Лабораторная работа №2

Задача:

- Написать приложение/веб-приложение, реализующее указанные в варианте методы обработки изображений.
- На проверку сдаются: exe, который должен работать на любом ПК под Windows XP /веб-приложение, выложенное в общий доступ; исходный код; сопроводительная документация.

Требования и критерии оценки:

- Графический интерфейс.
- База соответствующих изображений для тестирования (зашумленные, размытые, малоконтрастные, и т.д.) согласно варианту.

Использованные средства разработки:

- Windows Forms .NET, C#
- Библиотека Emgu (реализация OpenCV)

Ход работы:

- Создание интерфейса пользователя с помощью встроенного конструктора;
- Реализация методов своего варианта :
  Глобальная пороговая обработка (Оцу, Порог градиента изображения);
  Адаптивная пороговая обработка;
  Реализация высокочастот ных фильтров (увеличение резкости)
- Реализация методов для открытия/сохранения картинки

Выводы:

В ходе выполнения работы я:

- Создал приложение, позволяющее применять к картинкам разричные методы фильтрациимоделях.
- Реализовал конвертацию цвета в одной цветовой модели в другую • Углубил знания в Windows Forms .NET, C#
- Поработал с системой контроля версий Gi

Пояснения с реализованным методам:

Метод Оцу

Процесс выбора порога с помощью метода Оцу включает следующие шаги:

- Вычисление гистограммы изображения: Сначала строится гистограмма яркости пикселей в изображении. Гистограмма показывает распределение яркости пикселей и позволяет определить, какие значения яркости наиболее часто встречаются.
- Вычисление общего числа пикселей: Суммируются все значения гистограммы, чтобы получить общее число пикселей в изображении.
- Вычисление весовых коэффициентов: Для каждого значения яркости вычисляются вероятности появления объекта и фона. Вероятность объекта определяется как сумма значений гистограммы яркости от начала до данного значения, поделенная на общее число пикселей. Вероятность фона определяется как сумма значений гистограммы яркости от данного значения до конца, также поделенная на общее число пикселей.
- Вычисление средних значений яркости: Для каждого значения яркости вычисляются средние значения объекта и фона. Среднее значение объекта рассчитывается путем деления суммы произведений значений гистограммы на вероятности объекта на общее число пикселей объекта. Аналогично рассчитывается среднее значение фона.
- Вычисление межклассовой дисперсии: Для каждого значения яркости вычисляется межклассовая дисперсия с использованием формулы Otsu: w_object * w_background * (mean_object - mean_background)^2, где w_object и w_background - вероятности объекта и фона соответственно, mean_object и mean_background - средние значения объекта и фона.
- Поиск оптимального значения порога: Межклассовая дисперсия вычисляется для каждого значения яркости, и выбирается значение порога, при котором межклассовая дисперсия максимальна.
- Применение порога для бинаризации: Полученное оптимальное значение порога используется для бинаризации изображения. Все пиксели с яркостью выше порога считаются объектом, а все пиксели с яркостью ниже порога считаются фоном.

Листинг данного метода

```
public static byte CalculateOtsuThreshold(Image<Gray, byte> image)
```

```
{
    // Вычисление гистограммы яркости
    int[] histogram = new int[256];
    var pixels = image.Data;

    for (int i = 0; i < image.Rows; i++)
    {
        for (int j = 0; j < image.Cols; j++)
        {
            histogram[pixels[i, j, 0]]++;
        }
    }

    // Общее количество пикселей
    int totalPixels = image.Rows * image.Cols;

    // Вычисление суммы яркостей
    int sum = 0;
    for (int i = 0; i < 256; i++)
    {
        sum += i * histogram[i];
    }

    // Вычисление глобальной дисперсии
    double globalVariance = 0;
```

```
double globalMean = (double)sum / totalPixels;
double maxVariance = 0;
byte threshold = 0;

for (int i = 0; i < 256; i++)
{
    int backgroundPixels = 0;
    int foregroundPixels = 0;
    int backgroundSum = 0;
    int foregroundSum = 0;

    for (int j = 0; j <= i; j++)
    {
        backgroundPixels += histogram[j];
        backgroundSum += j * histogram[j];
    }

    foregroundPixels = totalPixels - backgroundPixels;
    foregroundSum = sum - backgroundSum;

    if (backgroundPixels == 0 || foregroundPixels == 0)
        continue;

    double backgroundMean = (double)backgroundSum / backgroundPixels;
    double foregroundMean = (double)foregroundSum / foregroundPixels;
```

```
            double variance = backgroundPixels * foregroundPixels *
Math.Pow(backgroundMean - foregroundMean, 2);
            if (variance > maxVariance)
            {
               maxVariance = variance;
               threshold = (byte)i;
            }
         }

         return threshold;
      }
```

Метод подбора на основе градиента изображения

Процесс выбора порога на основе градиента изображения включает следующие шаги:

- Вычисление градиента изображения: Градиент изображения может быть вычислен с использованием операторов градиента, таких как оператор Собеля или оператор Превитта. Операторы градиента вычисляют разности яркости между соседними пикселями в разных направлениях, чтобы получить информацию о изменении яркости и ориентации границ.
- Расчет гистограммы градиента: Гистограмма градиента строится на основе значений градиента пикселей изображения. Гистограмма показывает распределение значений градиента и помогает определить наиболее значимые границы на изображении.
- Анализ гистограммы градиента: Анализируется гистограмма градиента, чтобы определить пороговое значение, которое будет использоваться для бинаризации изображения. Порог выбирается таким образом, чтобы отделить значимые границы от шума и незначительных изменений яркости.

- Бинаризация изображения: Пороговое значение, полученное из анализа гистограммы градиента, применяется для бинаризации изображения. Все пиксели с градиентом выше порога считаются границами или контурами, а остальные пиксели считаются фоном.

Листинг метода

```
public static Image<Gray, byte> CalculateThreshold(Image<Gray, byte> gray)
    {
        // Вычисление градиентов по горизонтали и вертикали с помощью оператора Собеля
        Image<Gray, float> gradientX = gray.Sobel(1, 0, 3);
        Image<Gray, float> gradientY = gray.Sobel(0, 1, 3);

        // Вычисление абсолютного значения градиента
        Image<Gray, float> gradientMagnitude = gradientX.AbsDiff(gradientY);

        // Нормализация значений градиента в диапазоне [0, 255]
        Image<Gray, byte> normalizedGradient = gradientMagnitude.ConvertScale<byte>(255, 0);

        // Поиск локальных максимумов градиента
        Image<Gray, byte> threshold = normalizedGradient.ThresholdBinaryInv(new Gray(0), new Gray(255));

        return threshold;
    }
```