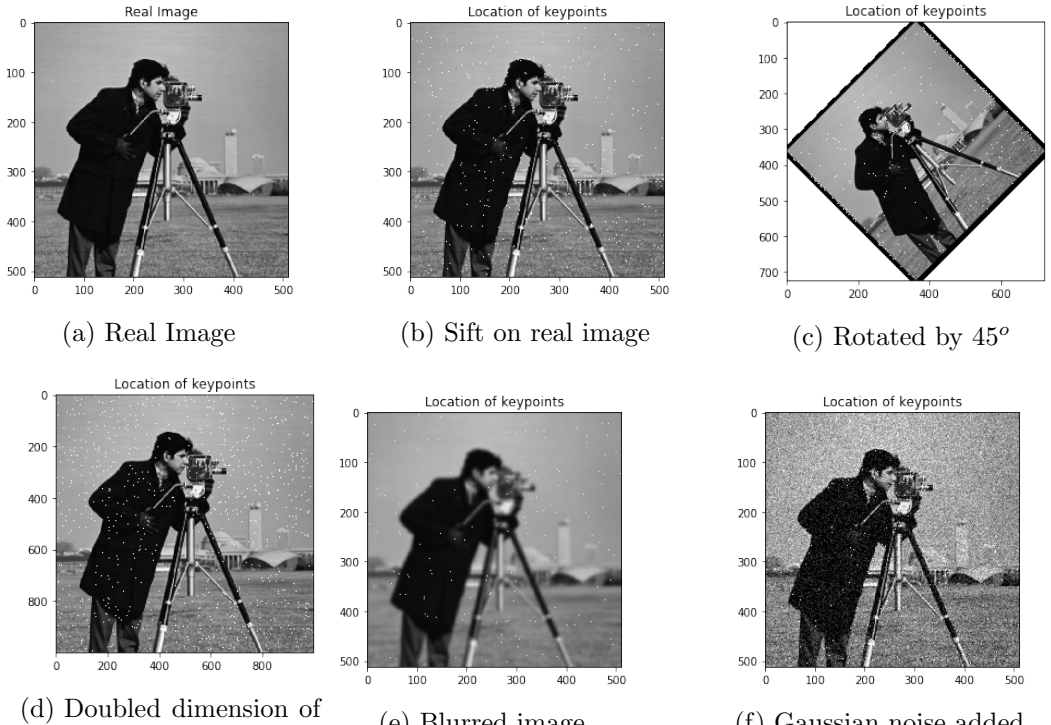
E9 246 — Advance Image Processing

Name: Alokendu Mazumder
Department and Course: PhD (EE)
Student Number: 20134
Link for Code(s): Click here

Problem 1 **Choose two images of your choice. Implement the first step of SIFT: i.e. scale-space extrema detection. Modify the images (rotate, scale, blur, add noise, etc.) and then re-do the keypoint detection step. Qualitatively analyze the keypoints that are detected.**

(a) (i) **Camera-man**



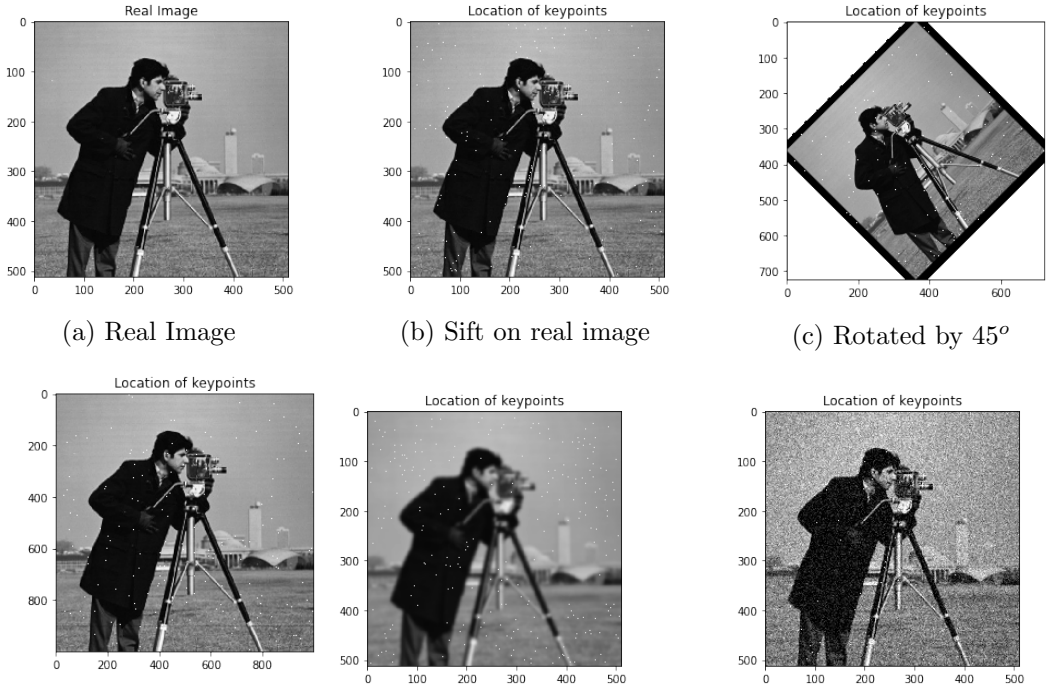
(a) Real Image (b) Sift on real image (c) Rotated by 45^o

(d) Doubled dimension of image (e) Blurred image (f) Gaussian noise added

Figure 1: SIFT with **sigma**: 1.6 and 8 as octave size

Observations: In Fig(1) one common thing is, most key points are detected when image is rotated, these key points are courtesy of python language, a frame is created to accommodate the rotated image, that edge of frame contains maximum keypoints. Hence, keypoints are sensitive towards drastic intensity change.

Least number of keypoints are detected when image is noisy (gaussian noise) and blurred, i.e when its being smooth. It supports our observation of previous paragraph, as the keypoints are sensitive to prominent edges/corners, soothing the images makes it hard to detect any edge or corner, which in turn supports the theory



(a) Real Image (b) Sift on real image (c) Rotated by 45^o

(d) Doubled dimension of image (e) Blurred image (f) Gaussian noise added

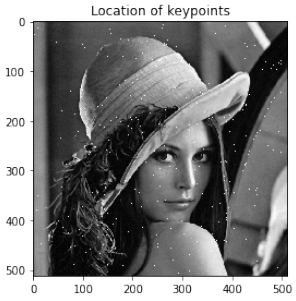
Figure 2: SIFT with **sigma**: 2.6 and 8 as octave size

of sift, it's hard to detect keypoints over very smooth region. I cannot come up with a concrete reasoning on why SIFT is detecting less points when image in noisy (gaussian), intuitively i think it's unable to find good features.

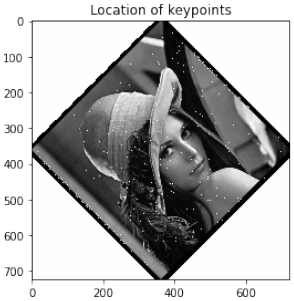
Now, in Fig(2) setup, when I increased the sigma value in SIFT, number of keypoints detected is less as compared to previous setup. Same with this setup, noisy and blurred images have least keypoints. Same observation follows. **Note:** In all the setting(rotate/blur/noise/rescale), good amount of common keypoints can be observed in both figures. keypoints are not detected over plain and smooth regions, they are present in important regions like edges, corners and regions of changing texture. Hence, fully supporting theory of SIFT, i.e scale and rotation invariant.

(b) **Woman painting**

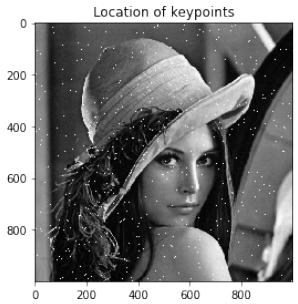
(a) Real Image



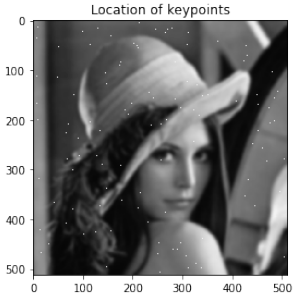
(b) Sift on real image



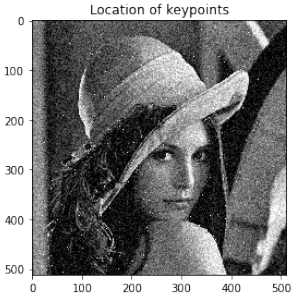
(c) Rotated by 45^o



(d) Doubled dimension of image



(e) Blurred image



(f) Gaussian noise added

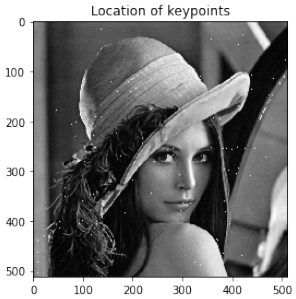
Figure 3: SIFT with **sigma**: 1.6 and 8 as octave size

Observations: This image is richer as compared to the previous one, it has strokes and textures, varying intensities over face and other regions and less amount of constant regions. In resize, I stretched it double of it's size.

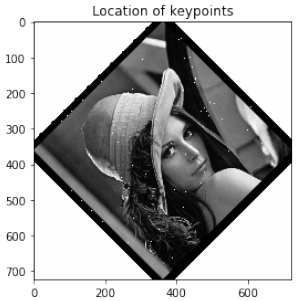
In previous set of images, it happened that some features which are common in all changes, but they are not present in blur version, in this set of images, we can see that features around the nose and eyes are common in all versions, even in blurred image.

Similar observation can be made, as we increased sigma, number of keypoints detected reduces. Less amount of keypoints are detectd in noisy and blur version of images. **Note**: In both the Fig(3) and Fig(4) configuration, sufficient amount of common keypoints can seen, hence supporting the theory of SIFT as it's invariant to scaling and rotation.

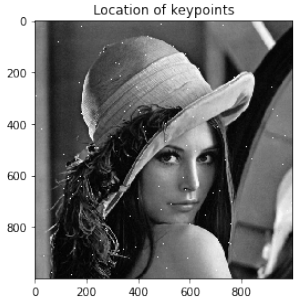
(a) Real Image



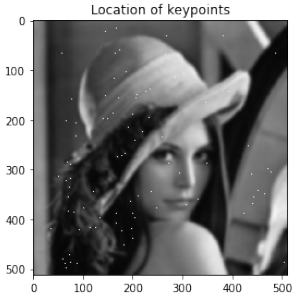
(b) Sift on real image



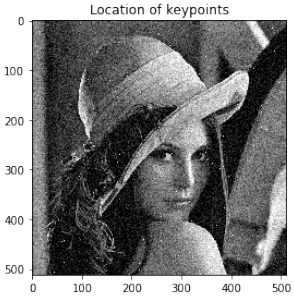
(c) Rotated by 45^o



(d) Doubled dimension of image



(e) Blurred image



(f) Gaussian noise added

Figure 4: SIFT with **sigma**: 2.6 and 8 as octave size

Table 1: Number of keypoints detected

Image	**Real**	**Rotate**	**Rescale**	**Blurred**	**Normal Noise**
Camera-man (with **sigma**: 1.6)	1270	1109937	7826	1202	698
Woman Painting (with **sigma**: 1.6)	940	1109518	4994	617	665
Camera-man (with **sigma**: 2.6)	445	1002998	2077	707	293
Woman Painting (with **sigma**: 2.6)	403	1002928	1466	404	307

Note: Original dimension of image used is (512x512) [for both images].

Question 2 Classification using CNN and deep features

(a) (i) **Theory**: I obtained 93.33 percent accuracy over testing data for **k**=8. This high accuracy are obvious for two sole reasons, first is, the feature vectors are pulled from last full connected layer of VGG16, so they are meant to be rich in features as they are already trained over the vast imagenet dataset. Secondly, in Bayesian learning theory, it is said that error of k-nn classifier is upper bounded by twice of error of optimum Baye's classifier (assuming true posterior densities), hence giving high accuracy when this theory is coupled with rich feature vectors.

One more observation I made, there is not much of intra class variation in test data when compared to training data, hence even if any classifier gets little overfit, it can still give good accuracy.

(ii) **Experimental Results**:

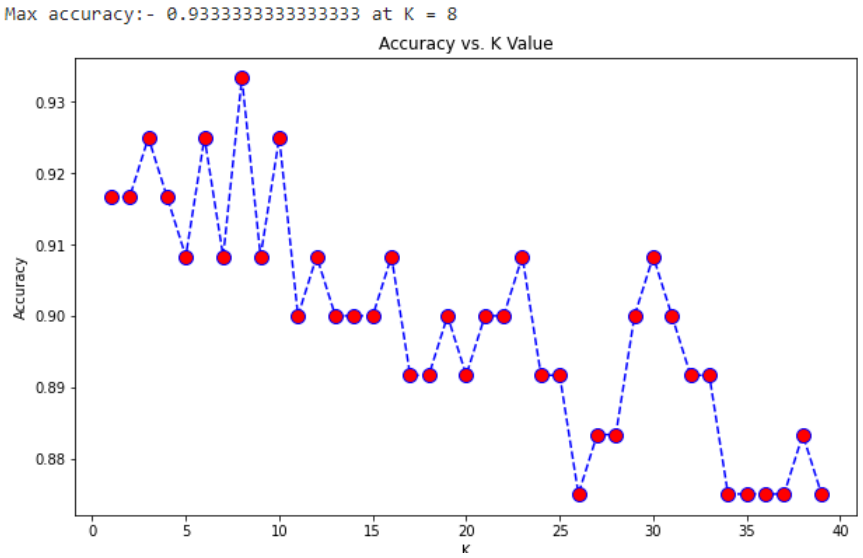


Figure 5: Plot of **Test** Accuracy v/s K value

I trained k-nn classfier over 1-40 vales for parameter k. Obtained the value of k for which test accuracy is maximum, i.e k=8. Input to k-nn classifier is the feature vectors obtained from last fully connected layer of VGG16 pretrained model with dimension of (4096,1). We resized each image to (224x224x3) before feeding it to pretrained VGG16.

(b) (i) **Theory**: Accuracy of 94.16 percent is obtained when only the last layer of pretrained VGG16 network is trained again. This is the highest classification accuracy obtained among the three parts of question 2. Reason is similar to above, the network is already trained over huge imagenet dataset and hence the frozen weights are meant to provide rich feature vector to the classification layer.

We can clearly see that there is some sort of co-adaption happening here, or we can say the whole model is generalized, i.e the VGG16 didn't have any **"represen-**

tation specificity", hence when only its last layer is trained, it's providing rich representation of feature vectors even over different dataset.

(ii) **Experimental Results**:

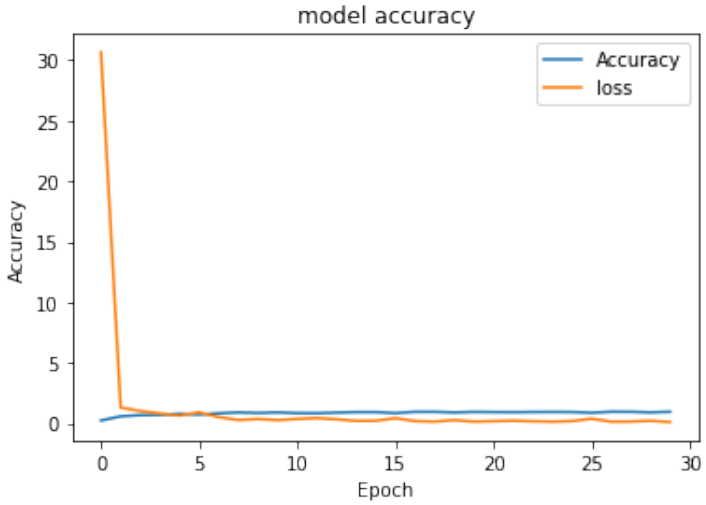


Figure 6: Training Accuracy/Loss v/s Epochs plot

Adam is used as optimizer with learning rate of 0.001, softmax is used in last classification layer with 6 neurons with cross entropy as loss function and for 30 epochs. Default batch-size of 32 is used. Dataset augmentation of training set is done using ImageDataGenerator from keras API. Each image (train as well as test) is resized to (224x224x3) before feeding it into pretrained VGG16.

Problems faced/Learning: No problems faced impementing or understanding this part. Learning is, co-adaption among pre-trained and fine tuned model is observed, i.e VGG16 has good generalizing power.

(c) (i) **Theory**: Below is the diagram of network that I have build for classification task from scratch. For any model to learn properly, it required adequate amount

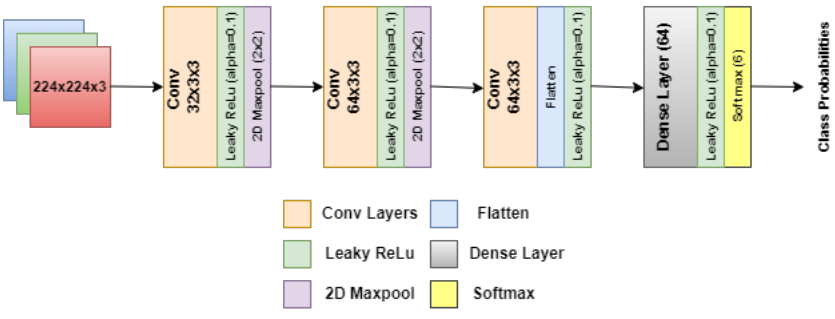


Figure 7: Neural Network architecture

of data, here training data has size of 240 samples only. The model was suppose to

overfit. The high test accuracy is obtained due to the fact that intra class variation is less among train and test set.

(ii) **Experimental Results**: An accuracy of 91.66 percent is obtained over test set. Leaky ReLu is used as activation function with aplha value of 0.1. Each image is resized to 224x224x3. It's a 5 layer CNN with first three layer being Convolutional Blocks of 32x3x3, 64x3x3, 64x3x3 respectively. The last layer is a softmax layer to predict class probabilities. Model is trained for 30 epochs with Adam optimizer with learning rate 0.0002. Default batch-size of 32 is used

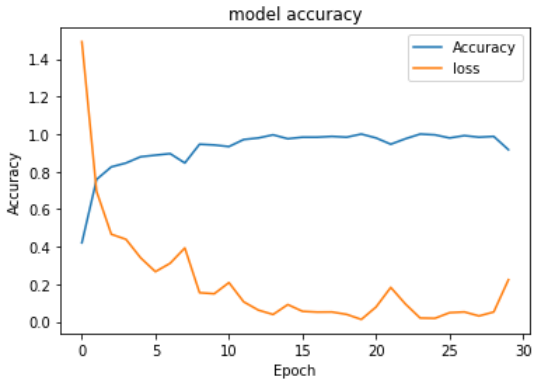


Figure 8: Training Accuracy/Loss v/s Epochs plot

The model I created is a light one, with only 3 Conv2D layers, as dataset is small, it's better not to use a big model(complex model with large no. of parameters) as it will definitely overfit. Dataset augmentation of training set is done using ImageDataGenerator from keras API. Each image (train as well as test) is resized to (224x224x3) before feeding it into pretrained VGG16.

Table 2: Comparison table

Model	**Accuracy (in percentage)**
K-NN with K=8	93.33
Fine-tuned VGG16	94.16
My Model	91.66

* accuracy with different values of k can be found at Fig(5).

Conclusion: It's obvious that fine-tuned model is performing best as it already has access to rich feature vectors, on the other hand my network has to learn everything from scratch that too with very limited sample size, hence it gave less accuracy as compared to pretrained one.

Same explanation is applicable for the k-nn classifier, which is mentioned clearly in it's "**Theory**" portion.