Hand Detection using Convolution Net

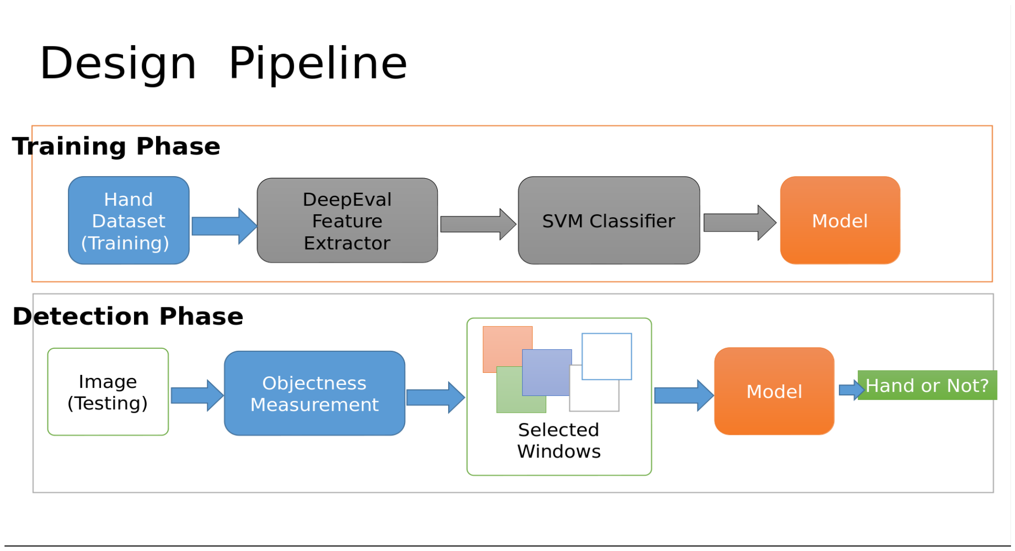
Md Nahid Hossain, Student ID 109722087
Syed Masum Billah, ID 109363705

Overview

Detecting human hands in still images is a very challenging task for obvious reasons. Some of the reasons are as follows: hands can be varied in shape and viewpoint, can be closed or open, can be partially occluded, can have different articulations of the fingers, can be grasping other objects or other hands, etc. In the paper "*Hand detection using multiple proposals*", Arpit Mittal et al. proposes a hand detector technique by applying a two-stage hypotheses and then feeding a classifier. We've tried detecting hand and compared our result with this paper.

Our Proposal

Deep learning is a set of algorithms in machine learning that attempts to model high-level abstractions in data by using architectures composed of multiple non-linear transformations. As an emerging AI technique, deep learning has achieved very impressive results and significantly outperformed state-of-the-art on various research topics. In our project we've used the deep-eval encoder to find the features which is detailed in the paper "*Return of the Devil in the Details: Delving Deep into Convolutional Nets*" by K. Chatfield, K. Simonyan, A. Vedaldi, A. Zisserman. After finding the features we've trained an SVM classifier using the LibSVM library. In the testing phase using a objectness proposal technique mentioned in the paper "*Segmentation as Selective Search for Object Recognition" by Koen E. A. van de Sande* et al. we generated objects from the images instead of doing an exhaustive search in different scale. These objects are later classified as hands or not, thus detecting the hands in an image. The model of our approach is given below:



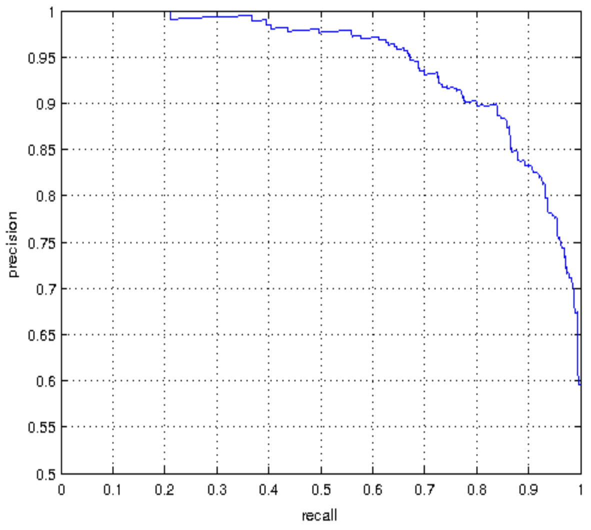
Dataset

We've used the dataset used by *Arpit Mittal et al.* that contains 13,050 hand instances and is divided into Training, Validation and Test sets.

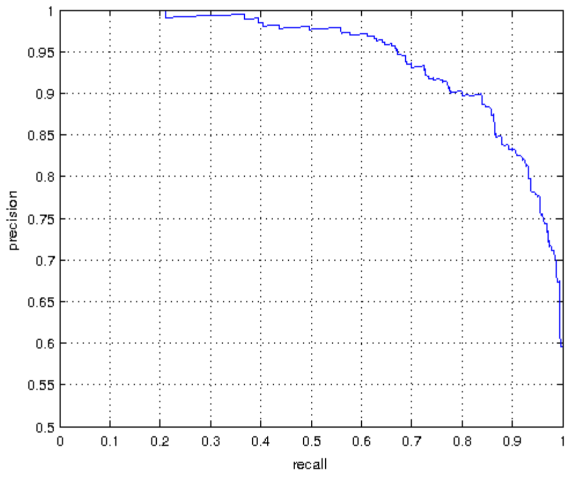
Training

For training at first all the images were rotated so that all the hands are aligned in a single axis, then we've cut out the hands by padding all the sides by 20%. We have 9000 hands sample which were cut out from the images. All the images were resized so that the smaller dimension of the images were 256. These are the positive images and the negative images were created by taking random patches from the images which doesn't include hand from the images. We have 73500 negative images. The dimension of the negative images are 256x256. The files *Create_Train_hand_dataset.m* and *neg_patch.m* cuts out the positive image and negative image patch respectively. After creating the positive and negative image patches we've extracted the features of these patches using the deep-eval encoder. The deep-eval encoder codes are given which we've used. In the *Train.m* file we've used the given encoder function to obtain the features of the image patches. We have used the CNN_M_128 network of the deep-eval encoder. It generates 128 features for each image patch. Also we've used the non-augmentation features instead of the augmented features. The augmented features cuts out five images patches from the 256x256 image. Four from the four corners and one from the middle. They are then flipped along the y axis thus as a result gives 10 feature vector of 128 features. Then using max or sum pooling they are merged as one. But the non augmented feature extracts a 224x224 patch from the middle. Sample of hand images:

We tested both the features using 500 positive data and 500 negative data. Incase of using the augmentation with sum pooling we get an average precision of 0.9457.

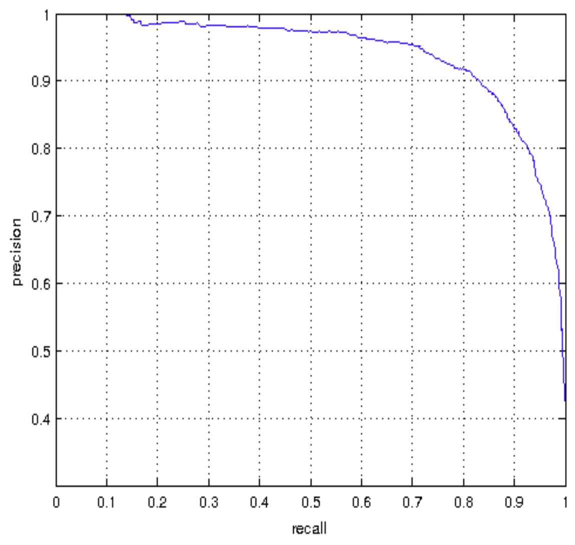


Using the non-augmented feature we got an AP of 0.9165.



We've used non-augmentation features because it takes 6 times less time to extract the features and the performance is almost the same. For features including augmentation it takes approx 0.63seconds/image to extract features but without including augmentation it takes only approx 0.1seconds/image to extract features.

After extracting the image patches we trained a svm classifier using the LibSVM library and created the model of the training data. The function *svmtrain()* is from the library. We then again tested the performance of our classifier with 1980 positive data and 5790 negative data. These data were collected the same way as the training data but this time they were collected from the test data set. Now we get an AP of 0.9411. We've improved the model by feeding hard negatives to it which as a result gave a slight increase of the AP, which is 0.9421.



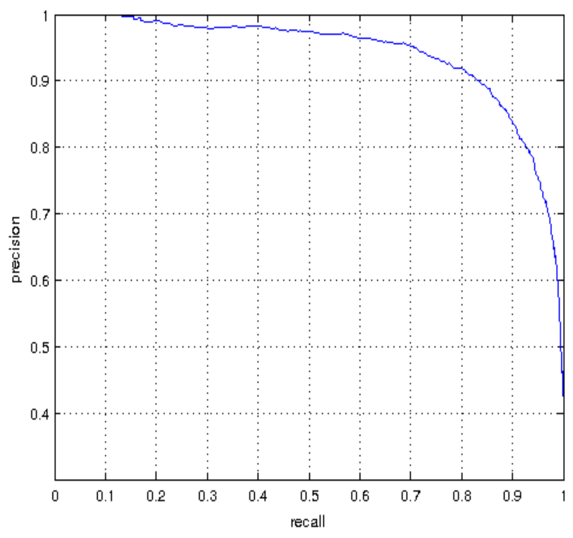
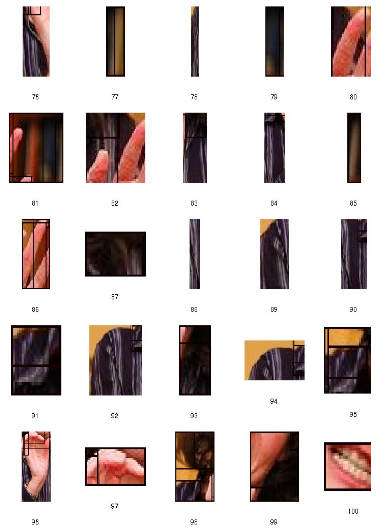


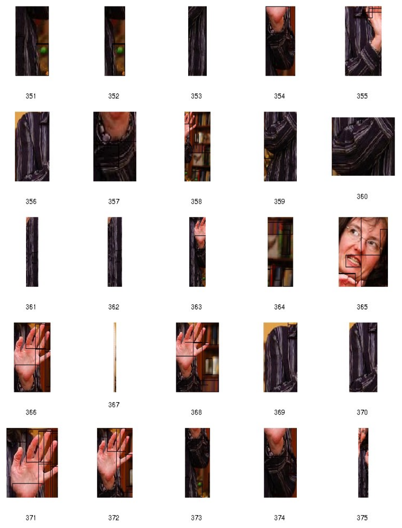
fig: Before bootstrapping(left), after bootstrapping(right)

The model after feeding hard negatives was later used in the detection phase.

Detection

There can be lots of hands in a single image and not all of them would be the same size. Also different images could have different number of hands of different scales. If we searched for a hand using a sliding window of different scales it would give us more than 100k proposal. Also we would have to check each of the separately and identify it as a hand or not. Instead we've used an objectness proposal method mentioned before which gave us patches of proposals in different scale. We've tested it and found out that the top 100 proposals included different hands of different scales in an image. So we've managed to decrease the number of search space from more than 100K to 100 only per image using "*Segmentation as Selective Search for Object Recognition"*. At the beginning of the *extractProposal.m* file we've called those library function to find the segmentations. There are different color types to extract the features. We've used the HSV color type because it gave us the best result. The file read all the images from directory and extract their proposal. After extraction proposals for each of the images are saved in the proposal directory. Number of proposals varies from image to image. In order to cut processing time, we consider top 100 proposals for each image. In our experiments, it is found that we are not missing any hand ground truth by taking only 100 proposals. Here are some of the object proposals from the given image:





Now after getting the proposals we need to find the orientation of the hand because the hands with the same orientation as the training images (which had all the hands aligned in one axis) will give high confidence value. We rotated each of the proposals 8 times with 45 degree to get the orientation. Next each of these orientated proposals are checked using the LibSVM library function *svmpredict()* with a threshold of -0.6 if it is a hand or not. The prediction of each images where it included of 8 orientations are stored in the prediction folder under the same name. These task is done by the *runHandDetector(file_name, encoder)* function which has a parameter of the file name and the CNN feature encoder which extracts the features of the proposals. This features are the ones that are compared with the model.

The *analyzeHandDetector(file_name,num_boxes,num_rotation)* function does further processing so that the prediction can be used to calculate the AP of our result. We generate the ground truth from the given test dataset and label the proposals as positive and negative.

The *showProposal(file_name,100,8,2)* function displays the objects that are detected as hand in the image. Here we perform non-maximal suppression. As the image will have a lot of proposals for each image, there are lots of overlap in the bounding boxes. So we tried reducing the boxes using Median NMS. It finds the overlap of the boxes and draws only those with high median values. Next we perform regular NMS to combine the small bounding box to one box. The function *nms_median()* and *nms_regular()* perform these tasks respectively.

The *detection_ap()* functions generates the AP of our hand detector.

Results

We tested our hand detector on 40 images and comparing these images with the ground truth we got an AP of 0.415. According to the "*Hand detection using multiple proposals*", Arpit Mittal et al. paper they got a maximum precision of 0.481 after using post processing. The precision recall graph of our method and the paper's method is given below:

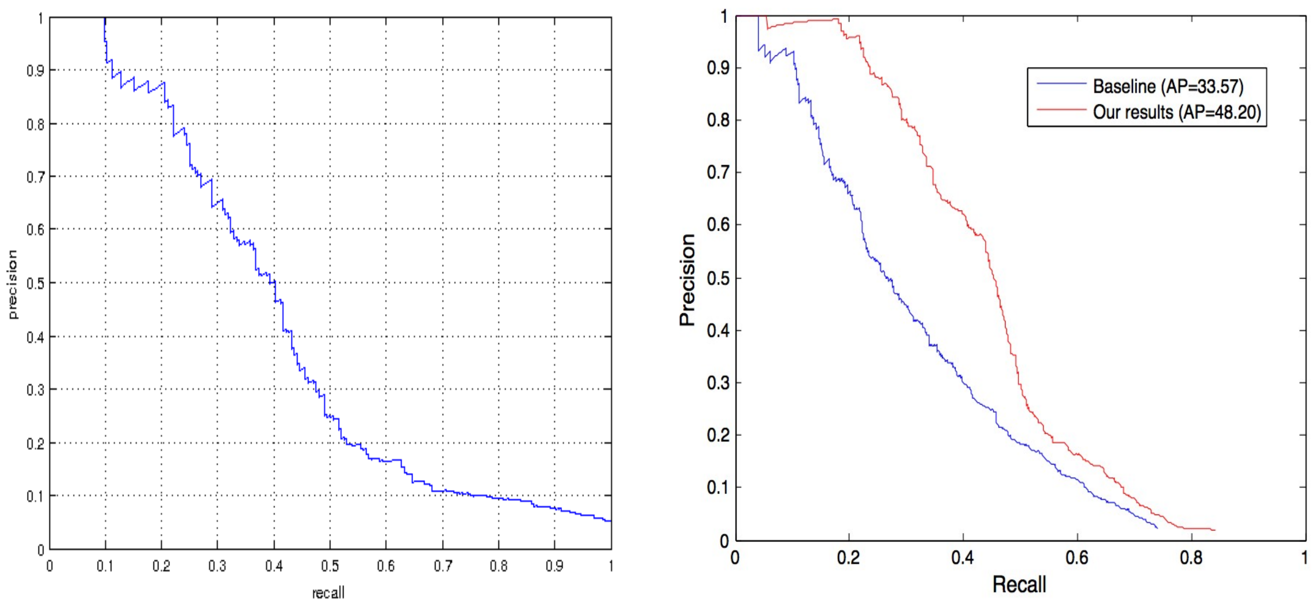
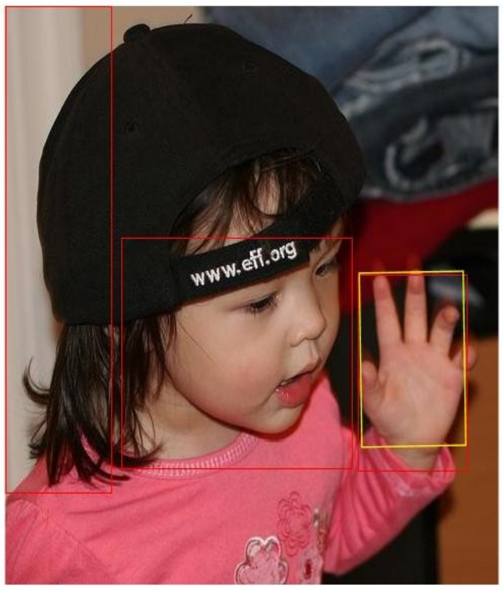
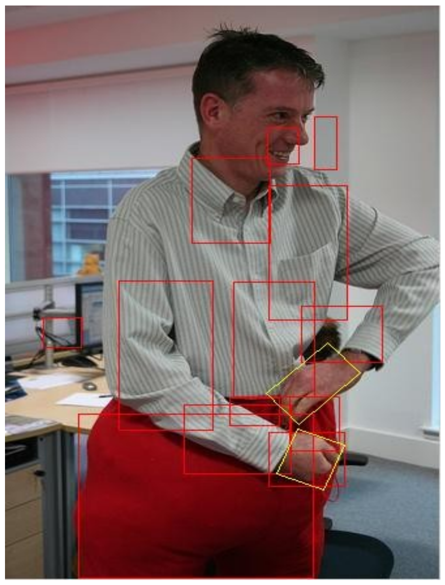
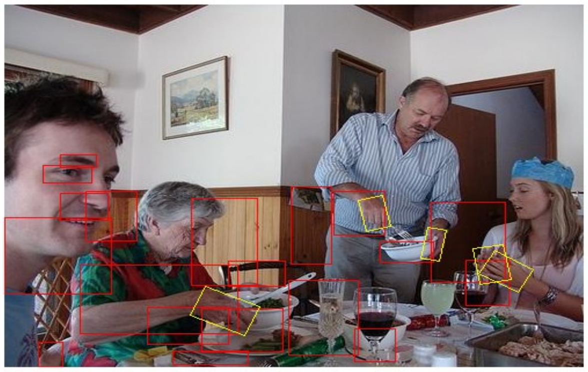
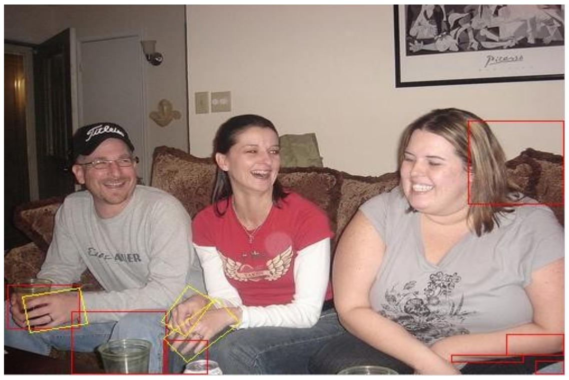


Figure: Our proposal(left), the paper's proposal(right)

As we can see from the graph our proposal's performance lies somewhere between the paper's proposal and the baseline. Here are some results. The rest of it is given in the result folder. The yellow boxes are the ground truth and the reds are our proposal.





Conclusion

We did not apply any post processing in our work. We could improve the performance of our method by doing some post processing method such as, detecting the skin color and can discard those negative windows that does not include the skin color. Also we could improve the performance of our detector by retraining the SVM model using the hard negatives generated after the detection phase instead of just the training phase. We see a lot of future potential of this proposed method for hand detection.