Classification of Images with and without the Presence of Pedestrians

WENXUN PENG
University of New South Wales

***Abstract*—This paper uses histogram of oriented gradients descriptor as the extracted feature descriptor, and then sends the extracted feature vector to the support-vector machine classifier for classification to find the optimal hyperplane to determine whether there is a pedestrian or not in a single image. Pedestrians are firstly detected using the human detector proposed by Dalal and Triggs which is called HOG descriptors and then classification using a linear SVM.**

Keywords—Single Pedestrian classification; histogram of oriented gradients descriptor; Support-vector machine

I. INTRODUCTION AND BACKGROUND

Pedestrians in tracking videos have many important computer vision applications, including visual surveillance, personnel recognition, intelligent environments, and human-computer interaction [1]. Pedestrian visual tracking includes two main phases: pedestrian detection and association of detected pedestrian windows [2]. In the individual component, all we need to do is to use python to implement a method to identify whether there is one or more pedestrians on a single image. Our data set used in here is published by CSIRO [3] containing a large collection of images with (positive) and without (negative) pedestrians in them. There are three parts in our data set:

1. Training_Data: This folder includes a subset of training data from the CSIRO database, including 3 positive sets (train_positive_A, train_positive_B and train_positive_C) and one negative set (train_negative_A);

2. Test_Data: This folder includes a subset of validation data from CSIRO database, including one positive set (test_positive) and one negative set (test_negative);

3. License.txt: License file

To be more specific, all the images used in the dataset are files in ".pnm" format, and all pictures have a resolution of 64 * 80.

Due to changes in human appearance, body posture, and perspective, detecting pedestrians in images is a challenging process [5]. In fact, many visual human detection methods have been proposed in the literature to solve these problems. Wu et al. [6] use a human body part detector to detect multiple people in a chaotic scene. Their body parts and whole body detectors are trained using nested cascade methods using edge features that reflect edge strength and direction. Their method can overcome partial occlusion, but it requires high-resolution images. Since mentioned before, all the images in the training set are low-resolution images, for low-resolution images, Dalal and Triggs [4] proposed a method using directional gradient histogram (HOG) descriptor and linear support vector machine (SVM) classifier to find pedestrians. Their method has stable and effective results when processing low resolution images. Therefore, I pretend to use HOG as a descriptor and SVM as a classifier in this project.

II. METHOD

A. Histogram of Oriented Gradients Descriptor

The concept of feature descriptor should be introduced before HOG (Histogram of Oriented Gradients) is introduced. A feature descriptor is a representation of an image or an image patch that simplifies the image by extracting useful information and throwing away extraneous information. The HOG feature detection algorithm was first proposed by Dalal and Triggs [4]. It is an image descriptor for detecting human target detection, which is used to characterize the local gradient direction and gradient intensity distribution characteristics of the image. The main idea is that when the specific position of the edge is unknown, the distribution of the edge direction can also well represent the outline of the pedestrian target. There are 4 main steps in HOG:

Step 1: Normalizing the image prior to description.

Typically, a feature descriptor converts an image of size width * height * 3 (channels) to a feature vector/array of length n. In the case of the HOG feature descriptor, the input image is of size 64 * 128 * 3 and the output feature vector is of length 3780. Of course, an image may be of any size. Typically patches at multiple scales are analyzed at many image locations. The only constraint is that the patches being analyzed have a fixed aspect ratio. In our case, the patches need to have an aspect ratio of 1:2. For example, they can be 64 * 128, 128×256, or 1000×2000 but not 64 * 80. As mentioned before, all of training set images are 64 * 80, so all of the images should be resized to 64 * 128 in the step 1. In addition, due to factors such as the image collection environment and devices, the collected image effect may not be very good, and it is prone to false detection or missed detection. Therefore, there are two main methods to improve performance of the HOG descriptor.

One is graying the image, which is converting the color images (have red, green and blue channel) into a single-channel grayscale image by using this formula:

$$gray = 0.3 * red + 0.59 * green + 0.11 * blue$$

Another is Gamma/power law normalization: In this case, we take the log(p) of each pixel p in the input image. However, as Dalal and Triggs demonstrated, this approach is perhaps an "over-correction" and hurts performance [4].

Also, there are some other normalization methods such as Square-root normalization (we take the sqrt(p) of each pixel p in the input image.). However, in most cases, it's best to start with no normalization. And in my result, I found it is good enough when starting with no normalization except resizing the images.

Step 2: Gradient computation

After normalized, the gradient and gradient direction of imaged are obtained. The calculations are performed in the

horizontal and vertical directions, respectively. Assuming that the gradient of the pixel (x, y) is currently being calculated, its horizontal gradient value Gx and vertical gradient value Gy are:

. Gx = I * Dx and Gy = I * Dy

where I is the input image, Dx is our filter in the x-direction, and Dy is our filter in the y-direction. Therefore, the final gradient magnitude representation of the images:

$$|G| = \sqrt{G_x^2 + G_y^2}$$

Finally, the orientation of the gradient for each pixel in the input image can then be computed by:

$$\theta = \arctan\left(\frac{G_x}{G_y}\right)$$

Given both |G| and θ, we can now compute a histogram of oriented gradients, where the bin of the histogram is based on θ and the contribution or weight added to a given bin of the histogram is based on |G|.

Step 3: Weighted votes in each cell

Now that we have our gradient magnitude and orientation representations, we need to divide our image up into cells and blocks. A "cell" is a rectangular region defined by the number of pixels that belong in each cell. For example, the input images is 64 * 128 image and defined our pixels per cell as 8 * 16, we would thus have 8 * 8 = 64 cells.

And then, for each of the cells in the image, we need to construct a histogram of oriented gradients using our gradient magnitude |G| and orientation θ mentioned above. But before we construct this histogram, we need to define our number of orientations. The number of orientations control the number of bins in the resulting histogram. The gradient angle is either within the range [0, 180] (unsigned) or [0, 360] (signed). In general, it's preferable to use unsigned gradients in the range [0, 180] with orientations somewhere in the range [9, 12].

Finally, each pixel contributes a weighted vote to the histogram -- the weight of the vote is simply the gradient magnitude |G| at the given pixel.

Step 4: Contrast normalization over blocks

To account for changes in illumination and contrast, we can normalize the gradient values locally. This requires grouping the "cells" together into larger, connecting "blocks". It is common for these blocks to overlap, meaning that each cell contributes to the final feature vector more than once. Again, the number of blocks is rectangular; however, our units are the cells not the pixels.

For each of the cells in the current block we concatenate their corresponding gradient histograms, followed by either L1 or L2 (prefer) normalizing the entire concatenated feature vector. Again, performing this type of normalization implies that each of the cells will be represented in the final feature vector multiple times but normalized by a different value. While this multi-representation is redundant and wasteful of space, it actually increases performance of the descriptor.

Finally, after all blocks are normalized, we take the resulting histograms, concatenate them, and treat them as our final feature vector.

B. *Support-vector Machine Classifier*

It can be known from the above that the HOG feature vector of the entire image has been obtained here. The HOG feature vector extracted using a series of samples is sent to the SVM classifier for training to find the optimal hyperplane for classification.

In practical applications, several small windows are often taken from a picture for feature extraction. In this way, windows with targets can be obtained as positive samples, and windows without targets can be used as negative samples. The positive and negative samples are sent to the SVM training, and a sliding window of the same size is taken during recognition, and the trained SVM is used to determine whether the sliding window contains the target.

Finally, I will briefly introduce the concept of support-vector machines (SVM). SVM is supervised learning model with associated learning algorithms that analyze data used for classification and regression analysis. Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier (although methods such as Platt scaling exist to use SVM in a probabilistic classification setting). An SVM model is a representation of the examples as points in space, mapped so that the examples of the separate categories are divided by a clear gap that is as wide as possible. New examples are then mapped into that same space and predicted to belong to a category based on the side of the gap on which they fall.

III. EXPERIMENT

A. *Experimental Setup*

Dependencies: Python3, Opencv, Numpy

Command format: *$ python3 Individual.py*

File directory: The data set folder "Individual_Component" should be the same folder directory with the Individual.py. The example is shown below:

After run Individual.py, a file called "svm.xml" will be generated, this file is used to store the trained SVM classifier.

Param set: For "cv2.HOGDescriptor", we adjust the winSize to (64, 128). The winSize means the input images' size (pixel * pixel). And all the training set images are resized to (64, 128) as I mentioned before. And for other param, such as blockSize, blockStride, cellSize, nbins, etc., using the default value is good enough. The corresponding default values are (16, 16), (8, 8), (8, 8) and 9 respectively. Although these values may affect the performance, it is good enough to use these default vales. I will discuss the effect in the next part. In addition, I used "hog.detectMultiScale" to detect positive images. This function returns two parameters. One parameter is composed of pixel coordinate points, that is (x, y, w, h). If we draw a rectangle on the image, the starting coordinate point (the upper left coordinate point) is (x, y). And the coordinate point at the end (coordinate point in the bottom right corner) is (x + w, y + h). In this rectangle, the pedestrian is detected. If no pedestrian is detected, this function will return a null value. Also, this function can be used to detected more other size images, not just in the test data set (all the images in the test data set have 64 *80 size.)

For " cv2.ml.SVM_create()", first, we should set the SVM_LINEAR since we used the SVM linear model in our project. As for other params, we should set the SVM formulation as EPS_SVR. And the EPS_SVR is class support vector regression machine. The distance between the feature vector in the training set and the fitted hyperplane needs to be less than p. The outlier penalty factor C is used. And if we used the default formulation "C_SVC", we will get poor performance since the EPS_SVR is more suitable than default formulation when detecting object is pedestrian. Other params are set default.

B. Evaluation Methods

Here, I used two methods to evaluate the performance of the algorithm. (We can consider images with pedestrians as positive and without pedestrians as negative to be calculated)

Accuracy: Accuracy is defined as: the ratio of the number of samples that the classifier correctly classifies to the total number of samples for a given test data set. More specifically, in this project, the test data sets have two types, positive and negative (with and without pedestrian), the accuracy is using the sum of the correct classified images (positive for positive images and negative for negative images) to be divided by the total number of samples in test data sets.

F1 score: However, Accuracy is not always effective, for example, Google crawled 100 pages of argcv, and it has a total of 10,000,000 pages in the index. Randomly selected a page and classified, is this a page of argcv? If I judge my work by accuracy, then I will judge all pages as "pages that are not argcv", because I'm very efficient (return false, one sentence), and accuracy has reached 99.999% (9,999,900 / 10,000,000). Therefore, we need another method to evaluate the performance which is called F1 score.

Before introducing F1 score, we need to introduce the precision and recall. The definition of **recall** is the number of true positives divided by the number of true positives plus the number of false negatives. And the definition of **precision** is the number of true positives divided by the number of true positives plus the number of false positives. However, **precision** and **recall** affect each other. In general, a good performance should have both high precision and high recall. However, in fact, if the precision is high, the recall will be low, and vice versa. Therefore, we can use a combination method which is called F1 (or F-score) to measure. Below showed the concept of TP, FP, FN and TN.

		Actual	
		Positive	Negative
Predicted	Positive	True Positive	False Positive
	Negative	False Negative	True Negative

Thus, the formulas of Precision, Recall and F1 score:

Precision: P = TP / TP + FP

Recall: R = TP / TP + FN

F1 score: F1 = 2 * P * R / (P + R)

IV. Results and Discussion

From below, left image is the original image extracted from positive test data set. After using " hog.detectMultiScale" and draw the rectangle on the original image, we can get the right image, which can be detected positive (with pedestrians). Also, the right image is resized to (64, 128) and original is (64, 80). We can find that below.

In fact, opencv provides a built-in pedestrian classifier. The upper picture below uses the results obtained by using the built-in classifier. The downside picture shows the results obtained by using the SVM classifier written by myself.

```
Test positive files test positive/total is 1458/3457.
Test negative files test positive/total is 2/6000.
Precision is 0.999
Recall is 0.422
F1 value is 0.593
Accuracy is 0.788
```

```
Test positive files test positive/total is 3446/3457.
Test negative files test positive/total is 814/6000.
Precision is 0.809
Recall is 0.997
F1 value is 0.893
Accuracy is 0.913
```

As mentioned earlier, if we change the parameters in cv2.HOGDescriptor (do not use the default value and randomly take some blockSize, blockStride and cellSize values) or use the default formulation "C_SVC" in the SVM classifier instead of "EPS-SVR", the obtained Accuracy and F1-score are (0.558, 0.423) and (0.454, 0.294) respectively.

In addition, in SVM training, the more pictures used in the training SVM, the better performance we can get. (More helpful for hyperplane classification). So I tried to use all the positive images (about 20,000 images) from the training set, and the corresponding number of negative images, and I got the following result.

```
Test positive files test positive/total is 3438/3457.
Test negative files test positive/total is 573/6000.
Precision is 0.857
Recall is 0.995
F1 value is 0.921
Accuracy is 0.937
```

Finally, the final training results obtained are quite satisfactory, but there are also obvious disadvantages: 1. The running time is long; 2. The amount of data required for the training set is large, 3. When Accuracy and F1-score reach above 90%, we need to consider whether this model is overfitting or not.

References

[1] Z. Jiang, D. Q. Huynh, W. Moran, S. Challa and N. Spadaccini, "Multiple Pedestrian Tracking Using Colour and Motion Models," 2010 International Conference on Digital Image Computing: Techniques and Applications, Sydney, NSW, 2010, pp. 328-334.

[2] A. Yilmaz, O. Javed, and M. Shah. Object Tracking: A Survey. ACM Computing Surveys, 38(4), Dec 2006.

[3] G. Overett, L. Petersson, N. Brewer, L. Andersson and N. Pettersson, "A new pedestrian dataset for supervised learning," 2008 IEEE Intelligent Vehicles Symposium, Eindhoven, 2008, pp. 373-378.

[4] N. Dalal and B. Triggs. Histograms of Oriented Gradients for Human Detection. In IEEE Computer Society Conference on Computer Vision and Pattern Recognition, volume 1, pages 886–893, Jun 2005.

[5] D. A. Forsyth, O. Arikan, L. Ikemoto, J. O'Brien, and D. Ramanan. Computational Studies of Human Motion: Part 1, Tracking and Motion Synthesis. Foundations and Trend in Computer Graphics and Vision, 1(2/3):77–254, 2005.

[6] B. Wu and R. Nevatia. Detection of Multiple, Partially Occluded Humans in a Single Image by Bayesian Combination of Edgelet Part Detectors. In Tenth IEEE International Conference on Computer Vision, volume 1, pages 90–97, 2005.