Practicum Assignment 1
(RO47004 / ME41105)
Visual Object Detection

Ronald Ensing, András Pálffy and Dariu Gavrila

Intelligent Vehicles group
Delft University of Technology

November 24, 2020

Document Version 1

TU Delft

About the Assignment

You are expected to do the practicum assignments in student pairs, you receive both one grade (if you do not have a partner, post to the ”find a lab partner” Brightspace forum of this course). A maximum of one group can contain a single student, in case of an odd number of students (if you think this involves you, please contact the Practicum Coordinator before starting the assignment).

Please read through this whole document first such that you have an overview of what you need to do.

This assignment contains *Questions* and *Exercises*:

- You should address all of the questions in a 2-3 page report (excluding plots and figures). *Please provide separate answers for each Question in your report, using the same Question number as in this document.* Your answer should address all the issues raised in the Question, but typically should not be longer than a few lines.
- The Exercises are tasks for you to do, typically implementing a function or performing an experiment. Therefore, first study the relevant provided code before working on an exercise, as the code usually contains comments referring to each specific exercise. *If you do not fully understand the exercise, it may become more clear after reading the relevant code comments!* Do not directly address the exercises in your report. Instead, you should submit your solution code together with the report. Experimental results may be requested in accompanying questions.

You will be graded on:

1. Quality of your answers in the report: Did you answer the Questions correctly, and demonstrate understanding of the issue at hand? All Questions are weighted more-or-less equal.
2. Quality of your code: Does the code work as required? Only then you obtain full credit for the corresponding Question.

Submission

- Before you start, go to the course's Brightspace page, and enroll with your partner in a lab group (found under the 'Collaboration' page).
- To submit, upload **two** items on the Brightspace 'Assignments' page:

pdf attachment	A *pdf* with your report. Do not forget to add your student names and ids on the report.
zip attachment	A *zip* archive with your code for this assignment Do NOT add the data files! They are large and we have them already ...
deadline	**Tuesday, December 8-th 2020, 23:59**

- Note that only a single submission is required for your group, and only your last group submission is kept by Brightspace. **You are responsible for submitting on time**. Do not wait till the last moment to submit your work, and *verify* that your files have been uploaded correctly. Connection problems and forgotten attachments are not valid excuses. The due deadline is automatically checked by Brightspace. If your submission is not on time,

points will be subtracted from your grade. Within 24 hours after the deadline, one point is subtracted. Between 24 and 48 hours, 2 points are subtracted, and so forth.

- You are only allowed to hand in work (code, report) performed by you and your practicum partner during this course this year. Passing other work as your group's work is considered *scientific fraud* (even if it consists a single line of code that you are "borrowing"). You are equally not permitted to share your work with anyone else outside of your group (including but not limited to WhatsApp, other social media, email, internet). This applies both to during the course and thereafter. Sharing your work will be considered as *abetting scientific fraud*. We have (and will use) manual and automatic means to detect possible instances of *scientific fraud* or *abetting scientific fraud*. We will always report them to the Examination Committee. Sanctions, if the suspicions materialize, could involve a failing grade for the course, an entry in the Academic Dishonesty registry of the Faculty up to expulsion from the MS program. It is not worth it!
- If code is submitted that was written with ill intent, e.g. to manipulate files in the user's home directory that were not specified by the task, you will also be reported to the Examination Committee.

Getting Assistance

The primary occasion to obtain help with this assignment is during the practicum contact hours of the course. An instructor and/or student assistant(s) will be present during the practicum to give you feedback and support. If you find errors, ambiguously phrased exercises, or have another question about this practicum assignment, please use the Brightspace practicum support forum. This way, all students can benefit from the questions and answers equally. If you cannot discuss your issue on the forum, please contact the Practicum Coordinator directly.

Practicum Coordinator: Ronald Ensing (R.M.Ensing@tudelft.nl)

Copyright Notice

Good Luck!

Visual Object Detection

In this practicum assignment we will study and evaluate feature extraction and pattern classification algorithms that can be used for video-based pedestrian detection.

Our first goal is to investigate which features and classifiers give the best result in distinguishing rectangular region proposals as belonging either to the 'pedestrian' or 'non-pedestrian' class. We have a dataset containing 3000 samples (1500 pedestrian and 1500 non-pedestrian) for training. For testing, 1000 samples (500 pedestrian and 500 non-pedestrian) are provided, see Figure 1. The pedestrian and non-pedestrian samples are provided in the file `new_data_int.mat` and the true labels are provided in the file `new_data_labels.mat`.

Figure 1: Examples of pedestrian and non-pedestrian class samples in the training data.

Getting started

Download the provided assignment files from Brightspace and unzip them in the `lab1` directory. The files `assignment_features.py`, `assignment_classification.py` and `assignment_detection.py` are the main files, scripts corresponding to the three assignments in this lab. All other python files contain functions that are called in the main scripts. In the exercises you will have to complete code in both the main scripts and the functions.

1 Features

Let us start by exploring the dataset a bit. Note that these 96 by 48 pixel samples have been reshaped to 1 by 4608 vectors.

> *Exercise 1.1.* Visualize five pedestrian and five background samples from the training data. In order to do this, you have to implement the function `int_to_image` in `int_to_image.py`. Note that the images are stored as a 1-dimensional array with a length of $rows \times cols$. Use the numpy function `reshape` to convert the data back to two-dimensional images. Include all plots in your report.

Classifiers use "features" to make decisions. Features are descriptive properties of the input

sample, in this case, an image. Features could be the original pixel intensity values, or derived properties thereof.

Select for the remainder of this question two pedestrian samples and two background samples from the 10 samples you chose originally, to visualize further.

Exercise 1.2. Complete the code in `visualize_hog.py` to calculate the HOG features and HOG visualization for selected samples using skimage[a]. Use 8 orientations, (8, 8) pixels per cell and (2,2) cells per block. Plot the HOG visualization for these samples. Now try 4 and 32 orientations instead and re-plot the selected samples. Include all plots in your report.

[a] https://scikit-image.org/docs/dev/auto_examples/features_detection/plot_hog.html

Exercise 1.3. Now we will observe the influence of the pixels per cell parameter. Use 8 orientations and (2,2) cells per block. Now try (4, 4), and (16, 16) as the number of pixels per cell. Plot these HOG visualizations for the same samples in `visualize_hog.py` as well. Include all plots in your report.

Question 1.1. Give a formula to calculate the length of a HOG feature vector given the number of orientations (O), pixel per cell (C), cell per block (B), and image size (W, H for width and height). Assume square cell and block shape, block step = 1, and that the image size is dividable with the cell size. C and B are per dimensions, so e.g B = 2 means 2x2 cells in a Block.

For the remainder of this practicum assignment, we will use HOG features with the number of orientations equal to 8, numbers of pixels per cell equal to (8, 8) and number of cells in a block equal to (2, 2).

Exercise 1.4. Compute the HOG features for all samples of the dataset.

Exercise 1.5. Now we will extract features with a pre-trained Convolutional Neural Network (CNN) titled MobileNet. Finish the code section in `assignment_features.py` to calculate the MobileNet features of the dataset samples (see Appendix A).

2 Pedestrian classification

For the three feature types (Intensity, HOG and MobileNet) we want to train the following classifiers:

1. Linear Support Vector Machine (SVM) (see Appendix C)
2. k-Nearest Neighbors (k-NN) [1]

[1] https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

Exercise 2.1. Complete the code for training the SVM classifier in `assignment_classification.py`. using svm.SVC of sklearn. Store the trained SVM in the variable `svm` that will be used in the remaining exercises. Train the SVM on all three feature sets using the samples from the training set.

Exercise 2.2. Complete the code for training the k-NN in `assignment_classification.py` using the neighbors.KNeighborsClassifier of sklearn. Until further notice, use k = 1 (nearest neigbor).

We will in the remainder of this Question consider performance in terms of classification error (i.e. percentage of wrongly classified samples).

Exercise 2.3. Use sklearn[a] to plot the confusion matrices for all (six) feature-classifier combinations on the test data. Include all plots in your report.

[a] https://scikit-learn.org/stable/modules/generated/sklearn.metrics.plot_confusion_matrix.html

Question 2.1. What do you observe in terms of performance for the various feature-classifier combinations? Do some features consistently perform better than others? Does one classifier consistently outperform the other? What is the best feature-classifier combination overall? Motivate your answers.

Robustness to Illumination Conditions Robustness to changes in illumination is a desirable property for a classification system.

Exercise 2.4. Construct a new, alternate test set by adding pixel intensity of 30 to the original test set (clipping at maximum intensity value of 255) to simulate images taken in conditions with more sunlight. Calculate and plot the six confusion matrices for this alternate dataset. Include all plots in your report.

Question 2.2. What do you observe in performance when comparing the same feature-classifier combinations on the original test set and on the brightness-adjusted test set. Please elaborate to what degree this is to be expected.

Curse of Dimensionality Some classifiers may especially be ill-equipped to handle high-dimensional data. Principal Component Analysis (PCA) is one method to reduce the data dimensionality by projecting it into a linear subspace that maintains most of the variance in the data, see Appendix B.

Exercise 2.5. For each of the three feature types, reduce the dimensionality of their samples in the dataset to 20 by PCA, and train new SVM and k-NN classifiers. Compute and plot the confusion matrices for the six feature-classifier combinations on the test data.

Question 2.3. What do you observe that dimensionality reduction achieves in terms of performance, based on the confusion matrices? Consider the result of a particular feature-classifier combination here compared to its results in the previous exercise.

Parameter selection Both the SVM and the k-NN have parameters that we can set before training the model, and which affect the final performance. In case of the SVM, this is the training parameter $C \in \mathbb{R}$, and for the k-NN we have k, the number of neighbors.

Till now we have kept these parameters fixed, but here we will experiment with optimizing them. We will consider for this optimization only HOG features. For the k-NN classifier we will apply PCA-20 first, for the SVM classifier we will use the original HOG features.

Exercise 2.6. Evaluate the k-NN classifier for various values of k on the test data. Plot a curve of classifier performance vs. parameter k.

Question 2.4. What is the effect of increasing k in a k-NN classifier, in general? Describe the shape of the curve you obtained.

Exercise 2.7. Run the code block that evaluates the SVM classifier for various values of C on test data.

Question 2.5. What is the effect you expect from increasing C in a SVM classifier, in general? Describe the shape of the curve you obtained.

ROC Curves can also be used to evaluate the classifiers (on y-axis: true positive rate [0,1], on x-axis: false positive rate). This requires logging the decision values (classifier outputs) on the test dataset.

Exercise 2.8. Complete the code for plotting the ROC curves in `assignment_classification.py`. Look at Appendix D and Figure 4 for more information on the ROC plots.

Question 2.6. We will only consider the SVM classifier here, without the use of PCA and with default parameter $C = 2$. With which feature performs the SVM classifier best based on the ROC plots? Include a single plot of the three corresponding ROC curves in your report to support your answer.

3 Pedestrian detection

Up to now, we have looked at pedestrian *classification*, i.e. deciding for a provided image sample whether it belongs to the pedestrian class or to the non-pedestrian class. In this last question, we move to pedestrian *detection*, which considers a broader question: In a given image, where are pedestrians located? See Figure 2. A simple and effective method to answer this question is to generate large set of candidate region proposals of the right size and shape covering the entire image, and classify each of these regions using a trained pedestrian classifier (i.e. like the samples of the test set in previous question).

In this exercise, you will train an optimized classifier and will try it out on a short video clip `pedestrian.mp4`. We provide code in `assignment_detection.py` which loads the video, crop region proposals in a sliding window style, and apply the given classifier to each of them. If a proposal is classified as a pedestrian, the code draws a green bounding box there. A video of the frames with detections overlaid is saved.

Figure 2: Pedestrian detection by classifying many region proposals. In this example, the green rectangles correspond to regions classified as pedestrians. Our aim is to avoid false positives and false negatives, though in practice misclassification errors may occur.

Exercise 3.1. Find your best-performing SVM-HOG and SVM-MobileNet combinations based on the classification test set. Either use the best performing feature-classifier combinations that you found in Question 2 (gives half Question credit) or perform a wider parameter optimization (HOG features, optional use of PCA with a certain reduced dimension, value of C - gives up to full Question credit). Plot the two resulting confusion matrices for the optimized classifiers. *Do not spend an excessive amount of time on parameter optimization.*

Question 3.1. Explain how you chose the optimized parameters including your decision (not) to use PCA.

Exercise 3.2. Now apply both optimized SVM classifier versions (HOG, MobileNet) on the region proposals of the video sequence, and visualize the bounding box regions which are considered 'pedestrian' (like in Figure 2). Create two video files, one for both classifiers. To do this, use the `Detector` class as in the example.

Question 3.2. Study the pedestrian detection results qualitatively (so by looking at the results, as opposed to quantitative evaluation using error statistics and ROC curves). What kind of false positives and false negatives do you observe? Show example screenshots. What would you suggest to counter these errors, and improve the results? Is there some noticeable difference between the outputs of the two SVM classifier versions, or do they perform similarly?

Acknowledgements

We thank Julian Kooij and Markus Enzweiler for their contributions in creating earlier versions of this Practicum Assignment.

Appendix

A MobileNet

A.1 Convolutional Neural Networks

Convolutional Neural Networks (CNN) are a sub-class of Artificial Neural Networks (ANNs) with a specific network architecture. They are widely used in modern machine learning, especially in computer vision applications. We will only provide a basic description of CNNs here, as they are not a main focus of this course. We provide ready-to-use code.

In general, in a CNN, the image is passed through a series of convolutional, pooling layers and finally a block of fully connected layers. The series of convolutional and pooling layers together are often referred as ”base” or ”backbone” network, while the block of the fully connected layers is often called ”the head”. The output of a CNN is dependent on the task. E.g. in case of classification, it can be a probability assigned to each possible classes (e.g. ”pedestrian”, and ”not-pedestrian”), see Figure 3. In a way, a CNN based Image classifier is *both* a feature extractor and a classifier: the base extract the features, and the head makes an output (i.e. a classification) based on them.

Note that in contrast to HOG features, which are hand crafted features, a CNN based image classifier *learns* what features to extract as the weights in the base network (which extracts the features) are not predefined/designed.

A.2 Transfer learning

To train a CNN, weights in the convolutional and fully connected layers (i.e. both the base and the head) needs to be tuned to match the task. This is done by a method called ”back-propagation” and often requires large amount of training samples, especially if the base network also needs to be trained (remember: number of required training samples are heavily dependent on the number of trained parameters).

However, this is not always the case. As presented above, in many of the widely used networks there is a clear border between the feature extraction part (base network) and the classification part (head, fully connected layers). This also means we can separate them and even replace one of them. For example, we can take a base network trained on some vision based task, and we can replace its head with a new one tuned to our needs. This approach does reduce the required number of training samples significantly as the base network is already trained. We only have to train the classifier part, which uses the features exported by the base network. This is method is often called ”transfer learning” as we ”transfer” knowledge from a task where the base network is already learned a lot on a task similar to ours. To further simplify the approach, the classifier part does not need to strictly be a neural network. I.e., in this assignment, we will take a pre-trained base network, remove its fully connected layers, and replace them with a simple SVM classifier.

To do this, we will use Mobilenet v2.

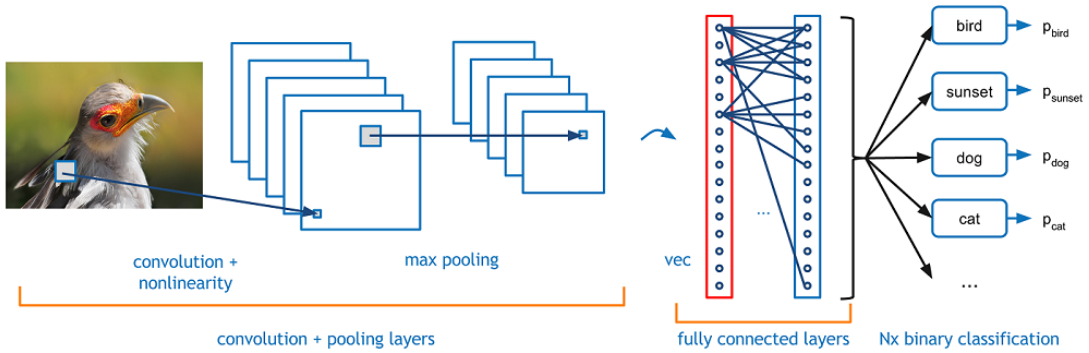

Figure 3: Structure of a typical CNN. The first block does convolutions and pooling, while a second block, the fully connected layers generates the final output. Source of image: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks.

A.3 Mobilenet V2

Mobilenets is a class of efficient, light weight neural networks for embedded vision based applications, e.g. object detection with a mobile phone. They were designed and published by Google [2]. In this assignment, when we say MobileNet, we mean a specific network, MobileNetV2 [3], pretrained on the ImageNet [4] dataset to classify images into 1000 classes. That means it has already seen literally millions of pictures and extracted several low and high level vision related features. To classify the inputs to 1000 classes in the original task, the authors used several fully connected layers. We do not need these, as our task is different (and simpler): we only need two classes.

Most Deep Learning libraries (i.e. Pytorch, Tensorflow) come with "famous" neural networks (i.e. Resnet, MobileNet, etc.) available, i.e. it is one line to load them. E.g. using tensorflow:

```
model = MobileNet(weights='imagenet', include_top=False)
```

loads a pre-trained version of Mobilenet (note the parameter `weights='imagenet'` : it was trained on that dataset) but we cut off the head/fully connected layers (note the parameter `include_top = False`. This means we only need the base network).

Thus the variable `model` is now a pre-trained Mobilenet without a head. As such, it outputs features instead of classifications:

```
features = model(input)
```

which can be directly used in a classifier like SVM or k-NN.

B Principal Component Analysis (PCA)

PCA is a technique for reducing the dimensionality of the feature space. By analyzing how the data is distributed from training samples, PCA computes a linear subspace that maintains

[2] https://arxiv.org/pdf/1704.04861.pdf
[3] https://keras.io/api/applications/mobilenet/
[4] https://http://www.image-net.org/

most of the variance of the input data. New data samples can later also be projected to this subspace, which is also sometimes referred to as the 'PCA subspace', or 'PCA projection'.

B.1 Computing the subspace

Consider that we have N data samples $\mathbf{x}_1, \cdots, \mathbf{x}_N$ in an M-dimensional feature space, given as a single $M \times N$ data matrix $\mathbf{X}$ with each column a data sample. We can compute a D-dimensional PCA subspace of $\mathbf{X}$ where $D \leq M$ to obtain a D dimensional data representation which maintains most of the variance. The subspace will be defined as a linear projection, consisting of an $M \times D$ transformation matrix $\mathbf{W}$, and the M-dimensional mean data vector $\mathbf{m}$.

First compute the mean data vector $\mathbf{m}$,

$$\mathbf{m} = \frac{1}{N} \sum_{j=1}^{N} \mathbf{x}_j. \tag{1}$$

The mean is then subtracted from the data to get the zero-mean $M \times N$ data matrix $\overline{\mathbf{X}}$ (with columns $\overline{\mathbf{x}}_j$), which is then used to compute the $M \times M$ covariance matrix $\mathbf{C}$,

$$\overline{\mathbf{x}}_j = \mathbf{x}_j - \mathbf{m}, \qquad \forall j, \text{where } 1 \leq j \leq N \tag{2}$$

$$\mathbf{C} = \overline{\mathbf{X}} \times \overline{\mathbf{X}}^\top. \tag{3}$$

Next, compute the eigen-vectors $\mathbf{w}_i$ and corresponding eigen-values λ_i of the covariance matrix $\mathbf{C}$. Recall that the eigen-vector and eigen-values fulfill the following property,

$$\mathbf{C}\mathbf{w}_i = \mathbf{w}_i \lambda_i. \tag{4}$$

The eigen-vectors should be sorted such that the i-th component is the eigen-vector with the i-th *largest* eigen-value λ_i, hence the first eigen-vector $\mathbf{w}_1$ has the largest eigen-value λ_1. These vectors M-dimensional $\mathbf{w}_i$ are called the *principal components* of the data $\mathbf{X}$, and are all orthonormal to each other.

The eigen-value λ_i is proportional to the amount of variance the data has along PCA subspace dimension i, hence fraction of variance kept by principal component i is $\lambda_i / \sum_{j=1}^{M} \lambda_j$. Therefore, we only keep the first D eigen-vectors $\mathbf{w}_1, \cdots, \mathbf{w}_D$, as they correspond the D dimensions that retain most of the data variance. These D eigenvectors can be represented as single $M \times D$ matrix $\mathbf{W}$, where the i-th column is the i-th component $\mathbf{w}_i$.

B.2 Projecting data to the subspace

Once the subpace is computed, we can apply the transformation to reduce the dimensionality of any M-dimensional data vector $\mathbf{x} \in \mathbb{R}^M$ to its reduced D-dimensional 'PCA' representation $\mathbf{x}^\star \in \mathbb{R}^D$ using the following linear equation (assuming all vectors are column vectors),

$$\mathbf{x}^\star = \mathbf{W}^\top (\mathbf{x} - \mathbf{m}). \tag{5}$$

B.3 Back-projecting from the subspace

The inverse back-projection can also easily be achieved,

$$\mathbf{x}' = \mathbf{W}\mathbf{x}^\star + \mathbf{m} \tag{6}$$

such that $\mathbf{x}'$ is the (approximate) reconstruction in the original M-dimensional feature space. If we keep all dimensions in the projection, such that $D = M$, then $\mathbf{W} \times \mathbf{W}^\top = I$. In other words, $\mathbf{W}$ is an orthonormal projection and therefore its transpose is its inverse $\mathbf{W}^\top = \mathbf{W}^{-1}$. However, typically we use $D \ll M$ so back-projection does not restore the original feature space exactly.

C Linear Support Vector Machine (SVM) classifier

Support Vector Machines are an advanced topic in Machine Learning. Linear SVMs on two-class data, form the basis of more complicated approaches.

A Linear SVM learns a hyperplane in the feature space that separates the data points of each class with maximal margin. The model parameters of the hyperplane are the M-dimensional weight vector $\mathbf{w}$ and a bias b which define the normal and offset of the plane. To test a new M-dimensional feature vector $\mathbf{x}$, the Linear SVM computes the following decision value:

$$d(\mathbf{x}) = \mathbf{w}^\top \cdot \mathbf{x} + b \tag{7}$$

The sign of this decision value determines the assigned class label, i.e. either it is assigned to the positive (i.e. pedestrian) class, $d(\mathbf{x}) \geq 0$, or to the negative (non-pedestrian) class, $d(\mathbf{x}) < 0$.

For learning Linear SVM parameters, built-in sklearn functions can be used.

Note that there is a *training* parameter $C \in \mathbb{R}$) which influences how the optimizer computes the linear decision boundary: For large C, the optimizer tries very hard to separate both classes, which means that its boundary is sensitive to outliers. For low C it results in a *soft margin* where a few samples may lie close to or on the wrong side of the boundary, if this enables a wider margin to separate most data points of both classes. While C is used during training, it is not part of the learned model in Equation (7).

Non-linear transformations of the input can be applied using certain kernels if the data is not linearly separable. A commonly used kernel is the 'Radial Basis Function': https://scikit-learn.org/stable/modules/kernel_approximation.html.

D Receiver Operating Characteristic (ROC) curves

Instead of reporting a single test error value for a fixed decision threshold, one can also consider changing the threshold to trade-off different types of classification error. Lowering the threshold results in more test cases being assigned to the positive (e.g. pedestrian) class, while increasing it results in more samples classified negatively (e.g. non-pedestrian). This trade-off is then reflected in the False Positive Rate (FPR) and True Positive Rate (TPR), both of which are numbers between 0 and 1.

Let d_i be the classifier's decision value for test sample i, and $y_i \in \{-1, +1\}$ the true class label of the sample. Also, let the function $\mathrm{count}_i[x(i)]$ count the number of samples for which

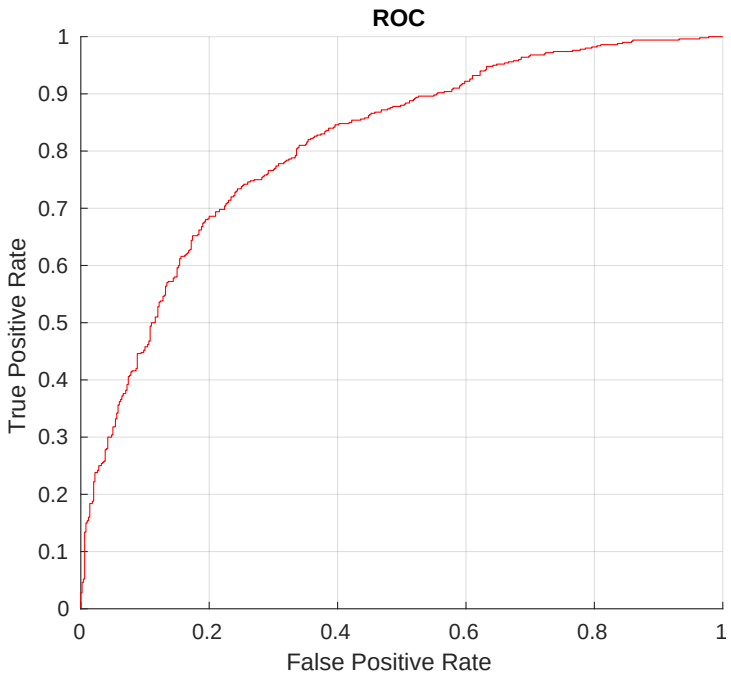

Figure 4: Example of an ROC curve for a certain classifier. We can see for instance that we obtain a True Positive Rate of about 70% (e.g. actual pedestrians classified as pedestrians) at a False Positive Rate of 20% (e.g. actual non-pedestrians classified as pedestrians).

condition x holds. Then the TPR and FPR for a given threshold τ are expressed as,

$$P = \text{count}_i[y_i \geq 0] \quad \text{number of samples in positive class} \tag{8}$$

$$N = \text{count}_i[y_i < 0] \quad \text{number of samples in negative class} \tag{9}$$

$$TP(\tau) = \text{count}_i[(d_i \geq \tau) \wedge (y_i \geq 0)] \quad \text{number of positive samples classified as positive} \tag{10}$$

$$FP(\tau) = \text{count}_i[(d_i \geq \tau) \wedge (y_i < 0)] \quad \text{number of negative samples classified as positive} \tag{11}$$

$$TPR(\tau) = \frac{TP(\tau)}{P} \qquad FPR(\tau) = \frac{FP(\tau)}{N} \quad \text{true and false positive rates.} \tag{12}$$

By computing the TPR and FPR for varying thresholds, one can create a so-called Receiver-Operating Characteristic (ROC) curve. Figure 4 shows an example of an ROC plot for a single classifier. The ROC curve always starts at (0,0), which corresponds to setting the threshold so high that all test samples are assigned to the negative class, hence there would be no false positives, but also no true positives. The other extreme is obtained when the threshold is so low that everything is assigned to the positive class, in which case both FPR and TPR become one as the curves reaches the top-right corner at (1,1). The curve of an ideal classifier would touch the top-left corner, corresponding to TPR of 1 for a FPR of 0.