

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

SCHOOL OF ELECTRONICS AND TELECOMMUNICATIONS



PROJECT REPORT

GASTROINTESTINAL ENDOSCOPIC IMAGES CLASSIFICATION

Name of students: *Programming Languages – BME K63*

Do Thi Khanh Linh - 20182963

Ho Nguyen Khang - 20182958

Name of instructor: Ascc. Prof. Tran Thi Thanh Hai

Hanoi, 6-2021

1. Subject topic

1.1. Reasons for choosing this topic

The human digestive system may be affected by several diseases. Altogether esophageal, stomach and colorectal cancer accounts for about 2.8 million new cases and 1.8 million deaths per year. Endoscopic examinations are the gold standards for investigation of the GI tract. Endoscopic examinations are resource demanding and requires both expensive technical equipment and trained personnel. For colorectal cancer prevention, endoscopic detection and removal of possible precancerous lesions are essential. However, the ability to detect adenomas varies between doctors, and this may eventually affect the individuals' risk of getting colorectal cancer. Endoscopic assessment of severity and sub-classification of different findings may also vary from one doctor to another. Accurate grading of diseases are important since it may influence decision-making on treatment and follow-up.

Artificial intelligence (AI)-based applications have transformed several industries and are widely used in various consumer products and services. In medical industries, AI is primarily being used for image classification and has great potential to affect image-based specialties such as radiology, pathology, and gastroenterology (GE). For automatic algorithmic detection of abnormalities in the GI tract, there have been many proposals from various research communities, especially for the topic of polyp detection.

1.2. Purpose:

The aim of this project is that we want to achieve a model consisting of simple algorithms to process as input images from different pathological findings, and as output - they are classified into the correct class.

Image processing related to medical images is an active research field where various techniques are used to facilitate diagnosis and various image processing techniques can be used. There are a variety of image processing libraries, however OpenCV (open computer vision) has become mainstream due to its large community support and availability in C++. Our abstract about endoscopic images classification presents the implementation of the C++ programming language and the Open CV library.

2. Contributions of each member

Task ID	Task Name	Resource Name
#1	Doing overall research & create plans	All members
#2	Doing research about gastrointestinal endoscopic image database.	Do Thi Khanh Linh
#3	Read image's data.	Do Thi Khanh Linh
#4	Extracting image's feature using Histogram of Oriented Gradient algorithm.	Do Thi Khanh Linh
#5	Doing the calculation of the distance function.	Ho Nguyen Khang
#6	Implementing the KNN algorithm to determine the image's class.	Ho Nguyen Khang
#7	Making Report + Slide + Defend Project	All members

Table 1. Our Task Planning

3. Algorithms, Dataset & Data Structure

3.1. Algorithms

We will simply build a model that read input images's data, perform feature extraction to obtain the feature vectors of the images. Then, we will implementing the KNN classifiers to calculate distance from the above features vectors to all other data that is already classified, get the K nearest points & do major voting in K point, and finally predict class that appeared the most times.

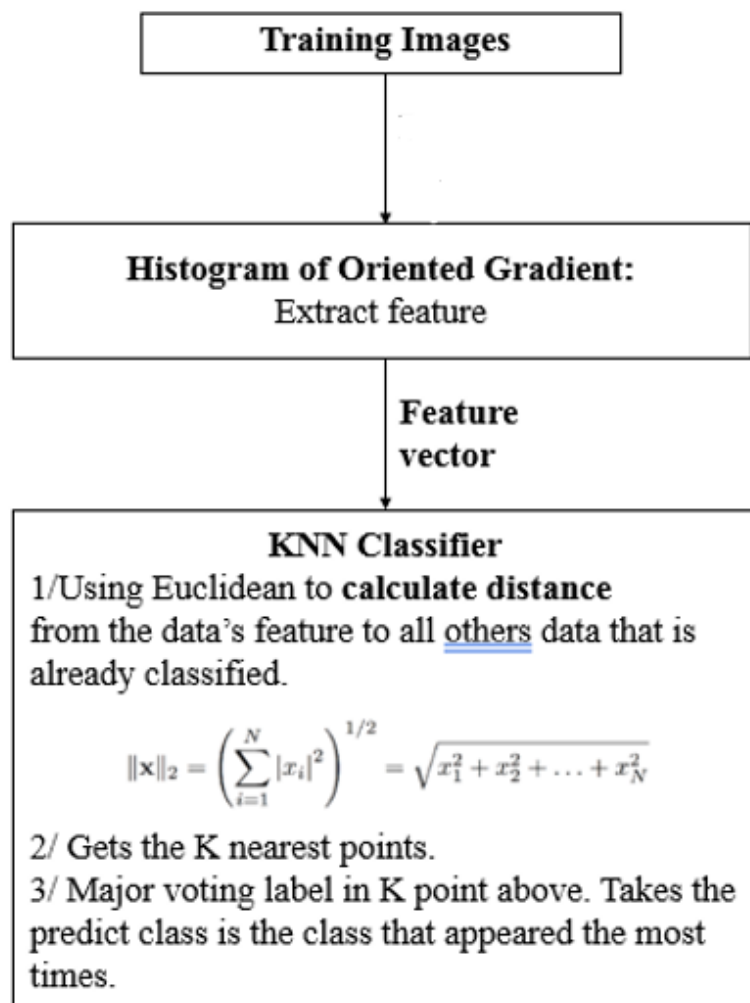


Fig 1. Our Classification Model using HOG & KNN

Feature Extraction – Histogram of Oriented Gradient

3.1.1.1. Why we have to extract image's feature ?

Image features, such as edges and interest points, provide rich information on the image content. They correspond to local regions in the image and are fundamental in many applications in image analysis: recognition, matching, reconstruction, etc. Image features yield two different types of problem: the detection of area of interest in the image, typically contours, and the description of local regions in the image, typically for matching in different images.

3.1.1.2. Histogram of Oriented Gradient (HOG) basic concepts

HOG, or Histogram of Oriented Gradients, is a feature descriptor that is often used to extract features from image data. The HOG descriptor focuses on the structure or the shape of an object. In the case of edge features, we only identify if the pixel is an edge or not. HOG is able to provide the edge direction as well. This is done by extracting the gradient and orientation (magnitude and direction) of the edges. Additionally, these orientations are calculated in 'localized' portions. This means that the complete image is broken down into smaller regions and for each region, the gradients and orientation are calculated. Finally the HOG would generate a Histogram for each of these regions separately. The histograms are created using the gradients and orientations of the pixel values.

- *Step 1: Preprocess the Data (64 x 128)*

Preprocessing data is a crucial step in any machine learning project and that's no different when working with images.

We need to preprocess the image and bring down the width to height ratio to 1:2. The image size should preferably be 64 x 128. This is because we will be dividing the image into 8*8 and 16*16 patches to extract the features. Having the specified size (64 x 128) will make all our calculations pretty simple.

- *Step 2: Calculate the Gradient Images*

To calculate a HOG descriptor, we need to first calculate the horizontal and vertical gradients; after all, we want to calculate the histogram of gradients. This is achieved by filtering the image with the following kernel.

The next step is to create a histogram of gradients in these 8×8 cells. The histogram contains 9 bins corresponding to angles 0, 20, 40 ... 160. The following figure illustrates the process. A bin is selected based on the direction, and the vote (the value that goes into the bin) is selected based on the magnitude.

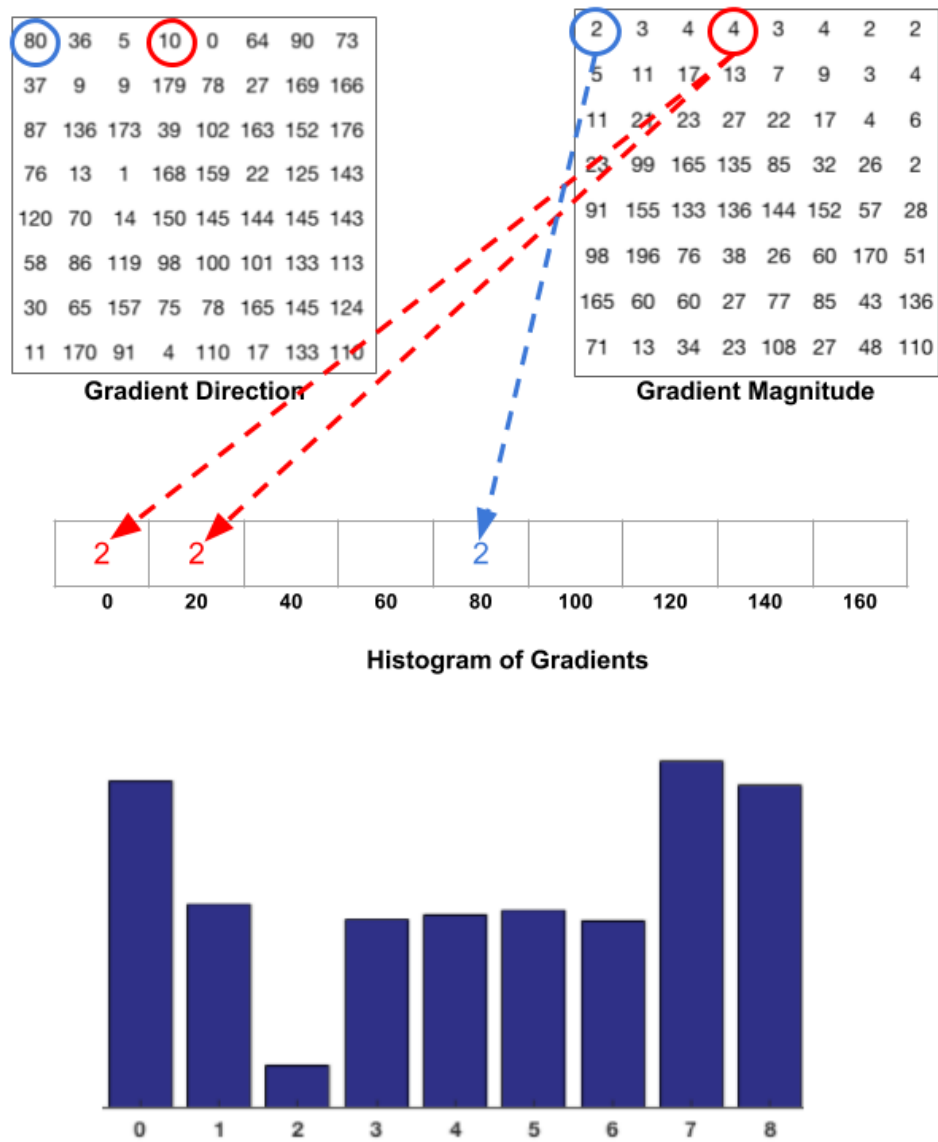


Fig 2. Process of create a histogram of gradients

- **Step 4: 16×16 Block Normalization**
“Normalize” the histogram so they are not affected by lighting variations.
- **Step 5: Calculate the Histogram of Oriented Gradients feature vector**

3.1.1.3. HOG implementation

We use implementation of HOG (Histogram of Oriented Gradients) descriptor and object detector in OpenCV library.

[OpenCV: cv::HOGDescriptor Struct Reference](#)

```
#include <opencv2/objdetect.hpp>
```

More specifically, we use member function *compute()* to compute HOG descriptors of given image.

```
virtual void cv::HOGDescriptor::compute ( InputArray      img,
                                          std::vector< float > & descriptors,
                                          Size             winStride = Size() ,
                                          Size             padding = Size() ,
                                          const std::vector< Point > & locations = std::vector< Point >()
                                          ) const
```

Parameters

img	Matrix of the type CV_8U containing an image where HOG features will be calculated.
descriptors	Matrix of the type CV_32F
winStride	Window stride. It must be a multiple of block stride.
padding	Padding
locations	Vector of Point

3.1.2. K Nearest Neighbors (KNN) Classifier

K Nearest Neighbor algorithm falls under the Supervised Learning category and is used for classification (most commonly) and regression. It is a versatile algorithm also used for imputing missing values and resampling datasets. As the name (K Nearest Neighbor) suggests it considers K Nearest Neighbors (Data points) to predict the class or continuous value for the new Datapoint.

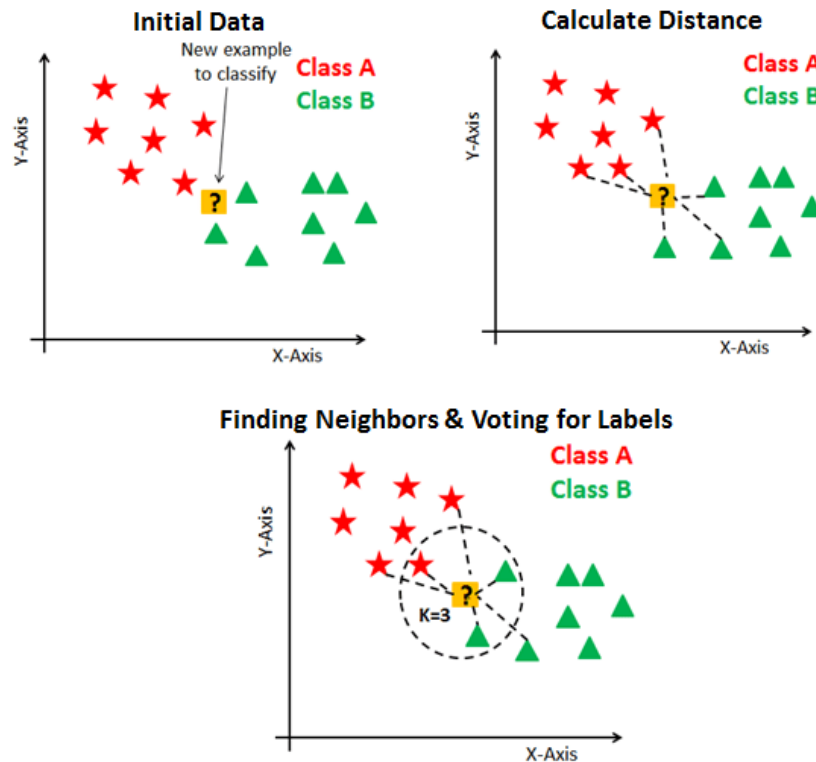


Fig 3. KNN explanation

In KNN, K is the number of nearest neighbors. The number of neighbors is the core deciding factor. Suppose P1 is the point, for which label needs to predict. First, you find the k closest point to P1 and then classify points by majority vote of its k neighbors. Each object votes for their class and the class with the most votes is taken as the prediction. For finding closest similar points, you find the distance between points using distance measures such as Euclidean. KNN has the following basic steps:

1. Calculate distance.
2. Find closest neighbors.
3. Vote for labels.

3.2. Dataset & Data Structure

- Open Source: [The Kvasir Dataset \(simula.no\)](https://simula.no)

- Data Collection:

The data is collected using endoscopic equipment at Vestre Viken Health Trust (VV) in Norway. The VV consists of 4 hospitals and provides health care to 470.000 people. One of these hospitals (the Bærum Hospital) has a large gastroenterology department from where training data have been collected and will be provided, making the dataset larger in the future. Furthermore, the images are carefully annotated by one or more medical experts from VV and the Cancer Registry of Norway (CRN). The CRN provides new knowledge about cancer through research on cancer.



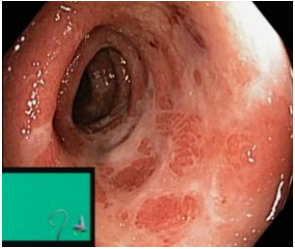
- Dataset Details:

The Kvasir dataset consists of images, annotated and verified by medical doctors (experienced endoscopists), including several classes showing anatomical landmarks, pathological findings or endoscopic procedures in the GI tract, i.e., hundreds of images for each class. The number of images is sufficient to be used for different tasks, e.g., image retrieval, machine learning, deep learning and transfer learning, etc. The anatomical landmarks include Z-line, pylorus, cecum, etc., while the pathological finding includes esophagitis, polyps, ulcerative colitis, etc. The dataset consist of the images with different resolution from 720x576 up to 1920x1072 pixels and organized in a way where they are sorted in separate folders named accordingly to the content.

- Our data usage:

In our project, we want to classify input images to different pathological findings, so we just use 3 images folders: Esophagitis, Polyps and Ulcerative Colitis.

A pathological finding in this context is an abnormal feature within the gastrointestinal tract. Endoscopically, it is visible as a damage or change in the normal mucosa. The finding may be signs of an ongoing disease or a precursor to for example cancer. Detection and classification of pathology is important in order to initiate correct treatment and/or follow-up of the patient.

<p>Esophagitis</p> 	<p>Esophagitis is an inflammation of the esophagus visible as a break in the esophageal mucosa in relation to the Z-line. The grade of inflammation is defined by length of the mucosal breaks and proportion of the circumference involved. This is most commonly caused by conditions where gastric acid flows back into the esophagus as gastroesophageal reflux, vomiting or hernia.</p>
<p>Polyps</p> 	<p>Polyps are lesions within the bowel detectable as mucosal outgrowths. The polyps are either flat, elevated or pedunculated, and can be distinguished from normal mucosa by color and surface pattern. Most bowel polyps are harmless, but some have the potential to grow into cancer. Detection and removal of polyps are therefore important to prevent development of colorectal cancer. Since polyps may be overlooked by the doctors, automatic detection would most likely improve examination quality. The green boxes within the image shows an illustration of the endoscope configuration. In live endoscopy, this helps to determine the current localisation of the endoscope-tip (and thereby also the polyp site) within the length of the bowel.</p>
<p>Ulcerative Colitis</p> 	<p>Ulcerative colitis is a chronic inflammatory disease affecting the large bowel. The disease may have a large impact on quality of life, and diagnosis is mainly based on colonoscopic findings. The degree of inflammation varies from none, mild, moderate and severe, all with different endoscopic aspects. For example, in a mild disease, the mucosa appears swollen and red, while in moderate cases, ulcerations are prominent. This is an example of ulcerative colitis with bleeding, swelling and ulceration of the mucosa. The white coating visible in the illustration is fibrin covering the wounds.</p>

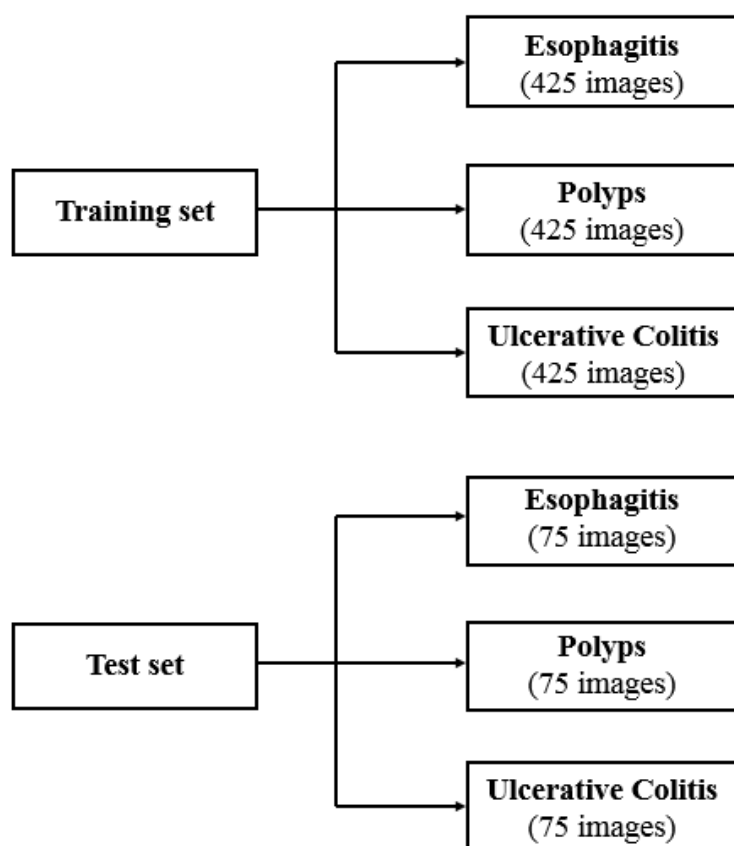


Fig 4. Our Data Usage

4. Programming Techniques

We use Object Oriented Programming by defining class Image

```
class Images {
private:
    vector <float> des;
    int label, size;
public:
    Images() { des.clear(); label = 0; size = 0; }
    //~Images(){}
    Images(vector <float>& d, int l, int s) { des = d; label = l; size = s; }

    int getLabel(void) { return label; }

    void print()const {
        For(i, 0, des.size()) printf("%f ", des[i]);
        printf("\n%d %d\n", label, size);
    }

    float Distance(Images B) {
        float res = 0;
        For(i, 0, B.size) res += (des[i] - B.des[i]) * (des[i] - B.des[i]);
        return sqrt(res);
    }
};
```

We implement HOGDescriptor – compute() of OpenCV Library

```
Images Image_train[N], Image_test[N];

void ComputeHOG(string path, int label) {
    HOGDescriptor hog;
    vector <float> descriptor;

    FOR(i, 1, 500)
    {
        if (label == 1 && i == 491) break;

        cv::Mat img = cv::imread(path + "(" + to_string(i) + ").jpg", 0);

        cv::resize(img, img, Size(64, 128), 0, 0);
        hog.compute(img, descriptor, Size(16, 16), Size(0, 0));

        Images A(descriptor, label, descriptor.size());
        if (i <= 425) Image_train[++cnt_train] = A;
        else Image_test[++cnt_test] = A;
    }
}
```

We wrote KNN function based on algorithms:

```
bool cmp(pair <float, int> a, pair<float, int> b) { return a.first < b.first; }

int Max = 0, id = 0;
float res = 0;
pair <float, int> dis[N];
int cnt[4];

float KNN(int K) {
    printf("%d %d\n", cnt_train, cnt_test);
    res = 0;
    FOR(i, 1, cnt_test) {
        Max = 0;
        FOR(j, 0, cnt_train) dis[j] = mp(0, 0);
        FOR(j, 1, 3) cnt[j] = 0;
        FOR(j, 1, cnt_train) dis[j] = mp(Image_test[i].Distance(Image_train[j]), Image_train[j].getLabel());

        sort(dis + 1, dis + cnt_train + 1, cmp);

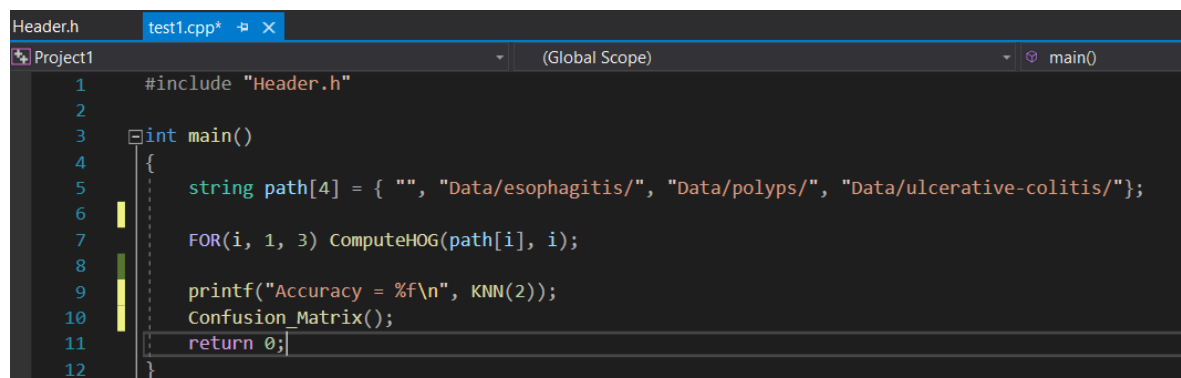
        FOR(k, 1, K) ++cnt[dis[k].second];

        id = 0;
        FOR(j, 1, 3) if (Max <= cnt[j]) Max = cnt[j], id = j;

        res += (cnt[Image_test[i].getLabel()] == Max);
    }
    cout << res << " ";
    return 100.0 * res / cnt_test;
}
```

After that, we put all of above code into 1 file called Header, and use it as an library. So that you can see in main function, we easily call function ComputeHOG to get the feature from image data.

Our main function:



```
Header.h  test1.cpp*  X
Project1  (Global Scope)  main()
1  #include "Header.h"
2
3  int main()
4  {
5      string path[4] = { "", "Data/esophagitis/", "Data/polyps/", "Data/ulcerative-colitis/" };
6
7      FOR(i, 1, 3) ComputeHOG(path[i], i);
8
9      printf("Accuracy = %f\n", KNN(2));
10     Confusion_Matrix();
11     return 0;
12 }
```

5. Result

```
Microsoft Visual Studio Debug Console
Tranining image: 1275, Testing image: 215
True prediction = 178 K = 2, Accuracy = 82.790695
Tranining image: 1275, Testing image: 215
True prediction = 160 K = 12, Accuracy = 74.418602
Tranining image: 1275, Testing image: 215
True prediction = 154 K = 22, Accuracy = 71.627907
Tranining image: 1275, Testing image: 215
True prediction = 155 K = 32, Accuracy = 72.093025
Tranining image: 1275, Testing image: 215
True prediction = 153 K = 42, Accuracy = 71.162788

D:\NNLT20202\project\Project1\x64\Debug\Project1.exe (process 9248) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->
```

This is our result of KNN implementation with different K values. KNN with K = 2 give us the best result. Accuracy = 82.79%

We can see that increasing K will reduce accuracy.

Moreover, we create a Confusion Matrix function to evaluate our result.

```
void Confusion_Matrix(void) {
    printf("Confusion Matrix:\n");
    FOR(i, 1, 3) {
        int scale = 65 + 10 * (i != 1);
        FOR(j, 1, 3) printf("%f ", 1.0 * Label[i][j] / scale);
        printf("\n");
    }
}
```

```
Microsoft Visual Studio Debug Console
Tranining image: 1275, Testing image: 215
True predicted = 178 Accuracy = 82.790695
Confusion Matrix:
1.000000 0.000000 0.000000
0.053333 0.506667 0.440000
0.000000 0.000000 1.000000

D:\NNLT20202\project\Project1\x64\Debug\Project1.exe (process 6548) exited with code 0.
To automatically close the console when debugging stops, enable Tools->Options->Debugging->Automatically close the console when debugging stops.
Press any key to close this window . . .
```

References

- [1] Tran Thi Thanh Hai, Lecture Slides (C/C++ Programming Language).
- [2] Kvasir Dataset: [The Kvasir Dataset \(simula.no\)](https://simula.no/kvasir/)
- [3] Struct HOGDescriptor in OpenCV:
[OpenCV: cv::HOGDescriptor Struct Reference](https://docs.opencv.org/3.4.12/d8/d90/struct_cv__HOGDescriptor.html)
- [4] Histogram of Oriented Gradients basic learning:
[Histogram of Oriented Gradients explained using OpenCV \(learnopencv.com\)](https://learnopencv.com/hog/)
[Tìm hiểu về hog\(histogram of oriented gradients\) \(viblo.asia\)](https://viblo.asia/tim-hieu-ve-hog-histogram-of-oriented-gradients)
[Feature Descriptor | Hog Descriptor Tutorial \(analyticsvidhya.com\)](https://analyticsvidhya.com/tutorial/hog-descriptor/)
- [5] KNN Algorithms basic learning:
[KNN Classification using Scikit-learn - DataCamp](https://datacamp.com/courses/knn-classification-using-scikit-learn/)
[KNN Algorithm | What is KNN Algorithm | How does KNN Function \(analyticsvidhya.com\)](https://analyticsvidhya.com/tutorial/knn-algorithm/)