# Scale invariant features

The cost of extracting these features is minimized by taking a cascade filtering approach, in which the more expensive operations are applied only at locations that pass an initial test. Following are the major stages of computation used to generate the set of image features:

1. **Scale-space extrema detection**: The first stage of computation searches over all scales and image locations. It is implemented efficiently by using a difference-of-Gaussian function to identify potential interest points that are invariant to scale and orientation.
2. **Keypoint localization**: At each candidate location, a detailed model is fit to determine location and scale. Keypoints are selected based on measures of their stability.
3. **Orientation assignment**: One or more orientations are assigned to each keypoint location based on local image gradient directions. All future operations are performed on image data that has been transformed relative to the assigned orientation, scale, and location for each feature, thereby providing invariance to these transformations.
4. **Keypoint descriptor**: The local image gradients are measured at the selected scale in the region around each keypoint. These are transformed into a representation that allows for significant levels of local shape distortion and change in illumination.

SIFT identifies keypoints that are distinctive across an image’s width, height, and most importantly, scale. By considering scale, we can identify keypoints that will remain stable (to an extent) even when the template of interest changes size, when the image quality becomes better or worse, or when the template undergoes changes in viewpoint or aspect ratio.

## Scale space and image pyramid:

First, we double the input image in size and apply Gaussian blur. We assume the input image has a blur of ‘0.5’, if we want our resulting base image to have a blur of , we need to blur the doubled input image by (). We blur the input image with kernel size () and then blurring the resulting image by (), which is equivalent to blurring the input image by ().

Then we have to calculate the number of times to applying the function “computeNumberOfOctaves()” repeatedly halve an image until it becomes too small. Next, we generate gaussian kernels, which creates a list of the amount of blur image in a particular layer. And using these kernels to generate gaussian images. We halve the third-to-last image, since this has the appropriate blur we want, and use this to begin the next layer.