Exercise Lab02 – Jorge Lorenzo

1. Understanding the code

This is a Python script that uses OpenCV to perform optical flow analysis on video frames.

FBackMVFITemplates class

The script defines a **FBackMVFITemplates** class that initializes OpenCV's video capture and sets the parameters passed to it. It creates an output directory for the resulting files and initializes video writers for the colour and optical flow maps. It also initializes the flow arrays and sets some other OpenCV parameters.

The **draw flow** method is used to draw the optical flow vectors onto the images. The method iterates over the flow array, drawing a line from each pixel in the previous frame to the current pixel plus the flow vector at that pixel, with colour depending on the length of the flow vector. It also can fill squares around the vectors with colours depending on the length of the flow vector.

The **run** method reads the frames of the video file, calculates the optical flow, and calls **draw flow** on each pair of frames. It saves the resulting images and videos in the specified output directory.

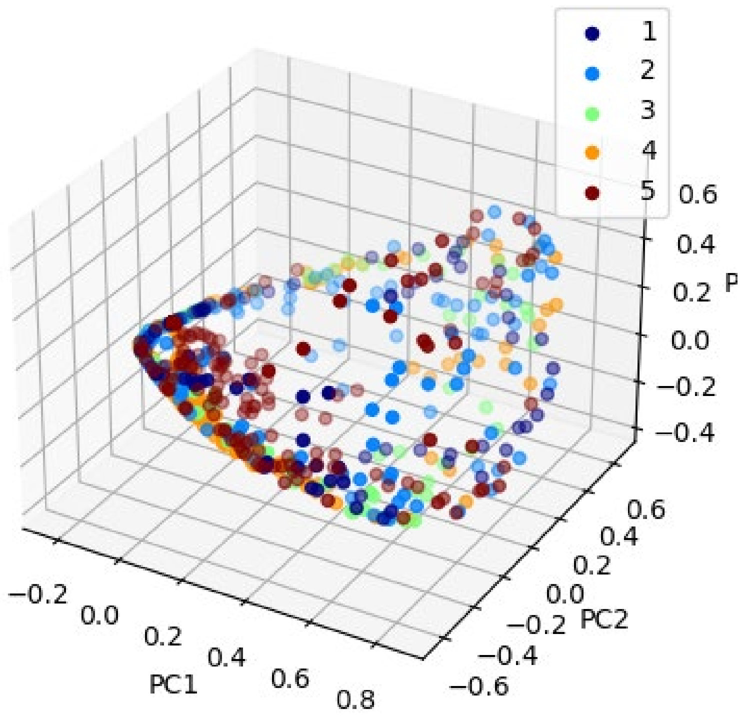
PCA/LDA code

The code is for classifying optical flow templates with PCA and LDA, using two-step LDA followed by PCA. It loads and pre-processes images, performs dimensionality reduction with PCA and LDA, and then visualizes the data. The functions can be used to perform analysis on images for a variety of purposes, such as computer vision and machine learning applications.

1. **load_images_from_directory(directory_path, target_size=(100, 100))**: This method loads images from a given directory and returns a list of images and their corresponding labels. It uses OpenCV to read and resize the images.
2. **flatten_images(images)**: This method takes a list of images and flattens them into a one-dimensional array.
3. **plot_pca_lda(pca_data, lda_data, labels)**: This method plots the PCA and LDA transformed data in a 2D scatter plot. The **pca_data** and **lda_data** are numpy arrays that contain the transformed data and labels is a list of corresponding labels.
4. **plot_3d_pca (pca_data, labels)**: This method plots the PCA transformed data in a 3D scatter plot.
5. **analysis_pca(X_train, y_train)**: This method performs PCA on the given training data and plots the transformed data in a 3D scatter plot.

6. **analysis_pca_lda(X_train, y_train)**: This method performs PCA followed by LDA on the given training data and plots the transformed data in two 2D scatter plots.
7. **analysis_kernelpca(X_train, y_train)**: This method performs Kernel PCA on the given training data and plots the transformed data in a 3D scatter plot. It also applies StandardScaler to pre-process the data before applying the kernel PCA.

2. Testing with provided datasets



Accurately estimating motion is easier when actions are simple, smooth, and follow predictable patterns, such as walking, and this is the case for the first 3 videos. On the other hand, the algorithm seems to face difficulties when it comes to complex actions that involve rapid changes, like a man suddenly falling to the ground. Similarly, optical flow may struggle with videos that have changing or poor lighting conditions, such as varying illumination, shadows, or reflections. Additionally, the complexity of backgrounds in videos seems to affect the performance of the optical flow method. For instance, videos with simple and static backgrounds, such as a walking person, do result in accurate motion estimation, whereas those with moving backgrounds, such as a racing car or leaves blowing in the wind challenge the algorithm in distinguishing between the action being performed and the background.

3. Identifying strengths and weaknesses

Strengths:

The method can calculate motion for every pixel in the frame thanks to its dense optical flow approach. Uses well-known methods with a strong foundation that have been proven effective in numerous scenarios. Creates colour images that allow for easy visualization of the estimated motion. The size of the rectangles used to display the motion can be adjusted therefore the method provides flexibility in controlling the level of detail in the output representation.

Weaknesses:

Does not take into account any “higher-level” information about the scene, such as object boundaries or semantic segmentation. This means that the motion estimation may not be as accurate in regions with complex motion.

The method does not include any smoothing or filtering, which means that the estimated motion may be noisy. Assumes constant brightness, which may not hold in all scenarios. For example, if the lighting conditions change between frames, the estimated motion may be incorrect. Finally, the method requires a significant amount of computational resources, which may make it difficult to use in real-time applications.

4. Changing/Improving the algorithm

One potential modification that could improve the performance of the existing optical flow algorithm code is to use a more advanced method for background subtraction. MOG2 is a commonly used method for background subtraction that models the background as a mixture of Gaussians and adapts the model over time to handle changes in the scene.

By using this method for background subtraction, we can obtain more accurate foreground masks, which can then be used to estimate the optical flow with greater accuracy. This can improve the performance of the optical flow algorithm, especially in scenarios with complex backgrounds.

Modifications have been made creating a new method named **run2(self)** inside the class FBackMVFITemplates