Comparison of Different Masks On Six MPII Human Pose Images

Amir Khayatzadeh

In this document, I'll start with the code I used for applying some masks, and saving them for their later analysis.

1. Code

This is mostly the code for this project.

```
    images_directory = "./mpii_human_pose/"
    for image_name in os.listdir(images_directory):
        original_img = cv2.imread(images_directory + image_name)
        gray_scale_img = cv2.cvtColor(original_img, cv2.COLOR_RGB2GRAY)
        cv2.imwrite(f"gray{image_name}", gray_scale_img)

        # Laplacian of Gaussian(LoG)(edge detection)
        kernel = np.array([[-1, -1, -1], [-1, 8, -1], [-1, -1, -1]])
        result = cv2.filter2D(gray_scale_img, -1, kernel)
        cv2.imwrite(f"LoG{image_name}", result)

        # box filter(smoothing kernel) (3, 3)
        kernel = np.ones((3, 3), np.float32) / 9
        result = cv2.filter2D(gray_scale_img, -1, kernel)
        cv2.imwrite(f"box3_3{image_name}", result)

        # box filter(smoothing kernel) (5, 5). i.e., more smooth
        kernel = np.ones((5, 5), np.float32) / 25
        result = cv2.filter2D(gray_scale_img, -1, kernel)
        cv2.imwrite(f"box5_5{image_name}", result)

        # box filter(smoothing kernel) (7, 7). i.e., even more smooth
        kernel = np.ones((7, 7), np.float32) / 49
        result = cv2.filter2D(gray_scale_img, -1, kernel)
        cv2.imwrite(f"box7_7{image_name}", result)

        # Gaussian filter(smoothing kernel)
        kernel = np.array([[1, 2, 1], [2, 4, 2], [1, 2, 1]]) / 16
        result = cv2.filter2D(gray_scale_img, -1, kernel)
        cv2.imwrite(f"gaussian{image_name}", result)
```

```
        # upper triangular with -1 as k
        kernel = np.triu([[1,1,1],[1,1,1],[1,1,1]], -1)
        result = cv2.filter2D(gray_scale_img, -1, kernel)
        cv2.imwrite(f"triu-1{image_name}", result)

        # upper triangular with 0 as k
        kernel = np.triu([[1,1,1],[1,1,1],[1,1,1]], 0)
        result = cv2.filter2D(gray_scale_img, -1, kernel)
        cv2.imwrite(f"triu0{image_name}", result)

        # upper triangular with 1 as k
        kernel = np.triu([[1,1,1],[1,1,1],[1,1,1]], 1)
        result = cv2.filter2D(gray_scale_img, -1, kernel)
        cv2.imwrite(f"triu1{image_name}", result)

        # lower triangular with -1 as k
        kernel = np.tril([[1,1,1],[1,1,1],[1,1,1]], -1)
        result = cv2.filter2D(gray_scale_img, -1, kernel)
        cv2.imwrite(f"tril-1{image_name}", result)

        # lower triangular with 0 as k
        kernel = np.tril([[1,1,1],[1,1,1],[1,1,1]], 0)
        result = cv2.filter2D(gray_scale_img, -1, kernel)
        cv2.imwrite(f"tril0{image_name}", result)

        # lower triangular with 1 as k
        kernel = np.tril([[1,1,1],[1,1,1],[1,1,1]], 1)
        result = cv2.filter2D(gray_scale_img, -1, kernel)
        cv2.imwrite(f"tril1{image_name}", result)
```

The original images in gray scale are as follows.

Original images

2. Masks

Now I'll go through the different masks which I applied on these images.

2.1. LoG

LoG Mask

2.2. Box Filter

2.2.1. 3x3 Box Filter

3x3 Box Filter

2.2.2. 5x5 Box Filter

5x5 Box Filter

2.2.3. 7x7 Box Filter

7x7 Box Filter

2.3. Gaussian Filter

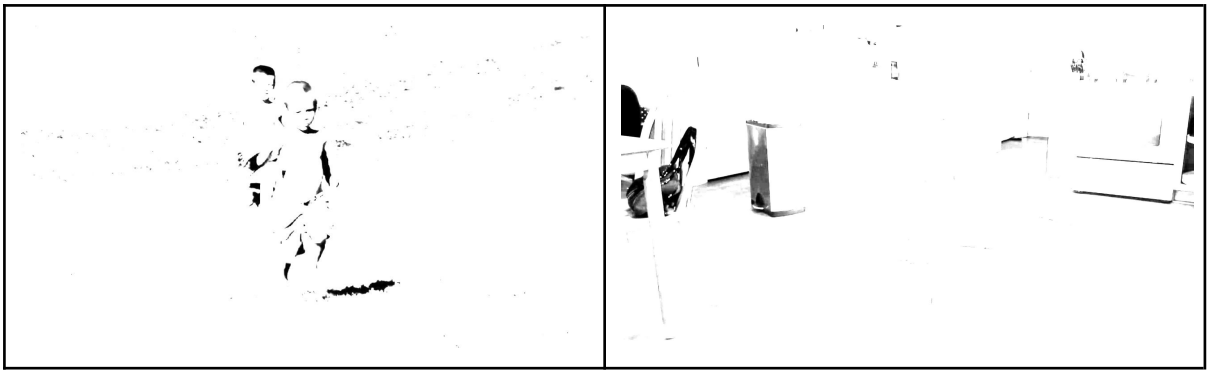
Gaussian Filter

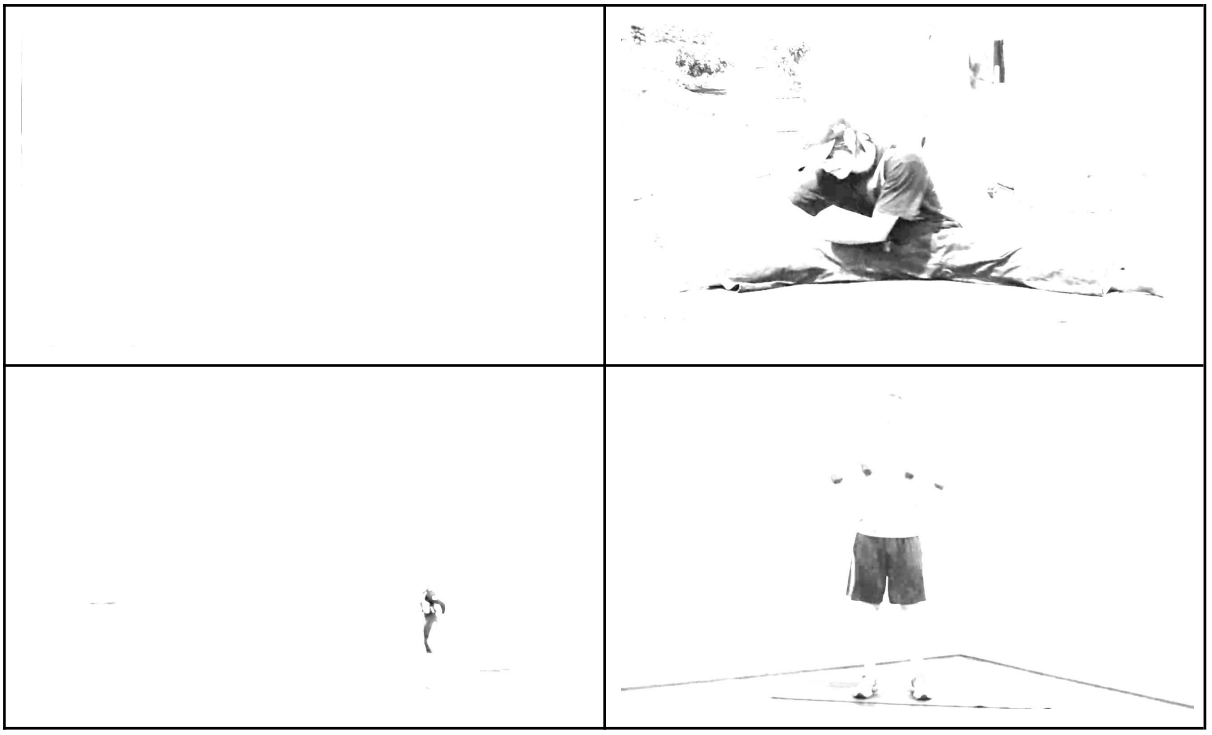
2.4. Upper Triangular Filter

2.4.1. Upper Triangular Filter with -1 as k

The kernel for this filter is

[[1, 1, 1],
 [1, 1, 1],
 [0, 1, 1]].



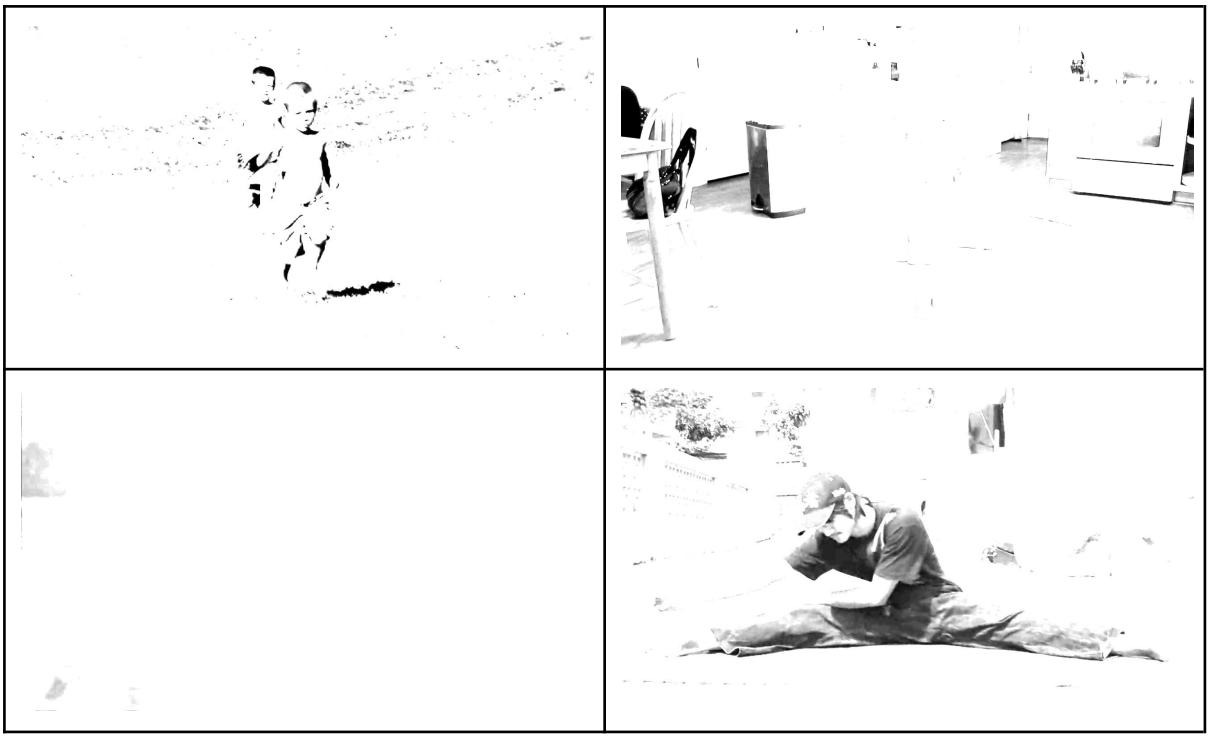


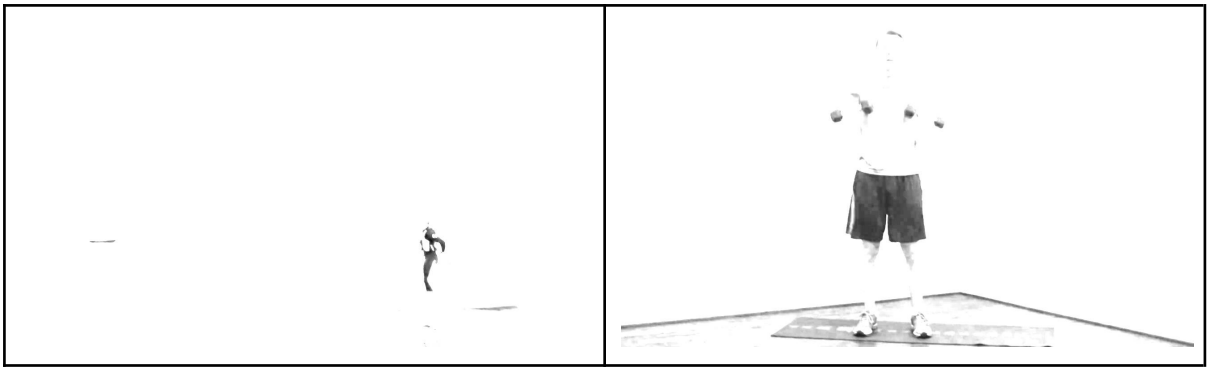
Upper Triangular with -1 as k

2.4.2. Upper Triangular with 0 as k

The kernel for this filter is

```
[[1, 1, 1],
 [0, 1, 1],
 [0, 0, 1]]
```



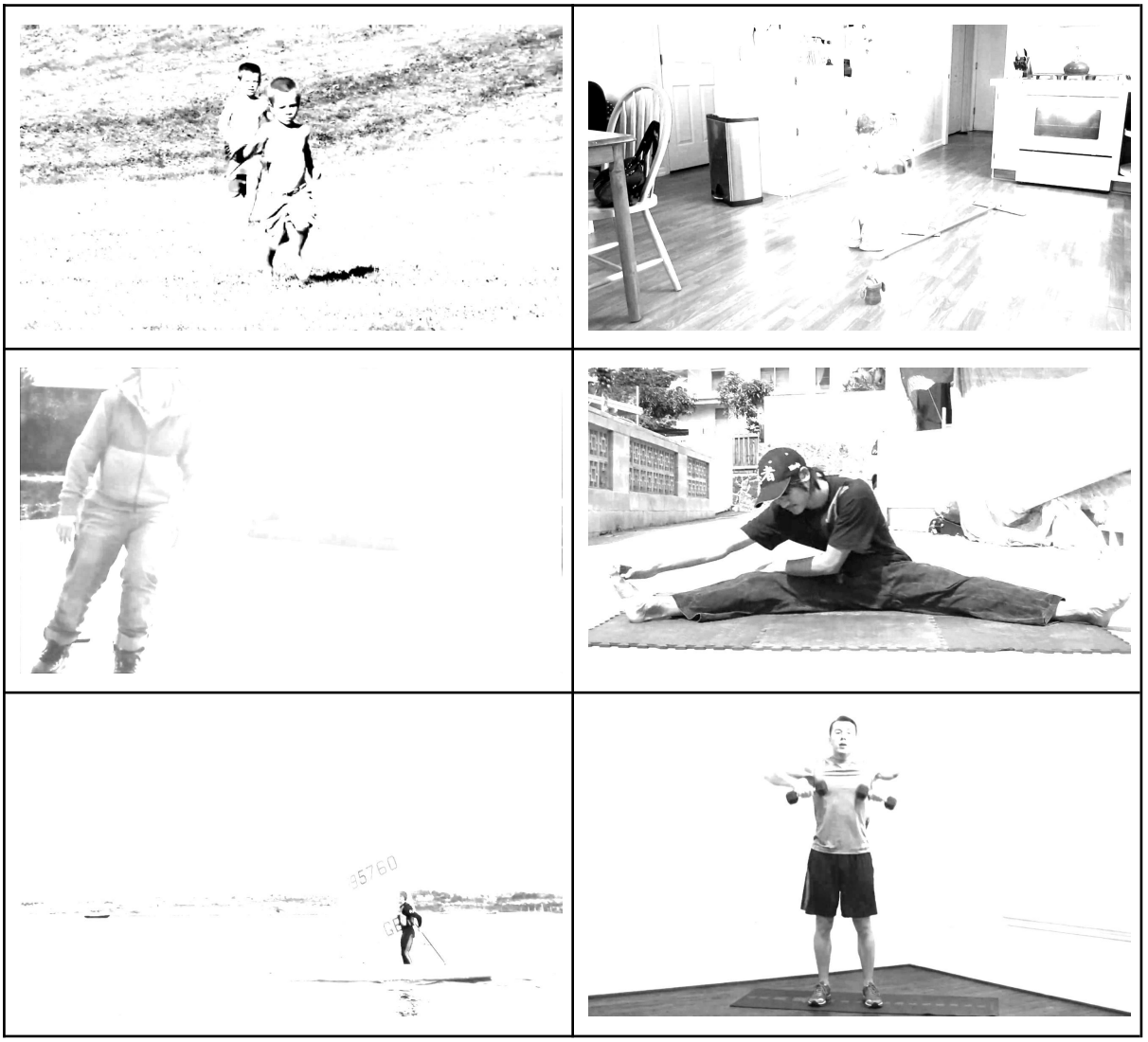


Upper Triangular with 0 as k

2.4.3. Upper Triangular with 1 as k

The kernel for this filter is

[[0, 1, 1],
[0, 0, 1],
[0, 0, 0]].



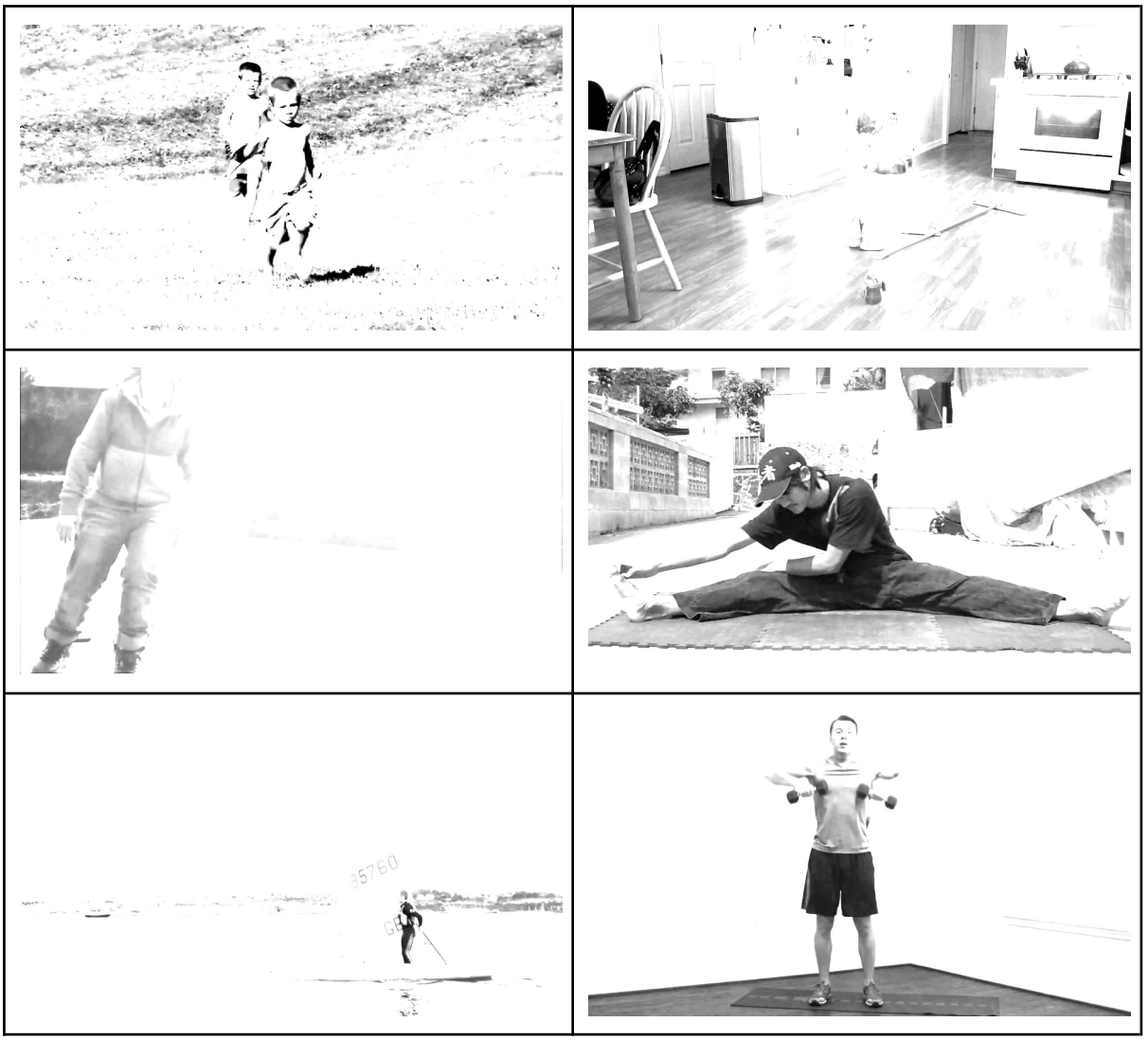
Upper Triangular with 1 as k

2.5. Lower Triangular Filter

2.5.1. Lower Triangular Filter with -1 as k

The kernel for this filter is

[[0, 0, 0],
[1, 0, 0],
[1, 1, 0]].

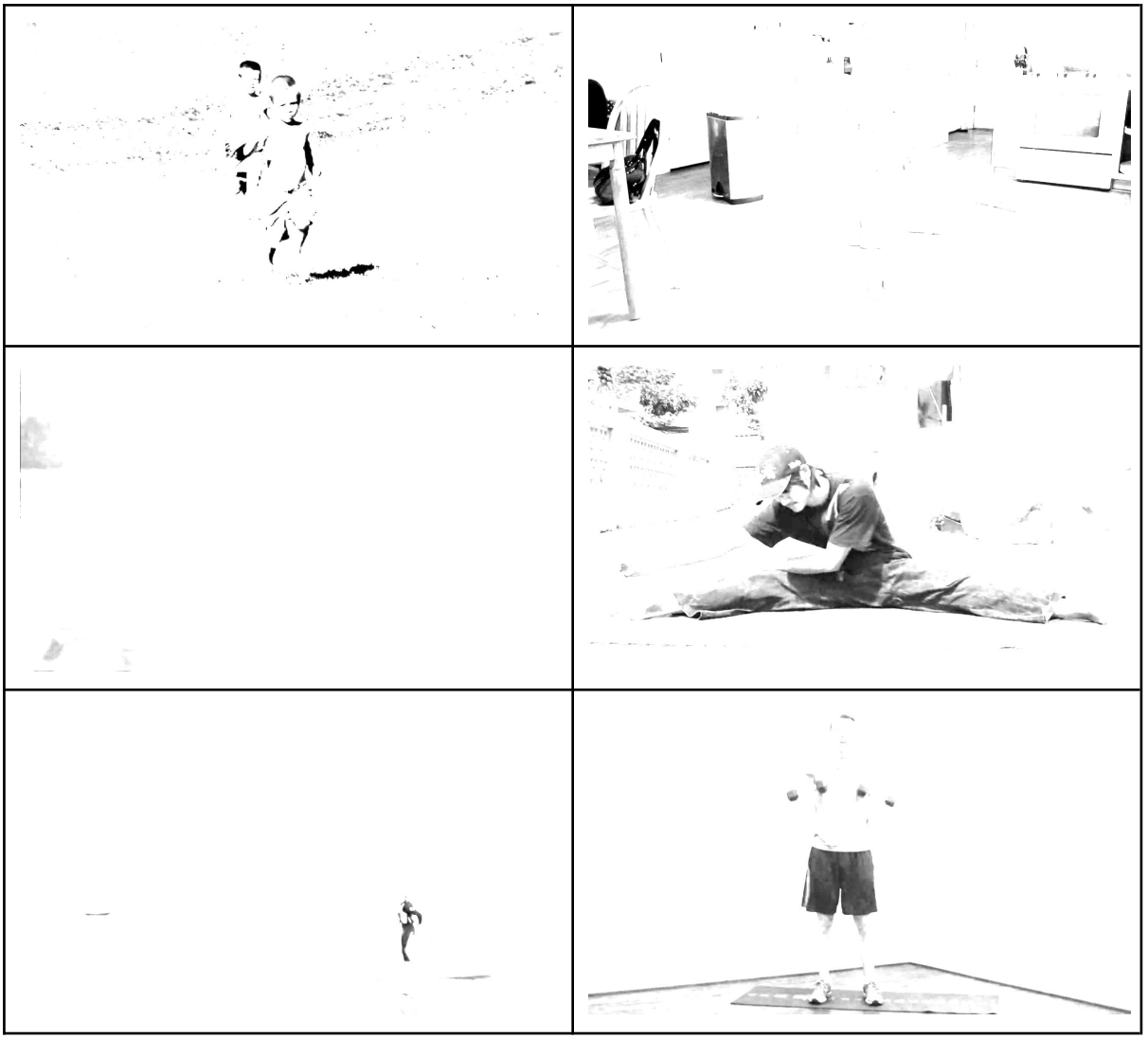


Lower Triangular Filter with -1 as k

2.5.2. Lower Triangular Filter with 0 as k

The kernel for this filter is

[[1, 0, 0],
[1, 1, 0],
[1, 1, 1]]

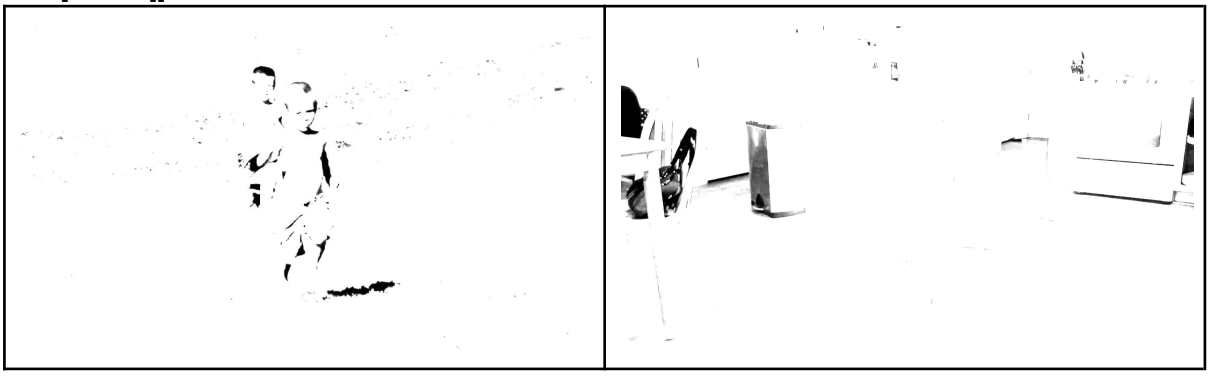


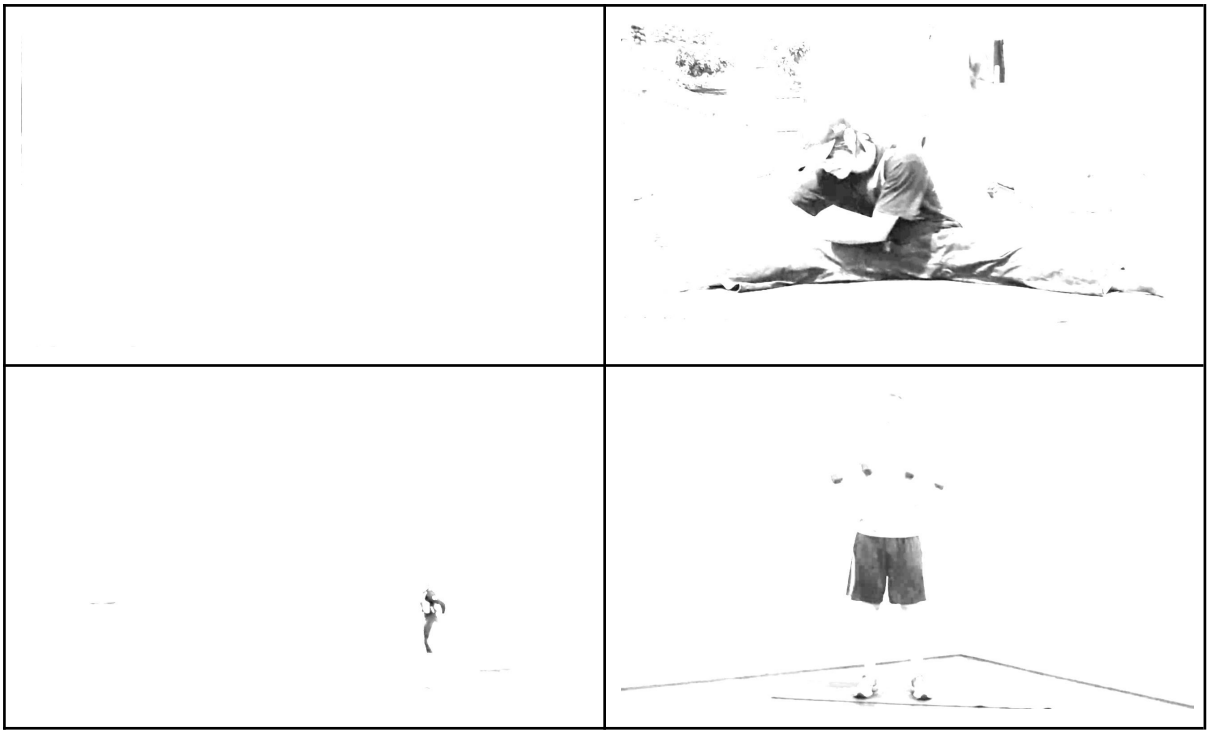
Lower Triangular Filter with 0 as k

2.5.3. Lower Triangular Filter with 1 as k

The kernel for this filter is

```
[[1, 1, 0],
 [1, 1, 1],
 [1, 1, 1]]
```





Lower Triangular Filter with 1 as k

3. Conclusion

After comparing the results, and looking at the effect of changing the kernel matrix, I have come to the conclusion that while edge detection filters like the LoG filter might be better for human pose estimation; I couldn't say it's the best filter. Therefore, for the future steps, I'll try mixing some blur and edge detection filters to see if that helps.
Looking at the results of the triangular filters, we can see how zeros in the kernel matrix affects the resultant image. Also the box filters denoise and blur the images more powerfully as you heighten the shape size of the kernel matrix.