How to use this code

1 Introduction

In this document, we will explain how to use the code provided. The code is a modification of code provided by X.L.X Stokkel for his bachelor thesis, there used to find humans. The code has been modified to use a larger area of the image, in order to evaluate suitable landing sites.

In the following sections, we will look at which data can be used as input, describing the data (creating feature vectors), training the Support Vector Machine (SVM) and classifying to test its performance. In the last section, we will provide some examples.

It should be noted, that while code for multiple classes is given, this code has not been tested in this research.

2 Data

2.1 Gathering Data

Before we can use this code, we need data. In the thesis we used the bottom camera of a *Parrot AR.Drone 2* to gather images. Other types of drones may be used as well. It is important to note that:

- The drone should be able land on the area represented by the image.
- Every type of terrain which the drone may encounter, should be represented in the dataset.
- The code assumes the images are .JPEG.

2.2 Labeling the data

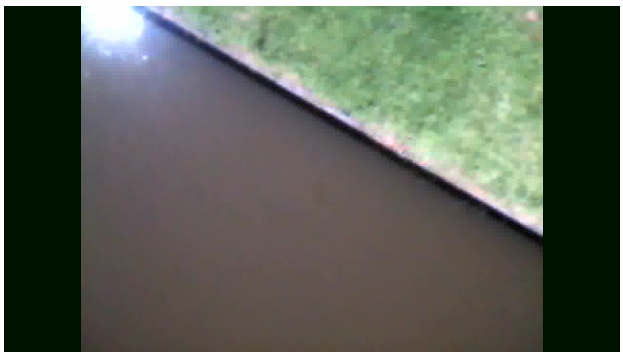

Figure 1: Landing in water will seriously harm the drone.

The code assumes that the files are labeled in the following way: the filename represents the classes present in the picture. For example, image 1 contains both water and grass. If the drone would try to land here, it will most likely fall into the water. Therefore, this place is dangerous. The filename will be: *dangerous water grass*. The order of the keywords does not matter.

The code provides a script for labeling. To use this script, the unlabeled files should be placed in the *doubtful* folder, after which *labelize.m* can be executed. A figure should pop up, which shows two squares. The outer square is one of 720x720 pixels, centered on the image. The inner square, sized 500x500 pixels, can be used as a better reference than the outer square to determine whether the landing site is safe or dangerous.

To add a label to the image, press its key. When all labels are added, press enter. The labeled image will now move to the *labeled* folder. *labelize.m* assumes that the images are .JPEGs of 1280x720 pixels.

The keys supported by *labelize.m* are:

- *s* for safe.
- *d* for dangerous.
- *b* for bushes.
- *r* for road.
- *g* for grass.
- *w* for water.

Pressing any key that is not supported, wil display a list of supported keys.

To add classes to keys, edit *getKey_labelize.m*.

2.3 Organizing the data

When all data is properly labeled, it will have to be organized in order to allow the code to find it.

crossdescribe.m and *multicrossdescribe.m* assume, by default, a folder *crossfold* which has 10 subfolders, named *training (1)* to *training (10)*. Each of those folders should have the same amount of images. The location of the *crossfold* folder can be edited in *crossdescribe.m* and *multicrossdescribe.m*.

3 Describe

There are 4 functions that can be used to describe the data:

- *describe.m* if training or testing on a single folder, using only safe/dangerous as classes.
- *crossdescribe.m* if performing a 10-fold crossfold, using only safe/dangerous as classes.
- *multidescribe.m* if training or testing on a single folder, using custom classes.
- *multicrossdescribe.m* if performing a 10-fold crossfold, using custom classes.

Each function takes as input: the size of the cells (CELLSIZE); the amount of cells per block (BLOCKSIZE); the amount of bins (HOGBINS) and the size of the square centered on the image (SQUARESIZE). In the thesis, a CELLSIZE of 72, a BLOCKSIZE of 2, a HOGBINS of 9 and a SQUARESIZE of 720 were used. *describe.m* and *multidescribe.m* take an extra argument DATA_DIR; a string stating the location of the data. Additionaly, *multidescribe.m* and *multicrossdescribe.m* both require an extra argument: a cell-array of strings, where each string is the name of a class. The order given, will be the order in which they are processed: if a landing site contains both water and grass, whichever class is listed first will determine the label.

The output is a .mat file saved as *descriptors.mat*, *crossdescriptors.mat*, *multidescriptors.mat*, or *multicrossdescriptors.mat*. This file contains the data, colordescriptors, hogdescriptors, labels, and the CELLSIZE, BLOCKSIZE, HOGBINS and SQUARESIZE used for calculating the feature vectors. Additionaly, *multidescriptors.mat* and *multicrossdescriptors.mat* both contain a cell-array called *classes*, which lists their classes in the order that they were given.

4 Train

There are 4 functions which can be used to train a SVM on the data, depending on which file was used to describe it. Use:

- *trainsvm.m* to train on a single folder, using only safe/dangerous as classes.
- *crosstrainsvm.m* to train 10 seperate models for a 10-fold crossfold, using only safe/dangerous as classes.
- *multitrainsvm.m* to train on a single folder, using custom classes.
- *multicrosstrainsvm.m* to train 10 seperate models for a 10-fold crossfold, using custom classes.

Each function takes as input: the cost, the kernel function to be used, descriptors, labels and a FILENAME. The cost is a n by n matrix, where n is the amount of classes. An element at *index* (*row*,*column*) lists the cost of an instance of *column*, being classified as *row*. For example, if we have classes 'positive' and 'negative', our matrix will represent [TruePositive FalsePositive; FalseNegative TrueNegative]. The order of the classes, is the same order given as input by describing the data.

multitrainsvm.m and *multicrosstrainsvm.m* take an extra argument, listing their classes in the order they were given.

The SVMModel(s) will be saved in a .mat file FILENAME. If FILENAME is empty, SVMModel(s) will not be saved. SVMModels will also be given as output.

5 Classify

There are 4 functions which can be used for classification, depending on which SVMModel has been trained. Use:

- *classify.m* to classify a single SVMModel on a single folder, using only safe/dangerous as classes.
- *crossclassify.m* to classify the 10 SVMModels in a 10-fold crossfold, using only safe/dangeorus as classes.

- *multiclassify.m* to classify a single SVMModel on a single folder, using custom classes.
- *multicrossclassify.m* to classify the 10 SVMModels in a 10-fold crossfold, using custom classes.

Each function takes as input: the data, SVMModels and the descriptors. The data and descriptors can be obtained, by using the *describe* function. In the case of *classify.m* and *multiclassify.m*, data and descriptors should be those of the test data. In the case of *crossclassify.m* and *multicrossclassify.m*, the data and descriptors generated by *crossdescribe.m* or *multicrossclassify.m* can be used.

multitrainsvm.m and *multicrosstrainsvm.m* take an extra argument, listing their classes in the order they were given.

6 Sample Code

6.1 Performing a 10-fold Crossfold

First, make sure all data is correctly labeled and divided between 10 subfolders called *training (0)* on to *training (10)*, located in the *crossfold* folder in *data*.

Then, execute the following command in the command window.

```
crossdescribe(72,2,9,720,456)
```

This will create a file called *crossdescriptors.mat*, using a CELLSIZE of 72, a BLOCKSIZE of 2, 9 HOGBINS, and a SQUARESIZE of 720x720 pixels. It assumes there are 456 files per folder.

We can now load *crossdescriptors.mat* by executing

```
load('crossdescriptors.mat');
```

This will add the following variables to our workspace:

- *data* is a 1x10 struct; listing information over the data (e.g. filenames)
- *hogdescriptors* is an array of HOG-descriptors, obtained by ExtractHOGFeatures, used to describe the data
- *colordescriptors* is an array of HSV color histograms, used to describe the data
- *labels* is an array containing the true labels of the data
- *CELLSIZE* is the size (in CELLSIZExCELLSIZE pixels) of the cells, used by the HOG-descriptors and color histograms
- *BLOCKSIZE* is the size (in BLOCKSIZExBLOCKSIZE cells) of the blocks, used by the HOG-descriptors and color histograms
- *HOGBINS* is the amount of bins, used by the HOG-descriptors and color histograms
- *SQUARESIZE* is the size of the square (in SQUARESIZExSQUARESIZE pixels), centered on the image, used for describing
- *nfiles* is the amount of files per folder
- *DATA_DIR* is the directory where the *training* folders are located

If we would like to use both hogdescriptors an colordescriptors, we can concatenate them by executing:

```
descriptors = cat(3,hogdescriptors,colordescriptors);
```

this will create a new variable called *descriptors*, which contains both the HOG-descriptors and the color histograms.

We can now use these variables to train SVMModels. Executing:

```
SVMModels = crosstrainsvm([0 100; 1 0], 'linear', hogdescriptors, ...
                          labels, 'Linear HOG [0 100; 1 0].mat');
```

will create a .mat file called *Linear HOG [0 100; 1 0].mat*, which contains *SVMModels*. *SVMModels* is an array of 10 SVMModels, each trained on a different subset of the data (9 out of 10 folders). The SVMModels have a cost matrix of [0 100; 1 0], which means that misclassifying dangerous landing sites as safe is discouraged. The SVMModel is linear and trained only on HOG-descriptors. Executing:

```
SVMModels = crosstrainsvm([0 1; 1 0], 2, descriptors, labels, ...
                          'Polynomial 2 [0 1; 1 0].mat');
```

will create a .mat file called *Polynomial 2 [0 1; 1 0].mat*, which again contains *SVMModels*. This time, the SVMModels have a cost matrix of [0 1; 1 0], which is the standard cost matrix. Furthermore, the SVMModels are using a 2nd order polynomial kernel and are trained with both HOG-descriptors and color histograms. Note that this assumes that *descriptors* is initialized, as shown in the previous example.

Lastly, we can use these SVMModels for classification:

```
crossclassify(data, SVMModels, descriptors)
```

This gives us information about the performance of the model. It lists a confusion matrix and it displays performance measures, like True Positive Rate and True Negative Rate.

6.2 Evaluating Multiple Classes

First, make sure all data is correctly labeled and divided between two folders - one for training, and one for testing. Then, think of which classes you want to be able to evaluate.

Let us assume our images are of grass, bushes, water and road. We would like to avoid water at all cost. If an image has any water on it, it should be recognized as such. Next, we would like to avoid bushes more than we want to avoid grass. For this reason, an image with both grass and bushes will be labeled 'bushes', instead of 'grass'. Road will be the third class to consider, and grass would be the last. Let us assume we have placed these images in two folders, called 'data/train' and 'data/test', located in the same directory as our code.

Our classes cell array will be:

```
classes = {'water','bushes','road','grass'};
```

Therefore, when determining the label to train the SVM with, we will consider water first, then bushes, then roads and then grass. If an image contains both water and grass, using this order, it will be classified as water. If we would rather have it classified as grass, simply put grass before water.

Now, we can describe the images using *multidescribe.m*.

```
multidescribe({'water','bushes','road','grass'}, 'data/train', 72, 2, 9, 720);
```

Like the previous example, this command will create the descriptors using a CELLSIZE of 72, a BLOCKSIZE of 2, 9 HOGBINS and a SQUARESIZE of 720x720 pixels. It will then store these into a file, along with some important parameters. Unlike the previous example, the name of the .mat file is now *multidescriptors.mat*. Furthermore, it does not keep track of *nfiles*, since there is only one folder. It does save the order of classes.

We can load *multidescriptors.mat* by executing:

```
load('multidescriptors.mat');
```

This will add the following variables to our workspace:

- *data* is a struct listing information about the data (e.g. filenames)
- *hogdescriptors* is an array of HOG-descriptors, obtained by ExtractHOGFeatures, used to describe the data
- *colordescriptors* is an array of HSV color histograms, used to describe the data
- *labels* is an array containing the true labels of the data
- *classes* is an cell array containing the classes and their order
- *CELLSIZE* is the size (in CELLSIZExCELLSIZE pixels) of the cells, used by the HOG-descriptors and color histograms
- *BLOCKSIZE* is the size (in BLOCKSIZExBLOCKSIZE cells) of the blocks, used by the HOG-descriptors and color histograms
- *HOGBINS* is the amount of bins, used by the HOG-descriptors and color histograms
- *SQUARESIZE* is the size of the square (in SQUARESIZExSQUARESIZE pixels), centered on the image, used for describing

Now, we can use these descriptors to train a SVM. By executing:

```
SVMModel = multitrainSVM(classes, [0 100; 1 0], 'linear', hogdescriptors, ...
                         labels, 'multiclass linear HOG SVM [0 100; 1 0].mat');
```

we can create a multiclass linear SVM, trained on HOG-descriptors, with a cost of [0 100; 1 0]. The order of the classes will be: 'water', 'bushes', 'road', and lastly, 'grass'.

In order to use the SVMModel for classification, we have to create descriptors for the test files as well. We can do this, by executing *multidescribe.m* again, this time on the test folder. We are then able to load the .mat file it has created:

```
multidescribe({'water','bushes','road','grass'}, 'data/test', 72, 2, 9, 720);
load('multidescriptors.mat');
```

Lastly, we can use *multiclassify.m*:

```
multiclassify(classes, data, SVMModel, hogdescriptors);
```

This will output a confusion matrix.