Project 1 vision and objects recognition

Christian Dansereau

Abstract—The objective of Project 1 is to explore the various tools found in the context of vision and object recognition. During the project we are face with various problems typical of vision that need to be address and that are discuss in the following pages.

Keywords—Vision, Object recognition, IEEE.

1 Introduction

The objectives of the projects are the following: 1) identify the objects in motion in a scene and extract some meaningful features. 2) classify the nature of the moving objects namely if they are pedestrians or cars. 3) Be able to track the moving objects and there respective trajectories.

1.1 Public code and data

The code and data used in this experiment are available on a GitHub repository at the following URL: https://github.com/cdansereau/vision_or. A IPython Notebook is also provided with most of the figure generation scripts.

2 Method

The project was realized in python with the following libraries: scipy [1], scikit-learn [2], matplotlib [3] and scikit-image [4].

2.1 Identification of objects in motion

The dataset used for classification was composed of 150 images (webcam at https://www.cs.dal.ca/cams/university.jpg) of those we use 30 images to tune the parameters of the feature extraction algorithm. The median of the first 30 images was used to compute the background. Algorithm 1 was use to extract the moving objects from the background and remove artefact of the image. We found that repeting the Algoritm 2 multiple time was improving the mask substantially in order to remove small objects and fuse the relevant ones, resulting in more consistent blobs for each object.

```
Data: rgbImg, background,
Result: imgLabels
im = mean(rgbImg) %Grayscale image;
imDif = im - background;
mask = adaptativeThreshold(imDif);
mask = binaryClosing(mask, square(2));
mask = binaryDilation(mask, square(4));
imgLabels = getLabels(mask);
imgLabels =
removeSmallObj(imgLabels,100);
for i=range(1,5) do
    imgLabels =
    maskCorrection(imgLabels);
end
return imgLabels;
```

Algorithm 1: extractAllLabels

2.2 classification

The features used in the classification are based on resized RGB (Red Green Blue) images 32x32 pixels extracted from the the centroid of the detected object preserving the structure of the image independently of its size and position in the scene. Two examples of the resulting images from Algorithm 3 can be seen in Figure 1.

• *Mr. Dansereau is with the Department of Computer science, University of Montreal, Montreal, CA.*
E-mail: christiandansereau@gmail.com

Manuscript received March 19, 2015

Data: imgLabels
Result: imgLabels
mask = $imgLabels > 0$
binaryDilation(mask, square(10));
mask = putBorderZero(mask);
binaryErosion(mask, square(10));
imgLabels = getLabels(mask);
imgLabels =
removeSmallObj(imgLabels,100);
return imgLabels;
Algorithm 2: maskCorrection

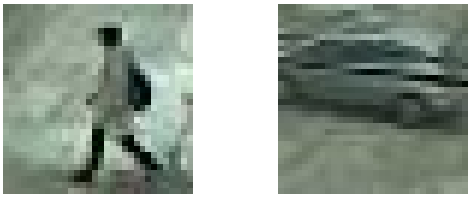
Data: originalImage,
Result: imagePatch
limg = largest side of originalImage;
sqImage = square image limgXlimg;
imagePatch = resample(sqImage);
Algorithm 3: Features extraction

In term of the features used for the classification we evaluated the performance of various features 32x32 RGB, 32x32 gray-scale and 16x16 RGB the performance are marginally different we therefore decided to go with the 32x32 gray-scale features resulting in a 1024 features after gray-scale conversion due to the fact that a large patch may capture in some case more relevant information in some case and it appear that the color component do not contribute drastically to the classification.

We choose to use SVM (Support Vector Machine) [5] as our classifier since it's a fast and robust state of the art classification algorithm with a Gaussian radial basis function kernel (RBF).

Some basic preprocessing was perform on the patch image of 32x32 pixels. We first vectorized the RGB images and removed the mean and variance from each of the features in an attempt to normalize the data.

In order to find the most adapted hyper-parameters (C and Gamma) for our classification problem we choose to use a grid search approach. The grid search parametrised as follow for C ($10^{-}5$ to 10^{2}) and for Gamma ($10^{-}5$ to 10^{3}) The grid search and the final classification were evaluated using a stratified cross-validation due to the unbalanced number of examples available for each class. We also included a parameter in the SVM classifier to account for the unbalance dataset by automatically re-weight each class by the inverse of its frequency.



(a) Pedestrian (b) Car

Fig. 1. 2 features side by side of 32x32 pixels in RGB. a) is a pedestrian and b) a car

2.3 Tracking

For the tracking we are using the information from Part 1 to make our decision of tracking. We are using up to two time points in the past to assess if an object belong to a particular tracked trajectory. We first take the tracking label of the previous frame closest to the current object in a radius of 150 pixel around the object, if a second frame is available we compute the angle of the trajectory of the two vectors to see if we are not turning more that $90\,\mathrm{deg}$ as described in Algorithm 4. The last verification is to make sure that the trackingId is not already attributed to an other object in the current frame, if so we run Algorithm 4 with objectT0 and listObjT2 (in the place of listObjT1) removing the already attributed trackingID. This will resolve problem of crossing when two objects are fused together for one frame.

3 Results

3.1 Part1: feature extraction

From the 150 images analysed the algorithm was able to identify 238 moving objects (93 cars and 145 pedestrians). Figure 2 show an example of moving objects identified by the algorithm and there respective bounding box. We also include an overlay of the actual object detected, has we can see from the car some areas are not completely covered due mainly to light reflection.

Data: objectT0, listObjT1, listObjT2
Result: trackingId
initialization;
trackingId = listObjT1.maxId()+1
smallestDist = Inf;
while *listObjT1 not empty* **do**
  objT1 = listObjT1.nextObj();
  if *objT1 distance is closest and* < 150 **then**
    if *objT2 is available* **then**
      if *trajectory using objT2* $< 90\,\mathrm{deg}$ **then**
        trackingId = objT1.trackingId();
      end
    else
      trackingId = objT1.trackingId();
    end
  end
end

Algorithm 4: Tracking objects

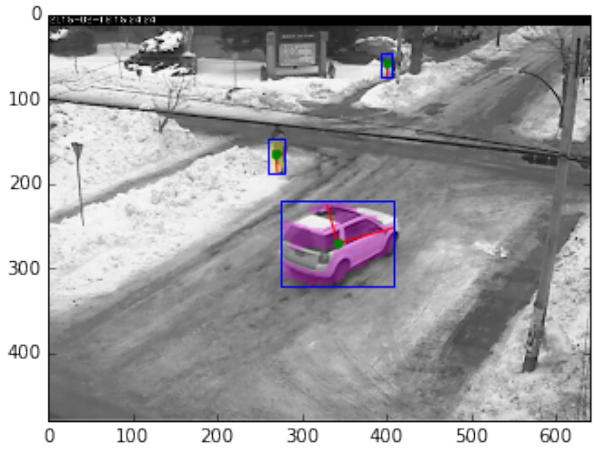


Fig. 2. Features extraction: the bounding box identify the detected moving objects in the scene and some of the feature extracted (angle of the major/minor axis and centroid).

3.2 Part2: classification

The best classifier is obtain from the grid search was $C = 10^1$ and $Gamma = 10^-3$.

We were able to achieve an accuracy of 90% (+/- 0.18) estimated with a 10 fold cross-validation with stratified test and validation sets. Figure 3 show an example of the classification in a scene with 3 moving objects 2 pedestrian (in red) and one car (in yellow).

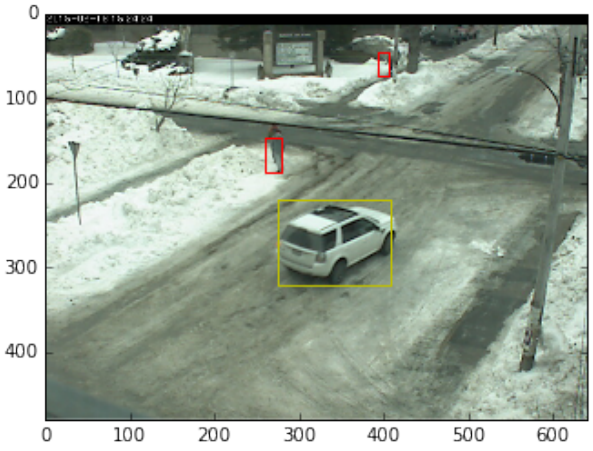


Fig. 3. Classification: the bounding box identify the detected moving objects in the scene, the color represent the classification result (yellow=car, red=pedestrian).

3.3 Part3: tracking

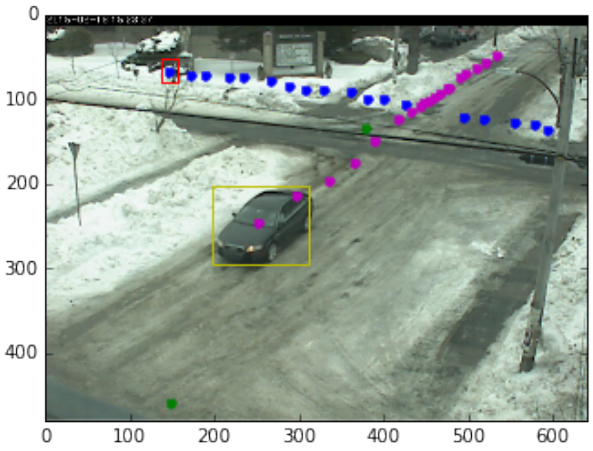


Fig. 4. Final tracking result: the bounding box identification the detected moving objects in the scene, the color represent the classification results (yellow=car, red=pedestrian) and the color dots represent the unique path of a particular moving object in the scene.

Figure 4 show a successful tracking example of two moving objects in a scene, in this particular case the two objects cross the path of each other and fuse at the intersection point but the algorithm was able to overpass this difficulty by looking at previous frames and the angle of the trajectories of each object.

4 Conclusion

This project introduce us to the various problematic that one can face in the context of vision and tracking. We were able to achieve the three requirements of the project namely 1) the identification of mobile object and there features, 2) the classification of the object in a two class problem (pedestrian or car) and finally 3) the tracking of the objects across the scene. We also made our software able to identify a trajectory and be able to resolve interference due to two objects crossing path.

An important note relate to the choice of the features for classification, the images use in this project were taken in winter and are not particularly colourful explaining the performance of the 32x32 gray-scale image as equivalent to the RGB one. Event though this choice is acceptable for this particular dataset it will most likely be more appropriate to use a color version for a classifier that would generalized better in other scenery.

References

[1] E. Jones, T. Oliphant, P. Peterson *et al.*, "SciPy: Open source scientific tools for Python," 2001–, [Online; accessed 2015-03-15]. [Online]. Available: http://www.scipy.org/

[2] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[3] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing In Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[4] S. van der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, T. Yu, and the scikit-image contributors, "scikit-image: image processing in Python," *PeerJ*, vol. 2, p. e453, 6 2014. [Online]. Available: http://dx.doi.org/10.7717/peerj.453

[5] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995. [Online]. Available: http://dx.doi.org/10.1023/A%3A1022627411411