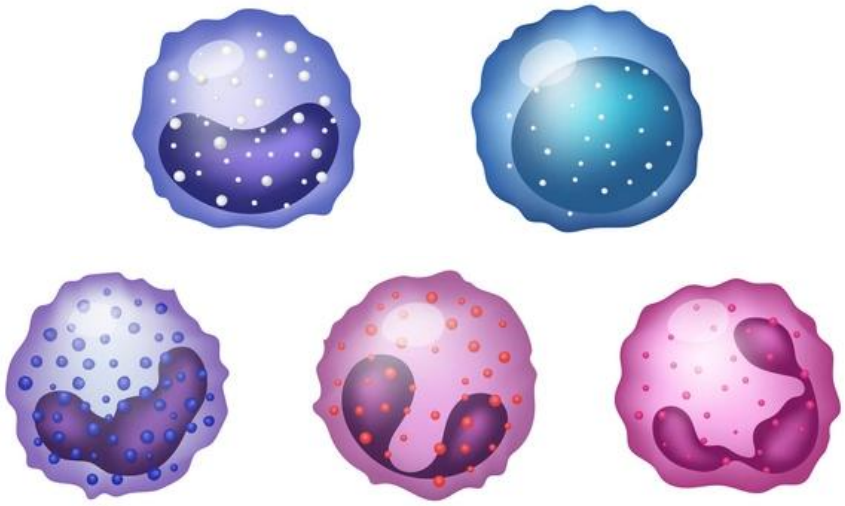
Cairo University – Faculty of Engineering
Image Processing & Computer Vision (CMPS446)

JARVIS

Just A Rather Very Intelligent System



Optimizers
Omar Mahmoud
Seifeldin Mohamed
Karim Mahmoud
Ahmed Mohamed

Supervised By

Eng. Roba Mohamed
Dr. Youssef Ghattas

Table of Contents

Introduction

Blood smear image analysis is a core task in medical laboratories, used to identify, segment, classify, and count different types of blood cells such as red blood cells (RBCs), white blood cells (WBCs), and platelets. Manual analysis under a microscope is time-consuming and highly dependent on the technician's experience, especially in cases of overlapping cells and variable staining conditions. This creates a need for automated and reliable computer vision–based solutions.

This project proposes JARVIS ***(Just A Rather Very Intelligent System)***, an image processing and computer vision framework designed to analyse blood smear images automatically. The system applies preprocessing techniques to enhance image quality, followed by segmentation methods to isolate individual blood cells, including overlapping cells. Extracted features are then used to classify cell types and compute statistical parameters such as cell count, size, and shape descriptors.

In addition to image analysis, the system integrates a gesture-based interaction module that enables hands-free control of the interface. Using contour analysis, convexity defects, and motion tracking, users can navigate results and trigger processing steps without physical contact.

This feature is particularly beneficial in sterile or laboratory environments where touch-based interaction is undesirable.

Dataset

The system is developed and evaluated using publicly available blood smear image datasets containing labelled microscopic images of RBCs, WBCs, and platelets.

These datasets include variations in lighting, staining, and cell overlap, providing a realistic testing environment for segmentation and classification algorithms.

Overall, this project aims to demonstrate how classical image processing techniques and gesture recognition can be combined into a unified system that improves efficiency, accuracy, and usability in blood cell image analysis.

Preprocessing

Purpose of Preprocessing

Blood smear images captured under a microscope often suffer from noise, uneven illumination, staining variations, and background artifacts. These issues negatively affect downstream segmentation and classification tasks.
The goal of the preprocessing stage is to enhance cell visibility while preserving diagnostically important details, particularly small structures such as platelets and overlapping cell boundaries.

Specifically, preprocessing aims to:

- Reduce image noise without destroying fine cellular details
- Enhance contrast to make cell boundaries clearer
- Normalize intensity variations caused by inconsistent lighting
- Improve the separability of RBCs, WBCs, and platelets for reliable segmentation

Colour Space Transformation

The input image is first converted from the RGB color space to the HSV color space. The HSV representation separates color information from intensity, which is beneficial for biomedical images affected by staining variations.

Only the **Value (V) channel** is processed at this stage because:

- **V (Value)** represents brightness and intensity, which is more stable across different staining conditions
- **Hue (H)** varies significantly with stain type and illumination
- **Saturation (S)** is sensitive to noise and may suppress small structures such as platelets

By processing only the V channel, cell shapes and boundaries are enhanced while color distortions are minimized.

Noise Reduction

To remove noise while preserving cellular structures, a **two-stage filtering strategy** is applied to the V channel.

Gaussian Smoothing:

A Gaussian filter is first applied to remove high-frequency sensor noise and minor intensity fluctuations:

- Kernel size: **(5 × 5)**
- Standard deviation (σ): **0** (automatically computed)

Gaussian filtering provides smooth intensity transitions and prepares the image for further denoising.

Median Filtering:

A median filter is then applied to eliminate isolated noise artifacts such as salt-and-pepper noise:

- Kernel size: **3 × 3**

Median filtering is particularly effective in:

- Removing isolated bright/dark pixels
- Preserving edges better than Gaussian

Applying the median filter **after** Gaussian smoothing ensures that platelet structures remain visible. If median is done first, the Gaussian could blur the edges again, reducing platelet visibility.

Contrast Enhancement Using CLAHE

Despite noise reduction, blood smear images often exhibit uneven illumination, especially in dense or overlapping regions. To address this, the image is converted to the LAB color space.

Only the **L (Lightness)** channel is enhanced using **Contrast Limited Adaptive Histogram Equalization (CLAHE)**.

CLAHE parameters:

- Clip limit: **2.0**
- Tile grid size: **(8 × 8)**

CLAHE is used over global histogram equalization because:

- It enhances **local contrast**
- Prevents over-amplification of noise
- Preserves edges in small and overlapping cells

The LAB color space isolates brightness from color components, allowing contrast enhancement without altering the original staining colors.

Final Preprocessed Output

After contrast enhancement, the image is converted back to the RGB color space. The resulting image exhibits:

- Reduced background noise
- Clearer RBC and WBC boundaries
- Enhanced visibility of platelets
- Improved robustness against lighting and staining variations

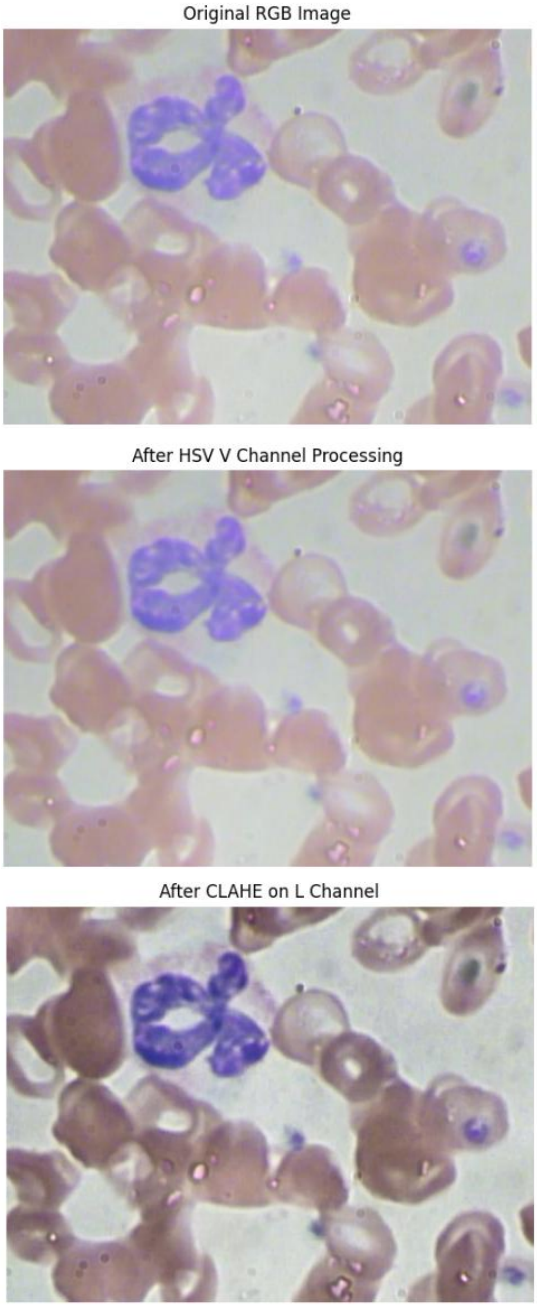
This processed image serves as a reliable input for subsequent **segmentation and feature extraction** stages.

Parameter Summary

Operation	Parameter	Value	Purpose
Gaussian Blur	Kernel Size	(5 × 5)	Smooth high-frequency noise
Gaussian Blur	Sigma	0	Automatic smoothing
Median Blur	Kernel Size	(3 × 3)	Remove isolated noise while preserving edges
CLAHE	Clip Limit	2.0	Prevent Over Enhancement
CLAHE	Tile Grid Size	(8 × 8)	Local contrast enhancement

Resulting image after preprocessing

The next figure shows a representative image **before and after preprocessing**:



Red Blood Cells Segmentation

1. Objective

Red blood cells in blood smear images frequently appear in **dense clusters with significant overlap**, making simple thresholding insufficient for accurate separation.
The objective of the RBC segmentation stage is to:

- Accurately isolate RBC regions from the background
- Separate touching and overlapping RBCs
- Preserve true cell boundaries while suppressing noise and artifacts

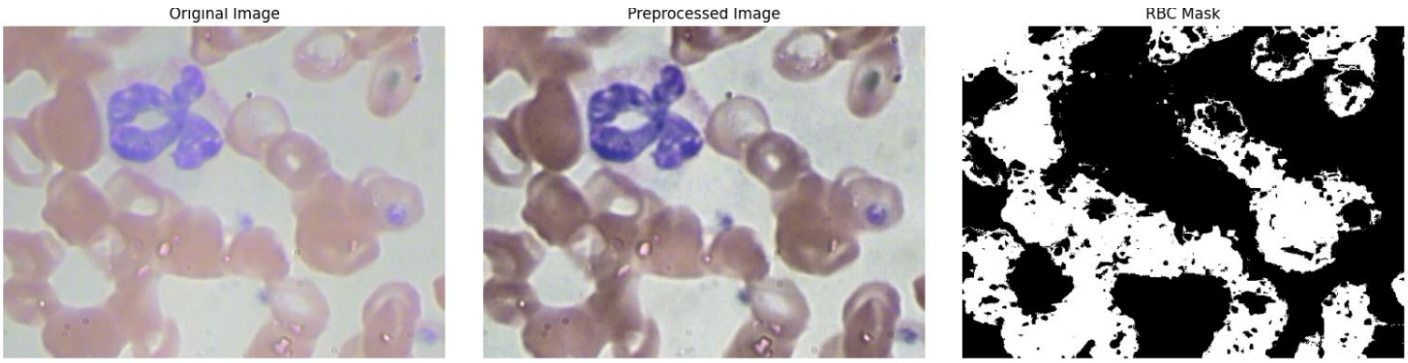
To achieve this, a **marker-controlled watershed approach** is employed, guided by **distance transform-based markers** and **Hessian-based edge constraints**.

2. RBC Extraction Using Color Thresholding

Following preprocessing, RBC regions are initially extracted using **HSV color-based thresholding**. Since RBCs appear in reddish-pink tones, two hue ranges are used to account for hue wrap-around in the HSV color space:

- Lower red range: $H \in [0,10]$
- Upper red range: $H \in [160,179]$

The following figure shows the image after segmentation:



3. Morphological Refinement

To improve mask quality, morphological operations are applied:

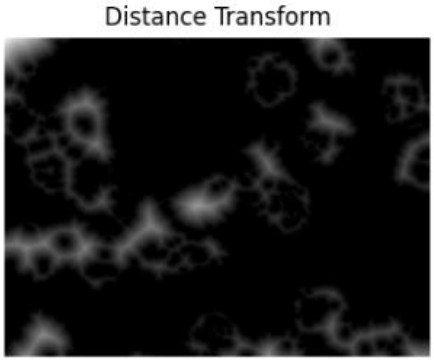
- **Opening** using a 3×3elliptical structuring element:
  - Removes small, isolated noise
  - Suppresses tiny artifacts without affecting RBC bodies
- **Closing** using a 5×5elliptical structuring element:
  - Fills small holes inside RBCs
  - Improves contour continuity

These operations produce a clean binary mask suitable for distance-based analysis.

4. Distance Transform for Cell Center Estimation

A **Euclidean distance transform** is applied to the refined binary mask:

$$\boldsymbol{D(x) = \textit{distance to nearest background pixel}}$$



Key properties:

- Maximum values occur at the center of each RBC
- Values decrease smoothly toward cell boundaries

Parameters:

- Distance type: cv2.DIST_L2 (Euclidean)
- Mask size: 5

This transform provides a robust basis for generating **internal markers** required for watershed segmentation.

5. Adaptive Foreground Marker Generation

To improve separation in clustered regions, connected component analysis is applied to the RBC mask.
Each connected region (representing one or more overlapping RBCs) is processed independently.

For each connected component:

- A **local threshold** is computed as:

$$\boldsymbol{T = 0.325 \times D_{max}}$$

- Pixels with distance values greater than T are selected as **sure foreground.** This adaptive strategy ensures:
  - Robust center detection for both small and large RBCs
  - Improved separation compared to a global threshold

6. Background and Unknown Region

To complete watershed marker preparation:

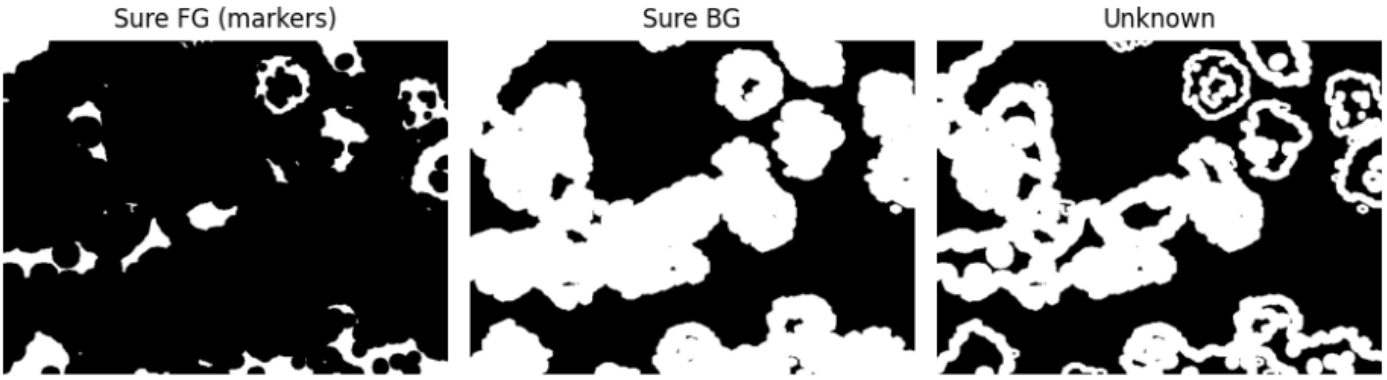
- **Sure background** is obtained by dilating the RBC mask
- **Unknown region** is defined as:

$$Unknown = Sure\ Background - Sure\ Foreground$$

Connected components are then labeled:

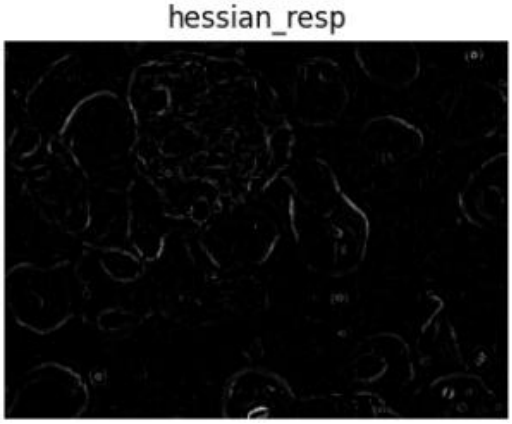
- Background → label 1
- RBC centers → labels 2, 3, 4, ...
- Unknown region → label 0

These markers guide the watershed expansion process.



7. Hessian-Based Boundary Enhancement (Frangi Filter)

To prevent watershed over-segmentation and leakage between touching cells, **Hessian-based Frangi filtering** is applied. The Frangi filter analyzes the **second-order derivatives (Hessian matrix)** of the image to enhance structured boundaries while suppressing noise.



Parameters used:

- Scales (σ): 1 to 7
  - Small scales detect fine edges
  - Medium scales detect typical RBC boundaries
  - Large scales suppress noise and thick artifacts
- $\alpha = 0.2$: favors round, blob-like structures
- $\beta = 0.2$: emphasizes strong boundaries
- $\gamma = 5$:
  - *low:* includes more noise, keeps low contrast regions
  - *high:* suppresses noise, keeps only strong contrast regions,

The Frangi response is normalized to an 8-bit image and used as a boundary constraint during watershed segmentation.

8. Marker-Controlled Watershed Segmentation

The watershed algorithm is applied using:

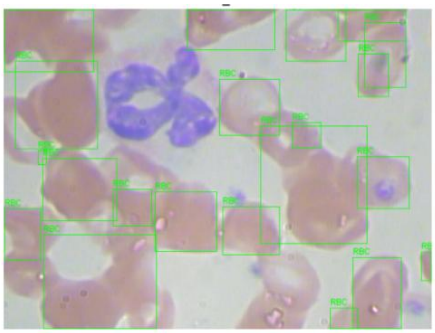
- *Foreground and background markers derived from the distance transform*
- *Hessian-enhanced image as the topographical surface*

The watershed grows regions from the markers until they meet strong boundary responses, effectively separating overlapping RBCs.

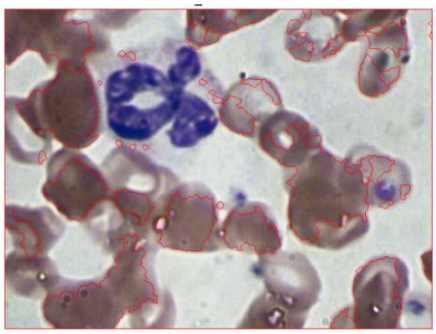
Watershed labelling:

- *Boundary pixels → −1*
- *Background → 1*
- *Individual RBCs → 2, 3, 4, …*

Final RBC boundaries are overlaid with red lines on the pre-processed image for visualization.



Bounding Boxes



Cells Boundaries

Watershed Results

9. Post-Watershed Label Refinement

Although marker-controlled watershed segmentation effectively separates overlapping RBCs, it may still produce **spurious regions**, including small noise fragments and over-segmented cell parts. To address this, a post-processing stage is applied to refine watershed labels and produce biologically valid RBC instances.

a. Extraction of RBC Regions from Watershed Output

The watershed output contains multiple region types:

- *Background regions*
- *Boundary pixels (labeled as −1)*
- *Individual cell regions (labeled ≥ 2)*

To isolate RBCs, only regions with labels greater than 1 are retained, while background and boundary pixels are discarded. This produces a clean mask containing only RBC candidate regions.

b. Connected Component Labelling

The refined RBC mask is subjected to connected component labeling to assign a unique label to each detected region. Each labeled region is then analyzed independently using region-based geometric properties, including:

- *Area*
- *Centroid*
- *Bounding box*

These properties are computed using region-based analysis to support noise removal and fragment merging.

c. Noise Removal Based on Area Thresholding

Very small regions are unlikely to represent true RBCs. To suppress these artifacts, regions with area below a predefined threshold are removed: ***Noise area threshold – 120 pixels***

This step is performed before merging to prevent noise fragments from being incorrectly merged into valid cells.

d. Fragment Merging Based on Spatial Proximity

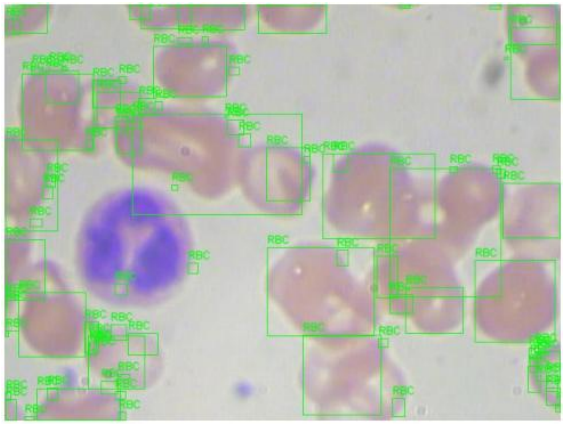
Due to over-segmentation, a single RBC may occasionally be divided into multiple fragments. To reconstruct complete cells, a merging strategy based on **area and centroid distance** is applied.

For each detected region:

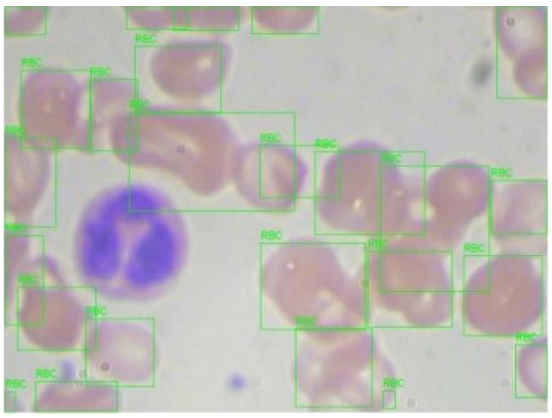
- *If its area is smaller than the minimum expected RBC area:*
  - *Minimum RBC area: 1200 pixels*
- *The centroid distance to neighbouring regions is computed*
- *If the distance is less than a predefined threshold:*
  - *Maximum merge distance: 63 pixels*
- *The fragment is merged with the nearest region*

This iterative merging process continues until no further merges are possible, ensuring stable and consistent labelling.

Before post-watershed

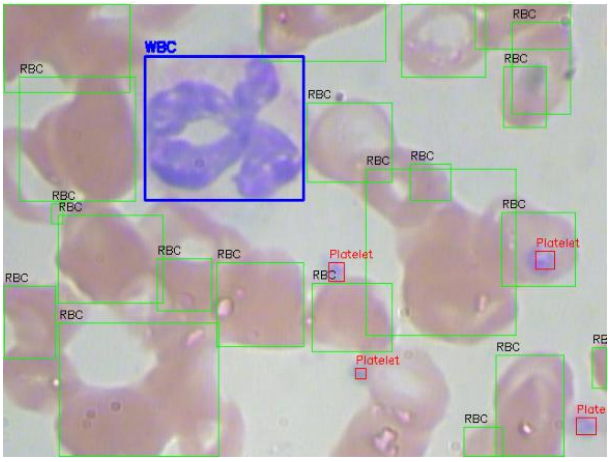


After post-watershed



As you can notice, the small fragments are merged. The counting and feature extraction now will be more accurate.

We did the same labelling for the WBC and platelets where we detected both as well and added each in its own box and merged the 3 detected cells into one image:



Red Blood Cells Features' Extraction

Methodology & Algorithm Design

Features' Extraction

RBC Features: `extract_filtered_rbc_features(img,labels,min_area,max_area)`

1. **Area:** Parameter in the `regionprops()`.
2. **Circularity:** A measure of how close the shape of the rbc is to a perfect circle.
   Perimeter: A parameter in the `regionprops()`.

$$Circularity = 4\pi \times \frac{area}{perimeter^2}$$

3. **Aspect Ratio:** uses the `regionprops()` parameters to calculate the aspect ratio

$$Aspect\ Ratio = \frac{major\ axis\ length}{\min or\ axis\ length}$$

4. **Cell Count:** The number of red blood cells found in the image, in the extract features we use min area and max area to select some red blood cells that would be good for analysis so not too big from the overlapping of multiple cells and not partially shown due to part of the cells are on the edge of the image.
5. **Cell Density:** cells density in the image in proportion to the image per 10k pixels.

$$Density = \frac{count}{total\ area} \times 10000$$

6. **Overlap Ratio:** How many of the detected red blood cells are overlapping with each other.

$$Overlap = \frac{clump\ area\ sum}{total\ cell\ area} \times 100$$

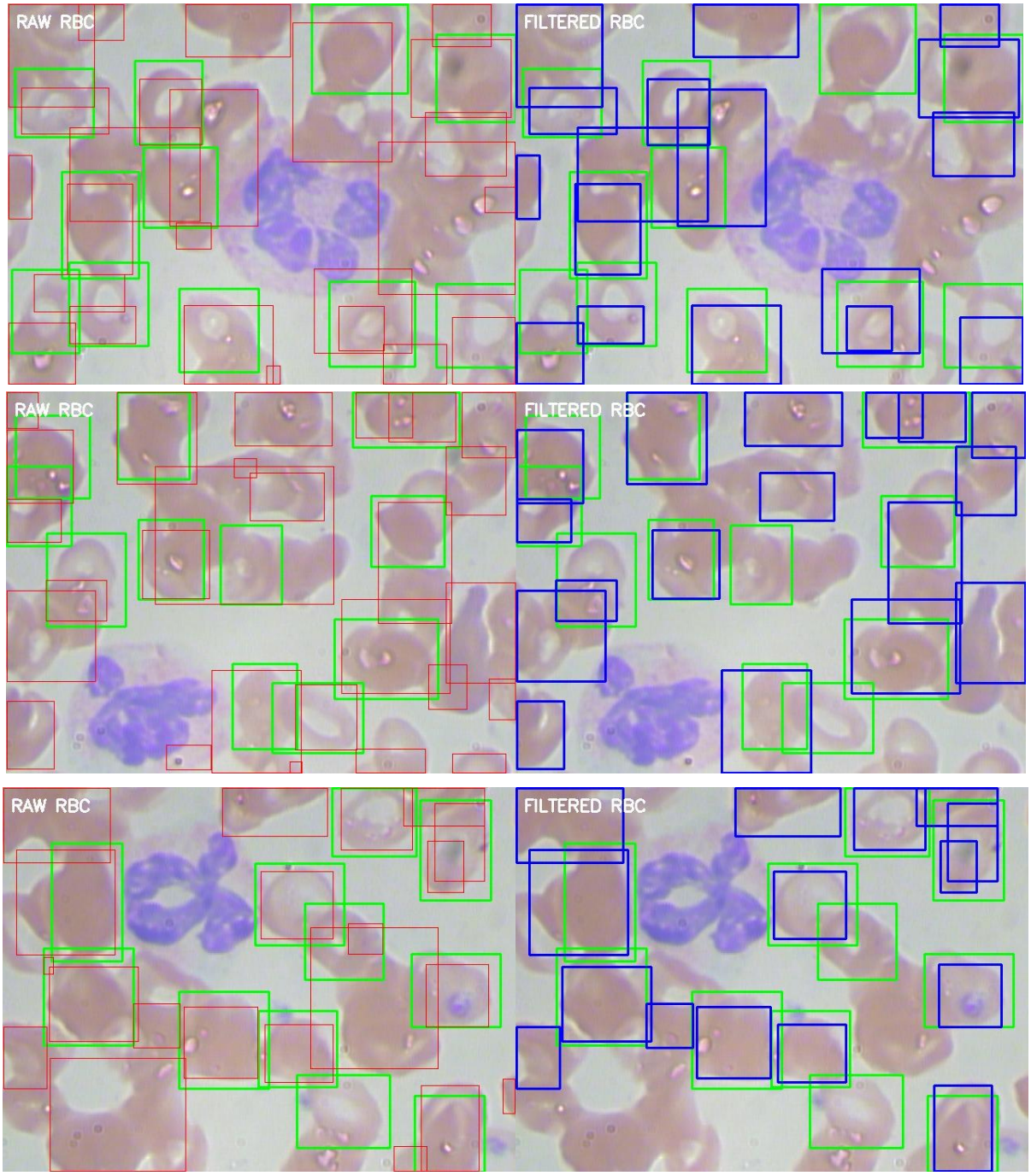
   Clump Area refers to the total area occupied by groups of RBCs that are touching or overlapping, rather than being distinct, single cells.
   Used `cv2.connectedComponentsWithStats(merged_mask)`

   The merged mask is the result of:
   - `cv2.dilate(binary_mask,kernal,iterations=1)`
   - `kernal=np.ones((3,3),np.unit8)`
   - binary mask is the result of applying a threshold on the RBC mask.
     `binary_mask=(rbc_mask>0).astype(np.unit8)`

`regionprops(labels)` Parameters:

1. Area.
2. Bbox.
3. Centroid.
4. Eccentricity.
5. Major_axis_length.
6. Minor_axis_length.
7. Orientation.
8. Perimeter.
9. Extent.
10. Mean intensity

Here are some testcases we can see that:



- The green boxes are the annotation's detected RBC.
- The red boxes are our detected RBC before filtering by area.
- The blue boxes are the filtered RBC.

During our analysis we had contact with a doctor who specialized in this area and while our detected RBC are not like the annotations, the doctor said that the annotations results are not as accurate as ours and we have detected more RBC than the annotation.

Here is an example of the features generated from one of the images:

Parameter	Value	Key Discriminator
Raw RBC Count	35	The total number of objects initially detected by the software before any filtering.
Filtered RBC Count	21	The final count of valid Red Blood Cells after removing noise and non-RBC objects. This is the accurate count used for diagnosis.
Avg, Cell Size	4177.57 pixels	The average area of a single red blood cell in pixels.
Circularity Index	0.378	A measure of how round the cells are (1.0 is a perfect circle). A low value like **0.378** indicates irregular shapes (e.g., Sickle Cells, Ovalocytes) or overlapping cells that distort the shape analysis.
Aspect Ratio	1.617	The ratio of length to width. A value of **1.617** means the cells are significantly elongated (oval or crescent-shaped) rather than round, consistent with Sickle Cell patterns.
Cell Density	0.68/10k pixels	The concentration of cells in a specific image area. This indicates how "crowded" the slide is; a low number suggests a thin smear, while a high number suggests a thick smear.
Overlap Ratio	70.76%	The percentage of cells that are touching or stacked on top of each other.

White Blood Cells Type Classification & Platelets

Methodology & Algorithm Design

Segmentation of WBC

- **Colour Space:** Conversion of RGB images to HSV to isolate the purple cells more easily through hue.
- **Thresholding:** HSV colour-based thresholding (range: Hue 115-130° for purple colour).
- **Morphological Operations:** Opening (to remove background noise/platelets) and Closing (to fill gaps within the segmented nucleus) using erosion and dilation.

Features' Extraction

We extracted 15 distinct biometric features, categorized into three groups:
Nucleus Features: `extract_nucleus_features(mask)`

1. **Area:** The total number of pixels in the nucleus. `cv2.contourArea(cnt)`
2. **Circularity:** A measure of how close the shape of the nucleus is to a perfect circle
   Perimeter: `cv2.arcLength(cnt, True)`
   $$Circularity = 4\pi \times \frac{area}{perimeter^2}$$
3. **Solidity:** The ratio of the nucleus area to its convex hull area, indicating how dense/compact the nucleus is.
   Hull Area: how concave or irregular a cell is `cv2.convexHull(cnt)`
   $$Solidity = \frac{Nucleus\ Area}{Hull\ Area}$$
4. **Number of Lobes:** Estimated by applying morphological erosion `cv2.erode` with 3 iterations to break the connections between lobes. The remaining distinct parts are then counted using `cv2.connectedComponentsWithStats`.

Cytoplasm Features: `extract_cytoplasm_features(img, mask)`

5. **Mean Hue:** The average color (Hue) of the cytoplasm in the HSV color space.
6. **Texture Variance:** A Laplacian filter `cv2.Laplacian` is applied to the grayscale version of the cytoplasm to highlight edges and intensity changes. The statistical variance of these filtered values represents the "graininess" of the cell.
7. **C/N Ratio:** ratio of pixel count of the cytoplasm mask to pixel count of the nucleus mask.

Regional & Metadata Features (Not Used in Type Classification):

8. **Centroid X:** The horizontal center of the detected cell.
9. **Centroid Y:** The vertical center of the detected cell.
10. **Box X1:** Top-left X coordinate of the bounding box.
11. **Box Y1:** Top-left Y coordinate of the bounding box.
12. **Box X2:** Bottom-right X coordinate of the bounding box.
13. **Box Y2:** Bottom-right Y coordinate of the bounding box.
14. **ID:** A unique numerical identifier for the cell within a specific image.
15. **Type:** The final categorical classification of the cell

Experimental Results & Analysis ***(Approach One: IF Conditions)***

Cell Type	Per-Type Accuracy	Recall	Key Discriminator
Lymphocyte	81.25%	76.47%	High Solidity & Low Hue
Neutrophil	82.68%	77.08%	Multi-lobed nucleus & High Texture
Eosinophil	46.15%	47.37%	Orange-pink Hue range (78-105°)
Monocyte	60.00%	14.29%	Large area, low solidity (Challenging)
Basophil	0.00%	0.00%	Insufficient data (only 3 photos)
Overall Accuracy 65.54%			

METRIC DEFINITIONS:

- ***Precision:*** *Of all predicted as this class, % that were correct*
- ***Recall:*** *Of all actual instances of this class, % we found*

Attempt Number	Overall Accuracy
#0	22.58%
#1	53.04%
#2	58.13%
#3	64.02%
#4	65.54%

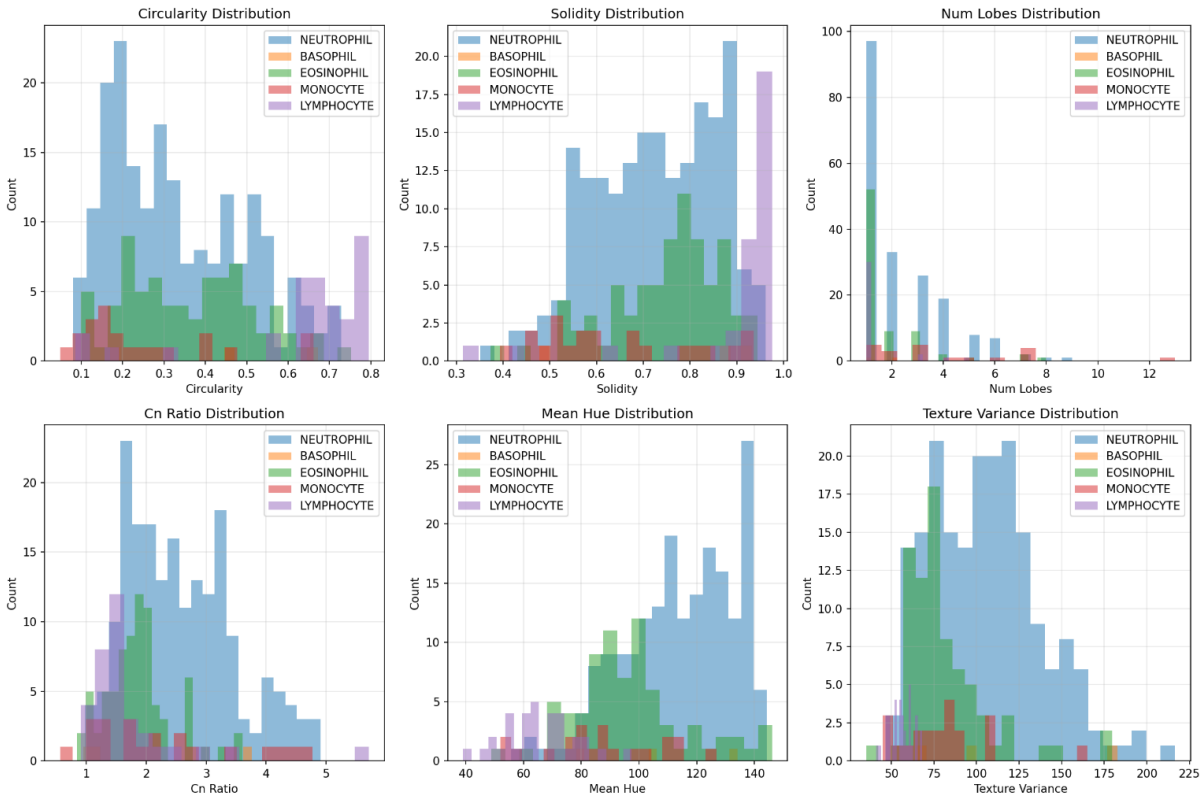
How was this accomplished:

- A python script was created and used to evaluate accuracy of our results compared to actual type (available with dataset).
- Script gives ranges for each type which is reused each time to increase accuracy until overall accuracy converged

Common Misclassification:

```
NEUTROPHIL → EOSINOPHIL :  34 cases
EOSINOPHIL → NEUTROPHIL :  24 cases
EOSINOPHIL → BASOPHIL   :  14 cases
NEUTROPHIL → BASOPHIL   :   7 cases
MONOCYTE → EOSINOPHIL   :   6 cases
MONOCYTE → NEUTROPHIL   :   6 cases
LYMPHOCYTE → BASOPHIL   :   5 cases
MONOCYTE → BASOPHIL     :   5 cases
NEUTROPHIL → LYMPHOCYTE :   3 cases
LYMPHOCYTE → EOSINOPHIL :   2 cases
BASOPHIL → NEUTROPHIL   :   1 cases
EOSINOPHIL → MONOCYTE   :   1 cases
MONOCYTE → LYMPHOCYTE   :   1 cases
EOSINOPHIL → LYMPHOCYTE :   1 cases
BASOPHIL → LYMPHOCYTE   :   1 cases
LYMPHOCYTE → MONOCYTE   :   1 cases
```

Features Distribution:



Experimental Results & Analysis ***(Approach Two: KNN)***

Cell Type	Per-Type Accuracy	Recall	Key Discriminator
Lymphocyte	74.36%	85.29%	High Solidity & Low Hue
Neutrophil	75.89%	88.54%	Multi-lobed nucleus & High Texture
Eosinophil	59.32%	46.05%	Orange-pink Hue range (78-105°)
Monocyte	66.67%	9.52%	Large area, low solidity (Challenging)
Basophil	0.00%	0.00%	Insufficient data (only 3 photos)
Overall Accuracy 72.62%			

METRIC DEFINITIONS:

• ***Precision:*** *Of all predicted as this class, % that were correct* • ***Recall:*** *Of all actual instances of this class, % we found*

The training dataset was prepared through the following steps:

1. Features were extracted from labelled WBC images using the custom feature extraction functions created in ***Approach One***
2. Extracted features were organized in an Excel spreadsheet with three main columns:
   - True WBC Type (ground truth labels)
   - Nucleus Features
   - Cytoplasm Features
3. Data quality control was performed by excluding rows with inconsistent or unreliable measurements to ensure accurate model training
4. The cleaned dataset was used to train the KNN classifier

The classification of unknown WBC samples follows this workflow:

1. The custom feature extraction functions ***(from Approach One)*** process the input WBC image
2. Nucleus and cytoplasm features are computed
3. The extracted features are provided as input to the trained KNN classifier
4. The KNN identifies the k-nearest neighbours in the feature space based on the training data
5. The WBC type is classified based on the majority class among the nearest neighbours

Common Misclassification:

```
EOSINOPHIL → NEUTROPHIL       :  39 cases
NEUTROPHIL → EOSINOPHIL       :  18 cases
MONOCYTE → NEUTROPHIL         :  11 cases
MONOCYTE → EOSINOPHIL         :   5 cases
NEUTROPHIL → LYMPHOCYTE       :   4 cases
MONOCYTE → LYMPHOCYTE         :   3 cases
LYMPHOCYTE → NEUTROPHIL       :   3 cases
EOSINOPHIL → LYMPHOCYTE       :   2 cases
BASOPHIL → NEUTROPHIL         :   1 cases
BASOPHIL → LYMPHOCYTE         :   1 cases
LYMPHOCYTE → MONOCYTE         :   1 cases
LYMPHOCYTE → EOSINOPHIL       :   1 cases
```

*Comparison with a Referenced Deep Neural Network **(DNN)***

Metric	Our Systems		DNN
	(1) IF Conditions	(2) KNN	
Lymphocyte Accuracy	81.25%	74.36%	100.00%
Neutrophil Accuracy	82.68%	75.89%	98.00%
Eosinophil Accuracy	46.15%	59.32%	98.00%
Monocyte Accuracy	60.00%	66.67%	99.00%
Overall Accuracy	65.54%	72.62%	99.00%
Feature Extraction	**Manual:** Features like Circularity, Solidity, and Hue are "hand-crafted" based on medical knowledge.	**Manual:** Uses same hand-crafted features as IF Conditions	**Automated:** The network learns its own hierarchical features (edges → shapes → cell types) from raw pixels
Logic	**Rule-Based:** Uses "If-Then" gates	**Probabilistic:** Finds k-nearest neighbours in feature space	**Probabilistic:** Uses complex mathematical weights to calculate a probability for each class
Speed	**Very Fast:** Runs nearly instantly	**Fast:** Quick nearest-neighbour lookup	**Slow:** Requires significant matrix math, often needs a GPU for speed.
Data Requirements	**Low:** requires tens/hundreds of images	**Low:** requires tens/hundreds of images	requires thousands of images
Interpretability	**White Box:** You can look at the code and see exactly why a cell was classified as a "Neutrophil"	Gray Box: Can see which training samples influenced decision	**Black Box:** It is extremely difficult to explain why the internal weights chose one class over another.
Robustness	**Low:** Changes in lighting or staining chemicals can break the hard-coded Hue thresholds.	**Medium:** Somewhat robust if training data covers variations	**High:** Can be trained with Data Augmentation to be immune to lighting and rotation changes.

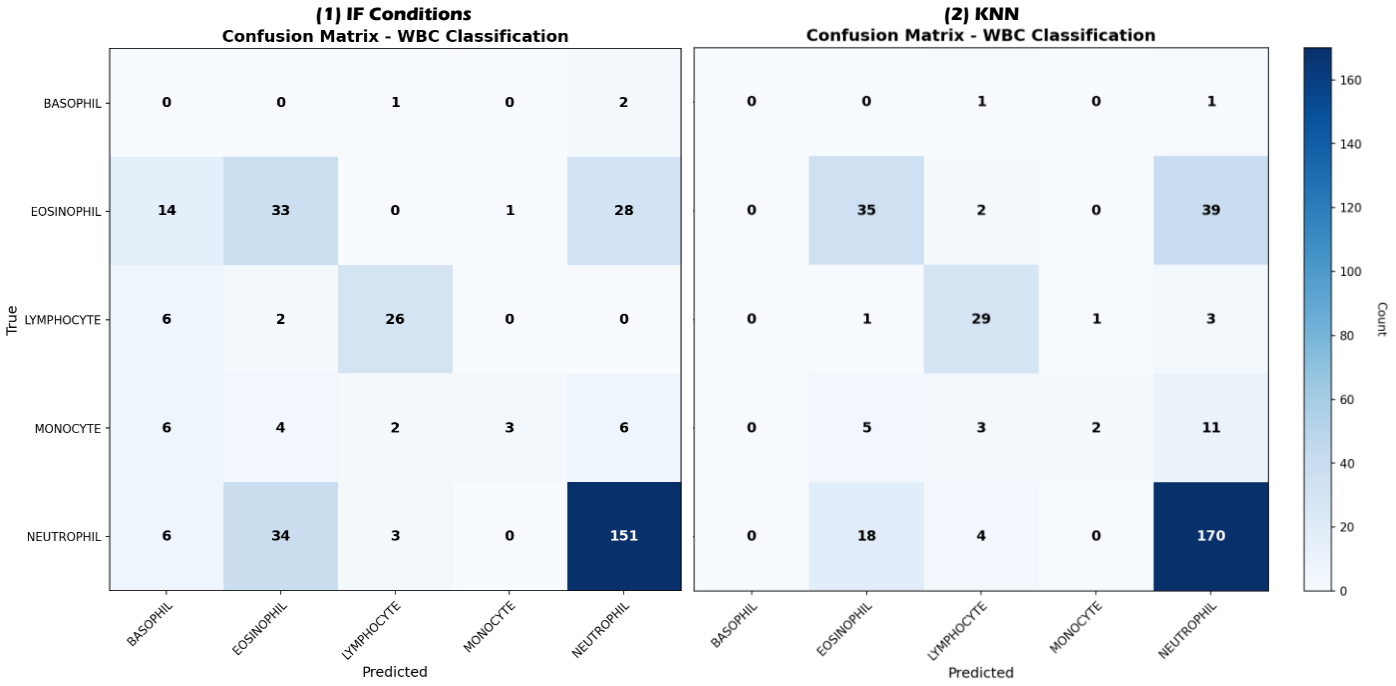
Traditional Approach Challenges (Our Systems)

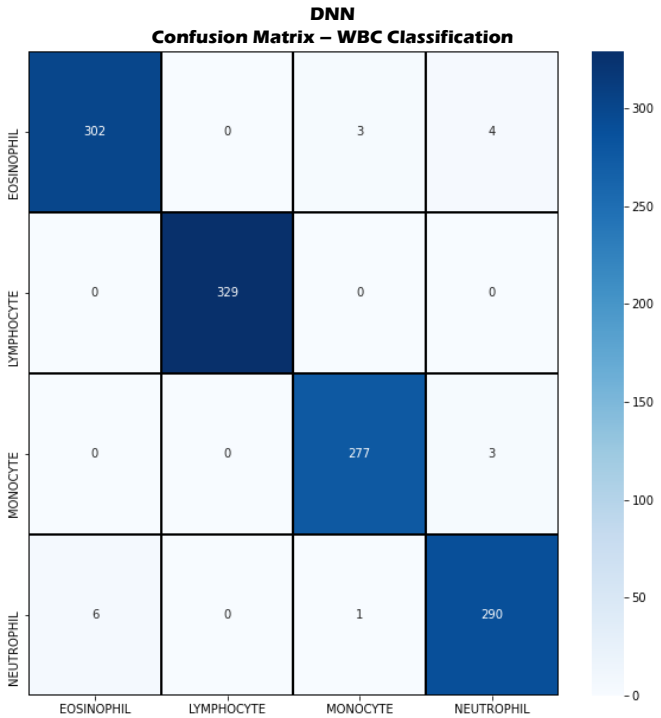
- **The Overlap Problem:** In our system, *Neutrophils* and *Eosinophils* are being confused. This is because their Hue and Texture ranges overlap in a way that a flat threshold cannot separate them.
- **Staining Variation:** If a laboratory uses a slightly different Giemsa stain (dyes used on samples to give it a colour, the entire code must be manually re-tuned for a specific stain.

DNN Approach Challenges

- **Small Sample Classes:** A DNN would likely fail to learn a class entirely because it doesn't have enough examples to find a pattern, whereas you can manually code a rule for it.
- **Segmentation Errors:** DNNs are often better at classification but can be tricked by artifacts like overlapping red blood cells if not trained specifically to ignore them.

Confusion Matrix Comparison (Our System vs DNN)





Platelets

Methodology & Algorithm Design

Segmentation of Platelets (Same as WBC), but we differentiate between them using area

- **Colour Space:** Conversion of RGB images to HSV to isolate the purple cells more easily through hue.
- **Thresholding:** HSV colour-based thresholding (range: Hue 115-130° for purple colour).
- **Morphological Operations:** Opening (to remove background noise/platelets) and Closing (to fill gaps within the segmented nucleus) using erosion and dilation.

Features' Extraction `extract_platelet_features(img,labels)`

Used `regionprops(labels)` which already gets the area of the cells:

1. **Area**
2. **Circularity:** A measure of how close the shape of the Platelets is to a perfect circle.

$$\boldsymbol{Circularity = 4\pi \times \frac{area}{perimeter^2}}$$

3. **Aspect Ratio:** uses the `regionprobs()` parameters to calculate the aspect ratio

$$\boldsymbol{Aspect\ Ratio = \frac{major\ axis\ length}{\min or\ axis\ length}}$$

4. **Cell Count:** The number of platelets found in the image.
5. **Cell Density:** cells density in the image in proportion to the image per 10k px.

$$\boldsymbol{Density = \frac{count}{total\ area} \times 10000}$$

6. **Overlap Ratio:** How many of the detected red blood cells are overlapping with each other.

$$\boldsymbol{Overlap = \frac{clump\ area\ sum}{total\ cell\ area} \times 100}$$

Hand Gestures

Hand Detection and Feature Extraction Overview

In addition to automated blood cell analysis, this project incorporates a **gesture-based interaction module** that enables intuitive human–computer interaction. This module allows the user to control or interact with the system using hand gestures captured through a camera, without the need for physical input devices.

The core objective of this stage is **robust hand segmentation and shape analysis** under real-time conditions. Unlike blood cell images, hand images are affected by varying illumination, background clutter, and skin tone variations. Therefore, a dedicated detection and feature extraction pipeline is required.

The hand processing pipeline is designed to:

- Accurately isolate the hand region from the background
- Reduce noise and small artifacts that may interfere with shape analysis
- Extract geometric and topological features that describe hand posture
- Analyze hand convexity properties rather than explicit finger counting, enabling more stable gesture recognition

Instead of directly counting fingers, this approach relies on **convexity defects**, which represent concave regions between protrusions of the hand contour. The number and distribution of these defects provide a more reliable descriptor of hand shape, especially in cases of partial occlusion, bent fingers, or varying hand orientations.

The overall process consists of two main stages:

1. **Hand detection and segmentation** using color-space thresholding and morphological filtering
2. **Feature extraction** based on contour geometry, convex hull analysis, and convexity defect evaluation

The following sections detail each stage of this pipeline, explaining the rationale behind the chosen techniques and the parameters used to achieve robust and consistent hand gesture representation.

Hand Detection and Feature Extraction Process

1. Color Space Conversion for Skin Detection (YCrCb)

The input frame is initially represented in the BGR color space, where color information and illumination are tightly coupled. In such representations, variations in lighting conditions such as shadows, highlights, or camera exposure directly affect pixel intensities across all three channels, making reliable skin detection difficult.

To address this limitation, the frame is converted to the **YCrCb color space**, which explicitly separates **luminance information** from **chrominance information**:

- **Y (Luminance)** encodes the brightness or intensity of the pixel.
- **Cr (Red-difference chroma)** and **Cb (Blue-difference chroma)** encode color information independent of brightness.

This separation is particularly advantageous for skin detection because **human skin tones occupy a compact and well-defined range in the Cr–Cb plane**, regardless of ethnicity or moderate illumination changes. As a result, skin pixels remain clustered even when lighting varies, while non-skin objects tend to disperse across a wider chrominance range.

By isolating color information in the Cr and Cb channels, the influence of shadows, highlights, and global illumination variations is significantly reduced. This allows skin segmentation to be performed primarily based on chrominance thresholds, rather than raw intensity values.

Consequently, converting to YCrCb enhances:

- Robustness against illumination changes
- Consistency across different skin tones
- Stability of threshold-based segmentation

This transformation forms a critical preprocessing step, enabling more reliable and accurate extraction of hand regions in subsequent thresholding and morphological operations.

2. Why HSV Was Not Used

Although HSV is widely used in color-based segmentation, it is less suitable for robust skin detection. The **Hue** component becomes unstable under low saturation, shadows, and highlights, which are common in hand regions and near boundaries. Additionally, **Saturation** is highly affected by illumination changes, making it difficult to define consistent threshold ranges.

In contrast, **YCrCb separates luminance from chrominance**, allowing skin color to be detected using the Cr and Cb channels with minimal sensitivity to lighting variations. Skin pixels also form a more compact and stable cluster in the Cr–Cb space, resulting in more reliable segmentation.

Therefore, YCrCb was preferred over HSV due to its superior illumination robustness and threshold stability.

3. Skin Color Thresholding

A binary skin mask is generated by applying predefined thresholds on the YCrCb image:

- Lower threshold: [0, 130, 70]
- Upper threshold: [255, 180, 145]

Pixels falling within this range are classified as skin, while all other pixels are suppressed.
This step produces a coarse segmentation of the hand region but may still contain noise and small false-positive areas.

4. Mask Smoothing Using Gaussian Filtering

To reduce high-frequency noise introduced during thresholding, a **Gaussian blur** is applied to the binary mask:

- Kernel size: 3 × 3
- Standard deviation: σ = 0 (automatically computed)

This operation smooths jagged edges and suppresses isolated noisy pixels, preparing the mask for morphological processing.

5. Morphological Refinement

Morphological operations are applied to clean and stabilize the segmented hand region:

1. **Erosion (2 iterations)**
    a. Removes small, isolated regions and thin noise artifacts
    b. Shrinks boundaries slightly to eliminate weak connections

2. **Dilation (2 iterations)**
    a. Restores the main hand shape after erosion
    b. Fills small gaps and holes inside the hand region

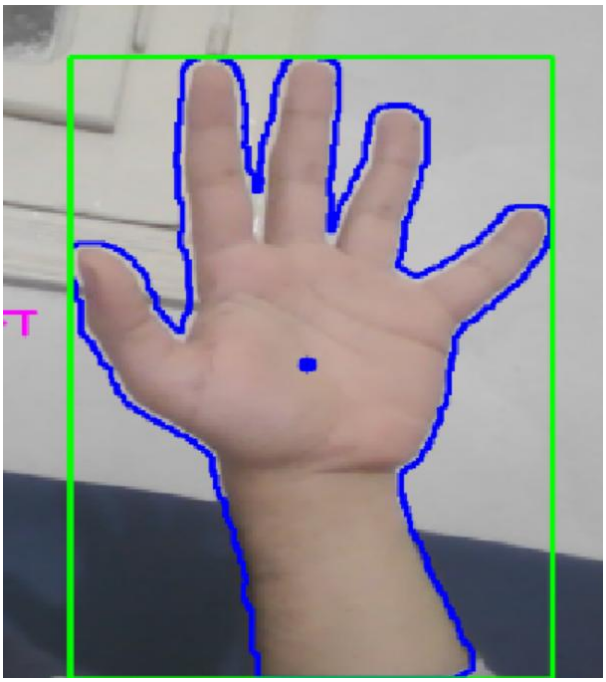
This sequence (erosion followed by dilation) effectively performs **opening**, which preserves large structures while removing noise.

6. Contour Extraction

Contours are extracted from the refined binary mask using:

- Retrieval mode: cv2.RETR_EXTERNAL
  → Only the outermost contours are detected
- Approximation method: cv2.CHAIN_APPROX_SIMPLE
  → Reduces the number of stored points while preserving contour shape

Among all detected contours, the one with the **maximum area** is selected as the hand contour, assuming the hand is the dominant object in the frame.



7. Centroid Computation Using Spatial Moments

The spatial moments of the hand contour are computed to determine its centroid:

- The centroid provides a stable reference point for tracking hand movement
- Division-by-zero is handled to ensure numerical stability

The centroid is later used for gesture interpretation and spatial reasoning.

8. Convex Hull Construction

A **convex hull** is constructed around the hand contour.
The convex hull represents the smallest convex shape that fully encloses the hand.

This step is essential for identifying concave regions between fingers and forms the basis for convexity defect analysis.

9. Convexity Defect Analysis

Convexity defects are extracted by comparing the hand contour to its convex hull.
Each defect corresponds to a concave region between two convex points, often located between adjacent fingers.
For each detected defect:

- The depth of the defect is computed
- The angle between defect points is calculated using geometric relations

A defect is considered valid if:

- The angle is less than **90°**
- The defect depth exceeds **80 units**

Unlike traditional finger-counting approaches, this system **counts convexity defects**, which provides a more stable descriptor of hand posture, especially when fingers are partially bent or occluded.

10. Geometric Feature Extraction

Additional geometric features are computed from the hand contour:

- **Bounding box**: Spatial extent of the hand
- **Area**: Size of the detected hand region
- **Perimeter**: Length of the hand contour
- **Aspect ratio**: Ratio between width and height of the bounding box
- **Circularity**: Shape compactness measure

These features form a compact representation of hand shape and are later used for gesture classification or interaction logic.

11. Feature Vector Construction

All extracted features are grouped into a unified feature structure containing:

- Centroid coordinates
- Convex hull
- Convexity defects count
- Bounding box parameters
- Area, aspect ratio, and circularity

This feature vector provides a robust and interpretable representation of hand posture suitable for real-time gesture-based interaction.

Static and Dynamic Hand Gesture Interpretation

After extracting geometric and topological features from the detected hand region, a rule-based interpretation stage is applied to distinguish valid hands, recognize static gestures, and detect dynamic motion-based gestures.

This stage converts low-level shape descriptors into meaningful interaction commands.

1. Hand Validation Based on Geometric Constraints

Not every detected contour corresponds to a valid hand. False detections may arise from background objects, face regions, or partial occlusions.

To ensure reliable gesture recognition, a **hand validation step** is applied using geometric constraints derived from the extracted features.

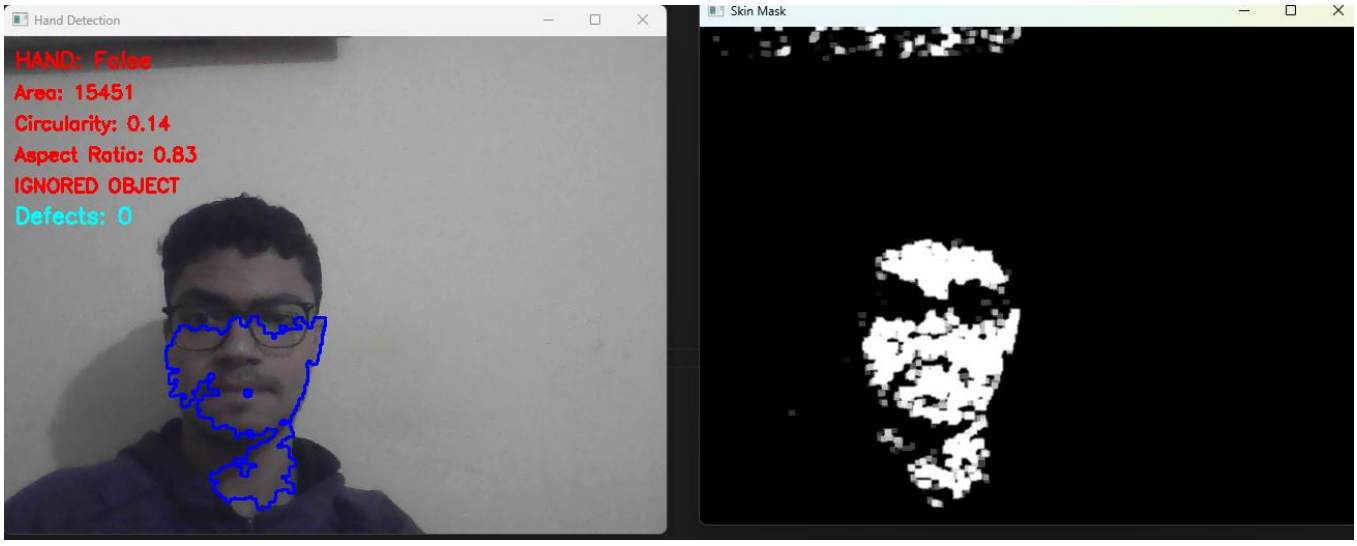
A detected object is classified as a valid hand if:

- *Its area lies within a predefined range:*
  - *Minimum area: 40,000 pixels*
  - *Maximum area: 100,000 pixels*
- *Its circularity lies within a realistic hand-shape interval:*
  - *Minimum circularity: 0.08*
  - *Maximum circularity: 0.75*

These constraints ensure that:

- *The hand is sufficiently close to the camera*
- *The hand does not occupy the entire frame*
- *Highly circular objects (e.g., faces) and extremely irregular shapes are rejected*

Only validated hand regions are passed to the gesture interpretation stage.



2. Static Gesture Recognition Using Shape Heuristics

Static gestures are identified using a combination of **convexity defect count**, **aspect ratio**, **area**, and **circularity**.

Unlike machine-learning-based approaches, this system employs **interpretable rule-based logic**, making it robust and computationally efficient for real-time applications.

Feature-Based Gesture Representation

The following features are used:

- *Convexity defect count: Represents concave regions between fingers*
- *Aspect ratio: Width-to-height ratio of the hand bounding box*
- *Area: Area of the hand*
- *Circularity: Indicates how the shape is circular*

Importantly, the convexity defect count is used **instead of direct finger counting**, providing more stability under partial occlusion and finger bending.

Static Gesture Classification Rules

Based on the extracted features, static gestures are classified as follows:

FIST Gesture

- *Convexity defects: 0*
- *Aspect ratio: greater than 0.6*

This configuration corresponds to a closed hand with minimal concavities and a compact shape.

PALM Gesture

- *Convexity defects: ≥ 3*
- *Area: greater than 45,000 pixels*

This represents an open hand with multiple concave regions between fingers.

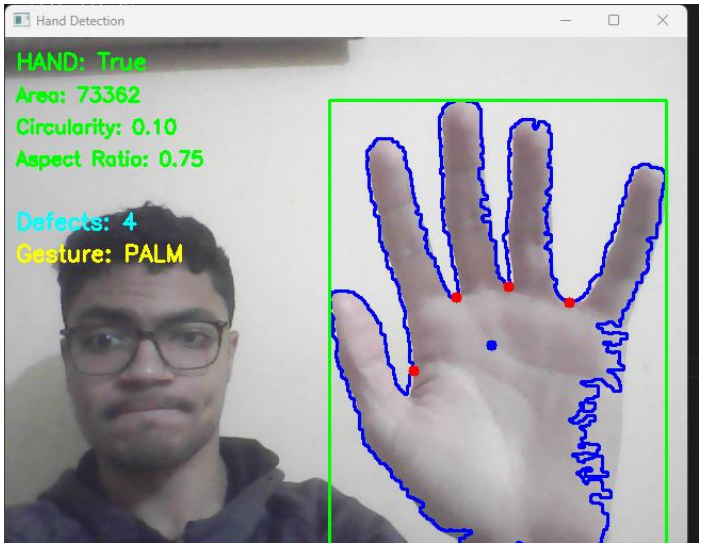
POINT Gesture

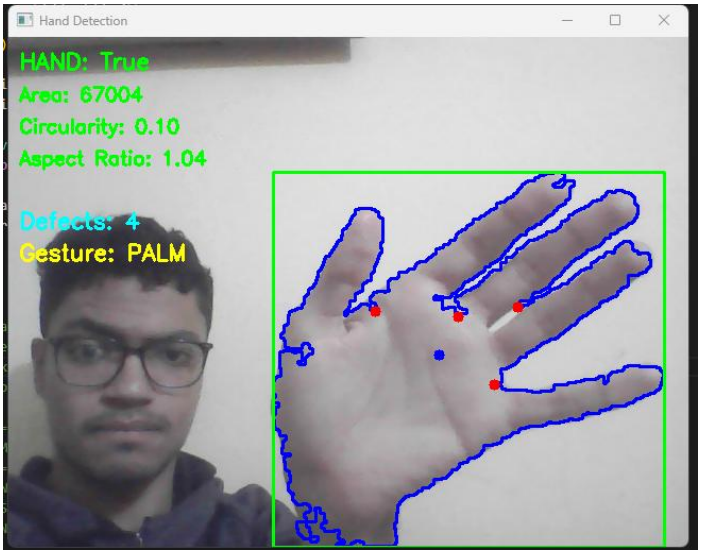
- *Convexity defects: 1*
- *Aspect ratio: between 0.4 and 0.7*

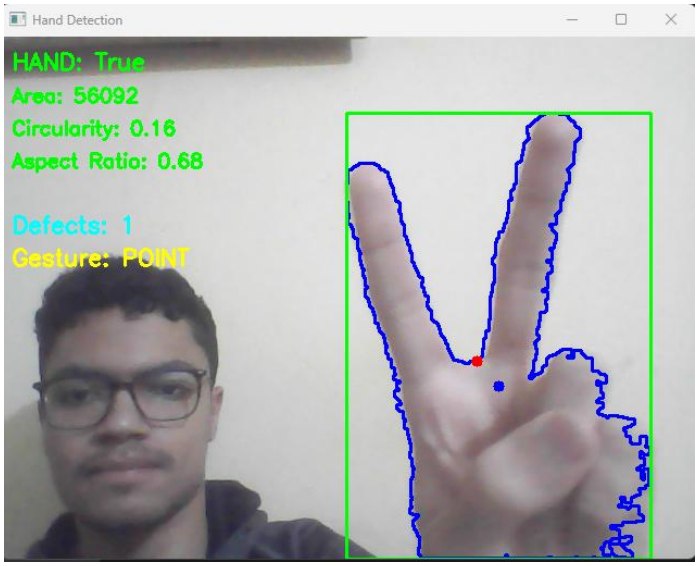
This configuration corresponds to a pointing or “victory-like” posture, typically used for cursor interaction.

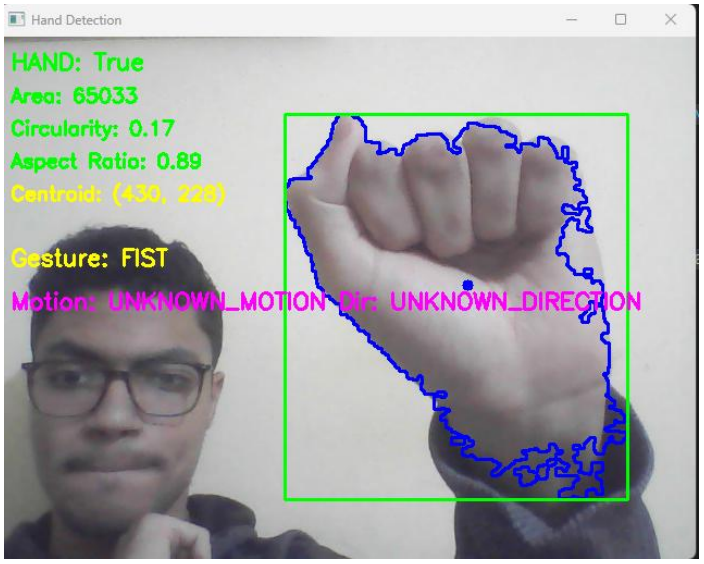
UNKNOWN Gesture

If none of the above conditions are satisfied, the gesture is classified as **UNKNOWN**, preventing ambiguous shapes from triggering unintended actions.









3. Motion Detection Using Centroid Displacement

Dynamic gestures are detected by analyzing the **movement of the hand centroid** over a sequence of frames. ***20 frames*** between the initial and final centroids gave the best results.

Given:

- *Initial centroid:* (x_1, y_1)
- *Final centroid:* (x_2, y_2)

The displacement vector is computed as:

- $dx = x_2 - x_1$
- $dy = y_2 - y_1$

The Euclidean distance of motion is then calculated to determine whether meaningful motion has occurred.
A motion is considered valid only if:

- *Displacement distance* $\geq$ *100 pixels*

This threshold prevents minor hand movements from being misclassified motion.

4. Motion Type Classification Using Gesture Transitions

The system incorporates **gesture state transitions** to distinguish different motion types.

The motion category is determined by comparing the static gesture at the start and end of the movement:

- *PALM → PALM*
- *PALM → FIST*
- *FIST → PALM*
- *FIST → FIST*

This allows identical motion directions to produce different semantic meanings depending on hand posture evolution.

5. Direction Estimation Using Dominant Axis Analysis

To avoid ambiguous diagonal motions, direction detection is constrained to **dominant axes** only.

A motion is classified as horizontal if:

- $|dx| > 1.4 \times |dy|$

A motion is classified as vertical if:

- $|dy| > 1.4 \times |dx|$

The resulting directions are:

- *LEFT / RIGHT*
- *UP / DOWN*

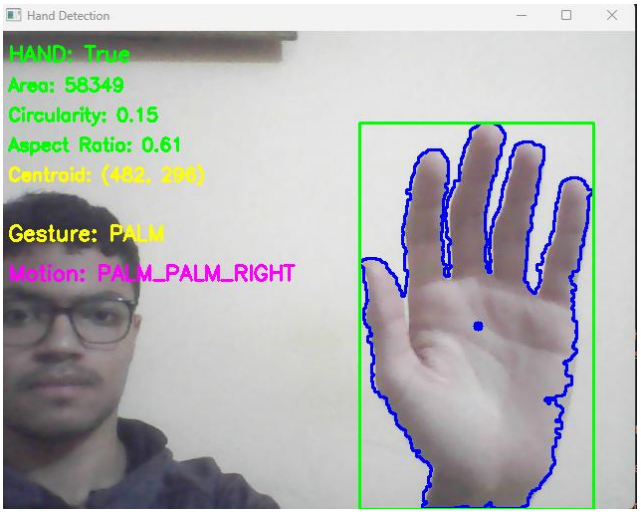
Diagonal or weakly dominant movements are ignored to ensure deterministic gesture behavior.

6. Final Motion Output Representation

Each detected dynamic gesture is represented by a tuple:

- *Motion type (gesture transition)*
- *Motion direction*

This structured output enables high-level interaction logic, such as command execution, navigation, or application control.



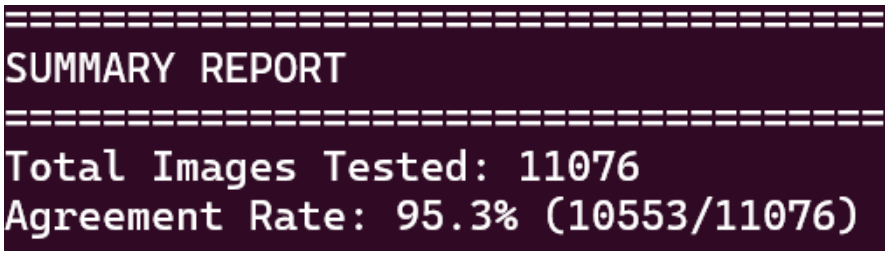
Testing Against Different cases

We conducted extensive testing of our custom hand detection algorithm on a dataset comprising over **10,000 images**, each containing hands in a wide variety of forms, poses, and orientations, set against diverse and complex backgrounds. The primary goal of our approach was to detect the hand itself with high precision, rather than performing general object detection or gesture recognition.

Our method is based on **classical computer vision techniques**, including color-based segmentation, contour analysis, and morphological operations. These techniques are specifically tailored to focus on the hand region, making the algorithm highly efficient for controlled detection tasks. However, we acknowledge that classical CV approaches are inherently limited: they perform best when the target (hand) is the main object in the scene and may struggle in general everyday scenarios with cluttered or unpredictable environments. For broader, real-world hand detection applications, **deep learning methods** such as MediaPipe are more suitable because they can generalize better across diverse conditions and backgrounds.

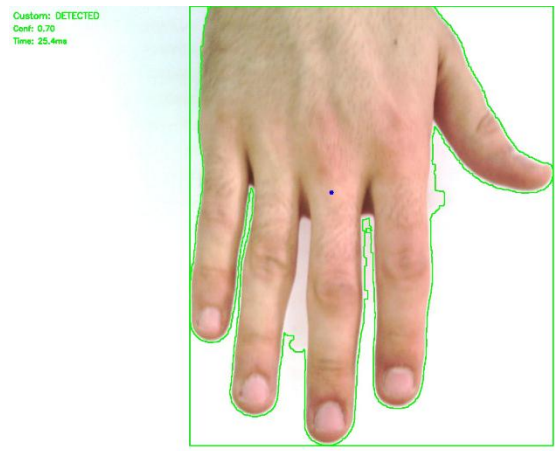
To evaluate the effectiveness of our approach, we compared our custom hand detection algorithm against **MediaPipe's hand_detection.task deep learning model**, which represents a state-of-the-art solution for hand detection in a wide range of scenarios. Our evaluation metric focused solely on the presence or absence of a hand in the image, ignoring finer-grained tasks such as hand landmarks or gesture classification.

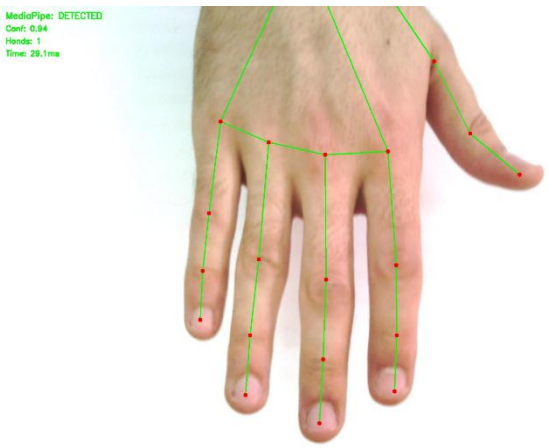
The results were highly encouraging: our classical CV method achieved an **accuracy of 95.3%**, which is very close to MediaPipe's performance. This demonstrates that, despite its simplicity, classical computer vision can be extremely effective for targeted hand detection tasks, especially in controlled or semi-controlled environments. Furthermore, our approach offers advantages in terms of **speed, computational efficiency, and ease of deployment**, since it does not require GPU acceleration or large-scale model training.



In conclusion, while deep learning-based solutions like MediaPipe are essential for everyday, real-world hand detection and tracking, our experiments show that classical computer vision techniques can still provide **highly accurate and reliable results** for focused hand detection problems. This suggests a complementary approach: classical methods can be used for lightweight, fast hand detection, while deep learning methods can handle more complex and unpredictable scenarios.

Sample Test Cases





Conclusion

This project presents JARVIS, an integrated computer vision system designed for automated blood smear image analysis combined with gesture-based interaction. By applying classical image preprocessing, segmentation, and feature extraction techniques, the system aims to accurately detect, separate, classify, and count different blood cell types, even in the presence of overlapping cells and variable imaging conditions.

The inclusion of a gesture recognition module enables hands-free control of the system, enhancing usability in sterile and laboratory environments where touch-based interaction is impractical. This dual approach demonstrates how image processing and human–computer interaction techniques can be effectively combined to support medical analysis workflows.

Overall, the project highlights the potential of classical computer vision methods to provide reliable, interpretable, and efficient solutions for medical image analysis, while also improving user interaction through touchless control. The proposed system serves as a foundation that can be further extended with advanced learning-based methods and larger datasets in future work.

References

[1] "Blood Cells Counting using Image Processing,"
International Journal for Research in Applied Science and Engineering Technology (IJRASET) (2020).

[2] Y.-M. Chen, J.-T. Tsai, and W.-H. Ho
"Automatic identifying and counting blood cells in smear images by using single shot detector and Taguchi method," in Proc. Int. Conf. Biomed. Eng. Innovation (ICBEI) 2019–2020, Singapore: Springer, 2022, pp. 241–247.

[3] G. K. Chadha, A. Srivastava, A. Singh, R. Gupta, and D. Singla,
"An Automated Method for Counting Red Blood Cells using Image Processing," Procedia Computer Science, vol. 167, pp. 769–778, 2020

[4] BCCD Dataset (Kaggle)
[5] Identify Blood Cell Subtypes From Images (Kaggle)
[6] Blood Cell Images (Kaggle)
[7] CNN Train(0.99)- Val(0.98) - Test (0.986) Referenced DNN for White Bloob Cells (Kaggle)