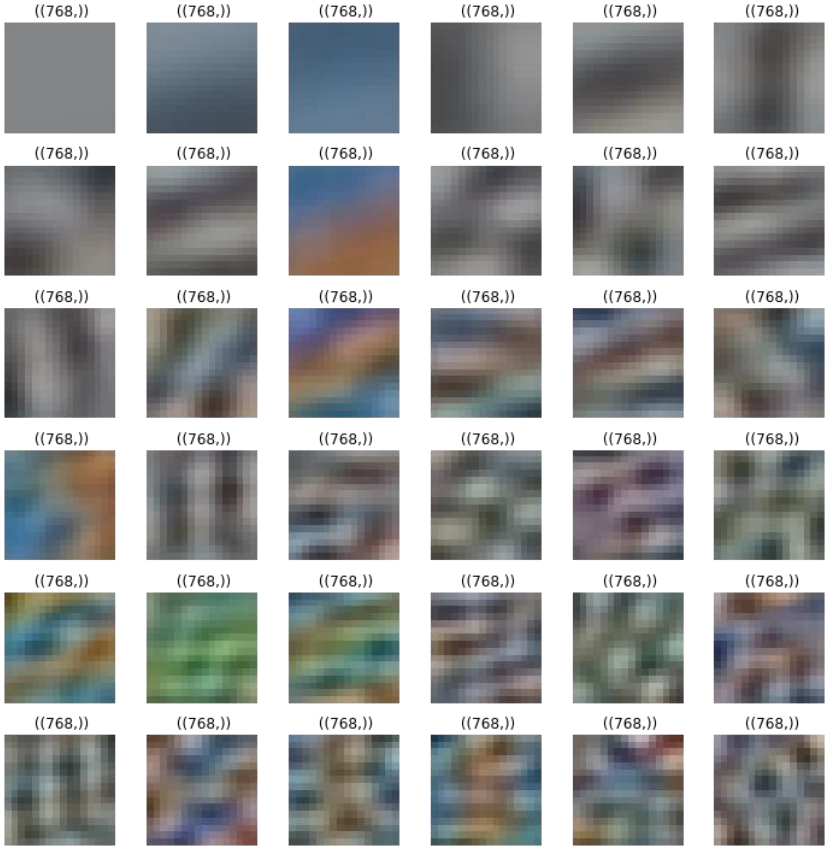

Figure 8. First 36 of station image's principal components as 16 x 16 images

Figure 9. Test image 2 – parrot image

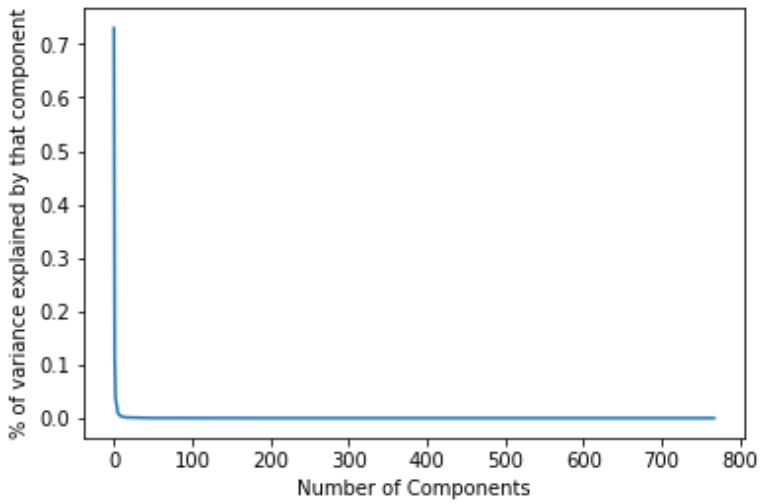

Figure 10. PCA % of variance captured by all principal components.

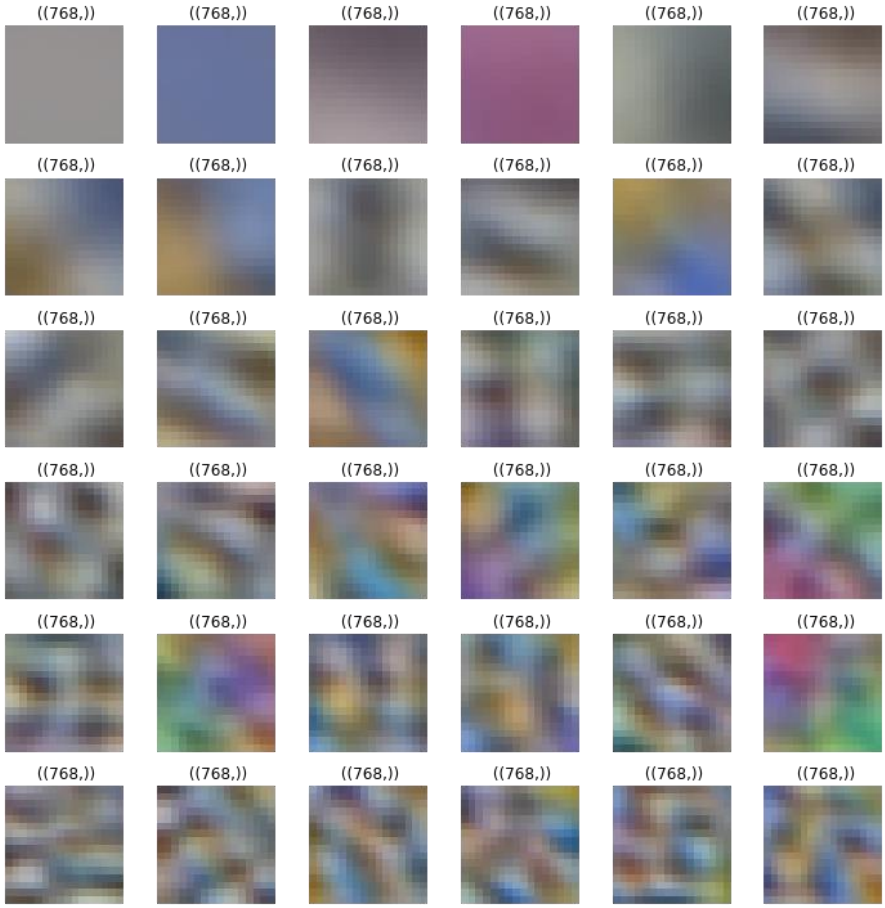

Figure 11. First 36 of parrot image's principal components as 16 x 16 images

2.2 Compress the image (5 points)

Show image reconstruction results using 1, 3, 10, 50, and 100 principal components. To do this, divide the image into non-overlapping 16 x 16 patches, and reconstruct each patch independently using the principal components. Answer the following questions:

1. **Was one image easier to compress than another? If so, why do you think that is the case?**
   My Answer:
   I observed that the parrot image is easier to compress, because it has a simple background and a single object in the image center, which means is require less data to accurately represent the image, On the other hand, the station image have multiple object and details, it require more data to accurately represent the image.
2. **What are some similarities and differences between the principal components for the two images, and your interpretation for the reason behind them?**
   My Answer:

   One similarity between the principal components of the two images is that the first principal component for both images capture a large amount of the overall variation in the image. This is expected, as the first principal component represents the direction of maximum variance in the data.

   The difference between the principal components of the two images is that the 3 principal component for the station image captures much more of the overall variation than the 3 principal component for the parrot image. I think this is because of that the station image has much more complexity and

variation in the scene, with many people, a train, and various objects in the background, while the parrot image has a simpler scene with a single object on a plain background.

Overall, the principal components capture the main patterns and variations in the image data, and the specific patterns that are captured depend on the complexity and structure of the image.

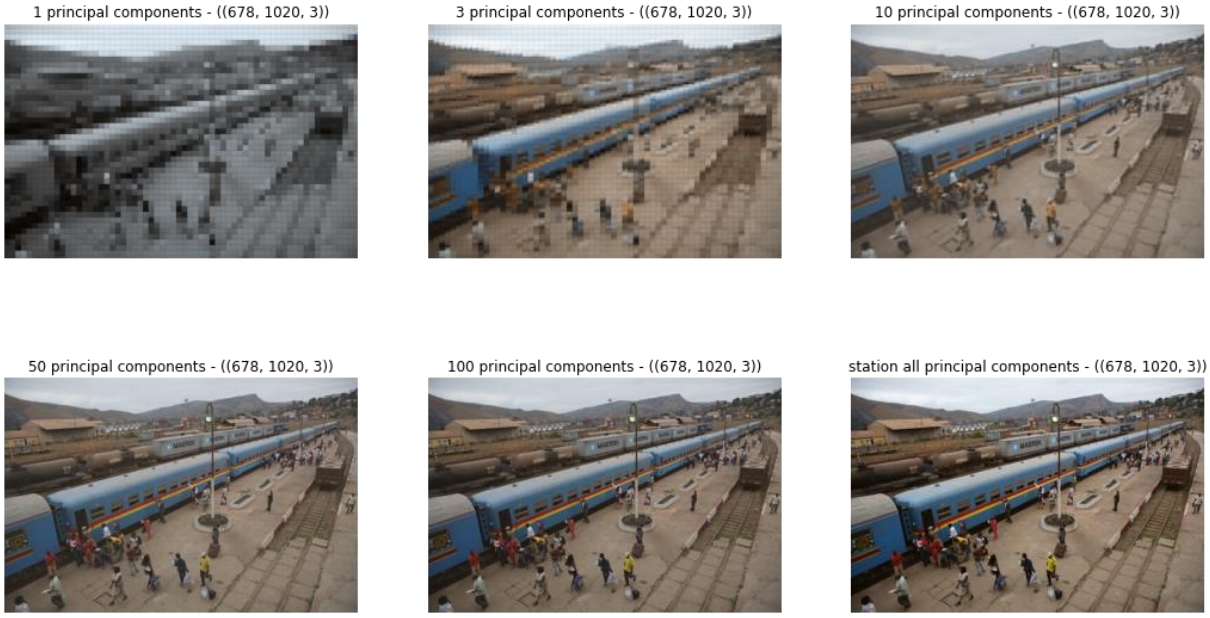

Figure 12. Reconstructed results of station image using 1, 3, 10, 50, and 100 principal components

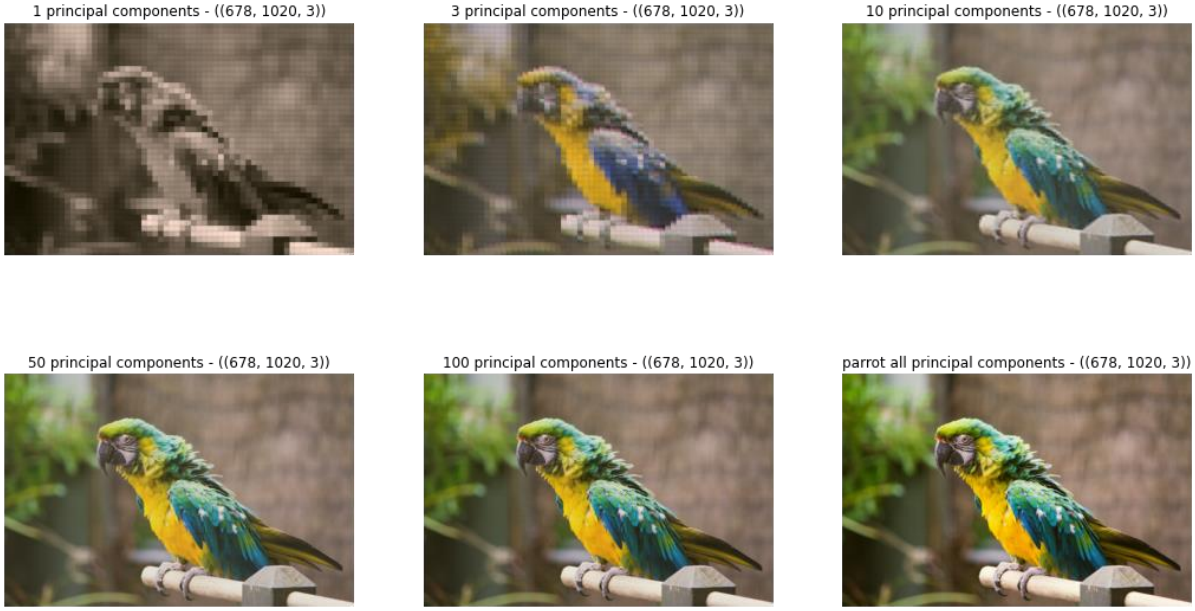

Figure 13. Reconstructed results of parrot image using 1, 3, 10, 50, and 100 principal components