**dVerse Technologies Internship Assignment**

Ayush Patel

Contact - 8238777873

Email - [ayushnpatel22@gmail.com](mailto:ayushnpatel22@gmail.com)

**Approaches :**

1. **Edge Detection Approach:**

* **Finger Detection and Cursor Control**: Utilised a webcam to track the finger's position in real-time using OpenCV. Mapped this position to screen coordinates to move the cursor accordingly.
* **Image Preprocessing**:

- Loaded and converted the target image to grayscale for further processing.

- Applied a Gaussian blur to the grayscale image to reduce noise and improve edge detection accuracy.

* **Edge Detection**:

- Used a Gaussian kernel to blur the image, helping to highlight regions that correspond to edges.

- Implemented the Sobel operator to find gradients of the image, which indicate edge boundaries. Calculate gradient magnitudes and directions.

* **Text Detection with OCR**:

- Convert the original image to grayscale to prepare for OCR processing.

- Perform OCR on the grayscale image to detect and extract text, using Pytesseract with a specified configuration for optimal text recognition.

* **Visual Marking Based on Edge Proximity**:

- For each detected text, draw bounding boxes around it on the original image.

- Determine the direction (left/right) of the detected text relative to the image centre.

- Identify the nearest edge to the text within its bounding box, using the gradient magnitudes obtained from Sobel filters.

- Mark a circle on the image at a calculated position based on the detected edge's length, adding visual cues to indicate text labels' proximity to edges.

* **Result Visualisation**:

Showed the final OCR results with bounding boxes and circles marking the text and its nearest edge, enhancing the interface for user interaction based on cursor position.

1. Design Decisions -

* The idea was to **include a real-time video feed from the webcam** to give users immediate visual feedback on what the camera sees, including the detection of their hand and finger movement. This way I can **directly map the finger movement to the screen**.
* I **used a bounding box around the detected finger tip**, to be used within the GUI. This indicator helps users see exactly which part of their hand is being tracked and how movements translate to actions within the interface.
* The movement of the cursor is dependent upon the finger and the mouse pointer can be used to detect the labelled part of the diagram.

**Future Suggestions:**

* Gesture Recognition Features: Extend the GUI with **control features like click and drag** that can be activated through specific gestures detected by the system. For example, a **certain gesture could simulate a mouse click**.
* Feedback and Instructions Panel: A feature can be incorporated in the GUI dedicated to **providing users with real-time feedback** (e.g., "Finger detected", "Finger not detected") and instructions on how to use the system effectively. This panel can also display troubleshooting tips for common issues, like inadequate lighting or incorrect hand positioning.

1. Bugs -

* The HSV colour range used for skin detection is highly sensitive to lighting conditions. Slight Variations in light can lead to inconsistent detection of the skin, causing the algorithm to either miss the finger or detect non-skin objects as fingers.
* The fixed HSV range sometimes leads to poor performance across different users that possess different skin tones, the ones that fall outside the HSV range.
* The algorithm assumes the largest contour detected to be the hand. So, in scenarios where multiple fingers or objects are present, this assumption leads to incorrect detections.
* Due to incorrect mapping of the finger's position in the webcam frame is sometimes not so smooth, causing jittery cursor movements.
* Addressing these issues would likely require incorporating more sophisticated techniques, such as machine learning models for more robust hand and finger detection, dynamic adjustment of parameters based on environmental conditions, and algorithms to smooth out cursor movements.

1. **CNN-Based Approach (Idea - Not implemented):**

Creating a Convolutional Neural Network (CNN) to segment an image and assign labels to each segment involves several key steps. Here is a 10-point plan to approach this task:

**1. Dataset Preparation**:

- Gather a substantial dataset of labelled diagrams similar to the one provided in the assignment document.

- Annotate each segment with precise boundaries and labels. Tools like LabelMe or VGG Image Annotator (VIA) can be used for annotation.

- Resize images to a uniform scale if necessary.

- Normalise pixel values to assist with the convergence of the network during training.

- Convert textual labels into a numerical format that can be processed by the CNN (e.g., one-hot encoding).

**2. Designing the CNN**:

- Define the architecture of the CNN which may include layers such as convolutional layers, max pooling, dropout for regularisation, and fully connected layers.

- For **segmentation**, I can use architectures like U-Net or FCN (Fully Convolutional Network).

**3. Loss Function**:

- Choose an appropriate loss function for segmentation, such as cross-entropy loss for multi-class segmentation.

**4. Training/Validation Split**:

- Split the data into training, validation, and test sets. A common split ratio could be 70% training, 15% validation, and 15% test.

**5. Model Training**:

- Train the CNN using the training dataset while validating on the validation set.

- Use techniques like data augmentation to improve generalisation and avoid overfitting.

**6. Hyperparameter Tuning**:

- Adjust hyperparameters like learning rate, batch size, and number of epochs based on performance on the validation set.

**7. Evaluation**:

- After training, evaluate the model on the test set to check its performance.

- Use metrics such as Intersection over Union (IoU), pixel accuracy, and mean IoU for evaluation.

**8. Post-processing and Label Linking**:

- After segmentation, use post-processing techniques like morphological operations to refine the segmentation results.

- Link each segmented region with its corresponding label by determining the centroid of each region and associating the closest label point (black dot in our case).

To implement this plan, I would typically use a deep learning framework like TensorFlow or PyTorch. These frameworks provide comprehensive libraries and tools to build, train, and evaluate neural networks. The key is to have a labelled dataset that is large and diverse enough to train a model effectively. After training, I would apply the trained model to new images to perform segmentation and label assignment.