Using Local Steering Kernels to Detect People in Videos

Arthur Louis Alaniz II, Christina Marianne G. Mantaring
Department of Electrical Engineering, Stanford University
{aalaniz, cmgmant} @stanford.edu

***Abstract*—Locally adaptive regression kernels (LARK) can be used as effective descriptors of generic objects, and have been used in object detection algorithms that do not require prior training [1]. This paper demonstrates how these kernels can be applied to the problem of people detection, taking into account the fact that a person can appear in a wide variety of poses, and can come in different shapes and sizes.**

I. Introduction

BEING able to automatically detect people in videos is the first step in a wide variety of tracking applications, which are in turn used in video surveillance systems, traffic monitoring, and human animation, among others. One way of doing people detection is through motion segmentation, where moving objects are identified and separated from non-moving ones. However, people may not necessarily be moving all the time. Moreover, in scenes with a lot of clutter, it can be difficult to differentiate moving people from other moving objects, like cars. Another way of doing people detection is through face detection. Unfortunately, more often than not, people's faces in videos can be partially or completely obscured, which leads to a decrease in the robustness of this kind of system. Finally, a third way of tackling this problem is by using external sensors, such as those on motion capture suits. Still, this is only really practical for closed and controlled environments, as you cannot expect every person to wear a sensor.

One effective way of approaching this problem would be to create a visual template of the person, and then use feature matching algorithms to search for this template in a scene. There has been a lot of recent work on doing visual recognition using only a single query image, and this paper makes use of the generic object detection algorithm proposed by Seo and Milanfar [1].

II. Previous Work

A. Kernel Regression

In kernel regression, regression functions are used to model picture data. In particular, we can represent the measured pixel data as

$$y_i = z(x_i) + \varepsilon_i, \quad x_i = \begin{bmatrix} x_{1i} \\ x_{2i} \end{bmatrix}, \quad i = 1, 2, \ldots, P$$

where the y_i's are the measured pixel data, x_i's represent the pixel coordinates, $z(\cdot)$ is our regression function, the ε_i's are zero-mean IID noise values, and P is the total number of pixels. If we assume that $z(\cdot)$ is locally smooth to some order N, then one method for estimating the function at a point x would be to expand it into the N-term Taylor series [2]:

$$z(x_i) \approx z(x) + z\prime(x)(x_i - x) + \frac{1}{2!}z\prime\prime(x)(x_i - x)^2 + \ldots + \frac{1}{N!}z^{(N)}(x)(x_i - x)^N$$

$$z(x_i) \approx \beta_0 + \{\nabla z(x)\}^T (x_i - x) + \frac{1}{2!}(x_i - x)^T \times \{\mathcal{H}z(x)\}(x_i - x) + \ldots$$

$$\mathcal{H} = Hessian$$

Since the Hessian is symmetric, and using the half-vectorization operator, $vech(\cdot)$[3], this simplifies to [2]

$$z(x_i) \approx \beta_0 + \beta_1^T (x_i - x) + \beta_2^T vech((x_i - x)^T (x_i - x)) + \ldots$$

Estimating the parameters $\{\beta_i\}$ would now allow us to estimate our regression function.

Since we assumed that the data was locally smooth, we need to be able to weight the data such that pixels which are close by have a larger weight than pixels which are far away. To do this, we use kernel functions $K(\cdot)$ [4], which can be any function that achieves our desired penalty characteristics.

To solve for our parameters, we now have [2]

$$\{\beta_i\} = \arg \min_{\{\beta_i\}} \sum_{i=1}^{P} (y_i - (\beta_0 + \beta_1^T (x_i - x) + \beta_2^T vech((x_i - x)^T (x_i - x)) + \ldots))^2 K_H(x_i - x)$$

where

$$K_H(z) = \frac{1}{\det(H)} K(H^{-1}z)$$

$$H \, \epsilon \, \mathbb{R}^2 (smoothing\, matrix)$$

B. Locally Adaptive Kernel Regression

In kernel regression, we made use of spatial differences to weigh the input values. In locally-adaptive kernel regression, we not only make use of spatial differences, but also the difference in data (pixel gradients). In particular, in steering

kernel regression [2], this is done by setting the smoothing matrix H_i to be

$$H_i = hC_i^{\frac{1}{2}}$$

where h is a global smoothing parameter and C_i is the covariance matrix at the ith pixel, An estimate of this covariance matrix can be obtained using the following formula [2]:

$$\hat{C}_i \approx \begin{bmatrix} \sum_{x_j \in w_i} f_x(x_j) f_x(x_j) & \sum_{x_j \in w_i} f_x(x_j) f_y(x_j) \\ \sum_{x_j \in w_i} f_y(x_j) f_x(x_j) & \sum_{x_j \in w_i} f_y(x_j) f_y(x_j) \end{bmatrix}$$

where f_x and f_y are the derivatives along the x and y directions, and w_i is a window surrounding the pixel.

If we choose our kernel function to be a Gaussian kernel, then the local steering kernel (LSK) at a pixel x_i will now be given by

$$K(x_i - x;\, H_i) = \frac{\sqrt{\det(C_i)}}{2\pi h^2} \exp(-\frac{(x_i - x)^T C_i (x_i - x)}{2h^2})$$

Because the smoothing matrix is now a function of the local pixel data (represented by the covariance matrix), this has the effect of spreading the kernel along local edges [2]. Figure 1 shows how a Gaussian kernel adapts to the image data inside the red box.

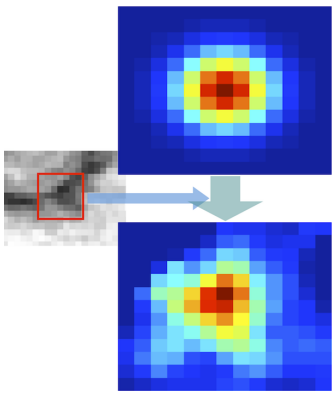

Figure 1. Steering Kernel

C. Object Detection Using Local Steering Kernels

Local steering kernels represent the local structures in images, and give us a measure of local pixel similarities. Given a query image Q, target image T, overall threshold τ_0, and a window size P^2, the generic object detection algorithm proposed by [1] involves the following:

First, the LSKs for the (grayscale) target and query images must be computed. Let these be denoted by $K_Q(x_i - x;\, H_i)$ and $K_T^j(x_i - x;\, H_i)$, where the subscripts Q and T denote the LSKs for the query and target, respectively, and the superscript j denotes that the kernels were computed for the j^{th} patch in T that is the same size as Q.

Now, the LSKs are too dense to use as effective descriptors, and so the next step would be to reduce the dimensionality of these vectors. Before this, however, we first need to normalize our data. Normalization of $K_Q(x_i - x;\, H_i)$ and $K_T^j(x_i - x;\, H_i)$ is given by the following formulas [1]:

$$W_Q(x_i - x) = \frac{K_Q(x_i - x;\, H_i)}{\sum_{l=1}^{P^2} K_Q(x_l - x;\, H_l)}$$

$$W_T^j(x_i - x) = \frac{K_T^j(x_i - x;\, H_i)}{\sum_{l=1}^{P^2} K_T^j(x_l - x;\, H_l)}$$

Let W_Q and W_T^j denote the collection of normalized LSKs for all the x_i's in the query, and all the x_i's in the j^{th} patch in the target, respectively. After normalization, Principal Component Analysis (PCA) [5] can then be used to reduce the dimensionality. Applying PCA to W_Q and extracting the top d eigenimages gives us the collection of eigenimages, F_Q, and the projection space A_Q that was used to obtain these eigenimages. We then project W_T^j onto A_Q to obtain F_T, which is a a collection of eigenimages for the target that are in the same space as that of the query.

With these reduced descriptors, we can now compute the similarity between two patches. For this, the Cosine Similarity measure is used. The similarity of the j^{th} patch in the target to the query is given by

$$\rho_j = < \frac{F_Q}{\|F_Q\|}, \frac{F_T}{\|F_T\|} >$$

From this measure, we generate the resemblance map by calculating the resemblance

$$f(\rho_j) = \frac{\rho_j^2}{1 - \rho_j^2}$$

for all patches in the target.

Finally, significance tests and non-maximum suppression are applied to find the objects. First, the resemblance map is thresholded by the overall threshold τ_0 (ideally set to 0.96; see [1] for details), to determine if there are any objects present in the target. If no values are above τ_0, then no objects are present. Then, the resemblance map is thresholded by a second threshold, τ, which is extracted from the PDF of $f(\rho_j)$ and is set so that only 1% of the resemblance values are above it. This gives a 99% confidence level in the produced data. The last step is to apply non-maximum suppression to find the locations of the objects.

III. Methodology and Results

A. Person Detection using the Generic Object Detection Framework

The first thing we did was to implement the generic object detection algorithm that was proposed in [1] and discussed in Section 2.3. Arbitrary query and target images were used, and the main purpose of this first step was just a sanity check as to how well the algorithm worked for detecting people. Figure 2 shows our query and target images, and the intermediate results we obtained as we stepped through the algorithm.

If our query and target images are the same size, then only one resemblance value is produced, and the significance tests can be replaced with a single comparison to some

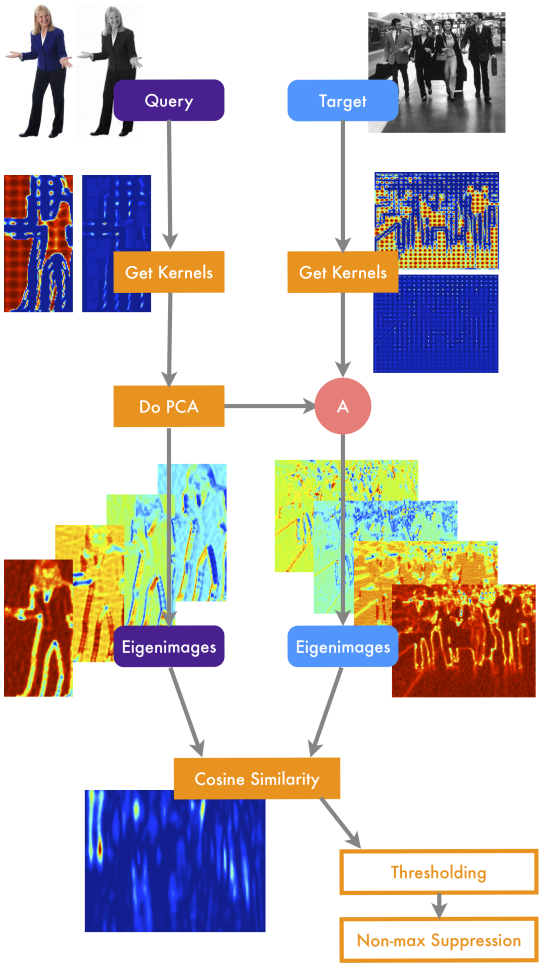

Figure 2. Generic Object Detection Using LARK

predetermined threshold to determine whether the resemblance value is good enough to classify the target. From hereon, we will refer to this special case of the algorithm as LARK Object Detection.

We quickly realized that, because people came in a wide variety of sizes, shapes, and forms, no single query image could capture all the information that could be used to describe a person. For instance, a person whose arms and feet were spread out might end up looking more like a starfish than a person who was standing straight. Another thing we realized was that, because of this variety of forms, the significance tests that the paper had proposed would not hold for our case. Determining the thresholds to be used seemed like a hit-and-miss process, and no value seemed to work for all images. Finally, a third realization was that PCA extracted information with the most variance. However, this did not necessarily mean that it extracted the vectors that identified people. If the query image had a background filled with detail, this information was likely to appear in the eigenimages as well.

B. Query Image Generation and Threshold Determination

To mitigate these effects, two methods were used: First, instead of using a single query image, we proposed to "train" query images from given pedestrian datasets. Generating queries in this manner would allow us to capture more generic human features. The second method would be to use logistic regression to properly determine the threshold that we needed to use in order to classify patches as either a person or not a person.

Our proposed method for generating a query image is as follows: Given a training set composed of images of humans and non-humans, we first take the LSKs of all the images of humans, as well as the LSKs of all these images mirrored horizontally. Mirroring removes the bias introduced by right- and left-facing images. Once we have obtained these kernels, we average them out. Averaging helps remove the background details that we do not want to appear in our feature vectors, and it also unifies the many different poses that are possible. In addition, we average the kernels and not the images themselves because the LSKs have the nice property of being invariant to noise and illumination changes.

After we have our query, we can now use it to do LARK Object Detection, using a second training set as the target images. Once we have our output resemblance values, we use logistic regression on these resemblance values (instead of thresholding), obtaining a training error for the second training set.

Figure 3 illustrates this methodology. The output of logistic regression is a weight vector that we can use for testing. To test, we simply take the generated query image, run LARK Object Detection, and then use the previous weights to determine how to classify the output.

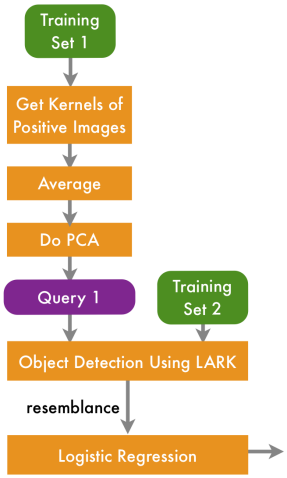

Figure 3. Initial Algorithm for Generating a Query Image and Performing Person Detection

We used the Daimler Pedestrian Classification Database [6] to do our training and testing. The database consists of 4,000 unique images each of pedestrians and non-pedestrians, and each image is an 18x36 grayscale image. In addition, each image is mirrored and shifted by a few pixels, to obtain 24,000 total images each.

Using this database, we created five subsets of images, where each subset was composed of 800 unique images. We ran our initial algorithm using two of these subsets (one to

produce the query image and one to train on) while varying the different parameters (eg. window size, number of PCA dimensions, global smoothing value, etc). This allowed us to obtain the values that would maximize the accuracy vs precision value, where accuracy and precision are given by

$$accuracy = \frac{true\,positives + true\,negatives}{total\,training\,samples}$$

$$precision = \frac{true\,positives}{total\,positives}$$

The last parameter that we had to determine was the bias parameter used in logistic regression. The accuracy vs. precision curve generated by varying this bias parameter is shown in Figure 4.

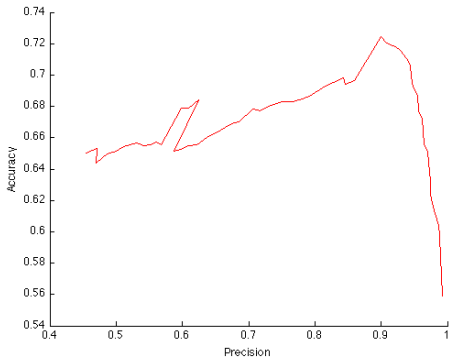

Figure 4. Accuracy vs Precision for Varying Bias Values

C. Chained Query Image Generation

Despite having a more general query image, we found that our results, although better than before, were still unsatisfying. Thus, we proposed to chain our initial algorithm, creating multiple query images out of the positive images that our classifier failed to classify, and then using logistic regression on the multiple queries. Figure 5 shows a version of this iterative algorithm that has three stages.

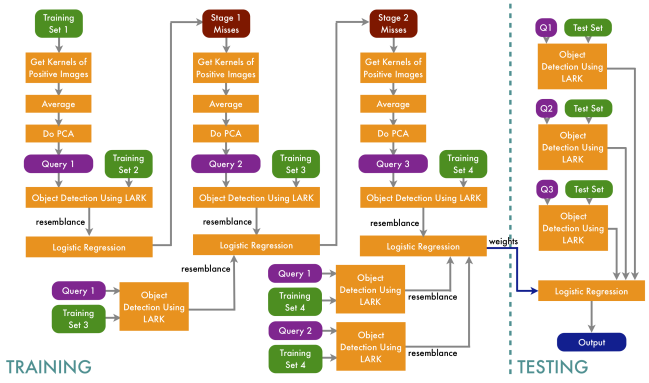

Figure 5. 3-Stage Algorithm for Query Generation and Person Detection

The query generated in the first stage is meant to capture the most generic information that can be used to identify persons. However, it will be unable to identify more unique poses or forms. What we do, then, is take all the positive images that Query 1 failed to classify in Training Set 2, and use these to generate a second query image. This second query will now have information that contains more details that the first query, since it is averaging over a smaller subset of images. These two queries are then used to perform classification on Training Set 3. Using two queries will produce two resemblance values, and logistic regression is used to classify the outputs. Again, the misclassified positive images are used to create Query 3, which will contain even more details than the previous two queries. Logistic regression is used to determine the final weight vector, which, along with the three generated queries, can now be used for testing.

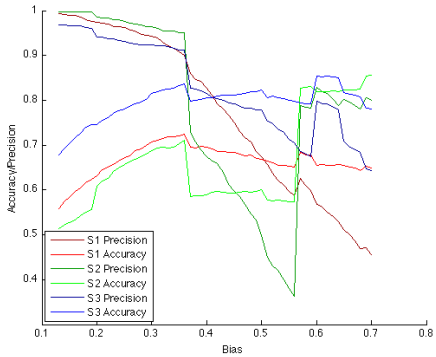

Figure 6. Accuracy and Precision vs Varying Bias

Figure 6 shows the accuracy and precision for obtained after testing with 1, 2, and 3 stages, vs the bias value used in logistic regression (where all instances of logistic regression use the same bias). Note that after $bias \approx 0.36$, both the accuracy and precision suddenly drop. This is due to the fact that the bias value used in Stage 1 greatly affects the output of the entire algorithm. In our case, making the bias value higher that 0.36 suddenly prevented a whole bunch of positive images from being classified. As a results, subsequent query images were more generic than we wanted, which made the performance of the algorithm suffer.

Holding the bias for Stage 1 constant at 0.36, we then repeated the same experiment, and obtained the plots shown in Figure 7. Note that these look much smoother than the previous plots obtained.

After consolidating this information into an Accuracy vs Precision plot, we obtain Figure 8.

Clearly, as we increase the number of stages, our overall test accuracy and precision increases. Our final figures after running the 3-stage algorithm were an accuracy of around 87% and a precision of around 88%.

D. Computational Complexity

From the previous section, we showed that increasing the number of stages increases the performance of the system. Unfortunately, this also results in increasing the number of computations needed, which makes the algorithm run much slower.

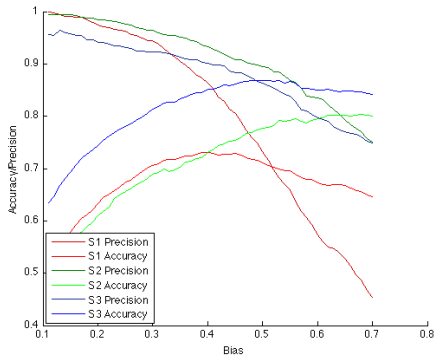

Figure 7. Accuracy and Precision vs Varying Bias, with first stage bias fixed at 0.36

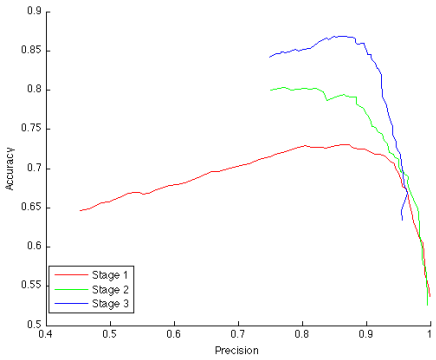

Figure 8. Final Accuracy vs Precision Plot

One disadvantage to using LARK as our base framework is that it is slow. The main bottleneck in our algorithm was in computing the LSKs, which involved several linear filters, as well as matrix multiplications.

Another bottleneck was memory. To generate the query images, we had to store the kernels of all positive training samples in memory so that they could be averaged later on (We found that the optimal window size to use was 9x9, and so each pixel would get converted into an 81-dimension vector). Thus, the number of training samples per stage was bounded by the number of kernels which we could store in memory. Increasing the number of samples beyond a certain number resulted in a much slower algorithm.

IV. Conclusion

Using a training-free generic object detection framework to detect persons is hard, because people come in many different forms, shapes, and sizes. Thus, no single query image can accurately describe each and every person. However, we can generate multiple query images from training sets, and then use logistic regression to unify the results produced by these multiple queries. We showed that chaining our initial algorithm increased both accuracy and precision, and adding more stages to this chain resulted in better performance for our classifier.

V. Future Work

As of now, our algorithm is a classifier that can determine whether image patches are human or non-human. To extend this so that we can detect people in bigger target images, we simply need to classify overlapping patches in the target as human or non-human, and then run non-maximal suppression to remove redundant detections.

Another line of work would be to extend our algorithm to be scale and rotation invariant. The former can be done by building an image pyramid consisting of target images that have been upsampled and downsampled, and then running the detection algorithm on the entire pyramid. Then, to make it rotation invariant, we can rotate the query image by varying degrees, and then run the detection algorithm. Note that since we are detecting people, it is highly unlikely to find rotated people (except maybe if a person is lying down, as compared to standing up), so rotation invariance isn't as important as scale invariance.

Finally, since we want to be doing this for video, then we can add a tracking algorithm so that we do not have to redetect and reclassify patches for every frame. Once a person has been detected, then we can use a method like optical flow to track that person in the scene.

References

[1] Hae Jong Seo, Peyman Milanfar, "Training-Free, Generic Object Detection Using Locally Adaptive Regression Kernels," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 32, no. 9, pp. 1688-1704, Aug. 2010, doi:10.1109/TPAMI.2009.153

[2] H. Takeda, S. Farsiu, and P. Milanfar, "Kernel Regression for Image Processing and Reconstruction," IEEE Transactions on Image Processing, vol. 16, no. 2, pp. 349-366, Feb. 2007.

[3] D. Ruppert and M. P. Wand, "Multivariate locally weighted least squares regression," The Annals of Statistics, vol. 22, no. 3, pp. 1346–1370, September 1994

[4] M. P. Wand and M. C. Jones, Kernel Smoothing, ser. Monographs on Statistics and Applied Probability. London; New York: Chapman and Hall, 1995.

[5] Jolliffe, I. T. (1986). Principal Component Analysis. Springer-Verlag. pp. 487. doi:10.1007/b98835. ISBN 978-0-387-95442-4

[6] S. Munder and D. M. Gavrila. "An Experimental Study on Pedestrian Classification". IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 28, no. 11, pp.1863-1868, Nov. 2006.

[7] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. LIBLINEAR: A library for large linear classification. Journal of Machine Learning Research 9(2008), 1871-1874.

[8] Hae Jong Seo; Milanfar, P.; , "Using local regression kernels for statistical object detection," Image Processing, 2008. ICIP 2008. 15th IEEE International Conference on , vol., no., pp.2380-2383, 12-15 Oct. 2008 doi: 10.1109/ICIP.2008.4712271