Pothole detection

Final Project Report

Lisa Achard
Université Paris-Saclay
lisa.achard@etu-upsaclay.fr

Manon Chartrin
Université Paris-Saclay
manon.chartrin@etu-upsaclay.fr

Sovanleng Ly
Université Paris-Saclay
sovanleng.ly@etu-upsaclay.fr

Abstract

Nowadays, deep learning methods are commonly used in computer vision tasks. It has become possible to achieve high performance easily with such models. However, our focus will be here on classical methods. We will indeed present a methodology to perform pothole detection on a set of images, mainly through feature extraction methods. This work does not pretend to be innovating but some effort was made to get the best results using classical methods. Their effectiveness was measured through binary classification: the presence or absence of potholes. Our results show that these methods don't outperform deep learning baselines, but some of them were still able to achieve similar performance, and progress was also seen after processing is done on images. The implemented code is publicly available.

1. Motivation

Roads are an important part of a country's infrastructure. Having proper roads allows inhabitants to stay connected with the community. For instance, in France, roads represent 1.1 million of kilometers. Therefore, **keeping track of roads quality can be challenging**. Among the various types of road damage, potholes are particularly problematic due to their impact on vehicle safety, driving comfort and maintenance costs. That is why **detecting potholes efficiently and accurately** is **essential for improving road safety**. One way of detecting potholes is through images. Some vehicles can have camera enabling them to take pictures of potholes, and using computer vision on these images, it is possible to detect them.

2. Problem definition

Given roads images $I \in \mathbb{R}^{p \times p}$, labeled with $Y \in [0, 1]$, containing potholes (1) or not (0), the goal is to predict Y knowing I. This is therefore a **binary classification problem**. However, predicting directly Y with I can be difficult, so feature extraction methods are used to get features $Z = g(I)$ from which predictions are made. The predicted results $\hat{Y} \in [0, 1]$ can thus be defined as $\hat{Y} = f(Z) = f(g(I))$.

In the study, the metric to maximize is the **F1-score**. It is defined as $2 \times \frac{\text{Recall} \times \text{Precision}}{\text{Recall} + \text{Precision}}$. It is an interesting metric as it is a **trade-off between recall and precision**, where precision is the number of positives that are truly positive (precision $= \frac{TP}{TP+FP}$), and recall is the number of actual positives that were correctly predicted (recall $= \frac{TP}{TP+FN}$). The **dataset** is approximately **balanced**, so basic F1-score was sufficient, and weighted F1-score was not used.

3. Related work

Pothole detection using computer vision techniques has been a well-researched area, with studies employing both classical and deep learning-based approaches. Traditional methods primarily focus on handcrafted feature extraction, while recent advancements leverage deep neural networks for more accurate detection.

Ryu et al. (2015) [21] introduced a **histogram-based thresholding** method coupled with **morphological filtering** to extract pothole contours, achieving a high detection accuracy of 91%. Similarly, Wang et al. (2017) [27] proposed a wavelet energy field-based method that combined grayscale and texture features to segment potholes from pavement images, achieving an accuracy of 86%. Their approach employed morphological processing and a Markov random field model to enhance the detection process. Akagic et al. (2017) [1] introduced RGB color space manipulation for pothole detection, utilizing dynamic road pixel selection to enhance visibility. Similarly, our project employs **Gaussian filtering** to smooth road images while preserving key features.

Makone and Rathod (2002) [13] used **mean shift-based filtering** for **edge detection**, which laid the foundation for later works focusing on boundary refinement techniques. Building on this, Koch and Brilakis (2011) [10] used histogram-based thresholding, morphological operations, and elliptic regression to enhance pothole boundary detection. Similarly, our project integrates **Sobel and Canny edge detection** to highlight intensity variations, making potholes more distinguishable. Expanding on the need for improved segmentation, Buza et al. (2013) [2] applied **Otsu's thresholding** and spectral clustering to segment potholes from road surfaces. Our project follows this direction by incorporating **Otsu's thresholding** and **adaptive thresholding** to handle lighting variations and enhance segmentation accuracy. To complement thresholding-based methods, Schiopu et al. (2016) [23] leveraged **histogram-based thresholding** and **geometric properties** to classify potholes. Their approach motivated our integration of the **Histogram of Oriented Gradients (HOG)** for structural feature extraction.

Taking a different perspective, Yousaf et al. (2018) [28] proposed a **bag-of-words (BoW)** approach combined with **Scale-Invariant Feature Transform (SIFT)** to establish a visual vocabulary for pavement surfaces. They employed a Support Vector Machine (SVM) to train and test histograms of extracted words, achieving 95.7% accuracy in pothole detection and 91.4% accuracy in pothole localization. This highlights the effectiveness of keypoint-based methods for pothole detection. Inspired by this, we explore **SIFT, ORB and Harris Corner Detection** for keypoint identification, improving feature extraction in our approach. Building on this, Silveira Rodriguez et al. (2022) [19] implemented Haar Wavelet Transform (HWT) on accelerometer signals for pothole detection. While we do not use sensor data, our work applies **Gabor filters** and **Local Binary Patterns (LBP)** to capture road texture variations.

While deep learning models such as **CNNs** have demonstrated high accuracy, our project prioritizes classical methods for interpretability and computational efficiency. However, for benchmarking, we still used some neural networks but we principally experimented with Logistic Regression and Random Forest as classifiers, similar to hybrid approaches in previous studies.

Finally, Fan et al. (2022) [6] also proposed a stereo vision-based method, combining semi-global matching, disparity transformation, and adaptive thresholding, achieving 98% detection accuracy. While we do not use stereo vision, their study supports the integration of multiple feature extraction techniques for enhanced robustness.

4. Methodology

The **methodology** used can be defined in **three main steps:** Image Processing, Feature Extraction and Classification.

4.1. Image Processing

The first step is **Image Processing.** The goal is to improve the images as much as possible before any feature extraction method is applied. Filters such as Gaussian, Median, or Bilateral filters, coupled with histogram equalization, are therefore applied. In the project, the folder Data processing is dedicated to image processing. The methods' implementation can be found in the **class *ImagesProcessing*** of the file `images_processing_class.py`. Its initialization allows to load and resize the images (256×256 by default), and its methods allow to process them.

Three filters can be applied on the images. The first one is the **Gaussian filter**. It is a linear filter used to smooth an image and reduce noise, while preserving the edges to some extent. It convolves the image using a Gaussian kernel, which allows to give more weights to pixels closer to the center because of the Gaussian distribution. It can be seen therefore as an improvement of the mean filter. The second one is the **Median filter**, a non-linear filter, that preserves the edges better than the Gaussian filter. The value of a pixel is replaced by the median value of its $N \times N$ neighborhood. It is the best choice to remove impulse noise (salt-and-pepper noise). The last filter is the **Bilateral filter.** It is a non linear filter that smooths an image while preserving edges better than the two previous filters. Indeed, while it also uses the Gaussian kernel, it uses an extra component, a weighting function based on intensity similarity, to ensure that edges are not blurred. Apart from filters, **histogram equalization** can also be performed on the images. It enhances contrast by redistributing pixel intensity values to use the full range of intensities. However, it should be used carefully, as in some cases, it might degrade the image.

Now, a question arises: **which processing should we do and which parameters should we choose?**

First, to determine which processing method to apply for each feature extraction method, a literature review was conducted alongside an analysis of which filters could be the most useful. Therefore, for each method, a **processing step was chosen.** It was indeed found the following processing methods: Bilateral filter and CLAHE histogram for SIFT [17], Bilateral filter for ORB, Harris, Edge, Gabor and Adaptive, and Gaussian filter for LBP [25] and Otsu's thresholding. The Bilateral Filter was chosen for methods that need the edges to be preserved the best, as this filter reduces noise while keeping edges sharp. In contrast, the Gaussian Filter softens the entire image [7]. The Bilateral Filter achieves noise reduction by taking into account both pixel location and intensity, making it more effective for preserving important details [26]. This is particularly efficient for edge detection techniques like Canny, which require sharp edge preservation. Recent studies have shown that applying the Bilateral Filter before Canny Edge Detection improves edge accuracy compared to Gaussian smoothing [9]. On the other hand, Gaussian filter reduces image noise and smooth intensity variation which leads to more stable and accurate feature extraction with LBP [14]. Similarly, the tutorial from OpenCV demonstrates applying a Gaussian filter to an image to remove noise, followed by Otsu's thresholding to achieve optimal binarization [16]. Apart from SIFT that uses a CLAHE histogram equalization as found in one article [17], for the other methods, standard and CLAHE histogram equalizations were tested, along with no equalization to find the best approach. **Not trying out each processing method** is of course a **limitation**: another filter than the one chosen might have shown better results. The idea was in fact here to **reduce the number of configurations tried.**

Then, once each processing method is chosen for each feature extraction method arbitrarily, **multiple parameters are tried** in the file `images_processing.ipynb`. For the Gaussian filter for instance, one of this parameter is the kernel size. The idea is for a given set of parameters, apply the image processing, get the features from the feature extraction method, and then perform Logistic Regression on the training set, and see which parameters have the best performance on the validation set (*Figure 1*). The best parameters are stored in a json file untitled `best_configs_processing.json` which allows to reuse those configurations without running the notebook `images_processing.ipynb` each time.

4.2. Feature Extraction

The second step is **Feature Extraction**; the folder features_extraction is dedicated to it. Multiple feature extraction algorithms are used. Their implementation can be found in the class *FeatureExtraction* of the file `feature_extraction_class.py`. Just like for the image processing methods, **multiple parameters were tried to get the optimal ones** (*Figure 1*). The file `feature_extraction.ipynb` allows to find the best hyperparameters for each method, and stores those parameters in the file `best_configs_methods.json`.

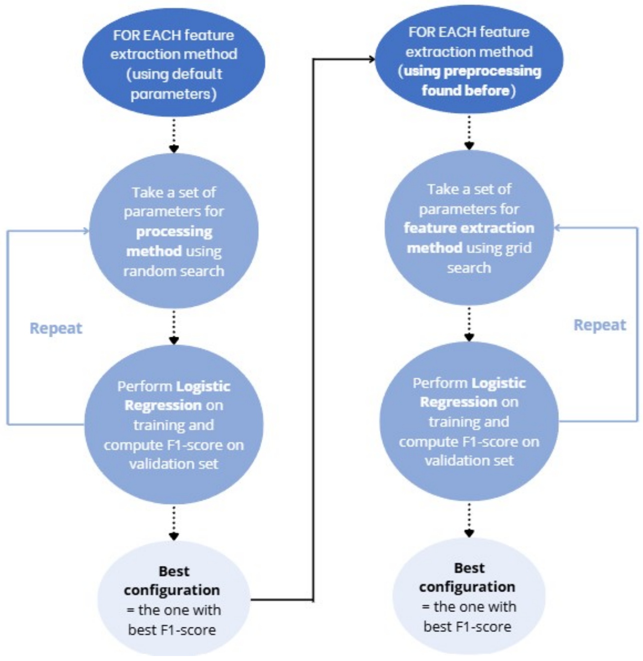
The methods used can be briefly listed as follows.

4.2.1. Low-Level Features

Low-Level Feature Extraction techniques focus on basic image characteristics like **edges, corners, or points of interest.**

- **1. Interest point detection algorithms**

Interest point detection algorithms focus on finding distinctive points that are easy to match across images.



FOR EACH feature extraction method (using default parameters)
Take a set of parameters for processing method using random search
Perform Logistic Regression on training and compute F1-score on validation set
Best configuration = the one with best F1-score
Repeat
FOR EACH feature extraction method (using preprocessing found before)
Take a set of parameters for feature extraction method using grid search
Perform Logistic Regression on training and compute F1-score on validation set
Best configuration = the one with best F1-score
Repeat

Figure 1. Methodology to find best parameters for image processing and best parameters for feature extraction methods. The choice of random and grid search is arbitrary. The idea behind the use of random search is to be able to get a greedy optimal solution if lots of parameters are used in a shorter amount of time.

- **SIFT (Scale-Invariant Feature Transform), 1999** [12]

SIFT first applies a convolution to all the images, with different Gaussian kernels, using different scales $\sigma_n = k^n\sigma_0$ (usually $k = \sqrt{2}$). If $I(x, y)$ is the original image and $G(x, y, \sigma)$ is the Gaussian function, then the blurred version of the image can be thus defined as:

$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$

The **Difference of Gaussians (DoG)** is then computed by subtracting images from two consecutive scales:

$$D(x, y, \sigma) = L(x, y, k\sigma) - L(x, y, \sigma)$$

Keypoints are defined as **local extrema in the DoG images.** To find them, each pixel in a DoG image is compared to its 26 neighbors: 8 pixels from the same image, 9 pixels in the previous scale and 9 pixels in the next scale. It is considered a keypoint if this point is an extremum in this neighborhood. Some techniques are also used afterwards to remove some unstable or weak keypoints such as the one along edges.

Once the keypoints are detected, **descriptors** are created to describe them and so that they can be matched across different images. To define them, for each pixel belonging to a 16×16 grid around a keypoint, are first computed the gradient magnitude M and orientation θ using finite differences. If we define $G_x = L(x+1, y) - L(x-1, y)$ and $G_y = L(x, y+1) - L(x, y-1)$ the gradients respectively in X and Y direction, then magnitude and orientation can defined as:

$$M(x, y) = \sqrt{G_x^2 + G_y^2}; \theta(x, y) = tan^{-1}(\frac{G_y}{G_x})$$

The 16×16 grid is divided into 16 smaller 4×4 cells so that each 4×4 region has its own orientation histogram, each containing 8 orientation bins. Therefore, each cell gives 8 orientation bins, and consequently the keypoint descriptor contains $8 \times 16 = 128$ values.

- ORB (Oriented FAST and Rotated BRIEF), 2011 [5]

ORB is a fast and free **alternative to SIFT and SURF**. It uses the **FAST algorithm** [20] to detect keypoints. FAST identifies corners by analyzing around each pixel a small circular neighborhood of 16 pixels. If N ($N = 9$ typically) contiguous pixels in the circle are all darker or brighter by a threshold compared to a pixel, then the pixel is considered a corner. FAST is efficient as instead of checking all pixels in the circle, it first checks if 4 of these pixels meet the condition, and if it is the case, the entire circle is checked.

- **Harris corner detection**, 1988 [8]

Harris corner detection is an algorithm that allows to identify corners in an image. Corners are points where there are significant changes in intensity in **multiple directions**, making them useful for feature detection and tracking.

First, it computes the matrix M, defined as:

$$M = \sum_{x,y} w(x, y) \begin{bmatrix} I_x^2 & I_x I_y \\ I_x I_y & I_y^2 \end{bmatrix}$$

where I_x and I_y are derivatives of the image in the X and Y direction:

$$I_x = \frac{\partial I}{\partial x}, \quad I_y = \frac{\partial I}{\partial y};$$

$w(x, y)$ is a window function (Gaussian for instance) that gives more weight to central pixels.

It then computes the Harris corner score:

$$R = det(M) - ktrace(M)^2$$

Typically k is equal to 0.04 - 0.06. The **values of R are positive near a corner**, negative near a contour, and low in a region of constant intensity.

- **2. Edge detection algorithms**

Edge detection algorithms focus on identifying object boundaries where there are significant changes in intensity. These methods help extract key features from images. Edge detection can be divided into **gradient-based methods** and **second derivative methods**.

- Gradient-Based Edge Detectors

Gradient-based methods compute the **first derivative** of the image to detect areas of high intensity change. They are **effective at finding edges** but tend to be **sensitive to noise**.

The **Canny Edge Detector (1986)** [3] is a popular method for detecting edges. It first applies **Gaussian smoothing** to reduce noise. Then, it computes the **image gradient** to find intensity changes. After that, **non-maximum suppression** is used to thin the edges. Finally, a **double thresholding** step classifies edges as strong or weak, and weak edges connected to strong ones are preserved.

The **Sobel Operator (1968)** [24] highlights edges by applying horizontal and vertical filters to detect changes in intensity.

The gradient magnitude is computed as

$$G = \sqrt{G_x^2 + G_y^2}$$

where G_x and G_y are gradients computed using Sobel kernels:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad G_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}$$

The Sobel operator is computationally **efficient and simple** but **sensitive to noise**, making it less reliable in high-noise environments.

The **Prewitt Operator (1970)** [18] is similar to Sobel but uses equal weights in the convolution kernels:

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}, \quad G_y = \begin{bmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

It detects edges effectively but is slightly less accurate in identifying sharp intensity changes.

The **Scharr Operator (2000)** [22] is an optimized version of Sobel that **reduces noise sensitivity** while maintaining sharp edge detection by improving **gradient estimation accuracy**. The convolution kernels are:

$$G_x = \begin{bmatrix} -3 & 0 & 3 \\ -10 & 0 & 10 \\ -3 & 0 & 3 \end{bmatrix}, \quad G_y = \begin{bmatrix} 3 & 10 & 3 \\ 0 & 0 & 0 \\ -3 & -10 & -3 \end{bmatrix}$$

The **Roberts Cross Operator (1963)** detects edges by calculating the difference between diagonally adjacent pixels.

$$G_x = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}, \quad G_y = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$$

It is **simple and fast** but not as precise as more advanced techniques due to its **high sensitivity to noise**.

- Second-Derivative Edge Detectors

Second-derivative methods compute the **Laplacian**, which highlights regions of **rapid intensity change.**

The **Laplacian operator** detects edges by computing the **second derivative** of the image, which enhances areas of rapid intensity change. The Laplacian Kernel is:

$$L = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

- Adaptive Edge Detection Techniques

These methods adapt thresholding techniques to improve edge detection in complex scenarios.

Adaptive thresholding dynamically adjusts threshold values based on **local intensity variations**, improving edge segmentation, especially in images with **non-uniform lighting conditions**.

Local Binary Patterns (LBP) (1996) [15] is used for texture-based edge analysis. It encodes pixel intensity variations in a small neighborhood, making it useful for capturing edge and texture information simultaneously.

4.2.2. Structural Features

Structural Feature Extraction techniques focus on capturing the **shape or geometry of objects in an image.**

The **Histogram of Oriented Gradients (HOG)** (2005) [4] is one of these techniques. It divides the image in cells, and for each pixel of this cell it computes the gradient magnitude and the orientation. It then computes for each cell a histogram of the oriented gradients. These histograms represent the distribution of edge directions in each cell, forming a set of local feature descriptors. HoG is typically used to describe larger image regions, whereas SIFT, that also uses orientations, focuses on keypoints.

4.2.3. Textural Features

Texture-based Feature Extraction methods focus on analyzing patterns, textures and fine surface details.

Gabor filters are among these methods. Gabor filters apply a convolution kernel to the image. Different Gabor filters are used to extract the useful features from an image, each one focusing on a special frequency and orientation. The kernel can be defined as:

$$G(x, y) = \exp\left(-\frac{x'^2 + \gamma^2 y'^2}{2\sigma^2}\right) \cos\left(2\pi f x' + \phi\right)$$

with $x' = x\cos\theta + y\sin\theta, \quad y' = -x\sin\theta + y\cos\theta$

4.3. Classification

Once the features are extracted, they are used for classification. For this purpose, two classes were implemented. Indeed, the file `classification_class.py` contains **two classes: *DataProcessing*** and ***BinaryClassification.***

The **first class, *DataProcessing***, allows to perform processing on the features dataset, such as scaling or PCA. It also splits the data into training, validation and testing sets (60, 20 and 20 % by default).

The **second class, *BinaryClassification***, allows to perform binary classification, and in particular Logistic Regression, Decision Tree Classification and Random Forest. Several methods are implemented in this class: they allow in particular to **find the best hyperparameters**, compare methods through cross validation, and compute diverse metrics. The features extracted using feature extraction methods are compared thanks to this class.

For **each created dataset**, the **same strategy** is **performed** (*Figure 2*). First, **for each classification method, a model** is **trained on the training set and its hyperparameters are tuned on the validation set**. Then, a K Fold cross validation is performed using a dataset composed of the training and validation sets but untouched (no data processing made on it). It is during the steps that the data processing will be made on the folds. For the training, the optimal hyperparameters found before are used. The **K Fold** allows to **determine the best classification method** using means metrics on the different test sets. For features extracted with SIFT for instance, it will help determine if it is Logistic Regression, Random Forest or Decision Classification Tree that has the best results. Finally, a **model** is **trained for the best method on the training set** and tested on the testing set, this time using the datasets transformed in the DataProcessing class.

All in all, this strategy enables to **get, for each features dataset, the results of its best classification method.** This methodology might seem a bit heavy but the idea is that, since we are applying classification methods, we might as well search for the best hyperparameters, and while doing it, why not try to be rigorous and avoid data leakage. Moreover, choosing only three classification methods limits the training time. It is also true that the goal is mainly to be able to compare feature extraction methods. That is why other classification methods were not used as they would have increased the code running time without much improvement. Nevertheless, the methodology performed (*Figure 2*) was thought so that it is really easy to add as many classification methods as one would want.

Simultaneously, the file `CNN.ipynb` was implemented. It contains the implementation of a **CNN** and a **Resnet50** applied directly to the images. Apart from different dropout values to avoid overfitting, a thorough search for their hyperparameters has not been done. Moreover, although we must keep in mind that their results could have been improved using some image processing, they perform easily well and they simply serve as a baseline **for comparing** our other **traditional feature extraction methods**, to see if they are able to beat Deep learning models.

5. Evaluation

The dataset used is downloadable on Kaggle [11]. It is composed of **road images with and without potholes**. In particular, it contains 329 images of roads with potholes and 352 images of roads without any issue: the **dataset** is therefore quite **balanced.** All kinds of roads and potholes are represented in this dataset.

The evaluation and comparison of feature extraction methods was done through classification, and in particular **binary classification (presence or absence of potholes)**. The methodology is the one talked about previously. Indeed, different types of features, obtained through diverse feature extraction methods (even combinations of different ones) are given as input for classification, so that performance results quantify their effectiveness. In particular, Machine learning methods like Logistic Regression, Decision Tree and Random Forest are used to get metrics like accuracy and F1-score. The code that allows to get the final scores is in the file `main.ipynb`. It uses the classes and the two json files created previously for efficiency.

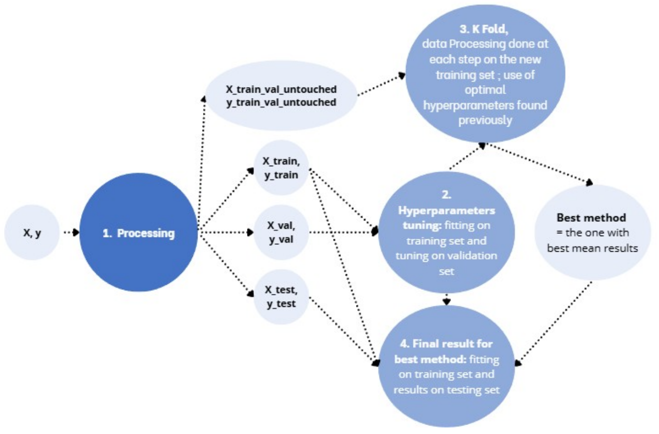

X, y
1. Processing
X_train_val_untouched
y_train_val_untouched
X_train, y_train
X_val, y_val
X_test, y_test
3. K Fold, data Processing done at each step on the new training set ; use of optimal hyperparameters found previously
2. Hyperparameters tuning: fitting on training set and tuning on validation set
Best method = the one with best mean results
4. Final result for best method: fitting on training set and results on testing set

Figure 2. Methodology to find best hyperparameters for each classification method (Logistic Regression, Decision Tree Classification and Random Forest in our case, but other methods could be added and the methodology would stay the same) and select the method that performs best.

The first table (*Table 1*) shows the classification performance obtained, in particular accuracy and F1-score on the testing set, when **no processing was done on the images.** The number of features kept was also indicated for information. For some methods, the number of features kept is quite small. For instance, for SIFT, a Kmeans clustering was performed to find the most important features, followed by a Bag of Words to describe each image. The number of clusters is in this case one parameter that we optimize in the step before. After deep learning models, **Edge and HOG are the methods that show the greatest performance** with a F1-score of respectively 82.39 and 83.30. LBP is also a close third with a F1-score of 79.33. Harris and SIFT also perform well. However, other methods don't perform as well. ORB or Gabor filters for instance are quite bad with a 50-60 F1-score. **CNN and Resnet50 perform easily well**, without even performing any processing beforehand, **beating traditional feature extraction methods.**

The second table (*Table 2*) represent the results obtained when using image processing. **HOG** is the method that **shows the best performance after image processing with a 90.44 F1-score.** Other methods, like LBP or Gabor filters also show nice improvement, which shows the relevance of image processing. However, some methods like SIFT or

	Test accuracy	Test F1-score	Features
SIFT	75.00	73.89	5
ORB	61.03	59.67	5
Harris	76.47	76.14	7
Edge	**83.09**	**82.39**	20
Otsu	63.97	59.97	256
Adaptive	56.62	55.75	256
Gabor	53.68	50.78	12
LBP	80.15	79.33	26
HOG	**83.82**	**83.30**	980
CNN	86.86	86.36	-
Resnet50	92.70	92.06	-

Table 1. **Classification results comparison between feature extraction methods on images with no processing, simply resized.** For all models but CNN and Resnet50, the choice of "best model" was done with the highest validation F1-score. For the deep learning models it was done using the highest accuracy (F1-score could have also been used for more equal comparison). The column features contains the number of features extracted by the methods.

	Test accuracy	Test F1-score	Features
SIFT	71.32	70.87	5
ORB	55.88	53.63	5
Harris	76.47	75.84	7
Edge	76.47	76.22	20
Otsu	69.85	64.92	256
Adaptive	61.76	58.88	256
Gabor	77.94	76.96	12
LBP	85.29	84.90	26
HOG	**90.44**	**90.11**	980

Table 2. Classification results comparison between feature extraction methods applied after image processing

	Test accuracy	Test F1-score	Features
SIFT + Edge	77.94	77.35	25
SIFT + HOG	88.24	87.71	985
HOG + PCA	86.03	85.62	209
SIFT + HOG + PCA	85.29	84.90	211

Table 3. Other results obtained from concatenating features from different methods and/or using PCA

Edge lose some performance, showing that the image processing performed is not successful for each method. The functions used are those implemented directly on Python, and consequently, some processing might already be done in these functions, making our processing redundant or even harmful. A more thorough work could be done on that part to ensure that it is not the case.

The first and last table (*Table 3*) show some results when classification was performed on some concatenated features and / or when using PCA. The first result computed is the one when using features from SIFT and Edge detection. It was done because we thought it might be interesting to get information on both keypoints and edges; sources also suggest the relevance of their combination [29]. These two methods also resulted in a small number of features, making it interesting to try out more features. For the other ones, the goal was to use HOG, which performed well previously. Adding SIFT doesn't increase the performance. HOG has indeed lots of features, so using PCA can also be interesting: even if it decreases a bit the performance, the model is simplified.

6. Conclusion

In conclusion, we performed a **classical approach** to computer vision by applying multiple **feature extraction methods** on a **dataset of roads with and without potholes.** By classifying these images, we were able to **compare** the **effectiveness** of those methods. To get the best comparison possible and let each method perform as well as it can, a good amount of time was spent searching for the right image processing and hyperparameters for the feature extraction methods.

Nevertheless, some **future work** could be performed. First, increasing the number of configurations tried for processing and feature extraction can improve performance. Furthermore, the feature extraction methods used are those directly implemented in Python and looking more in details at their implementation could help improve our processing step, by knowing if some is already done in their function. Other methods could also be tried, by merging multiple feature extraction methods for instance. Then, on the classification side, the range of classification methods used could be increased, while non linear dimension reduction methods could help reduce the number of features more effectively. Finally, a deeper analysis on the relevance of the features extracted could be done on the images: see if keypoints extracted for example truly focus on potholes and not also on other parts of images.

References

[1] Amila Akagic, Emir Buza, and Samir Omanovic. Pothole detection: An efficient vision based method using rgb color space image segmentation. In *2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, pages 1104–1109. IEEE, 2017. 1

[2] Emir Buza, Samir Omanovic, and Alvin Huseinovic. Pothole detection with image processing and spectral clustering. In *Proceedings of the 2nd International Conference on Information Technology and Computer Networks*, page 4853, 2013. 2

[3] John Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1986. 4

[4] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2005. 5

[5] Kurt Konolige Gary Bradski Ethan Rublee, Vincent Rabaud. An efficient alternative to sift or surf. *Proceedings of the IEEE International Conference on Computer Vision*, 2011. 4

[6] Rui Fan, Umar Ozgunalp, Yuan Wang, Ming Liu, and Ioannis Pitas. Rethinking road surface 3-d reconstruction and pothole detection: From perspective transformation to disparity map segmentation. *IEEE Transactions on Cybernetics*, 52(7):5799–5808, 2021. 2

[7] R. C. Gonzalez and R. E. Woods. *Digital Image Processing*. 2002. 2

[8] Chris Harris and Mike Stephens. A combined corner and edge detector. 1988. 4

[9] Xiangjian He, Daming Wei, Kin-Man Lam, Jianmin Li, Lin Wang, Wenjing Jia, and Qiang Wu. Canny edge detection using bilateral filter on real hexagonal structure. In *Advanced Concepts for Intelligent Vision Systems*, pages 233–244. Springer, 2010. 3

[10] Christian Koch and Ioannis Brilakis. Pothole detection in asphalt pavement images. *Advanced engineering informatics*, 25(3):507–515, 2011. 2

[11] Atulya Kumar. Pothole detection dataset. 6

[12] David G. Lowe. *Distinctive Image Features from Scale-Invariant Keypoints*. 2004. 3

[13] AB Makone and Mr Amitkumar G Rathod. Pothole dimensions measurement using mean shift-based speckle filtering. *Remote Sensing*, 40(11):1, 2002. 1

[14] Siyamalan Manivannan, Ruixuan Wang, and Emanuele Trucco. Extended gaussian-filtered local binary patterns for colonoscopy image classification. In *Proceedings of the IEEE International Conference on Computer Vision Workshops*, pages 184–189, 2013. 3

[15] T. Ojala, M. Pietikäinen, and T. Mäenpää. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. 1996. 5

[16] OpenCV Documentation. Image Thresholding - OpenCV Documentation, 2024. Accessed: 2025-02-19. 3

[17] Pedraza Ortega J. C. Ramos Arreguín J.M. Palma Olvera R. D., Martínez Zerón E. and Gorrostieta Hurtado E. A feature extraction using sift with a preprocessing by adding clahe algorithm to enhance image histograms. *International Conference on Mechatronics, Electronics and Automotive Engineering*, 2014. 2, 3

[18] Judith M. S. Prewitt. Object enhancement and extraction. 1970. 4

[19] Ricardo Silveira Rodrigues, Marcia Pasin, Alice Kozakevicius, and Vinicius Monego. Pothole detection in asphalt: An automated approach to threshold computation based on the haar wavelet transform. In *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, pages 306–315. IEEE, 2019. 2

[20] Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *European Conference on Computer Vision*, 2006. 4

[21] Seung-Ki Ryu, Taehyeong Kim, and Young-Ro Kim. Image-based pothole detection system for its service and road management system. *Mathematical Problems in Engineering*, 2015(1):968361, 2015. 1

[22] H. Scharr. Optimal operators in digital image processing. Technical report, 2000. 5

[23] Ionut Schiopu, Jukka P Saarinen, Lauri Kettunen, and Ioan Tabus. Pothole detection and tracking in car video sequence. In *2016 39th International Conference on Telecommunications and Signal Processing (TSP)*, pages 701–706. IEEE, 2016. 2

[24] I. Sobel and G. Feldman. An isotropic 3x3 image gradient operator. Technical report, 1968. 4

[25] Xiangping Sun, Jin Wang, Ronghua Chen, Lingxue Kong, and Mary FH She. Directional gaussian filter-based lbp descriptor for textural image classification. *Procedia Engineering*, 15:1771–1779, 2011. 2

[26] Carlo Tomasi and Roberto Manduchi. Bilateral filtering for gray and color images. *Proceedings of the IEEE International Conference on Computer Vision*, 1998. 2

[27] Penghui Wang, Yongbiao Hu, Yong Dai, and Mingrui Tian. Asphalt pavement pothole detection and segmentation based on wavelet energy field. *Mathematical Problems in Engineering*, 2017(1):1604130, 2017. 1

[28] Muhammad Haroon Yousaf, Kanza Azhar, Fiza Murtaza, and Fawad Hussain. Visual analysis of asphalt pavement for detection and localization of potholes. *Advanced Engineering Informatics*, 38:527–537, 2018. 2

[29] Wannan Zhang. Combination of sift and canny edge detection for registration between sar and optical images. *IEEE Geoscience and Remote Sensing Letters*, 2020. 7