МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНОЛОГИЧЕСКИЙ
УНИВЕРСИТЕТ «МИСИС»

Институт компьютерных наук
Кафедра инженерной кибернетики

Курсовая работа

по дисциплине «Обработка изображений» на тему

«Автоматическая детекция бластных клеток»

Выполнил:
студент 3-го курса,
гр. БПМ-21-3 Грачев К.Ю.

Проверил:
доцент, к.т.н. Полевой Д.В.

Москва
2024

ОГЛАВЛЕНИЕ

1. ТЕХНИЧЕСКОЕ ЗАДАНИЕ

Разработать алгоритм, который позволит автоматически определять на изображении области скопления бластных клеток.

1.1. Требования к функционалу

• Алгоритм автоматически определяет бластные клетки на изображении.

• У пользователя есть возможность детектировать объекты на своих изображениях.

• Пользователь может получить значение метрик в формате JSON, если предоставит ground_truth данные.

• При запуске программы пользователь обязан указать путь к исходному изображению.

2. ДАННЫЕ

Так как необходимо проверить качество алгоритма, было принято решение собрать датасет из изображений бластных клеток с микроскопа.

Разметка проводилась на платформе CVAT, после чего данные были приведены в удобный вид.

Особых требований к данным не было, поэтому они имеют разные размеры, разное количество объектов, разные размеры объектов, а также выполнены в разных цветовых гаммах.

Объекты размечены полигонами.

3. ИЗМЕРЕНИЕ КАЧЕСТВА

3.1. Логика оценки качества

Для оценки качества алгоритма детекции объектов была применена следующая методика:

1. Создание бинарных изображений полигонов:

• Имеется список полигонов - ground_truth.

• Для каждого полигона создается отдельное бинарное изображение, где пиксели, принадлежащие полигону, обозначены значением 1, а все остальные пиксели имеют значение 0.

2. Пересечение с детекцией:

• Выполняется пересечение каждого бинарного изображения полигона отдельного объекта с результатом детекции.

• Для каждого пересечения вычисляется площадь.

• Если площадь пересечения детекции и ground_truth >= SQUARE_THRESHOLD, то это считается как True Positive (TP).

• Если площадь пересечения меньше SQUARE_THRESHOLD, то это считается как False Negative (FN).

3. Подсчёт False Positive (FP):

• Все бинарные изображения полигонов объединяются в одно большое изображение groundTruth.

• На основании оставшейся части детекции (после удаления всех TP и FN) выявляются компоненты, которые не попадают на groundTruth.

• Эти оставшиеся кусочки считаются False Positive (FP)

4. Вычисление метрик:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

$$Precision = \frac{TP}{TP + FP}$$

$$Recall = \frac{TP}{TP + FN}$$

$$F1\text{-}score = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$$

Рисунок 1. Метрики оценки качества детекции

5. Обсуждение True Negative (TN):

• В данной задаче определение True Negative (TN) затруднено, так как оно обычно описывает случаи, когда ни детекция, ни ground_truth не содержат объекта на определенном участке изображения.

• Так как задача сосредоточена на полигональной детекции, определение TN становится менее очевидным и не включено в текущую методику оценки.

Данная методика позволяет четко разделить успешные (TP), пропущенные (FN) и ложные (FP) детекции, что дает возможность объективно оценить качество работы алгоритма.

4. АЛГОРИТМ

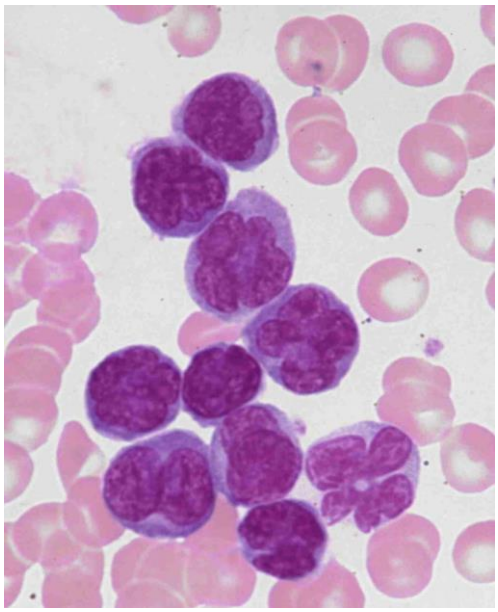

Рисунок 2. Пример оригинальной картинки

4.1. В ЦВЕТОВОМ ПРОСТРАНСТВЕ HSV

	accuracy	*precision*	*recall*	*f1*
micro	0.18	0.21	0.58	0.31
macro	0.28	0.37	0.55	0.39

1. Для бинаризации используются константные верхний и нижний пороговые значения для Hue, Saturation, Value.
   cv::Scalar lowerHSV(90, 50, 50), upperHSV(180, 200, 200);
   cv::inRange(hsvImage, lowerHSV, upperHSV, detectionsImg);

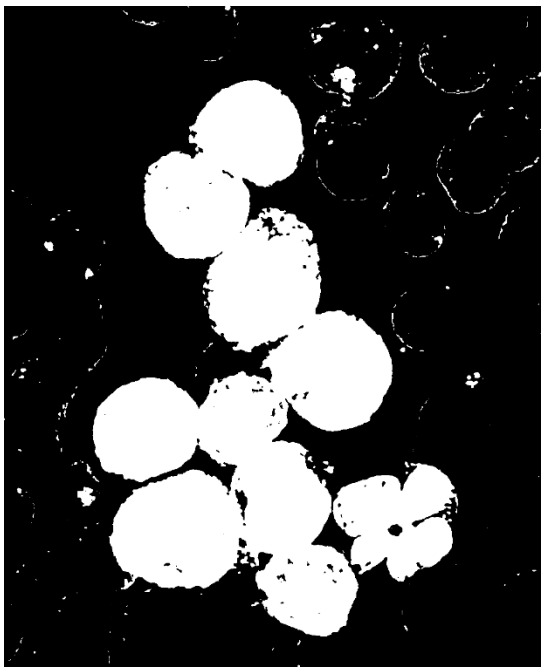

Рисунок 3. После бинаризации

2. Закрываем дыры внутри задетектированных компонент морфологическими операциями.
   cv::morphologyEx(detectionsImg, detectionsImg, cv::MORPH_CLOSE, cv::getStructuringElement(cv::MORPH_RECT, cv::Size(5,5));

3. Выделяем компоненты связности.
   std::vector<std::vector<cv::Point>> contours;
   std::vector<cv::Vec4i> hierarchy;
   cv::findContours(detectionsImg,contours,hierarchy,cv::RETR_CCOMP, cv::CHAIN_APPROX_SIMPLE);

4. На детекцию попадут компоненты, площадь которых выше PERCENT_THRESHOLD перцентиля.

```
std::vector<double> areas;
for (const auto& contour : contours) {
    areas.push_back(cv::contourArea(contour));
}
std::sort(areas.begin(), areas.end());
```

```
double threshold = areas[(int)(areas.size() * PERCENT_THRESHOLD / 100.0)];
detectionsImg = cv::Mat::zeros(detectionsImg.size(), CV_8UC1);
for (int i = 0; i < contours.size(); i++) {
    double area = cv::contourArea(contours[i]);
    if (area >= threshold) {
        cv::drawContours(detectionsImg, contours, i, cv::Scalar(255), cv::FILLED, 8, hierarchy, 1);
    }

}
```

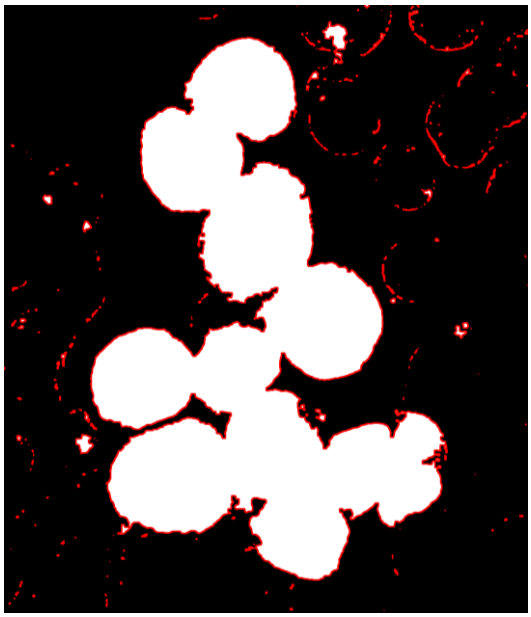

Рисунок 4. Результат детекции в режиме HSV

4.2. В ЦВЕТОВОМ ПРОСТРАНСТВЕ BGR

	accuracy	*precision*	*recall*	*f1*
micro	0.67	0.75	0.86	0.80
macro	0.70	0.78	0.87	0.79

1. K-Means кластеризация для поиска цвета, похожего на цвет клетки – фиолетовый (128, 0, 128) и темно-синий (30, 30, 90).

```
int k = 20; // Количество кластеров
cv::Mat labels, centers;
cv::kmeans(data, k, labels, cv::TermCriteria(cv::TermCriteria::EPS
+cv::TermCriteria::COUNT,10,1.0), 3,cv::KMEANS_PP_CENTERS,
centers);
centers.convertTo(centers, CV_8UC1);
centers = centers.reshape(3, centers.rows);
std::vector<cv::Vec3b> targetColors = {{128, 0, 128}, {30, 30, 90}};
for (const auto& targetColor : targetColors) {
    double minDistance = DBL_MAX;
    cv::Vec3b foundColor;
    for (int i = 0; i < centers.rows; i++) {
        cv::Vec3b color = centers.at<cv::Vec3b>(i, 0);
        double distance = cv::norm(color - targetColor);
        if (distance < minDistance) {
            minDistance = distance;
            foundColor = color;
        }
    }
    bool foundAlready = false;
    for (const auto& color : cellColors) {
        if (color == foundColor) {
            foundAlready = true;
            break;
        }
    }
    if (!foundAlready) {
        cellColors.push_back(foundColor);
```

```
        }
    }
```

2. Изображение разделяется по цветовым каналам, после чего для каждого найденного цвета ищется эффективный диапазон по каждому каналу:

```
int histSize = 256; // from 0 to 255
float range[] = {0, 256};
const float* histRange = {range};
cv::Mat hist;
cv::calcHist(&channel,1,0,cv::Mat(),hist,1,&histSize,&histRange);

// Определение диапазона значений, близких к targetValue
int highBound = targetValue, lowBound = targetValue;
float threshold = hist.at<float>(targetValue) * 0.5; // Порог для определения значимых пикселей
// Расширение диапазона с проверкой на границы
while (highBound < 255 && hist.at<float>(highBound) > threshold)
highBound++;
while (lowBound > 0 && hist.at<float>(lowBound) > threshold)
lowBound--;
```

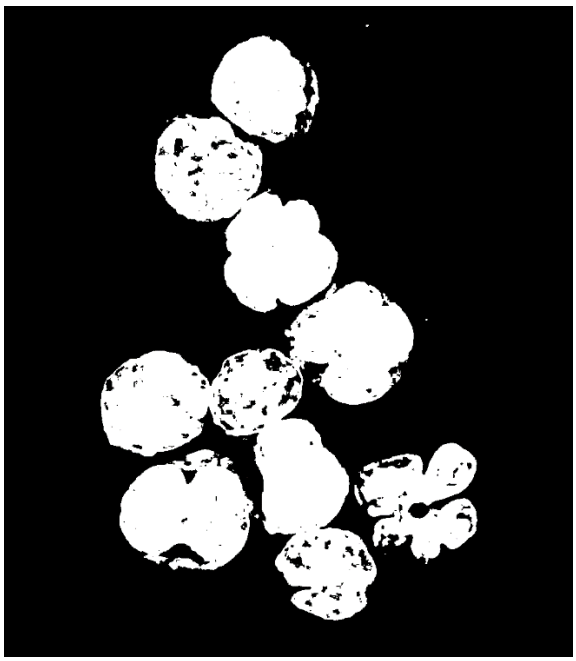

Рисунок 5. После бинаризации

3. Морфологическое замыкание для заполнения маленьких дырочек

```
cv::Mat kernel = cv::getStructuringElement(cv::MORPH_RECT, cv::Size(3, 3));
cv::morphologyEx(detectionsImg,detectionsImg,cv::MORPH_CLOSE, kernel);
```

4. Определение компонентов связности и повторное морфологическое замыкание для каждой компоненты связности.

```
cv::Mat labels, stats, centroids;
int nLabels = cv::connectedComponentsWithStats(detectionsImg, labels, stats, centroids);
// Заполнение оставшихся дыр внутри компонент связности
for (int i = 1; i < nLabels; i++) {
cv::Mat componentMask = (labels == i); // Создаем маску для компонента
// Находим контуры компонента
std::vector<std::vector<cv::Point>> contours;
```

```
cv::findContours(componentMask,contours,  cv::RETR_EXTERNAL,
cv::CHAIN_APPROX_SIMPLE);
// Закрашиваем внутренние области компонента
for (const auto& contour : contours) {
    cv::drawContours(detectionsImg, contours, -1, cv::Scalar(255),
    cv::FILLED);
    }
```

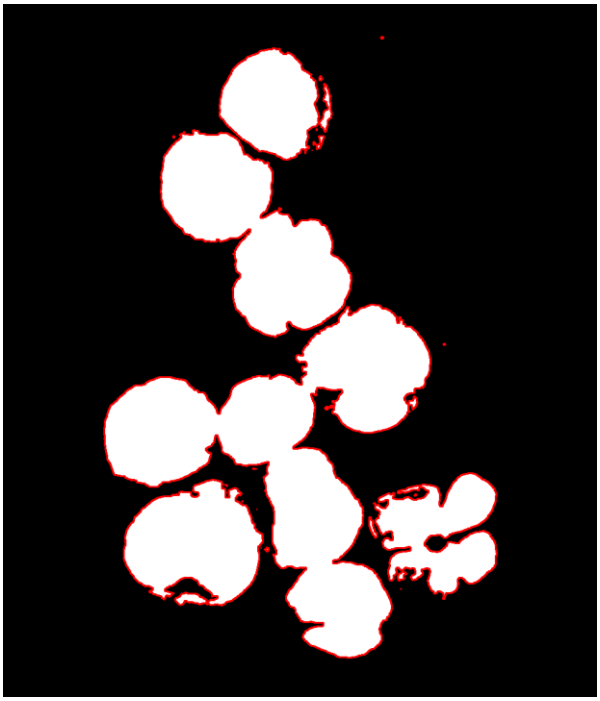

Рисунок 6. Результат работы BGR-алгоритма

5. ИНСТРУКЦИЯ ПО ИСПОЛЬЗОВАНИЮ

1. Скачайте датасет в папку с проектом (если необходимо):

git clone https://github.com/K-Grachev-2106756/blast_cell_dataset.git

2. Соберите проект и запустите “./main.exe”
3. При желании можно указать параметры:
   -mode («BGR» или «HSV»)
   -imgPath (путь до файла для детектирования)
   -groundTruthPath (путь до файла для валидации)

Примеры:

• ./main.exe -mode=HSV – запустит детекцию для каждой картинки датасета и посчитает метрики. Сохранит результаты в папку с датасетом.

• ./main.exe" -imgPath="../blast_cell_dataset/images/4.jpg" -mode=BGR - запустит без валидации детекцию одной картинки. Результат детекции сохранится в папку с файлом.

6. ТЕХНИЧЕСКАЯ ЧАСТЬ

- CMake версия не ниже 3.20.
- Язык C++ 17 стандарта.
- OpenCV версии 4.9.

6.2. Инструкция по установке

1. Скачать проект из репозитория
2. Перейти в папку *«misis2024s-21-03-grachev-k-y».*
3. Прописать в консоли «cmake -B ./build».
4. Прописать в консоли «cmake --build ./build --config Release».
5. Теперь для запуска нужно прописывать «./bin.rel/main.exe …».
6. При желании, можно запустить проверку на датасете, но для этого его необходимо скачать:

- git clone https://github.com/K-Grachev-2106756/blast_cell_dataset.git

7. Так как BGR алгоритм работает долго, советую запускаться на датасете с параметром -mode=HSV

7. ВЫВОД

Мною были разработаны алгоритмы для детекции бластных клеток.

- В алгоритме, который работает в цветовом пространстве HSV, были использованы константные параметры определения цвета объекта детектирования, в связи с чем метрики получились очень плохими. Тем не менее, алгоритм работает очень быстро.

- В алгоритме, который работает в цветовом пространстве BGR, был реализован адаптивный поиск оттенка объекта с использованием k-means кластеризации цветов и поиска эффективных диапазонов по цветовым каналам. Данный алгоритм работает в 20 раз медленнее вышеописанного, но его можно назвать «хорошими».

8. ИСТОЧНИКИ

• "Learning OpenCV 4: Computer Vision with Python 3" by Joseph Howse, Joe Minichino, Cody Brimhall, O'Reilly Media, 2019.
• "OpenCV 4 Computer Vision Application Programming Cookbook" by Robert Laganière, Packt Publishing, 2018.
• "Computer Vision: Algorithms and Applications" by Richard Szeliski, Springer, 2010.
• "Digital Image Processing" by Rafael C. Gonzalez, Richard E. Woods, Pearson Education, 3rd Edition, 2007.
• "Introduction to Machine Vision" by Davies E.R., Newnes, 2011.