Group 1

Aren Babikian (260684709)

Francis Duhamel (260746909)

Mohamed Nouri (260639472)

Maxence Rousseau-Regaudie (260632647)

Ammar Sarfaraz Aziz (260565232)

ECSE 415: Intro to Computer Vision, F2018

Final Project Report

December 2nd, 2018

# Classification

The first step of our project was image classification. Our goal for this part was to train two classifiers using the images from a given dataset such that the classifiers could be later used during localization. Our approach consists of 3 steps: step 1 involves data acquisition and feature extraction while steps 2 and 3 involve training a Support Vector Machine (SVM) classifier and a K-neighbours classifier, respectively. These steps, as well as other details of our implementation, are discussed over the following subsections of this report.

## Used libraries

We use various libraries in our implementations. Besides the standard *cv2*, *numpy* and *matplotlib* libraries used throughout the semester, we use the *glob* library [CITATION?] to facilitate the import of many images from different subdirectories into the program. We also use the *datetime* library [CITATION?] for timing analysis of our implementation. Finally, we use various sub libraries of the *sklearn* library to complete all machine-learning related tasks within our program.

## Data Acquisition and Feature Extraction

### Data acquisition

As requested, we have used the MIO-TCD car dataset [CITATION?] as input data. The dataset contains 11 categories of images and a various number of images in each category. The total number of images in the MIO-TCD dataset is over 600 000. However, the number of images being read into the program for each category has been limited to 2000, which makes for a total of 22000 processed images.

On one hand, the reason we have set a limit is because processing the complete dataset would be very time consuming and would also require a lot of storage memory (which would mean that we would have less available memory to store the extracted features of each image). Furthermore, we noticed that when we were processing all images, predictions were very inaccurate due to the large variance in the number of elements in each category of the dataset. We noticed that test images were often being falsely predicted as part of the categories with a larger number of images. On the other hand, considering that the smallest category contains 1751 images and that there are only two categories with less than 2000 images in the whole dataset, we chose 2000 as the threshold value regarding the number of images to import per category. Experimenting with different threshold values showed that 2000 was a good limit to set, considering that high accuracy was achieved with the current image number limits.

Images from different categories are read into the program one by one and immediately resized to 64x64 pixels. Considering that the images in the dataset have very different dimensions, we decided to resize the images into squares to generalize our implementation. We chose 64 as the size of the square side because, on one hand, it is small enough not to cause any issues regarding memory and to avoid wasting too much time during feature extraction, and, on the other hand, it is large enough to avoid distorting the input images (which, as is, are not very high-quality).

### Feature Extraction

Once an image is read into the program, its HoG features are extracted using the same algorithm that we saw over the semester. Image dimensions are 64x64, cell size is 4x4, block size is 4x4 (cells) and we use 8 bins. These parameter values are similar to the ones seen in class. We have decided to use a simple feature extraction technique with typical parameters simply because of the fact that our implementation, as it is, gives good enough results in a short enough amount of time We did experiment with other values, particularly with other numbers of bins, and results were not as positive. Furthermore, our current configuration does not cause any memory and storage issues either.

Once features are extracted, they are stored in an array. At this point, it is important to note that although we view the images category by category, we store the images at random positions such that all positions of the array are eventually filled. We do this because had we not done this, we would run into issue during cross validation as training would occur mainly on all categories except for one and testing would occur n the categories for which no training had occurred, which is obviously a problem.

### Runtime Analysis

Our implementation requires around 1:08 minutes to import the portion of the dataset we are interested in importing. Considering that we import 22000 images, it takes approximately 0.0031s to import a single image, which we do not consider to be slow at all.

## SVM Classifier

We trained the SVM classifier, which is deep learning classifier, with the data acquired in the earlier step. We apply the k-fold cross-validation method with k = 10, as specified. This method involves separating the input data into 10 sections and running a loop 10 times. Each time, a different section is selected as the test section and the other sections are used for training the classifier. After predicting the labels for the test data, various metrics are calculated. The results yielded by these metrics are discussed in section 1.5, while the cross-validation method is discussed in detail over the following subsections of this report.

### Training the classifier

Our program gives 2 options when it comes to training the classifier. Users can choose either the “build” option, which trains the SVM classifier using the training data as we have seen throughout the semester, or the “load” option, which loads a classifier from external files within the project directory. Note that the “build” option allows users to choose whether or not they want to save the classifiers as files as well. We have implemented this feature because we have noticed that building the classifier is the most time-consuming part of the k-fold cross-validation method and loading it from a file would significantly speed up the process.

### Label Prediction, Metric Calculations and Confusion Matrix

Once the classifier has been trained, we predict the labels for the test data and, considering that we already know the real labels of the test data, we can calculate metrics to quantify the correctness of our results and we generate a confusion matrix. Before calculating the metrics, we must calculate four parameters:

* *True Positive (TP)*, for each label *X*, the number of times a test image is predicted to having label *X* and its real label is *X* as well.
* *True Negative (TN)*, for each label *X*, the number of times a test image is not predicted to having label *X* and its real label is not *X* either.
* *False Positive (FP)*, for each label *X*, the number of times a test image is predicted to having label *X* and its real label is not *X*.
* *False Negative (FN)*, for each label *X*, the number of times a test image is not predicted to having label *X* and its real label is *X*.

From these four parameters, we calculate the following three metrics, which are discussed in section 1.5:

For the last iteration of the cross-validation method, we generate a confusion matrix which allows us to visualize the correctness of our implementation. Results are discussed in section 1.5.

### Runtime Analysis

If we are loading and not building the classifiers, our implementation requires around 25:54 minutes to run all iterations of the cross-validation algorithm. Our code runs for ten iterations, thus, for each iteration, our program takes on average about 2:35 minutes to load the classifier, predict labels for test data calculate metrics and create the confusion matrix for the last iteration. This time seem relatively fast to us, specially considering that we have a total of 22000 data elements to consider.

## K-neighbours Classifier

Implementation for this classifier is exactly the same as for the SVM classifier. The only difference in the code is that we are training or loading a k-neighbours classifier instead of an SVM classifier. We use k=3 because it gives good enough results, as discussed in section 1.5.

Although implementation is identical to that of the SVM classifier, runtime is very different. Our implementation for the k-neighbours classifier takes 1:00:14 hours to run all iterations of the cross-validation algorithm, which means that it takes on average 6:01 minutes for each iteration. It can clearly be seen that runtime performance is significantly worse for k-neighbours compared to SVM. We associate this to the fact that SVM is a deep-learning classifier, while k-neighbours is not (faster performance for SVM is expected).

## Performance Analysis

In this section, we analyse and compare the performance of our implementations of classification based on the metrics calculated for each classifier, as explained in section 1.3.2.

### Classification accuracy

Details of the accuracy for each iteration of the cross-validation loop and for each category in each iteration are printed to the screen during runtime. Global average accuracy values and standard deviations are reported in table 1. We notice that both classifiers achieve excellent accuracy and very small standard deviation. K-neighbours seems to perform slightly better than SVM on average, but this improved performance is a trade-off with respect to the longer runtime.

Table : Average accuracy and standard deviation for both classifiers across validations

|  |  |  |
| --- | --- | --- |
| **Classifier** | **Average Accuracy** | **Standard Deviation** |
| **SVM** | 93.74% | 0.001062 |
| **K-neighbours** | 95.52% | 0.001789 |

### Classification Precision and Recall

Details of the precision and recall for each iteration of the cross-validation loop and for each category in each iteration are printed to the screen during runtime. Global average precision and recall values are reported in table 2. We notice that performance is significantly better for the K-neighbours classifier compared to the SVM classifier. However, once again, this improved performance is a trade-off with respect to the longer runtime.

We can notice that values for precision and recall are, for both classifiers, significantly smaller than the values for accuracy. Nevertheless, this is expected, mainly because the accuracy considers true negatives as well, which significantly boost its value. However, in real life application, we are only interested in true positives because although true negatives are closer to reality than false positives or false negatives, they are not very helpful in correctly labeling test data. This having been said, we can state that precision and recall are in fact more representative of the dataset and more relevant in our application, which involves localization at a later stage.

To add to the previous paragraph, we can state that precision and recall would better reflect performance compared to accuracy in cases where there is a large number of labels to classify images into. To illustrate this, let us consider an example where we have a dataset with 30 labels. A true negative would occur if, for a given test data element, we predict that it is not labeled *X* and, in fact, that is the case. Knowing that a data element has been correctly identified as not bearing one of the 30 labels does not help too much because we are still clueless regarding what its real label is, out of the 29 remaining labels. True negatives may be relevant in some cases where we do not want to make incorrect predictions, but in cases such as this project, where we are only interested in correct label predictions, true negatives become irrelevant. This is why, in the cases explained in this paragraph, accuracy would be a worse reflection of performance compared to precision and recall.

Table : Average precision and recall for both classifiers across validations

|  |  |  |
| --- | --- | --- |
| **Classifier** | **Average Precision** | **Average Recall** |
| **SVM** | 65.59 | 65.36% |
| **K-neighbours** | 76.97% | 75.22% |

### Confusion Matrices

Confusion matrices are generated and displayed for the last iteration of the cross-validation algorithm during runtime. They are also shown in figures 1 and 2. In both cases, we notice that the classifier has more difficulty handling classes that have a larger label number, particularly class 10, “non-motorized\_vehicle”. I DO NOT KNOW HOW TO JUSTIFY THIS

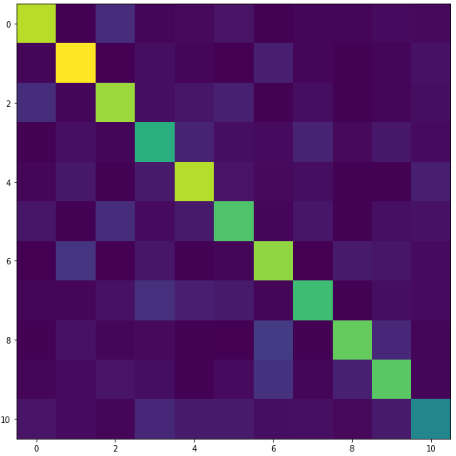


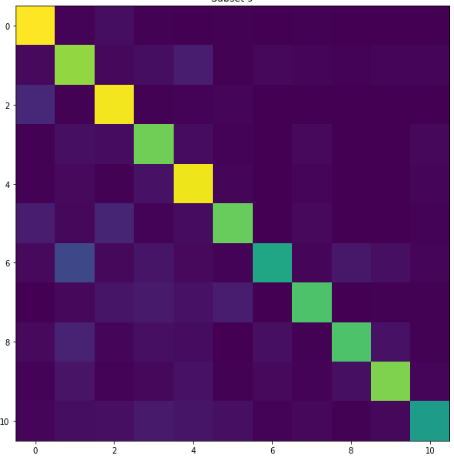
Figure : Confusion matrix for the SVM classifier

Figure : Confusion matrix for the K-neighbours classifier

# Localization