**Activity Detection using Optical Flow**

**Dataset:**

For activity recognition, KTH standard dataset is used. KTH publicly available dataset for action recognition was initiated at the KTH Royal Institute of Technology in 2004. Dataset contains 6 actions/classes: walking, jogging, running, boxing, hand-waving and hand clapping. Each action is carried out by 25 individuals and every individual task is recorded in 4 different environments: outdoor (s1), outdoor with scale variation (s2), outdoor with different clothes (s3), and indoor (s4). So, total video samples of the dataset are 25x4x6 = 600. The resolution of each frame of the video is 120x160 and frame rate is 25fps.

**Folder Structure:**

Dataset is divided into folders visualized below:



Figure 1: KTH Dataset folder structure

Frames from each video are extracted through code and placed in the perspective folder with the same name and hierarchy visualized in figure 1.

 **Optical Flow:**

Optical flow or optic flow is the pattern of apparent motion of objects, surfaces, and edges in a visual scene caused by the relative motion between an observer and a scene. This technique is used to describe image motion and direction of motion. It works on consecutive frames (video). Optical flow calculates a velocity for points within the images, and provides and estimation of where points could be in the next image sequence. It is used to track objects in the scenery.

Figure 2: Optical Flow Demonstration in Traffic

**Optica Flow Methodologies:**

Following are the types of optical flow based on the attaention mechanism of the image:

1. **Sparse Optical Flow**

Sparse optical flow provides the flow vectors for selected “interesting features” (such as the edges or corners of an object) in the frame.

1. **Dense Optical Flow**

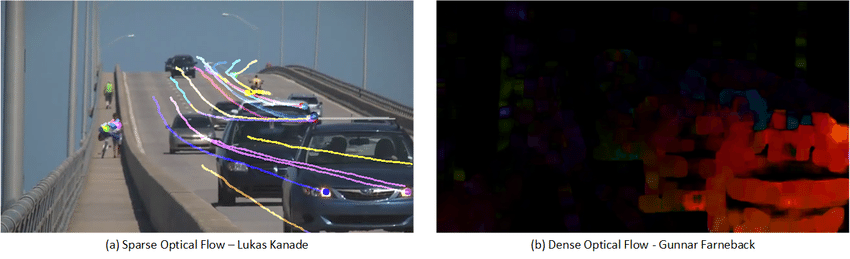
Dense optical flow provides the flow vectors for all pixels in the frame, up to one flow vector per pixel. As expected, Dense optical flow offers higher accuracy but comes at the cost of being computationally expensive and slow.

Figure 3: Sparse optical flow vs Dense optical flow

**Lucas–Kanade** algorithm is the example of sparse optical flow and **Gunnar Farneback’s** is an example of dense optical flow. Python library OpenCV provides both algorithms to use. This project uses Lucas-Kanade algorithm for faster results to meet the real time processing. Below is the description how Lucas-Kanade is integrated with the project.

**Lucas-Kanade:**

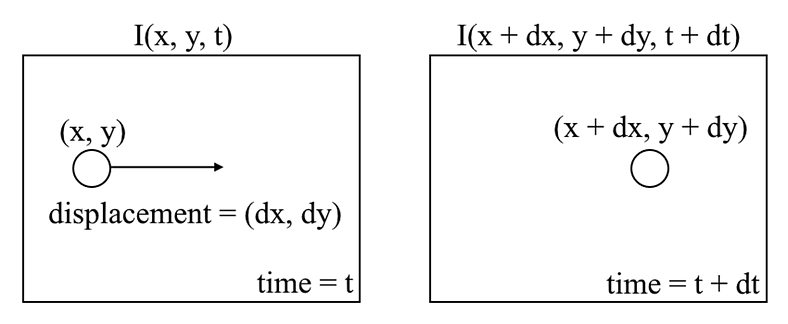
Lucas-Kanade is a popular technique for calculating optical flow under the category of sparse optical flow. This algorithm is based on the Brightness constancy assumption. The key idea here is that pixel level brightness won’t change a lot in just one frame. It assumes that the color of an object does not change between two consecutive frames. So, if we track the pixel as a good point or feature for that frame, the movement of that constant brightness containing pixel would tell us the movement of object in image.

Figure 4: Optical flow tracking mechanism

At code level implementation, first important points or features are highlighted with the OpenCV library function cv2.goodFeaturesToTrack that takes the image in gray scale and some feature parameters like maximum corner points in the image, quality of the corner point, etc. This function calculates the corner quality measure at every source image pixel using the cornerMinEigenVal or cornerHarris. The corners with minimal eigenvalue less then quality level is simply rejected. The remaining corners are sorted by the quality measure in the descending order. Function throws away each corner for which there is a stronger corner at a distance less than maximum distance specified in the feature parameter. The feature points highlighting can be visualized in figure 5 and 6.

Figure 5: Actual Frame

Figure 6: Good Points of the frame

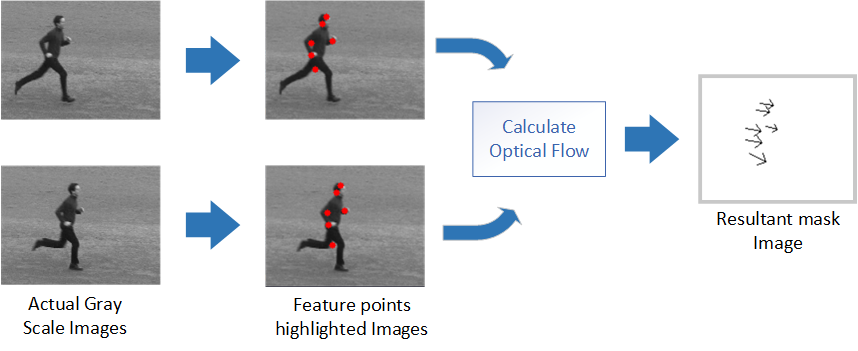
Now cv2.calcOpticalFlowPyrLK function is called with gray scale consecutive frames and first image good points with additional parameters like window size (in which good point has to be search), maximum level (maximum level of pyramid), etc. Then cv2.calcOpticalFlowPyrLK will return moved good features in second image. Now we have good features for both current frame and previous frame. We can draw directional arrows on mask image with the function cv2.arrowedLine that will show the direction of motion in the frames. This whole process can be visualized in figure 7.

Figure 7: Optical flow calculation steps

Then divide the resultant mask image into 4 segments. If arrows cutting any segment line, then that arrow line is split among those segments. For better visualization, every segment assigned its unique color. Like first, second, third and forth segments are assigned with color red, green, blue and black respectively. This can be visualized in figure 8.

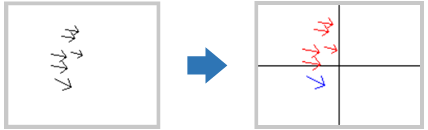
 Now every segment has feature arrows representing direction of motion and amount of motion (magnitude of x and y directions). Important point here is that If any segment has no feature arrow that represent no movement in that segment.

Figure 8: Feature Segmentation

Now normalize every segment of image, add all x directional resultant force and y directional resultant force. For every segment we have single x and y values representing x resultant force magnitude and direction and same for y. Also, we can find the angle between these directional forces.

= tan ()

Resultantly we have three features of each segment (x-directional force, y-directional force, angle). And each image has 4 segments. So, every result of two image optical flow has 4 x 3 = 12 features. These features are then stored in csv file for further model processing.

Standard KTH dataset has 6 actions or classes e.g. walking, running, jogging, boxing, hand clapping and hand waving. 3 of them (walking, running, and jogging) could be further divided as walking-left and walking-right. The resultant x directional force will determine this. If x is a positive (+) number then the moving object is moving in right direction like person walking left to right. Also, the negative x determines leftward direction. So, the dataset for classification here has 9 classes with distribution displayed in figure 9.

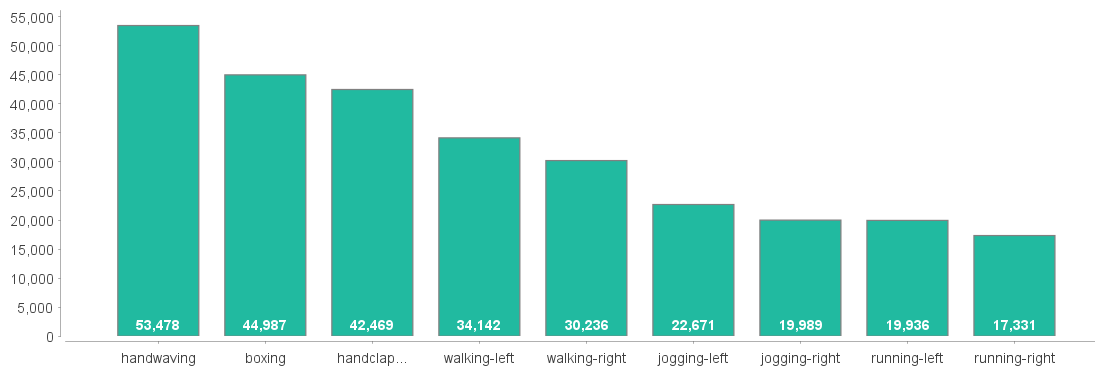


Figure 9: Class distribution of dataset

**Model:**

**Result:**