Queen's University Belfast

Group Assignment

Technical Report

Title	Pedestrian Detection
Lecturer	Dr Jesús Martínez del Rincón
Module	CSC3061
Group	Group 10
Word Count	6300
Date	Tuesday, 31 March 2020

Name	Student No.	Email
Dongming Li	40215999	dli01@qub.ac.uk
Jamie Thompson	40178456	jthompson83@qub.ac.uk
Dewei Liu	40216004	dliu08@qub.ac.uk

Contents

Introduction

This report outlines the process of building a full pipeline for pedestrian detection. Each section of this report details a stage of the pipeline and the reasoning behind the final choices for each part. Data backed justification for every stage is an important aim for this project; each section has been informed by experimental data, which is explained in each section of the report. Throughout each stage, a priority has been placed on following best practices, with all training of models being done only with the training data, and the evaluation of these models only using the training data to avoid bias, overfitting or a misleading result. The result of this experimental and data-driven approach is a final justified pipeline which can be tested against the testing dataset.

The project source code was developed on GitLab [1].

Image Acquisition

The dataset contains 2003 images labelled as pedestrians and 997 non-pedestrian images for training. For testing, it has 100 images which are a series of frames captured from a moving camera.

Function Definition

The function $src/image\_aquisition/loadImages$ takes a string of directory as input parameters and loads all the images under this directory excluding subdirectories. The return value of the function is a struct containing the following properties

paths

This value is a matrix of strings. Each string is the full file path to an image.

number

This value is a number. It is the total number of images loaded.

images

This value is a 4D matrix. $images(:,:,:,index)$ returns an RGB image in the index.

grayImages

This value is a 3D matrix. $images(:,:,index)$ returns a grayscale image in the index.

nRows

This value is a number. It the height of the images (assuming each image has the same height).

nColumns

This value is a number. It the width of the images (assuming each image has the same width).

Function Usage

The function is called at $src/setEnvir.m$ where pedestrian, non-pedestrian and testing images are loaded respectively.

```
pedestrians = loadImages("../dataset/pedestrian/");
positives = loadImages("../dataset/images/pos/");
negatives = loadImages("../dataset/images/neg/");
```

Data Processing

Image Augmentation

The training image set contains 2003 positives and 997 negatives images. In order to improve the performance of the model in further analysis, it is necessary to increase the variety of training images. Therefore, in this step, flipping horizontally images of all training images will be generated as new a candidate training image set. The current training image set contains 3000 original images and 3000 flipping horizontally images.

Image Pre-processing

For doing further image pre-processing, contrast enhance methods are implemented on current candidate training images. Because in the step of feature generation, the HOG method is applied to generate feature data for training images. The gradient of pixels in each image is a key element of the HOG method, so enhancing the contrast of images can increase the value of the gradient for most of the training images, which can effectively make feature data produced by HOG more significant.

Processed Images Selection

Figure 1 illustrates all processed images of the first pedestrian image.

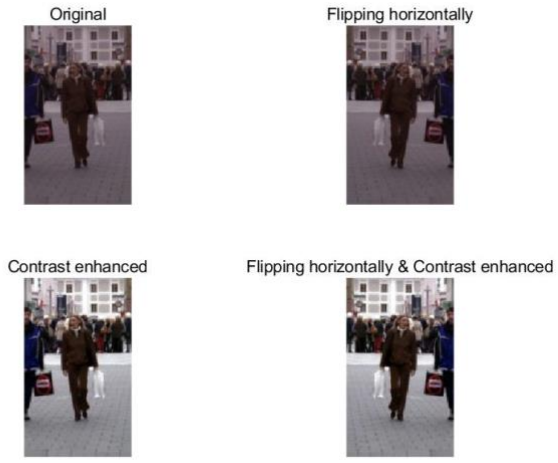

Figure 1, Processed images of the first positive image.

There are 12,000 images in total as candidate training images, if using all of them to generate features and fit models, the size of the training set is too big which will increase the running time and complexity substantially. To address this issue, firstly, in four types of training images, the first 350 images are selected to do a combination, as shown in the table below.

Combination 1	350 original images
Combination 2	350 original images + 350 contrast-enhanced images
Combination 3	350 original images + 350 flipping horizontally images
Combination 4	350 original images + 350 flipping horizontally & contrast-enhanced images

Table 1, Combinations of candidate training images.

For images of four types of combinations, generating features data using HOG method and fitting an SVM classifier on features, the classifier is evaluated by the build-in cross-validation method to check the error rate. For the combination having the lowest error rate, all images of this combination will be selected as the training set. The error rates table is illustrated below.

Combination 1	0.024286
Combination 2	0.001429
Combination 3	0.018571
Combination 4	0.001429

Table 2, Error rate table for combinations of candidate training images.

According to Table 2, also considering promoting the diversity of training images, the images of combination four is selected as the current training set.

In the step, the SVM classifier is in a simple version without testing by parameter optimization. The aim is selecting the most appropriate images as training images. The analysis of implementing the HOG method and SVM classifier are illustrated in the following content.

Demos

Running processed images selection

The following steps show how to select appropriate candidate images as training images:

1. Open MATLAB and set the working directory to `csc3061-1920-g10/src`.
2. Run the script `processedImagesSelection.m`.
3. The script will output its progress to the console.

Visualize processed images

The following steps show how to select appropriate candidate images as training images:

1. Open MATLAB and set the working directory to `csc3061-1920-g10/src`.
2. Run the script `visualizeProcessedImages.m`.

Feature Extraction

Feature Definitions

Histogram of Oriented Gradients

Histogram of Oriented Gradients (HOG) is an effective method to generate useful feature data in people detection. This method counts the direction of gradients that occur in a certain local area of the image and calculates on the grid of evenly spaced cells [2]. Therefore, in the case of images containing people, the gradient values of the junction of the human silhouette and background in the image will be significant. However, in images without people, the gradient values do not have this trend. Based on that, it will be a high probability that there is a considerable difference between the feature data of images with people and images without people.

To avoid overlapping or excluding some pixels in each image, the cell size in the HOG method is set to 12. Because the size of training images is 160×96, 12 is divisible by these two numbers and is not too large to cause loss of information.

Colour Histogram

The Colour Histogram feature is the prevalence of colours in an image, split into three channels. Features for both red, green, and blue channel histograms and hue, saturation, and value histograms were created. Before calculating the colour histograms, the images were quantized to a specified number of colours. The colour histogram calculation takes a tuning parameter of *Quantization Level*, which equates to the number of bins for each colour channel; for example, for the RGB histogram a *Quantization Level* of 3 would result in 3 bins of red, 3 of green, and 3 of blue. After determining the number of colours in the image, the histogram was then normalized to allow comparison between different scales of image.

In order to determine the level of quantization to use, a comparison of error rates when using colour histograms with different levels of quantization was created. Features were calculated for each quantization level (from 1 to 30) for RGB and HSV colour histograms before an error rate was calculated by using 5-fold cross-validated training and testing and being fed into an SVM classifier.

Figure 2 shows the error rates for RGB colour histograms with varying quantization levels. It levels off around quantization level of 7 – with an error rate of 0.41 - at later points in this report, the quantization level for RGB colour histograms will be at 7. Figure 3 shows the error

rates for HSV colour histograms with varying quantization levels. It levels off around quantization level of 7 – with an error rate of 0.41 - at later points in this report, the quantization level for HSV colour histograms will be at 7.

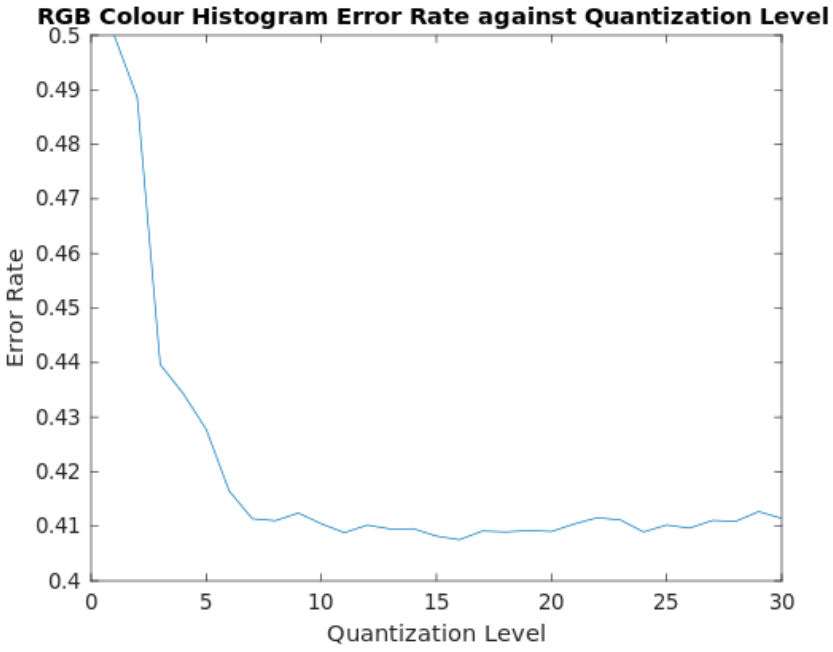

*Figure **2** RGB Colour Histogram Error Rate against Quantization Level*

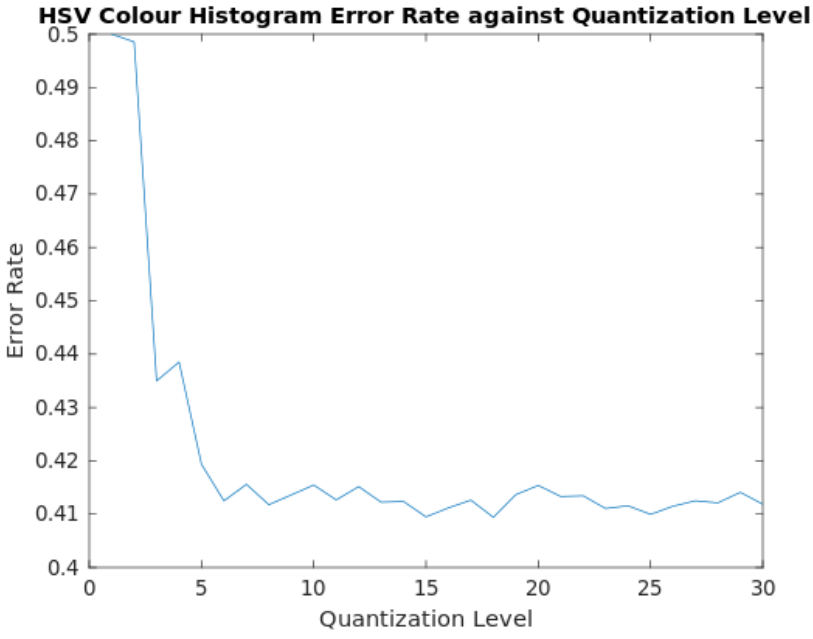

*Figure **3** HSV Colour Histogram Error Rate against Quantization Level*

Flattened Raw Data

The flattened raw data feature is simply taking the raw pixel values and flattening them into a single vector of features. These are split by colour channel (red, green, and blue). Before flattening the raw data, the image needs to be resized to a consistent size – either upscaling or downscaling to fit. This allows comparing different sized images and aspect ratios.

The images are resized to $width \times height$ dimensions. In order to determine which image size to resize to a comparison of dimensions error rates when using the raw data feature was calculated. Features were calculated for each square image size from 1 to 20 (1*1, 2*2, 3*3 etc.), before an error rate was calculated by using 5-fold cross-validated training and testing and being fed into an SVM classifier.

Figure 4 shows the error rates for raw data with varying image resizing. It levels off around quantization level of 16 – with an error rate of 0.35 - at later points in this report, the image resizes dimensions will be 16*16 for the raw data feature.

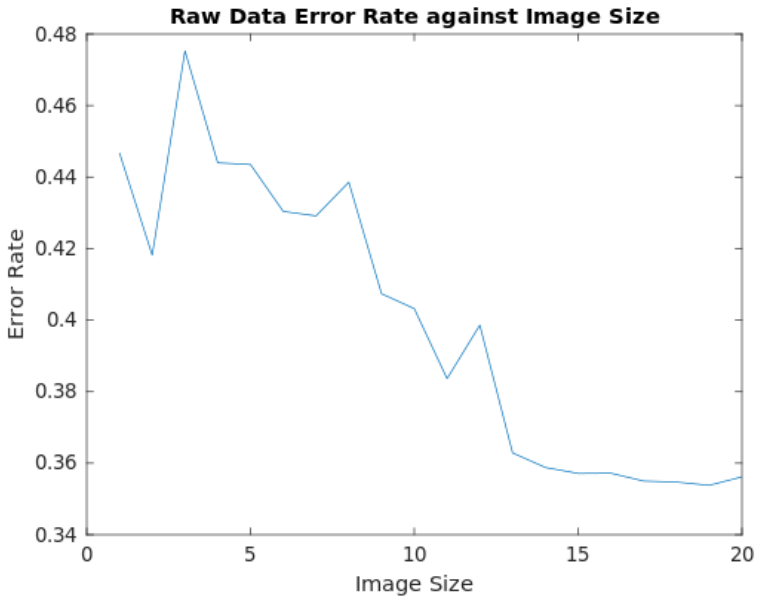

Figure 4 Raw Data Error Rate against Image Size

Feature Comparison

An experimental comparison of each feature was created in order to determine which feature or features to use in the final pipeline. The comparison involved calculating the error rate and accuracies of an SVM classifier when using all possible features, before comparing these values to error rates and accuracies of SVM classifiers when a single feature has been removed

at a time. The error rates are calculated using a 5-fold cross-validated approach to try to provide the fairest and stable results.

The results are laid out in Table 3, with Table 4 outlining the differences between the two compared values. As the results show, removing HOG increased the error rate, while removing the RGB and HSV colour histograms, alongside the flattened raw data decreased the error rate. The removal of the colour histograms appeared to decrease the error rate, however by a small amount; ~0.3% - meanwhile the removal of the flattened raw data decreased the error rate drastically; around 8%.

Using the results from the experimental comparison, the final pipeline will only use the Histogram of Oriented Gradients feature, as all other features evaluated increase the error rate for cross-validated training and testing.

Feature Removed	**Error Rate**	**True Pedestrian**	**False No Pedestrian**	**False Pedestrian**	**True No Pedestrian**
None	9.58%	94.53%	13.69%	5.47%	86.31%
HOG	10.25%	93.78%	14.29%	6.22%	85.71%
Colour Histogram RGB	9.33%	95.03%	13.69%	4.97%	86.31%
Colour Histogram HSV	9.21%	94.31%	12.74%	5.69%	87.26%
Flattened Raw Data	1.28%	99.45%	2.01%	0.55%	97.99%

Table 3 Error rates and accuracies when each feature is removed

Feature Removed	**Error Rate**	**True Pedestrian**	**False No Pedestrian**	**False Pedestrian**	**True No Pedestrian**
None	9.58%	94.53%	13.69%	5.47%	86.31%
HOG	+0.67%	-0.75%	+0.60%	+0.75%	-0.60%

Colour Histogram RGB	-0.25%	+0.50%	0.00%	-0.50%	0.00%
Colour Histogram HSV	-0.37%	-0.22%	-0.95%	+0.22%	+0.95%
Flattened Raw Data	-8.30%	+4.92%	-11.68%	-4.92%	+11.68%

Table 4 Difference in error rates and accuracies when each feature is removed

Feature Visualization

Based on the feature comparison, the feature generated by HOG is the most significant in distinguish positive and negative images. Therefore, we implement Principle Component Analysis as dimensionality reduction method on the HOG feature to visualize how does the feature data distribute in a low-dimensional space.

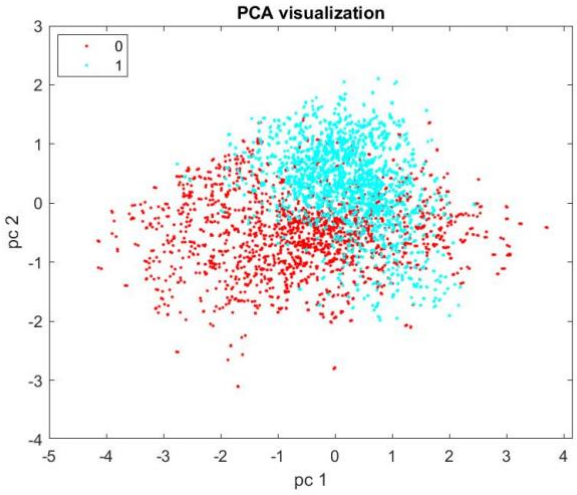

Figure 5 Visualization of feature HOG by applying PCA

Figure 5 illustrates how the data distributed in 2-dimensional space. It can be seen that positive samples (label 1) and negative samples (label 0) are separated well.

Demos

Colour Histogram Tuning Comparison

The following steps explain how to generate the graphs and results for comparing tuning Colour Histogram Quantization Levels:

1. Open MATLAB and set the working directory to csc3061-1920-g10/src.
2. Run the script histogramQuantizationComparison.m.
3. The script will output its progress to the console.
4. The generated graphs will be in the csc3061-1920-g10/graphs folder, called rgb_error_rate_quantization_compare.svg and hsv_error_rate_quantization_compare.svg

Running Flattened Raw Data Tuning Comparison

The following steps explain how to generate the graphs and results for comparing tuning Flattened Raw Data image size:

1. Open MATLAB and set the working directory to csc3061-1920-g10/src.
2. Install the Matlab component Bioinformatics Toolbox [3].
3. Run the script rawDataComparison.m.
4. The script will output its progress to the console (it took around 3 hours to run in our experiment).
5. The generated graphs will be in the csc3061-1920-g10/output folder, called raw_data_image_size_compare.svg

Running Feature Comparison

The following steps explain how to generate the results for comparing each feature:

1. Open MATLAB and set the working directory to csc3061-1920-g10/src.
2. Run the script featureComparison.m.
3. The script will output its progress to the console.
4. The script will print out the results tables to the console.

Running Feature Visualization

The following steps explain how to visualize feature HOG:

1. Open MATLAB and set the working directory to csc3061-1920-g10/src.
2. Run the script PCAImp.m.
3. The script generates a visualized graph and save it to csc3061-1920-g10/output/ pcaVisualization.

Classifier Building

Classifier Definition and Parameter Tuning

KNN Classifier

The KNN algorithm is a simple classifier that can be easily applied to a dataset of features to produce a classifier for binary discrimination. This is a versatile classifier, with no assumptions of the data that can be applied to the feature dataset. KNN, however, does have some limitations. It requires tuning. It is very computationally expensive and slow for predicting. KNN may be applicable for the prediction pipeline, which will be experimentally tested in this report.

KNN Classifier Parameter Tuning

In terms of KNN classifier parameter tuning, the key is how to define the K value for a specific dataset. In this case, setting up a range of K values from 1 to 99 with increment as 2. Only odd numbers are used as K value to avoid confusions in voting.

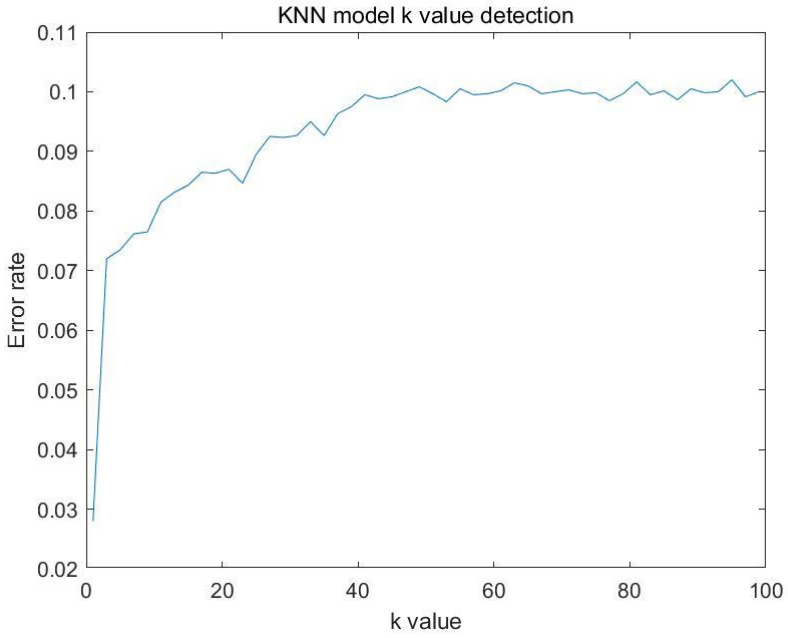

Figure 6 The error rate curve with the growth of k values

In Figure 6, when the k value is 1, the error rate is almost 0, this is because, for each sample in the dataset, it only picks itself as the nearest neighbour points, which means it sets the class label of itself as the prediction label. Thought the error rate is at a low level with one as K value, it will cause overfitting issue. When this classifier is applied to make predictions on an unknown testing set, it will have a high error rate. Therefore, in this case, the K value should

be selected where the curve ends increasing sharply and starts to grow smoothly, which is around 41.

KNN Classifier Testing

The K value is set as 41 to fit the KNN classifier and making predictions on the training set by this classifier. The confusion matrix of the prediction result is shown in Figure 7. The accuracy of the prediction is only 91%, and the Precision ($\frac{3571}{3571+107}$) is 97%

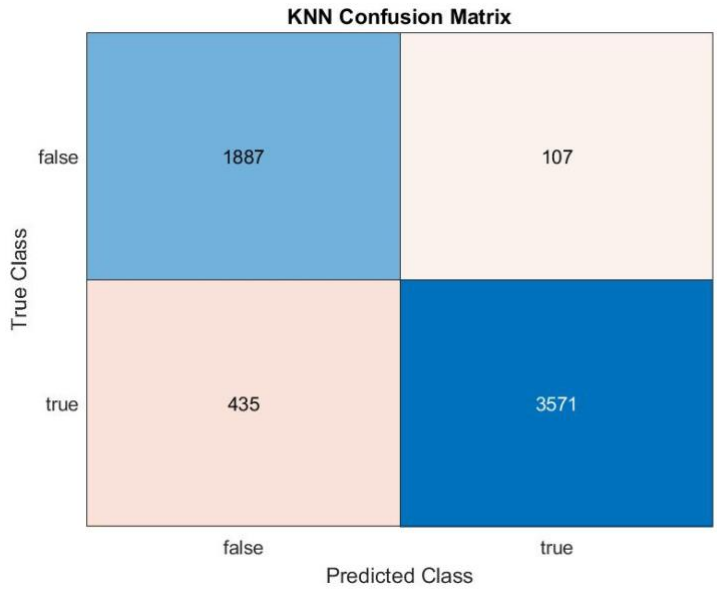

Figure 7 Confusion matrix for KNN

SVM classifier

SVM classifier is one of the most popular and efficient classifiers in classification methods. In this case, the number of training items and features is not excessively big, which will not consume much time on fitting classifiers. Besides, the method of classification is binary classification, and SVM is able to produce a separating hyperplane if the feature data is good. Therefore, SVM might be an appropriate classifier in this situation.

SVM classifier parameter tuning

In respect of the SVM classifier, the parameter tuning method is using "linear" as kernel function and applying "OptimizeHyperparameters" which is a built-in parameter optimization method to find best "BoxConstraint" and "KernelScale". Table 5 illustrates the best observed and estimated feasible points.

Optimization completed.

MaxObjectiveEvaluations of 30 reached.

Total function evaluations: 30

Total elapsed time: 2095.763 seconds.

Total objective function evaluation time: 2087.2172

Best observed feasible point:

BoxConstraint	KernelScale
1.2541	0.3364

Observed objective function value = 0.0135

Estimated objective function value = 0.01321

Function evaluation time = 10.7152

Best estimated feasible point (according to models):

BoxConstraint	KernelScale
841.75	0.095743

Estimated objective function value = 0.01321

Estimated function evaluation time = 11.0526

Table 5 Part of result of parameter optimization for SVM model

SVM classifier testing

The best observed feasible points are used to fit the SVM classifier and implementing built-in cross-validation methods [4] to test the error rate of this classifier, which is 0.010333. The confusion matrix is created based on making prediction by using cross-validation, as seen in Figure 8, the accuracy rate is at a higher level (99%). If a picture is predicted as a pedestrian, it has $\frac{3989}{3989+46} = 99\%$ chance actually containing a pedestrian.

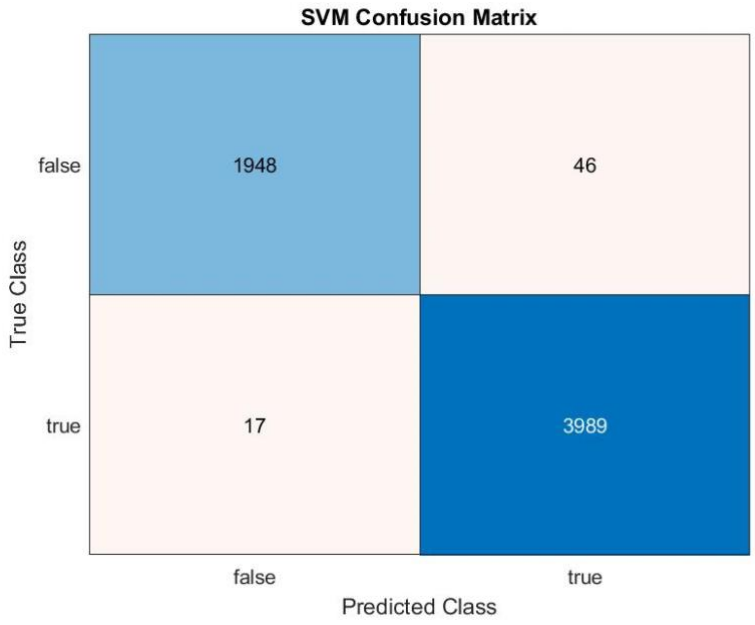

Figure 8 Confusion matrix for SVM

Random Forest Classifier

Random Forest (RF) is another classifier used in this project. The classifier is named as Tree Bagger in Matlab [5]. RF provides a more reliable and better performance than a single decision tree. It repeats the sampling many times by building different trees, and it gives the predictions based on the predictions on majority trees. Besides, it is easy to evaluate the classifier with the Out-of-Bog score [6]. However, the computation may costly and slow depending on the number and depth of trees, which is not useful in applications requiring fast computation [7].

In order to apply the most appropriate classifier on making the prediction on pedestrian images, for three classifiers involving KNN, SVM, and Random forest, a simple parameter tuning method is implemented with all 6000 images as the training set.

Random Forest classifier parameter tuning

The parameter in the random forest classifier is the number of trees, the number of predictors in each tree and the number of input data in each tree.

For the number of predictors, it is a convention that the optimized number is the square root of the number of total predictors [8] which is equal to $\sqrt{1600} = 40$. The number of input data in each tree is set to the default value of the number of total input data, but samplings are with replacements which means there are some data not sampled in each tree.

Thus, the parameter needed to be tuned is the number of trees. Figure 9 shows the OOB error rate declines as the number of trees increases. When the number of trees exceeds 30, the error rate remains flat at around 3%. Because the computational cost rises as when using a more significant number of trees, this parameter will be set to 30 in the testing phase.

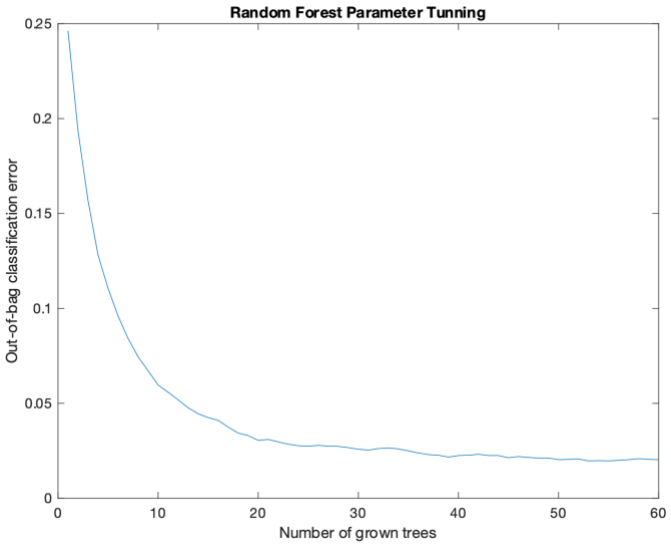

Figure 9 OOB error in Random Forest

Random Forest classifier testing

With the number of trees set to 30, OOB is performed, and the result is shown in Figure 10 (true means pedestrian pictures while false indicates non-pedestrian pictures). As we can see, the classifier has very high accuracy on the pictures of pedestrians (99%). If a picture is predicted as a pedestrian, it $\frac{3971}{3971+118} = 97\%$ chance actually containing a pedestrian.

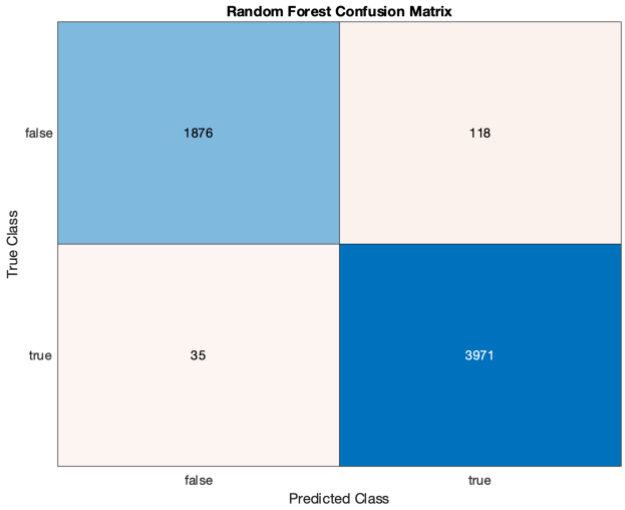

Figure 10 Confusion Matrix for Random Forest

Classifier Comparison

Based on the analysis for three classification classifiers, the accuracy and precision are aggregated in Table 6. It is noticeable that SVM classifier has highest accuracy and precision. Therefore, SVM classifier is applied to making predictions in the further analysis.

Classifier	Accuracy	Precision
KNN	91%	97%
SVM	99%	99%
RF (random forest)	99%	97%

Table 6 Comparison of accuracy and precision for three classifiers

Demos

Parameter Tuning and Classifier Comparison

The following steps explain how to implement parameter tuning and generate confusion matrix of each model:

1. Open MATLAB and set the working directory to csc3061-1920-g10/src.
2. Run the script classifierComparison.m (it took 1 hour to finish in our experiment)
3. Do not close the graph that appears when running this script, this may cause the script to crash.
4. The script will generate the process of parameter tuning to the console.
5. The script will generate graphs and save them into csc3061-1920-g10/output/classifier_comparison

Segmentation Method

Optical Flow

Optical flow detects moving objects in a video. In this project, we need to detect pedestrians on the street that they are usual moving objects in the videos.

Figure 11 demonstrates an example on how optical flow detect moving pixels from a moving camera. Most pixels are the background moving to the right, while the pixels of the dancer move to different directions. If we subtract all vectors by the average vector of the background, the vectors on the dancer will become outliers. Thus, the dancer can be segmented by identifying all pixels with their magnitudes greater than a threshold.

Figure 11 Optical Flow Detection Sample [9]

Implementation

In our example, we calculate the vector in each pixel. Using Euclidean distance formula, the magnitude for each pixel is calculated. The vectors are shown as blue arrows in top-left in Figure 12.

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}, \text{[10]}$$

Considering the images were captured from a moving camera, we calculated the average vector in each column, and subtract the vectors of the corresponding column by the average vector (top-centre in Figure 12).

The threshold value is set to 0.1, which means the pixels with magnitudes greater than 0.1 are classified as moving pixels (top-right corner in Figure 12).

As we can see, there are many noisy pixels in the image. Performing morphological closing [11] and opening [12] can reduce the noises (bottom-left corner in Figure 12).

There are many pedestrians wearing dark clothing, which results that the optical flow only detects the edges of the pedestrians, so we performed Image Fill Holes [13] to fill the objects (bottom-centre in Figure 12).

Finally, we performed the previous two steps again to make the objects more visible (bottom-right in Figure 12).

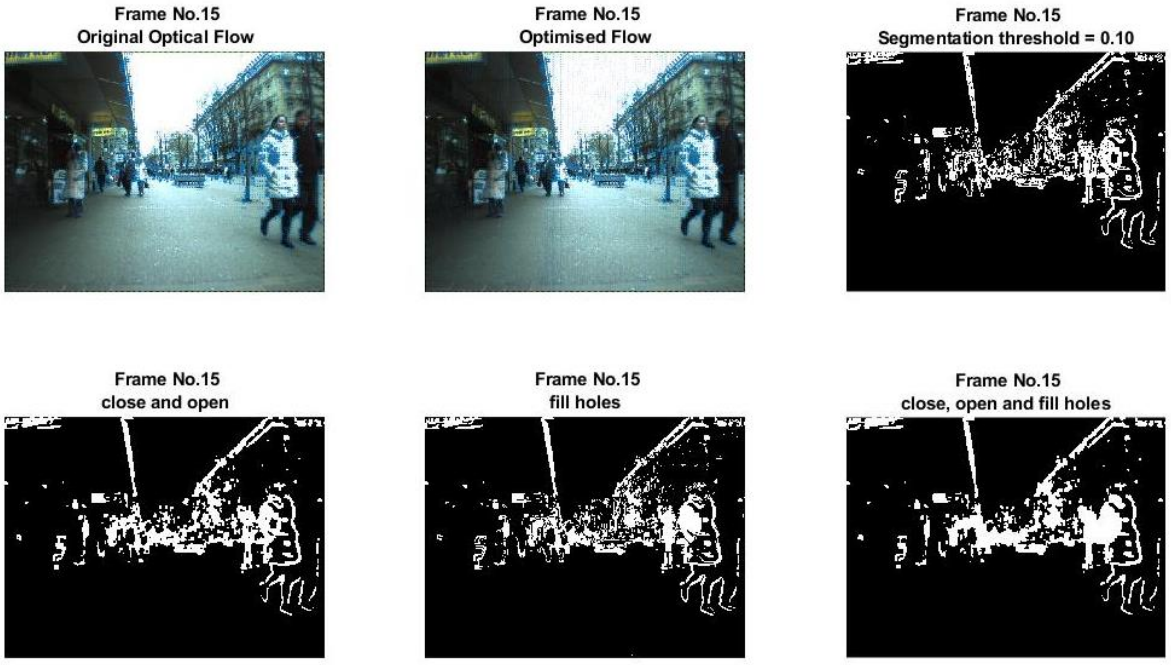

Figure 12 Optical Flow result

Result

In spite of the efforts above, the result of optical flow segmentation is not good. We concluded these three reasons.

- The temporal sampling interval is too larger, making it hard to detect how objects and pixels move from the previous frame.
- The resolution of the images is low. It is hard to detect how each pixel moves.
- The camera is going forward, which means the background pixels are moving to a different direction. In contrast, all background pixels in the example Figure 11 move

equally to the right. It is easy to find out the outlier pixels which are caused by the person's movement.

Hence, the optical flow is not a good technique to apply in this project.

Sliding Windows

The Sliding Window technique is the process of splitting an input image up into smaller images, which can be individually processed for calculating features and classified. The sliding window algorithm will iterate over an image and extract out individual smaller images from it.

The sliding window function takes in a struct containing the images to subdivide into smaller sliding window images with parameters for deciding how to split up the sliding windows, and outputs back into the struct the sliding windows, grouped by scale, and organised by frame, row, and column.

Our implementation of the sliding window takes in four parameters:

- **Width** – The width of images to be broken up into sliding windows.
- **Height** - The height of images to be broken up into sliding windows.
- **Scales** – A vector of scales determining which scales of sliding window to use, Figure 13 shows how each scale is determined. This allows for creation of sliding windows of a variety of scales.
- **Gap Percentage** – A decimal number representing the percentage gap between sliding windows, Figure 14 shows how the gap works. With a gap of 0.4 this means that after the first sliding window, the second sliding window will start from first sliding window left position + 0.4 * first sliding window width. This reduces processing time and the number of sliding windows to calculate for, while still giving wide coverage across the image.

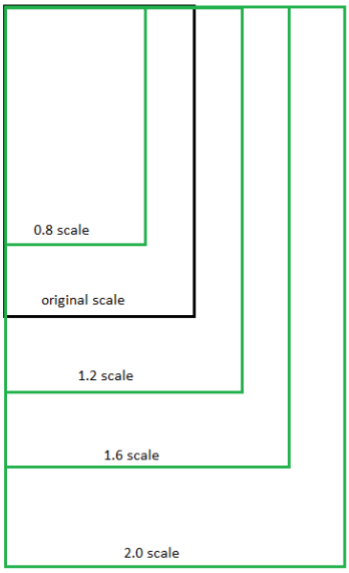

Figure 13 Sliding Window Scales

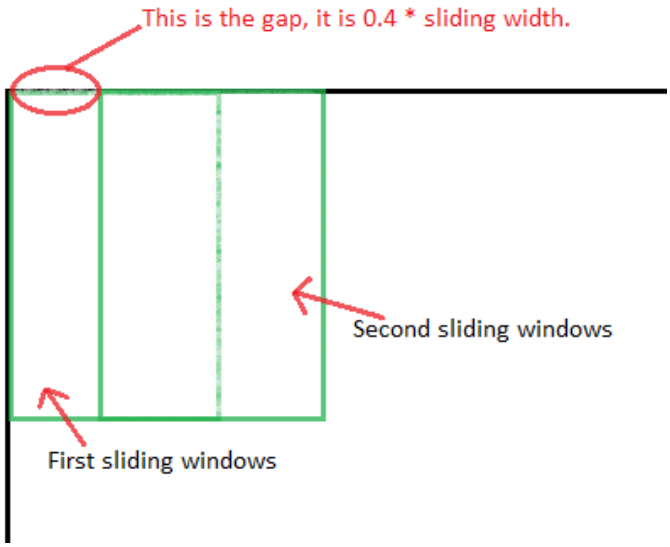

Figure 14 Sliding Window Gaps

Demos

Running Optical flow

The steps below guide you to run the script to view a result.

1. Open Matlab program
2. Switch the current directory to $csc3061-1920-g10/src$
3. Execute the script $opticalflow.m$

Proposal Filtering

From the steps above, we have been able to category each sliding window to as pedestrian or non-pedestrian image. However, some sliding windows contain the same pedestrian in the image, which means the pedestrian has been correctly detected more than one by our classifier. As we only want one proposal of this pedestrian in the final result, Non-maximum Suppression (NMS) and Low-coverage Suppression (LCS) are introduced to filter proposals based on some algorithms.

Non-maximum Suppression

Algorithm

Initially, all sliding windows marked as pedestrian images are included in a proposal list. For each frame in the video, the following NMS algorithm was implemented.

1. Scan for the sliding window with the highest probability score.
2. This sliding window is added to the FINAL LIST and marked as TARGET.
3. Iterate all sliding windows (including the TARGET) marked as a pedestrian, if the sliding window overlaps more than 50% of the TARGET, remove it from the proposal list.
4. Repeat the steps 1-3 until no sliding window is left in the proposal list.

After the process above, the FINAL LIST includes all the detections of the NMS [14].

IoU vs IoM

Intersection Over Union (IoU) and Intersection Over Minimum (IoM) are the two methods to calculate the overlapping score (Figure 15) [15].

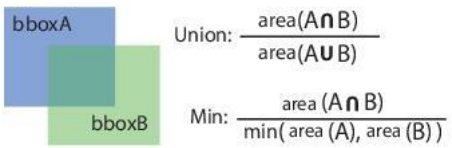

Figure 15 Intersection Over Union [15]

In our project, we will apply both methods, and if the coverage score is higher than 50%, two sliding windows will be considered the same.

Figure 16 illustrates the pedestrian sliding windows in frame ten on the testing video. After applying NMS with IoU and IoM, they have different results in Figure 17.

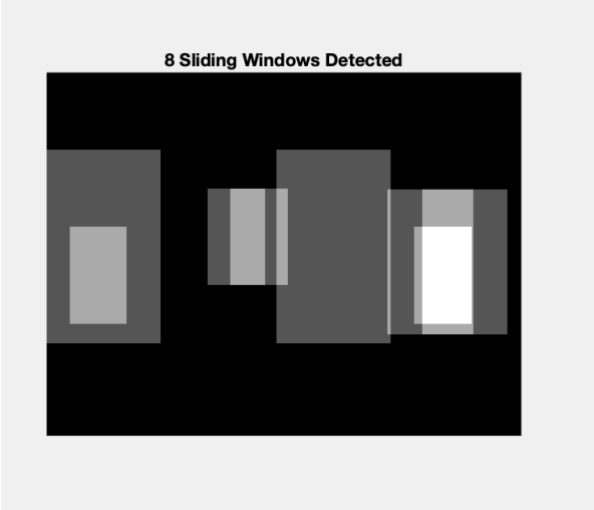

Figure 16 Frame 10 sliding windows

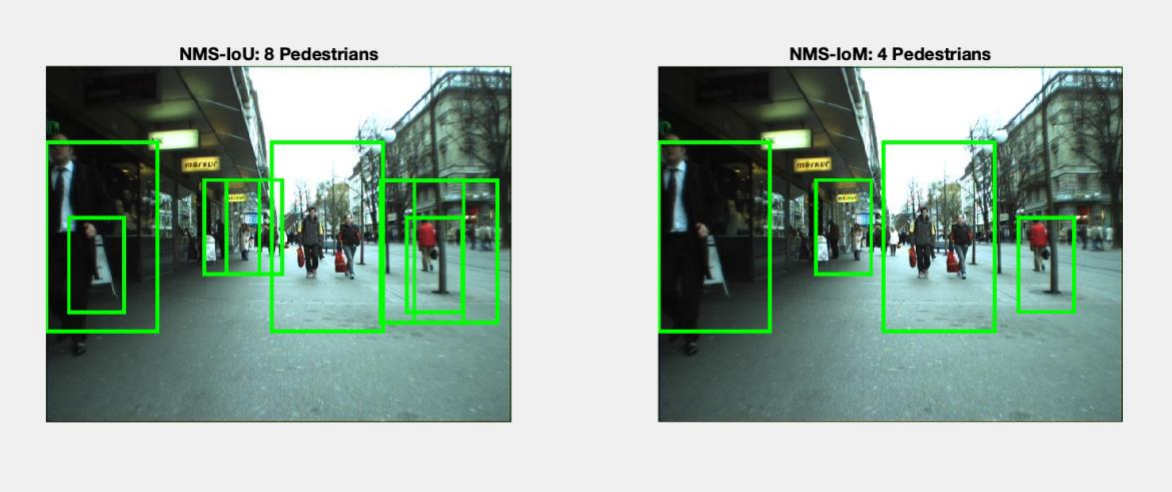

Figure 17 NMS result

Since multiple scales of sliding windows are employed in the previous section, there are some cases that a large sliding window is detected while a much smaller sliding window inside the large window is also detected. Thus, in the result of the IoU, they are identified as two different windows, because the denominator in the formula above will be equal to the area of the large window, making the overlapping score smaller than the threshold. In contrast, the result of IoM will filter the smaller sliding window as the larger one has a higher probability score, because the denominator in the formula above is equal to the area of the small window, making the overlapping score larger. In this specific frame, the IoM has better performance than the IoU.

Furthermore, in pedestrian detection, it is an unlikely event that a large pedestrian is covering a small pedestrian completely. We believe the IoM is more suitable for pedestrian detection.

Low-coverage Suppression

Apart from the NMS, we created another method called LCS. This proposal filtering method was exclusively created by our team.

Algorithm

Initially, all sliding windows marked as pedestrian images are included in a proposal list. For each frame in the video, the following three phases for LCS was implemented respectively.

1. Highlight sliding windows
2. Label pixels
3. Bounding boxing

Highlight sliding windows

This phase involves the following steps

1. Prepare a matrix with the same size as the target image. All values of the matrix are set to 0.
2. Take one sliding window from the proposal list, according to the location of the sliding window, find the corresponding locations in the matrix, and increment the values at those locations by the SCORE.

In first, we set the SCORE as one, so the final value of each element is equal to the number of times it is covered by a sliding window. After some discussion, we decided that the SCORE may not be accurate to set to 1, because we can calculate a probability value (p-value) for each sliding window indicating the likeliness of it to be a pedestrian. The SCORE should be higher in those sliding windows with higher p-values.

Thus, the SCORE is a value calculated using the p-value and the formula in Figure 18.

$$SCORE = 2^{10\times(pvalue-1)} + 1$$

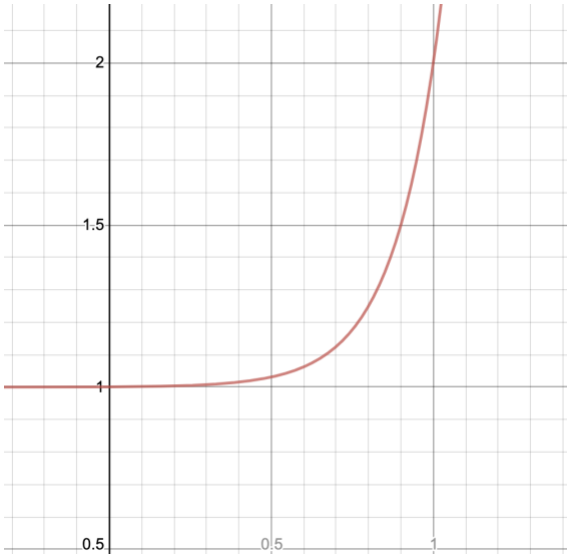

Figure 18 SCORE formula

To visualise this matrix, it has been converted into a greyscale image, and linear stretching is performed to this image. The subplot at the bottom-left in Figure 19 shows the visualisation of the matrix.

Label pixels

This phase identifies the pixels which were classified as pedestrian many times by the classifier. In our experiment, if a pixel is covered by two pedestrian sliding windows, we mark it a pedestrian pixel. The steps to implement it are the following.

1. For each value in the matrix from the previous phase Highlight sliding windows, if the value is higher than the threshold, set this value to 1, otherwise, set it to 0.
2. Using the *bwlabel* function in Matlab, the pixels in the matrix are labelled/grouped.

The subplot at the bottom-right in Figure 19 illustrates the labelling result.

Bounding boxing

According to the result labels, the rectangle bounding boxes in each group of pixels are created. The subplot at the top-right corner in Figure 19 visualizes the bounding boxes in the image.

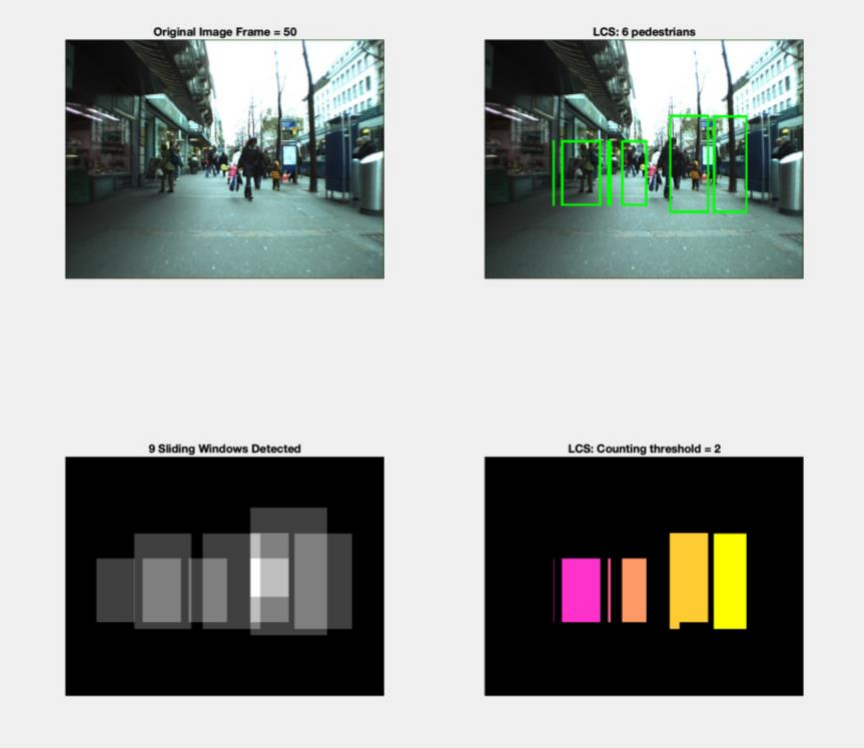

Figure 19 LCS result

Demos

Main program

In the sections above, we have compared all the classifiers, features and their tunable parameters. With these best configurations, we can now run the main thread of the program. It uses the information extraction techniques described below and detect pedestrians against the testing video.

To run the main program, follow these steps.

1. Open Matlab program and set the current directory to $csc3061-1920-g10/src$
2. Run the script $main.m$ (it takes 10 to 20 minutes to finish)

The output of the script will be saved at $csc3061-1920-g10/output$. At the end of the script, the result of the pedestrian detection against the testing video will be shown.

Evaluation

In our project, the models are evaluated both visually and quantitatively.

Quantitative Evaluation

First of all, the demo above will generate detection datasets using the NMS and LCS. For NMS, it has datasets for IoU and IoM methods, respectively.

These dataset files are located at $csc3061 - 1920 - g10/output/dataset$.

Implementation

Using Multi-target Detection Evaluation [16], each of these detection datasets will be compared with the $test.dataset$ provided in the project document, respectively.

The following steps are performed to evaluate the detection datasets against the test dataset.

1. Set a variable $count = 0$ indicating the total true positives (TP).
2. To evaluate the first frame of the video, iterate the detection bounding box in the first frame, and scan the bounding boxes in the first frame in the test dataset, if the detection bounding box overlaps more than 50% of the test bounding box, perform the following steps.
    a. the $count$ is incremented by 1
    b. remove this test bounding box from the test dataset
3. Repeat step 2 to evaluate the remaining frames in the video.

After these steps, the $count$ has the value of TP, which are the number of pedestrian bounding boxes detected.

To get the false negative (FN) which is the number of non-detected pedestrian bounding boxes, we calculated the total pedestrian bounding boxes (T) in the test dataset. Thus,

$$FN = T - TP$$

To get the false positive (FP) which is the number of non-pedestrian detection bounding boxes, we calculated the total bounding boxes detected (P) in the corresponding dataset. Thus,

$$FP = P - TP$$

Result

Using the formulas above, Table 7 was produced showing the confusion matrices for each method.

Confusion Matrix for LCS result

	positive	negative
pedestrian	180	667
non-pedestrian	235	Inf

Confusion Matrix for NMS IoU result

	positive	negative
pedestrian	584	263
non-pedestrian	1158	Inf

Confusion Matrix for NMS IoM result

	positive	negative
pedestrian	380	467
non-pedestrian	708	Inf

Table 7 Datasets Confusion Matrices

From the result, we can compare different methods by their precisions and recalls [17].

The precision of the LCS is $\frac{180}{180+235} = 43\%$. This value is $\frac{584}{584+1158} = 34\%$ in the NMS with IoU, while the NMS with IoM has a similar precision of $\frac{380}{380+708} = 35\%$. From this statistic, the LCS method is better than the other two. It is believed that the LCS algorithm selects the pixels covered by multiple detected sliding windows, which makes it only produce high-probable detections.

Similarly, the recall of the LCS is $\frac{180}{180+667} = 21\%$. The NMS with IoU has the recall of $\frac{584}{584+263} = 69\%$, and this figure is $\frac{380}{380+467} = 45\%$ for the NMS with IoM. Dramatically, the NMS with IoU enjoys the highest recall value leaving other two far behind. We believe that is because this method tries to select many potential sliding windows, which makes sure it does not filter out many pedestrians.

Finally, as proposal filtering methods are expected to have a high recall value [14], the NMS with IoU has excellent performance on recall and acceptable performance in precision, it seems the best method to apply on the pedestrian detection.

Visual Evaluation

In this section, we plotted the images generated from the proposal filtering methods above. Taking one frame from the video to explain, Figure 20 draws the results using three different proposal filtering methods.

Figure 20 Result of Frame 23

Referring to the analysis of the previous section Quantitative Evaluation, the LCS (bottom-right in Figure 20) only selects the areas which are covered by many detections (bottom-left), making it has the highest precision value. The NMS with IoU (top-centre) tends to select as many detections as possible which produces a high recall value while having the lowest precision value. The NMS with IoM (top-right) seats in the middle of these two methods, which performs in an acceptable manner.

In conclusion, the LCS produces high precision. In contrast, the NMS with IoU has high recall while the NMS with IoM has compromised effect between these two.

Demos

Before running this program, the **Error! Reference source not found.** must be run before, which generates $csc3061-1920-g10/src/result.mat$ in the directory. Then follow the steps below.

1. Open Matlab program
2. Switch the current directory to $csc3061-1920-g10/src$
3. Run the script $evaluationResult.m$

Other Methods

Except for the standard method described above, we also explored some online resources to detect pedestrians.

Tracking Pedestrians from a Moving Car

A script from Mathworks demonstrates an example of Tracking Pedestrians from a Moving Car [18]. This script uses an existing model to detect and track pedestrians, and it produces a reasonable result against our testing video (Figure 21).

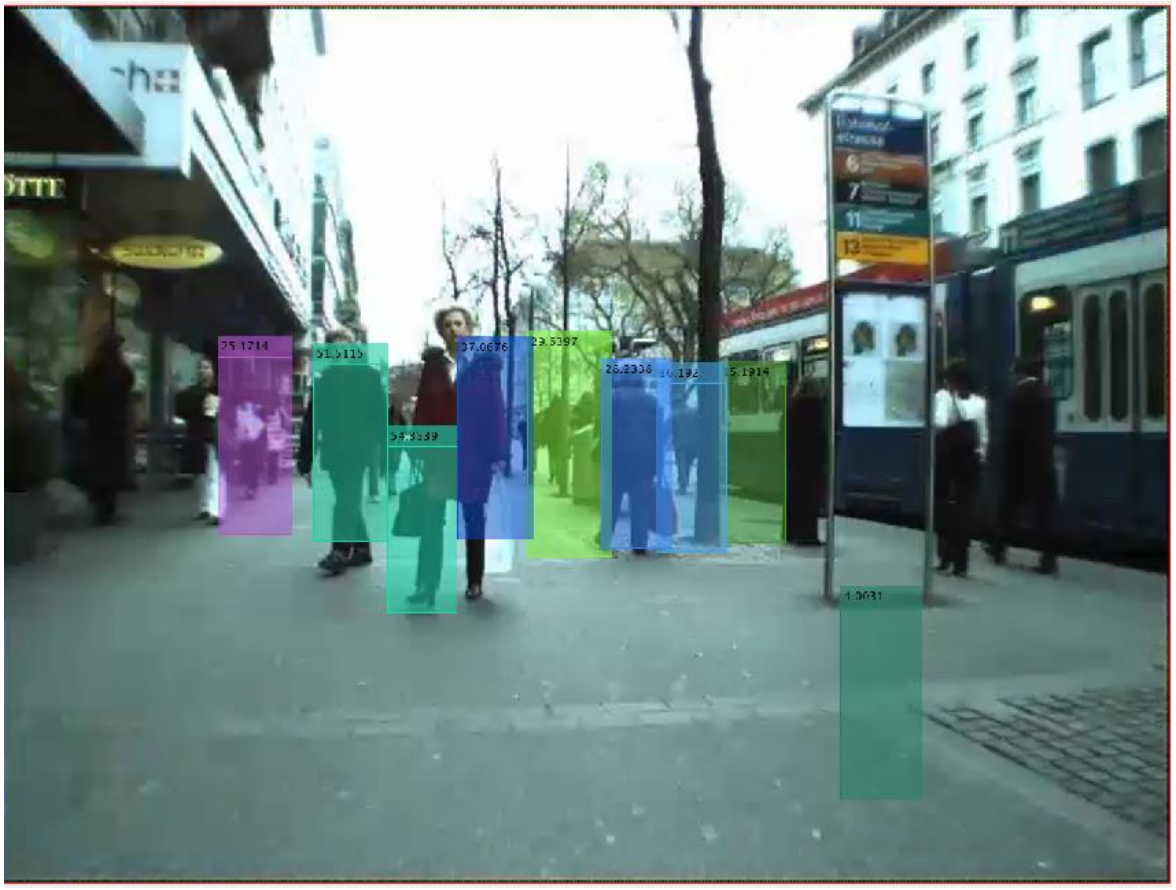

Figure 21 Result of Matlab example script

Demo

To view the result of this Matlab example script, follow the steps below.

1. Open Matlab program
2. Switch the current directory to $csc3061-1920-g10/src$
3. Run the script $matlabexample.m$

Detect people using aggregate channel features

When doing research for available resource, we find Matlab provide an effective built-in method (peopledetectACF) on area of people detection, which is using aggregate channel features to detect people [19].

The peopledetectACF is a strong method which is able to be directly used on testing images. It can automatically create rectangle boxes to find where are people in images and mark scores. Figure 21 is an example images from MathWorks showing how this method works on an image.

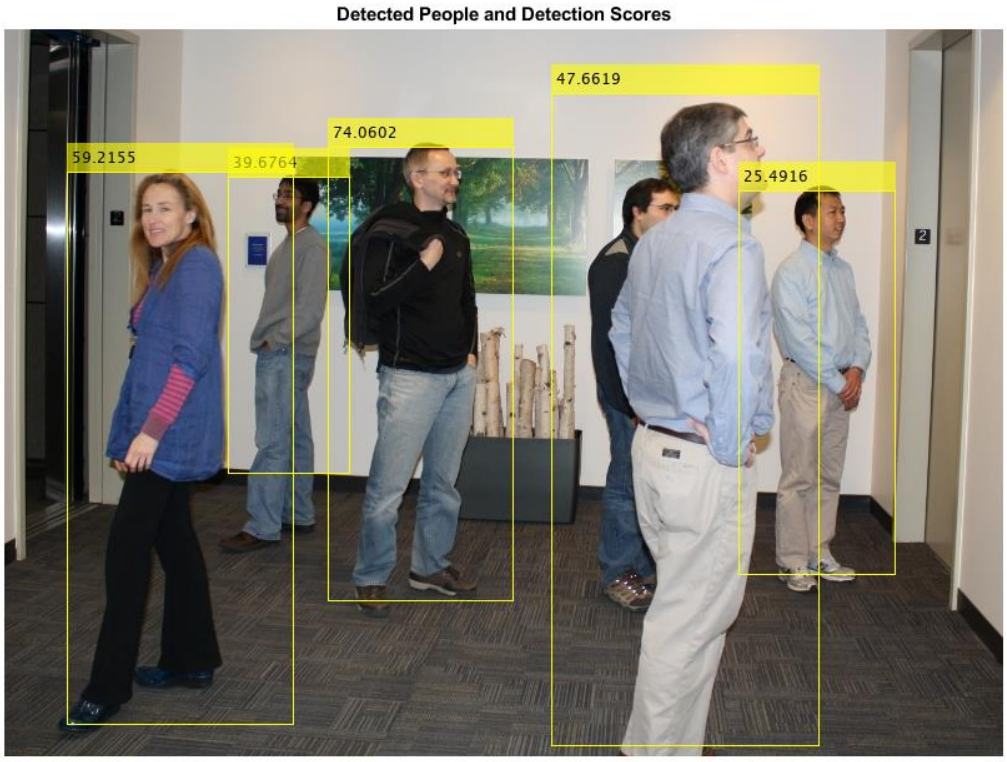

Figure 22 People detection example using peopledetectACF [19]

Demo

To view the result of this Matlab example script, follow the steps below.

1. Open Matlab program
2. Switch the current directory to $csc3061 - 1920 - g10/src$
3. Run the script $peopleDetectACFImp.m$

Conclusion

In this project, we explored many methods detect pedestrians from a video captured by a moving camera.

By a large number or trials and experiments, we conclude that using HOG to generate features and doing prediction by the SVM model is the most appropriate method in people detection. In the presentation of pedestrians, we implemented NMS and LCS filter proposals, and found out that the NMS method with IoU is the best among our methods.

The final pipeline of this project is the product of comparison and analysis of several techniques; with decisions justified by experiments and data. We have aimed to provide clear and repeatable justification for the decisions we have made at each stage and based on this we have created a pipeline that we believe has a good performance for pedestrian detection.

Also, we did some research on existing online resources that provide models to detect and track humans in images.

For the future work, to enhance the detection accuracy, it can be aimed at exploring more advanced techniques to filter proposals of sliding windows and add functions to track pedestrians from the previous frame.

References

[1] D. Liu, J. Thompson and D. Li, "CSC3061-1920 / CSC3061-1920-G10 · GitLab," GitLab, 2020. [Online]. Available: https://gitlab2.eeecs.qub.ac.uk/CSC3061-1920/csc3061-1920-g10. [Accessed 31 March 2020].

[2] A. R. A. B. A. B. F. Suard, "Pedestrian Detection using Infrared images and Histograms of Oriented Gradients," in *IEEE Xplore,* Tokyo, Japan, 2006.

[3] MathWorks, "Bioinformatics Toolbox - MATLAB," MathWorks, 2020. [Online]. Available: https://uk.mathworks.com/products/bioinfo.html. [Accessed 31 March 2020].

[4] MathWorks, "Cross-validate support vector machine (SVM) classifier," 1994-2020 The MathWorks, Inc., [Online]. Available: https://www.mathworks.com/help/stats/classificationsvm.crossval.html. [Accessed 25 3 2020].

[5] MathWorks, "Create bag of decision trees - MATLAB - MathWorks United Kingdom," 2020. [Online]. Available: https://uk.mathworks.com/help/stats/treebagger.html. [Accessed 26 March 2020].

[6] N. Bhatia, "What is Out of Bag (OOB) score in Random Forest?," 16 June 2019. [Online]. Available: https://towardsdatascience.com/what-is-out-of-bag-oob-score-in-random-forest-a7fa23d710. [Accessed 26 March 202].

[7] Oreilly, "Pros and cons of random forests - Hands-On Machine Learning for Algorithmic Trading [Book]," 2020. [Online]. Available: https://www.oreilly.com/library/view/hands-on-machine-learning/9781789346411/e17de38e-421e-4577-afc3-efdd4e02a468.xhtml. [Accessed 26 March 2020].

[8] L. Breiman, " RANDOM FORESTS," University of California, Berkeley, 2001.

[9] J. M. d. Rincón, "4-Detection-Students.pdf: CSC3061: Video Analytics and Machine Learning (2191_SPR)," 2020. [Online]. Available:

https://canvas.qub.ac.uk/courses/8435/files/597917?module_item_id=224544. [Accessed 31 March 2020].

[10] Rosalind, "Euclidean distance," Rosalind, 2020. [Online]. Available: http://rosalind.info/glossary/euclidean-distance/. [Accessed 26 March 2020].

[11] T. U. o. Edinburgh, "Morphology - Closing," 2020. [Online]. Available: https://homepages.inf.ed.ac.uk/rbf/HIPR2/close.htm. [Accessed 26 March 2020].

[12] T. U. o. Edinburgh, "Morphology - Opening," 2020. [Online]. Available: https://homepages.inf.ed.ac.uk/rbf/HIPR2/open.htm. [Accessed 26 March 2020].

[13] Mathworks, "Fill image regions and holes - MATLAB imfill - MathWorks United Kingdom," Mathworks, 2020. [Online]. Available: https://uk.mathworks.com/help/images/ref/imfill.html. [Accessed 26 March 2020].

[14] S. K, "Non-maximum Suppression (NMS) - Towards Data Science," Towards data science, 1 October 2019. [Online]. Available: https://towardsdatascience.com/non-maximum-suppression-nms-93ce178e177c. [Accessed 27 March 2020].

[15] The_Matrix_, "非极大值抑制 NMS 算法实现," CSDN, 10 December 2018. [Online]. Available: https://blog.csdn.net/cuixing001/article/details/84946990?fbclid=IwAR2qSUcVcPKJhuRBIVVUqIvACi3TrqJhln4Xuri4ZnRgKdYc7yM5dPRKaFU. [Accessed 27 March 2020].

[16] J. M. d. Rincón, "8-Metrics-Students.pdf: CSC3061: Video Analytics and Machine Learning (2191_SPR)," QUB, 2020. [Online]. Available: https://canvas.qub.ac.uk/courses/8435/files/597921?module_item_id=224547. [Accessed 27 March 2020].

[17] W. Koehrsen, "Beyond Accuracy: Precision and Recall - Towards Data Science," Toward data science, 3 March 2018. [Online]. Available: https://towardsdatascience.com/beyond-accuracy-precision-and-recall-3da06bea9f6c. [Accessed 27 March 2020].

[18] MathWorks, “MathWorks Help Center,” 1 March 2020. [Online]. Available: https://uk.mathworks.com/help/vision/examples/tracking-pedestrians-from-a-moving-car.html.

[19] MathWorks, “Detect people using aggregate channel features,” [Online]. Available: https://uk.mathworks.com/help/vision/ref/peopledetectoracf.html. [Accessed 30 3 2020].