Chapter 2

Partial Hand Keypoints Detection

2.1 Overview

The term partial hand pose estimation corresponds to creating a rough model of the hand and its keypoints motions. Instead of trying to locate and track 21 hand joints that are most commonly used for complete hand pose estimation (as described in Chapter 1), positions only a subset of these 21 hand keypoints are determined in the partial hand pose estimation tasks. And these keypoints can be single or multiple fingertips, the center point of the palm, the wrist point, or a combination of these points. The algorithm designed for partial hand pose estimation is mainly used to complement appearance-based HCI systems where the information of the motions of a few selected hand-keypoints are used for manipulation, navigation, or pointing tasks. Many vision-based HCI systems employ gesture-based techniques for real-time interaction. In Virtual Reality (VR) or Augmented Reality (AR) applications, the hand fingertips information has been used to interact with or manipulate virtual objects [42–48]. The sixth sense [49], an example of a technology based on hand-gesture estimation, manipulates the information project on a surface using colored markers placed on fingertips. In some hand-gesture recognition systems, wrist-point detection is performed as a preprocessing step for hand-forearm segmentation [50, 51].

The partial hand pose estimation as being a sub-field of the appearance-based HCI system benefits the application dependent on low DOF hand operations. The operation can be simple tasks like pointing or navigation, swiping, scrolling, etc. Because of having dedicated and goal-oriented applications, having algorithms for partial hand keypoints is as essential as they are for full hand keypoints detection. This chapter provides some brief details of existing algorithms or methods available for partial hand keypoints detection. It then provides the details of some algorithms designed especially for fingertip(s) detection from monocular RGB images.

2.2 Related Works

The task of fingertip detection can be broadly classified into two categories - vision-based and non-vision-based. The use of specialized hardware placed on the hand to monitor the movement and position of fingertips comes under the non-vision-based category. These hardware are equipped with multiple sensors to track the hand joints and usually are in the form of gloves. Hence, they are often referred to as data gloves. An example of data gloves is shown in Figure 2.1. The use case of data gloves ranges from VR/AR, urban planning, HCI, and others [52–55]. The data gloves have been used along with visible trackers like cameras [56, 57]. The data gloves are well-suited for real-time application and they are relatively more accurate. However, the setup required for capturing hand motion through these specialized gloves may be restrictive to hand motion. They may not be suited to capture all the DOF of hand motion and may require calibration before each setup. Due to certain inconveniences associated with their use, they may not be preferred by a large number of users. Therefore, the use of data gloves will be limited to a certain type of application only.

Vision-based hand motion tracking and pose estimation are actively studied and researched in computer vision [6] given the numerous potential applications. There are several approaches available for hand pose estimation in the vision-based category. They can be classified as

Figure 2.1: A type of data glove used for capturing hand joint motion.

Visible markers-based methods: Unique markers that can be easily detected and tracked are placed over hand joints for their motion tracking. These markers can be colored tapes, special forms of paints, special suits or gloves with markers placed on them, etc. Then, a single or multiple calibrated camera captures the movements of the position of these points and passes it to a computational system for processing. Nakamura *et al.* [58] placed an LED light source on fingertips and tracked them using the color emitted by the light source. Another variation to this is the use of colored tapes or paints to make the fingertips distinct from the other objects in the captured image frame. In the Sixth Sense Device [49], P. Mistry and P. Maes demonstrated the fingertips tracking process by putting colored tapes on them. These unique markers are easy to locate. However, from a user perspective, they are inconvenient since a user has put them at the time of every use. Therefore, the application range of fingertip detection methodologies using this approach will be limited.

Methods using hand silhouettes. The hand's silhouettes are an important feature and have been utilized for hand tracking [59]. Sophisticated methods are used to capture the full kinematic information of the human hand through the silhouette information [60] and then methods are proposed for partial pose estimation like fingertips detection.

A few of the methods used in the hand silhouette information to solve the fingertips detection problem in RGB images. Basically, these methods rely on feature information extracted or recovered from the object's appearance in a $2D$ image. The overall process can be roughly divided into two parts where the first part is the hand localization step. In the second stage, further feature extraction is conducted to estimate the position of the fingertips. The hand localization is generally carried out using skin color segmentation [48, 61–63] or hand motion-based methods [64]. In the skin color segmentation method, the distinct color of human skin is segregated from the rest of the image objects. The process may be conducted in RGB color space or the image is transformed into another color space like HSV, YCbCr, etc [65–67]. The other methodologies may involve lookup table matching [43], pattern classification [68, 69], multi-scale aggregation [70]. However, the task of skin segmentation is challenging due to various factors. The process will provide incorrect results if there are objects with color similarity to that of skin. The image lighting condition, the background complexity, the user ethnicity, etc., plays an important role in the skin segmentation process [45, 46, 51]. Therefore, hand localization with skin color segmentation is often conducted in a controlled environment like keeping the background static [71, 72].

After, the hand localization, the fingertips detection involves the centroid calculation of the hand silhouette and then the farthest distance of the centroid to the tip of the finger is considered to be the identified fingertip. Methods are developed using the information derived from the contours of hand silhouettes to estimate the position of fingertips [48, 62, 73–76]. These methods are simple to implement and computationally less expensive but have dependencies on hand localization. Some of these methods work well for single fingertip detection but provide unsatisfactory results in multiple fingertips' position estimation scenario [62].

Methods based on Binocular Image or Depth Sensors: Several methods are proposed that either use setup of more than one RGB camera (such as binocular camera

system [77, 78] or depth sensors [50, 79–81]. With both of these setups, it is comparatively easy to obtain the depth information of the subject in focus. Hence, a task like hand segmentation becomes easier. However, these types of setups are not particles for common applications like home device control or casual gaming. Additionally, the cost involved in setting up such a configuration is high.

DNN based methods. A deep neural architecture offers the capability to automatically extract complex features from the input image. These features are quite superior as compared to hand-engineered features like the one obtained through the hand contour or structure. Taking advantage of this capability of DNN models most of the research for hand pose estimation shifted from using traditional approach to DNN-based approaches. Even in the traditional two-step approach for fingertips detection, CNN-based architectures are being used for hand localization and pose estimation. Huang *et al.* [82] used two cascaded CNN architectures for fingertip detection. The first level in the architecture generated the bounding-box coordinates of the hand. The second level takes this information to locate the fingertip. Many other designs followed the same approach for fingertips detection with changes in the CNN architectures [47]. The problem CNN based architecture using a regression for fingertips detection is that either they are limited for single fingertip or their performance in multiple fingertips detection lacks adequate accuracy. This led the researcher to use the Gaussian heatmap regression for keypoints detection. The Gaussian heatmap regression was first known to be used for human keypoints detection [83]. The formula used to encode the position of a keypoint to the Gaussian heatmap is given in (2.1).

$$G_{i,j,k}(x_i, y_i) = \frac{1}{2\pi\sigma^2} \mathrm{e}^{-[(x_k-i)^2+(y_k-j)^2]/2\sigma^2} \tag{2.1}$$

Here, σ is the standard deviation, and typically $\sigma = 1.5$. The point $p_k = (x_k, y_k)$ is the position of a keypoint in the Cartesian coordinate system. An example of a heatmap mask generated after encoding a keypoints is shown in Figure 2.2. The bright

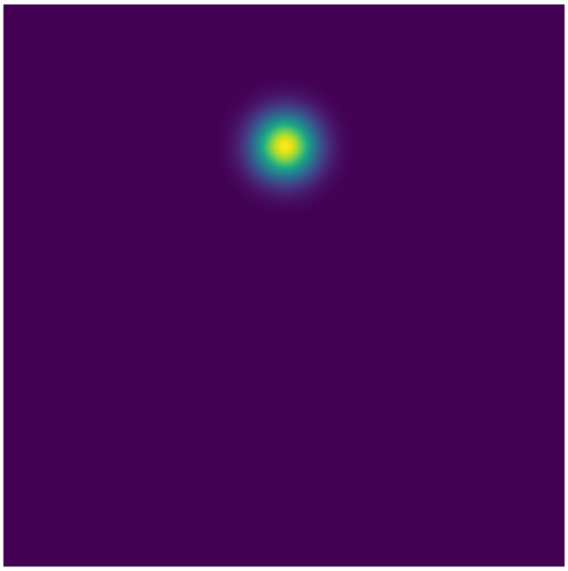


Figure 2.2: An example of a keypoint represent using the Gaussian heatmap.

yellow spot in the mask represents a keypoints. Each individual keypoint is encoded in separate channels of a 3D mask. An example of encoding the human keypoints, used to estimate the human pose, in the Gaussian heatmap is shown in Figure 2.3 [3]. A number of methods have adopted the heatmaps regression technique designed for human pose estimation in the process for estimation of hand keypoints [12,13,20,84–86]. However, in order to keep the computational cost of a CNN model, the output feature map is generally a low-resolution heatmap. It suffers from quantization error [87, 88]. Additionally, the value of standard deviation used for encoding all the keypoints is constant. This creates semantic confusion for keypoints present at different semantic scales [89]. Moreover, the CNN architecture performing heatmap regression has multiple stages of prediction stacked on top of each other. Each subsequent prediction layer in this design refines the prediction of the last layer. This design makes the model larger and increases the computation cost making it unsuitable for real-time application.

In this section, we described the process of fingertips detection from a monocular RGB image. Three different algorithms for fingertips detection were proposed. With each new approach, the performance of keypoints detection improved. Furthermore, in

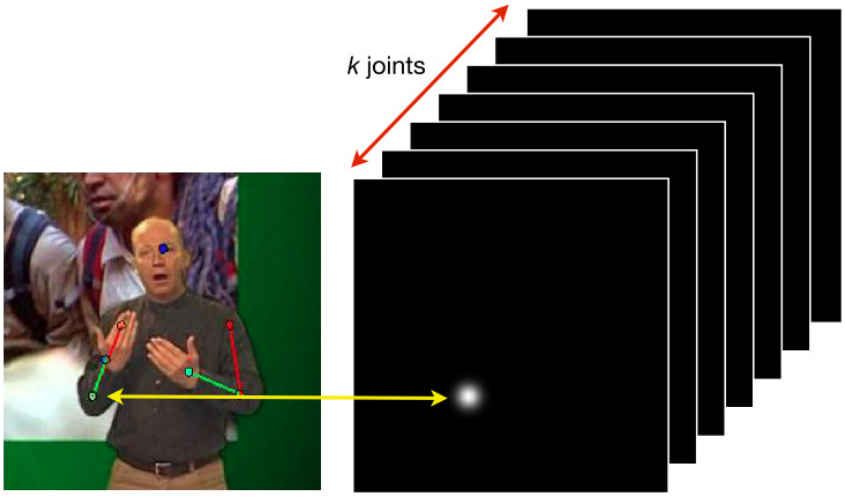


Figure 2.3: Encoding multiple keypoints in different channels with Gaussian heatmap (Source [3]).

each new approach, it was attempted to rectify the drawback associated with the last one. The details of each of the methods are provided next.

2.3 Fingertips Detection

This section and subsequent subsection cover the details of the algorithms for the estimation of fingertips' position in a monocular RGB image. The fingertip is a point on the top of each of the fingers namely the thumb, index, middle, ring, and little (pinky) fingers. These points are represented by numbers $5, 9, 13, 17, 21$ in Figure 2.4, respectively. The algorithms discussed in this section are for $2D$ fingertips detection from a monocular RGB image or a video frame.

Given, a monocular RGB image represented as $I \in \mathbb{R}^{w \times h \times 3}$, where h and w are the height and the width of the given image, respectively. The fingertips position can be represented as $p_n = (x_n, y_n) \in \mathbb{R}^2$ for $n \in [1, K]$. The value of K corresponds to the maximum number of hand keypoints to be detected. In cases of fingertips detection, confined only to a single hand, the value of $K = 5$. The values of x_n and y_n are along the x- and y- coordinates of the image. The x_n value ranges from $[0, w]$ and

the y_n ranges from $[0, h]$. These two values are measured in pixels (px). Additionally, a fingertip location can also be represented in the normalized form where $x_n \in [0, 1]$ and $y_n \in [0, 1]$. The x_n and y_n are divided by image width w and image height h to get the normalized representation of the point p_n, respectively. An algorithm designed for fingertips detection provides the probable location directly in the form of x and y coordinates or some encoded form that needs to be decoded to get the detected fingertips location as $p_n = (x_n, y_n)$. The subsequent sections cover the different variables that were explored for fingertips detection.

2.4 Partial Keypoints Detection with Multi-label Classification

This section describes the process of fingertip(s) detection in case of a multi-gesture scenario made using a single hand. The RGB images are an input source containing the hand gesture made using either extended or folded fingers or both. For example, a pointing gesture is made by extending the index finger of the hand, and by locating the tip of the index finger, it would be possible to identify the object the finger is pointing.

The process of detection of fingertips becomes difficult while a pose is being made using the hand due to the nature of hand movements and their varying articulation.

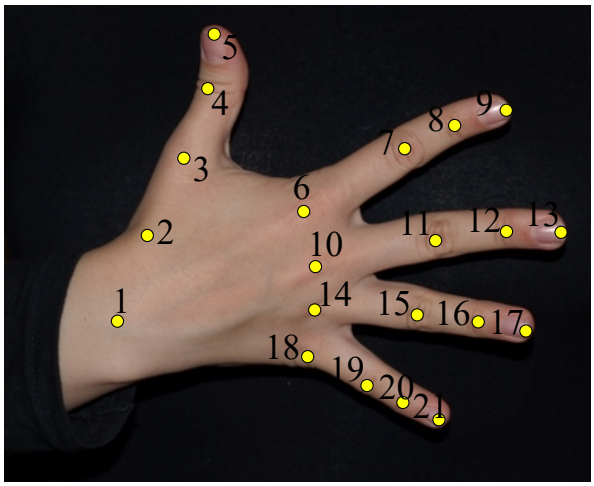


Figure 2.4: The positions of important hand keypoints

Compared to the hand, which occupies only a fraction of the image area, the fingers are relatively small objects in the image. Given these difficulties associated with the detection of fingertips even identifying finger-based gestures from RGB images, the whole process was generally divided into two steps. The first step is the detection of the presence of a hand in the image and locating it. Afterward, in the second step, given the hand is being detected, the fingertips are located from the segmented image. The same workflow has been shown in Figure 2.5

As aforementioned, the first step involved in this two-step process is to locate and segment the hand region from the given image. Before the CNN-based object detection processes became popular, hand region segmentation was performed through feature engineering. Skin segmentation process were often used for hand segmentation [16, 51, 62, 66, 73, 90–92]. However, the skin segmentation-based process was not so robust as the processes were affected by subject illumination, background color, skin color variations, etc. One way to improve the skin segmentation-based process is to augment the data with depth information [93, 94]. With the advent of CNN-based object detection algorithms [95–100], they become favorite for hand segmentation

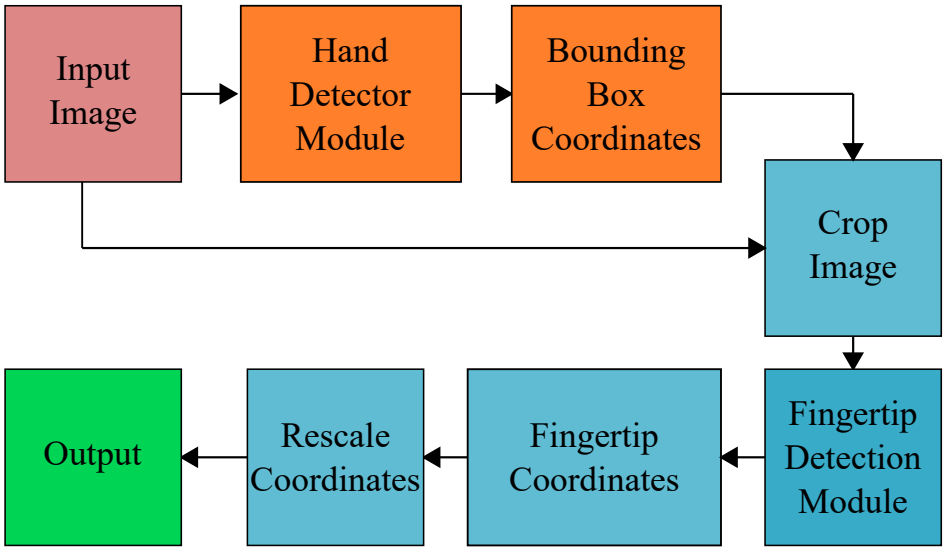


Figure 2.5: Flow diagram illustrating the process of fingertips detection.

Figure 2.6: Issue of ambiguous points being present with direct regression-based fingertips detection

task [12, 47, 82, 101].

Once, the hand region has been segmented from the input image, the next task is to process it to obtain the locations of the fingertips. With the help of a neural network-based non-linear regression technique, the fingertips' location can be determined from the segmented hand image. When either a single finger is involved in masking the gesture or the number of fingers involved in the gesture pose remains constant, it is possible to use the direct regression method for locating the fingertips. However, the process becomes difficult when multiple fingers are involved and the number of fingers varies from one gesture to another. This is due to the limitation of the neural network architecture where the number of output nodes cannot be dynamically varied due to changes in the fingers involved in making the hand gesture. For a given network architecture, the number of neurons in the output layers remains fixed. One way to solve this is to create a network with the number of output neurons equal to the maximum number of keypoints to be detected. The information should be available in advance. And, then perform regression and take the output of NN as the probable fingertip location. However, these types of networks provide ambiguous coordinates as opposed to when one or more fingers are missing or not involved in making the hand gesture. An example is shown in Figure 2.6. The points shown in blue and black obtained through

the regression technique represent the correct location of the index and the little fingers' tip, respectively. However, there are three other points present in the image shown in green, red, and cyan which are ambiguous and are not connected with any other fingers in the image. However, along with regression output if any other information like the identity of fingers involved in making the gesture, is included, will be helpful to remove ambiguous information. A multi-label classification technique was developed for the same and its details are presented next.

2.4.1 Fingertips detection with multi-label classification

This proposed technique combines non-linear regression and multi-label classification techniques to estimate the positions of fingertips used in making a hand gesture. The *non-linear regression* is a form of regression analysis in which data is fit to a model and then expressed as a mathematical function [102]. The *multi-label* classification also known as multi-output classification is a variant of classification problem where a single

Figure 2.7: Example of multi-label classification for finger identification.

model assigned multiple non-exclusive labels to each object instance in the input [103].

In the fingertips detection task, the multi-label classification technique has been utilized to identify different fingers involved in making a hand gesture. For example, as shown in Figure 2.7, using multi-label classification the fingers used in making the gesture are identified. The top-left image in Figure 2.7 represents a gesture whose meaning is numerical value two. The index and the middle fingers are forming this gesture and the same is being identified using a multi-label classification algorithm. Similar is for other hand gestures.

In a single DNN network, the fingers involved in making a hand gesture are identified. Along with it, the fingertips' position is estimated using non-linear regression. The ambiguous coordinates obtained through regression are discarded by combining the multi-label classification output. The final output of the model contains only the coordinates of the extended fingers present in the input image.

2.4.2 The CNN architecture

The DNN architecture used to perform the aforementioned steps is shown in Figure 2.8. The DNN architecture is composed of two parts- the feature extractor, and a group of sub-networks for specific tasks. The feature extractor layer mainly consists

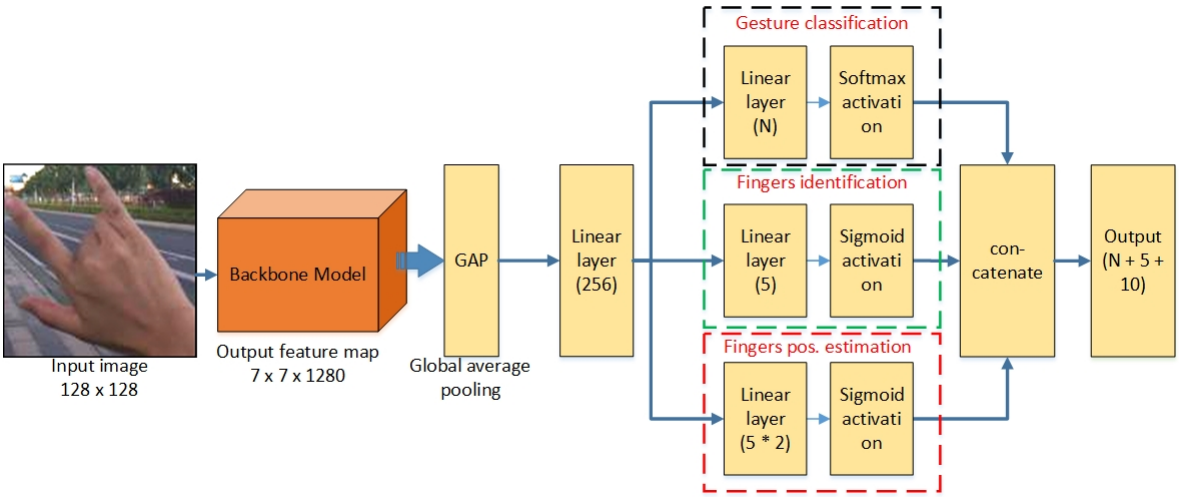


Figure 2.8: The DNN model architecture used for fingertip(s) position estimation.

of a state-of-the-art CNN-based classification model that transforms the input RGB image into a feature map which is used by subsequent layers for regression as well as prediction. In the proposed architecture, the MobileNetV2 [104] is used as the main feature extractor. Then, the feature map obtained for MobileNetV2 is passed through a global average pooling (GAP) layer. The GAP layer performs two tasks. First, it helps extract contextual information from the input feature map, and second, it converts the 2D array to a vector. A linear layer with 256 neurons is added after the GAP layer to reduce the dimension of the 1D features vector. The 256-D output feature vector of the linear layer is the input to three sub-networks each performing separate tasks namely gesture classification, finger identification (multi-label classification), and non-linear regression for fingertips' position estimation. In Figure 2.8, the gesture classification layer is composed of N neurons as there are N different gesture that needs to be classified. Each neuron out of N neuron in the gesture classification layer is responsible for the classification of a gesture. The multi-label classification layer has five neurons while the non-linear regression layer is composed of ten neurons, that is, twice the number of fingers in a hand. The output of the regression layer is x and y coordinates for each of the five fingers. The outputs of these three sub-networks are concatenated to obtain the final fingertips' position.

2.4.3 Model optimization

In order to train the model and optimize the model's parameters, its output is compared with the ground-truth fingertips' position. The error in the estimated and actual fingertips' position is performed with the help of the loss function. For the aforementioned process of fingertips' position estimation, the loss function is given by (2.2).

$$loss = w_0 * L_{classification} + w_1 * L_{identification} + w_2 * L_{regression} \tag{2.2}$$

where, $L_{classification}$ is the cross-entropy loss [105, 106], $L_{identification}$ and $L_{regression}$ are mean squared losses [107]. w_0, w_1 and w_2 are weights whose values are 0.1, 0.45, and 0.45, respectively. The value of weights was chosen empirically.

The model's parameters were optimized with Nesterov's accelerated SGD algorithm. The polynomial learning rate policy with warm restart [108] described in Section 1.2.5 is used for learning rate scheduling. The input image samples and ground-truth data are taken from the SCUT-Ego-Gesture dataset [12]. We used YOLO [95], a CNN-based object detection algorithm, for hand detection. The detected hand region is then segmented from the image and cropped to a size of 128×128 spatial dimension before passing to the CNN model for fingertips' position estimation.

2.4.4 Experimental Analysis

The methodology discussed in Section 2.4 is based on hand location followed by the fingertips detection process. As discussed above, the CNN-based object detection model called YOLOv3 has fine-tuned on SCUT-Ego-Gesture [12] dataset samples for hand localization. The qualitative performance of the model is shown in Figure 2.9. The

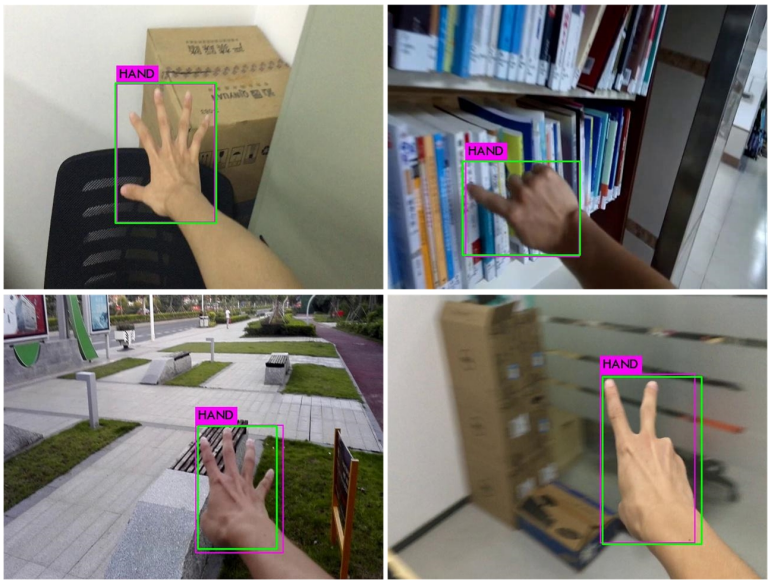


Figure 2.9: The hand localization results on the samples from the SCUT-Ego-Gesture dataset.

performance of the hand localization process is measured with respect to the ground truth in terms of the Jaccard similarity index also known as Intersection over union (IoU) defined as

$$IoU = \frac{S_p \cap S_g}{S_p \cup S_g}. \tag{2.3}$$

Here, S_p and S_g are the areas under the predicted bounding box, and the ground-truth bounding box, respectively. The predicted bounding box is shown in purple and the ground-truth bounding box is shown in green in Figure 2.9.

Apart from the YOLOv3 architecture, two other CNN-based architectures are tested for hand localization. The first architecture is referred to as DeepFinger [82] is designed for hand localization on the SCUT-Ego-Gesture dataset. Another architecture based on ResNet50 [40] architecture was modified to perform regression and generate four corner coordinates of the bounding box containing the hand region. The comparison results on hand location with three different CNN-based architectures are furnished in Table 2.1.

Among all these three models the highest mean IoU value of 0.9046 was obtained on the test samples of the SCUT-Ego-Gesture dataset with YOLOv3 architecture. The same is evident from the qualitative analysis of the results. Some results of which are shown in Figure 2.9. Therefore, for the task of hand localization, the YOLOv3 model is preferred.

The performance of the aforementioned described process for fingertips detection is conducted using the metrics - Precision, Recall, and $F1score$. Precision is defined

Table 2.1: **Performance on hand detection**

Type	Architectures	mean IoU
retrained	DeepFinger [82]	0.7693
modified	ResNet50 [40]	0.8061
retrained	YOLOv3 [96]	**0.9046**

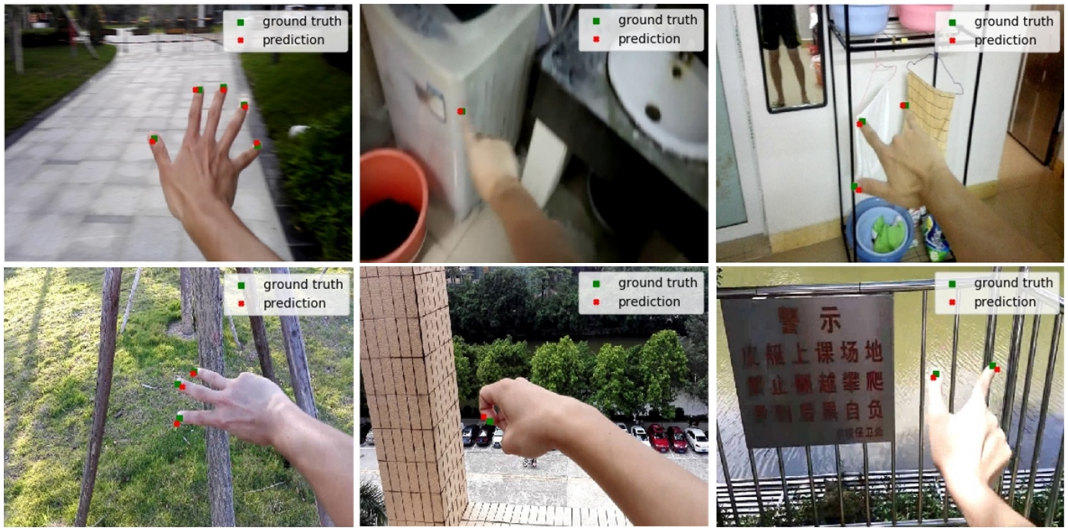


Figure 2.10: The fingertips detection results for different hand gestures on the samples from the SCUT-Ego-Gesture dataset.

as the ratio of true positive to true positive plus false positive while recall is the ratio of true positive to true positive plus false negative. A predicted fingertip location is considered to be true positive if it lies inside a circle centered at the ground-truth fingertip position and radius of 15 pixels. The false positive value is assigned to the fingertip lying outside this circle. If ground-truth is present for a fingertip but it is not detected by the algorithm then this particular fingertip is considered to be false negative. $F1score$ is the harmonic mean of precision and recall.

The comparison results on test samples of SCUT-Ego-Gesture are furnished in Table 2.2. The results were compared with the YOLSE algorithm for fingertips detection. Among these two methods, the $f1score$ of 0.9785 is obtained with the multi-label classification approach for fingertips detection. A few sample results are shown in Figure 2.10.

Table 2.2: **Comparative results for various performance metrics for fingertips detection with the multi-label classification process.**

Algorithm	Threshold (P)	Image size	Average error (pixels)	Precision	Recall	F_1score	Time (ms)
YOLSE [12]	0.2	640 × 480	8.6793	0.9617	0.9852	0.9702	9.018
	0.5		10.9832	0.9394	0.9746	0.9521	9.067
Proposed (with multi-label classification)	-	640 × 480	**6.1527**	**0.9711**	**0.9914**	**0.9785**	9.072

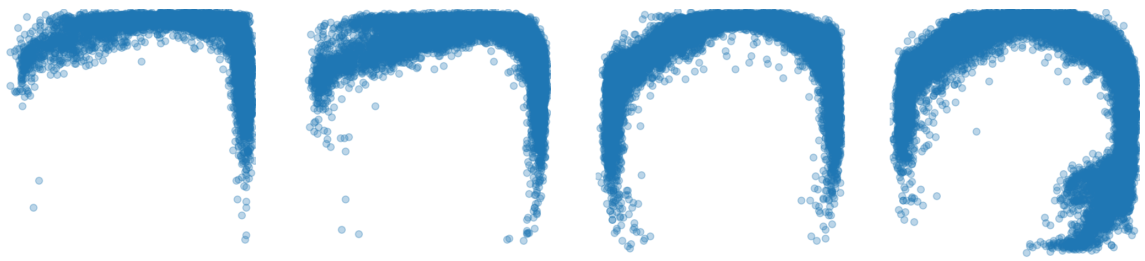
The ground-truth fingertips are shown with a green marker while the red marker shows the estimated position of the fingertips. The qualitative results suggest that the multi-label classification-based fingertips detection approach estimates the fingertips' position with adequate accuracy.

2.4.5 Discussion

The advantages and disadvantages of the aforementioned algorithm for fingertips' position estimation are listed below.

- Advantages
  - Lighter model architecture which results in faster inference time.
  - Ambiguous points present in the output due to regression can be removed.
  - Algorithm is suitable for simultaneous estimation of multiple fingertips' position.
- Disadvantages
  - Model is dependent on the hand localization algorithm and hence the process of fingertips estimation will fail if hand localization is not correctly performed.
  - The algorithm is based on a global regression technique and output accuracy lacks robustness [84].
  - The algorithm works well for hand gestures made using extended fingers. But for folded fingers in a hand gesture, the estimated fingertips position lacks accuracy.

Given the disadvantage associated with the global regression technique for keypoints detection [84] and an observation on the position of fingertips in the ground-truth data, an algorithm developed is to perform local regression for the task of fingertips position estimation. The details of the process are provided in the next section.



(a) Gesture two (b) Gesture three (c) Gesture four (d) Gesture five

Figure 2.11: The distribution of fingertips for different gestures along the hand bounding box.

2.5 Anchors-based Fingertips Detection

In the two-step approach of hand keypoints detection, the first step is the localization of the hand region. Having obtained the location of the hand in the image it is then cropped out. It was noticed that the fingertips of all the extended fingers often lie near the edge of the cropped hand image. This hypothesis has been verified by plotting the location of fingertips in the image spatial space. The ground-truth samples were obtained from the SCUT-Ego-Gesture dataset [12]. It was observed from the scatter plot of the fingertips distribution that the fingertips usually lie near the edges of the cropped image. Figure 2.11 shows the distribution of fingertips position of all five fingers during four hand gestures. All the blue points in the figure represent the position of a fingertip in proximity to the image's edge.

This observation leads to formulating a strategy where instead of searching for fingertips in the entire image. That is, instead of performing global regression, we can focus on an area close to the edge of the image. In this strategy, a set of N reference points is referred to as N-*markers* or *anchors*. The location of a fingertip is calculated with reference to these N-markers. These N-markers were placed circularly along the edge of the image. These circular curve is obtained after performing regression to find the output of the curve that best fits the distribution of fingertips location. The ground

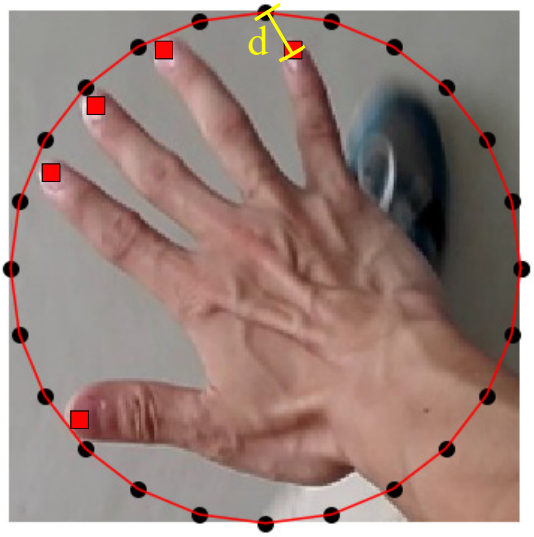


Figure 2.12: Local regression using prior known points called anchors for estimation of fingertips position.

truth from the SCUT-Ego-Gesture dataset is used for this purpose. The resultant curve is an ellipse which when scaled for a square image takes the form of a circle. Next, N-points are selected at equal intervals on this circular curve which acts as the reference points. The black points in Figure 2.12 represent these N-reference points or anchors. The position of a fingertip is estimated by identifying the nearest anchor to the fingertip. Next, its distance from the closest anchor point is estimated. For example, as shown in Figure 2.12, the Pinky finger is at a distance of d from the nearest anchor.

The framework for the anchors-based fingertips' estimation process is shown in Figure 2.13. It is similar to the framework described in Section 2.4. The complete process consists of two-step; First - a separate process is used for hand localization and in the second step, the anchor-based process is used to estimate the fingertips' position

.

Similar to Section 2.4, for the detection of hand in the RGB image, a DNN-based objector called YOLOv3 redmon2018yolov3 architecture is referred to because of its accuracy and inference speed. Next, the detected hand region is cropped from the

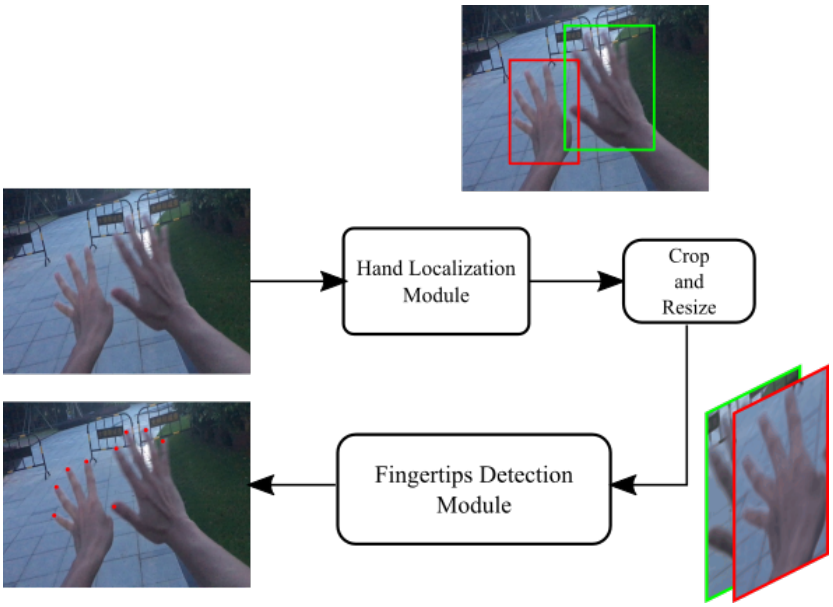


Figure 2.13: The framework used for fingertips detection in the anchor-based detection method.

input image using the bounding box coordinates. The cropped image is then resized to $S \times S$ spatial dimension. The value of S is 224. Afterward, the fingertips' positions are estimated using the anchor-based method from the resized image.

For the fingertips detection, the anchor points $Z = \{Z_k\}_{k \in N}$ are N reference points placed at equal intervals along the circumference of the circle inscribing the hand bounding box. Here, $z_k = (x_k, y_k) \in \mathbb{R}^2$ is the 2-D pixel coordinate of the $k-th$ marker. The complete process is two parallel processes with one for identifying the nearest marker to a fingertip and another for calculating the offset of the fingertip from the nearest identified marker. The process of identifying the nearest fingertip involves finding the minimum value of $d(z_k, p_n)$. Here, $d(z_k, p_n)$ is the Euclidean distance between the n-th fingertip and the k-th marker; that is

$$d(z_k, p_n) = \|Z_k - p_n\|_2 \tag{2.4}$$

After the nearest marker for the n-th fingertip is identified, its final position is calculated by finding the value of d shown in Figure 2.12. To identify the nearest

marker point and estimate the value of d a CNN model is used. The task of finding the nearest marker is treated as an object classification problem and to calculate the value of d regression is used.

2.5.1 The CNN architecture

The CNN architecture used for the aforementioned task is shown in Figure 2.14. The architecture has a backbone model as the main features extractor and two sub-networks. Additionally, atrous spatial pyramid pooling (ASPP) [35] layer is used to enlarge the receptive field of the proposed model. The backbone model has CSPDarknet53 [97,109]. It has about 27.6M trainable parameters and can perform prediction at a high frame rate on A GPU-enabled system. The input to the backbone model is an image of shape $S \times S$ and the output is a features map with the spatial dimension of $\frac{S}{32} \times \frac{S}{32}$. The ASPP layer is used to enlarge the receptive field of the model. It is connected to the last layer of the backbone model. The convolutional layer in the ASPP has sampling rate $rd = 1, 3, , 6, \text{and} 12$. The output feature map obtained for the ASPP layer is passed to the sequentially connected GAP layer and L_2 normalization layer. The resulting feature map is used as input to two sub-networks viz. *sub-net A* and *sub-net B*, respectively. Both of these sub-networks consist of a dense layer, an activation layer, and a reshape layer. The sub-network A finds the offset of a marker from the nearest fingertip point. The sub-network B identifies the nearest marker.

2.5.2 Inference

At inference time, a fingertip position can be obtained from the output of the CNN model using (2.5) and (2.6).

$$x_n = p + x_k, \tag{2.5}$$

$$y_n = q + y_k \tag{2.6}$$

Here, p and q are x and y offsets of n-th fingertips from k-th marker. The n-the fingertip, the value of each index $m \in [1, M]$ is the confidence score for each marker in its proximity, The confidence interval with $C \geq \delta_m$ represents the marker nearest to the n-th fingertip. The value of the confidence score lies in the range of 0 and 1. The value of m represents the linear index of a marker. The coordinate of the k-th nearest marker present at index m can be obtained as

$$x_{m_i} = U\frac{S}{M} \tag{2.7}$$

$$y_{m_i} = V\frac{S}{M}. \tag{2.8}$$

where,

$$U = m \bmod M, \tag{2.9}$$

$$V = (m - U)/M, \tag{2.10}$$

The value of $\delta_m = 0.65$ to identify the highest confident marker point.

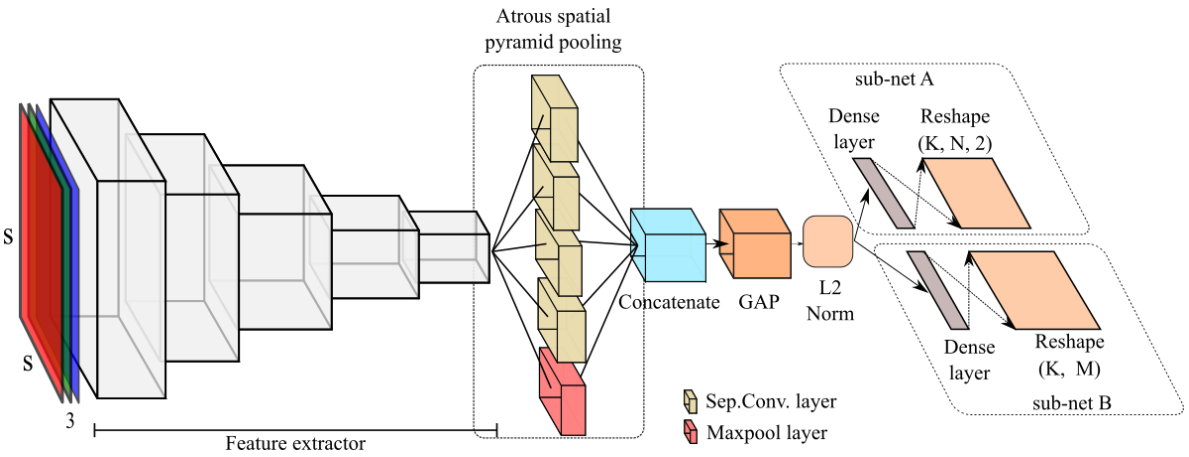


Figure 2.14: The model architecture used for fingertips detection with the anchor-based method.

2.5.3 Model optimization

The CNN architecture shown in Figure 2.14 is performing two different tasks. Hence, the model is optimized using two different loss functions. The final loss of the network is the weighted sum of these two loss functions. The loss function is given by

$$loss = l_{netA} + \lambda . l_{netB} \tag{2.11}$$

Here, l_{netA} and l_{netB} are the loss of sub-net A and sub-net B, respectively. The loss of sub-net A is calculated using Huber loss [110]. The SGD algorithm is used for model optimization. The model is trained for 100 epochs with a batch size of 16. In order to train end-to-end, the ground-truth fingertip pixel coordinates are encoded to match the output format of the CNN model shown in Figure 2.14.

The aforementioned processes for fingertips' position estimation follow the two-step approach. The first step is the localization of the hand from the given RGB image. The process of hand detection is in itself, a very challenging task. Therefore, the accuracy of fingertips position estimation will be affected by the performance of the hand detector. Furthermore, a separate detector is used which has its own computational cost, and hence total processing time for fingertips detection is relatively higher. Another downside is that since the partial hand keypoints detection of a single hand can be taken at a time, multiple hand keypoints detection will be affected. Taking note of all these problems with the two-step process for fingertips detection a different approach is required. A single-stage process for hand keypoints detection is developed which is discussed in the next section.

2.5.4 Experimental Analysis

Performance evaluation metric

In order to measure the performance of the fingertips detection algorithm a metric

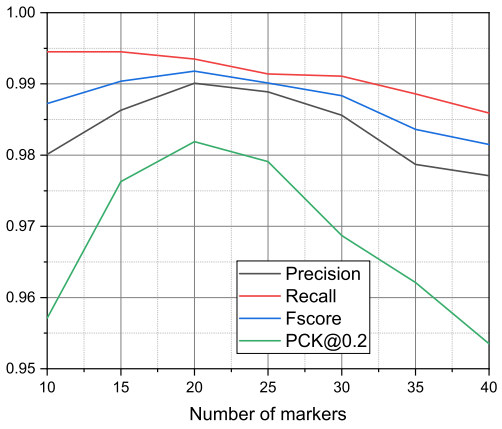


Figure 2.15: The effect of accuracy on fingertips estimation when the number of markers is varied.

called *Probability of Correct Keypoint*(PCK) [111]. The PCK for n-th fingertip position estimation on $\mathbb{N}$ samples is computed as

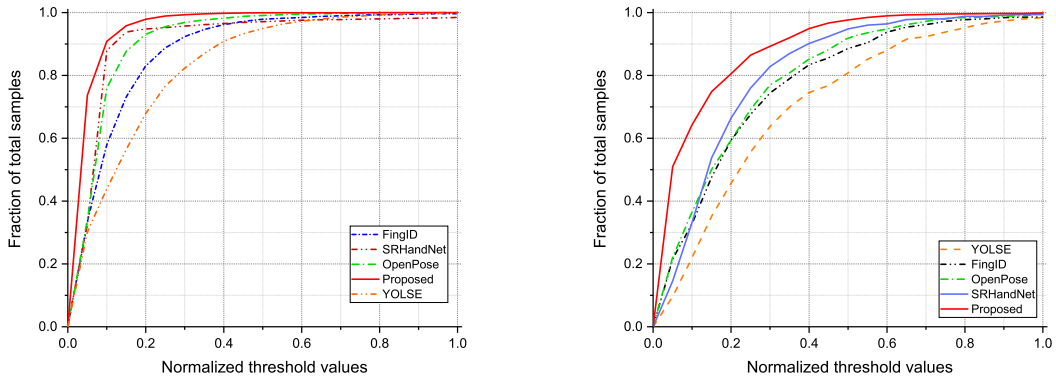
$$PCK_{\sigma}^{k} = \frac{1}{\mathbb{N}} \sum_{\mathbb{N}} \chi \left(\frac{\|f_k - g_k\|_2}{\max(w_b, h_b)} \leq \sigma \right). \tag{2.12}$$

The (2.12) gives the probability that predicted keypoints, f_k, lies with a threshold distance (σ), from the ground-truth keypoint position (g_k). The Euclidean distance between the ground truth and the estimated keypoint is normalized by the maximum length edge of the bounding box that encloses the hand region in the input image. In (2.12), h_b and w_b are the height and width of the bounding box enclosing the hand, respectively.

Effect of the number of markers: The anchor-based fingertips detection method is based on performing local regression with the help of markers placed at regular intervals on the circumference of the closed curve. An experiment was conducted on the number of markers, M, at which the best results are obtained. The value of M is varied from 10 to 40 with a step size of 5. The effect on performance is measured in terms of $f1score$, PCK value at $\sigma = 0.2$ (written as PCK@0.2). The results obtained are shown

in Figure 2.15. The best value of $f1score$ and PCK@0.2 is observed at $M = 20$. For larger values of $M > 20$, the performance does not improve instead the fingertips detection accuracy decreases.

Performance comparison. The performance comparison of anchor-based fingertips detection methods in conducted using the Precision, Recall, F1score, and PCK metrics. The presented method performance is compared with YOLSE [12], SRHandNet [14], FingID [101], and OpenPose [84]. The YOLSE method is based on heatmaps regression while the FingID method uses linear regression for fingertips detection. All of these methods follow a two-step approach. The performance comparison in terms of the PCK curve on SCUT-Ego-Gesture and HGR datasets are shown in Figure 2.16. The quantitative comparison results are furnished in Table 2.3. The higher values of F1scores and PCk for the proposed anchor-based fingertips detection methods suggested that the estimated fingertips' positions are closer to the marked ground-truth fingertips' position. Additionally, from the PCK curves shown in Figure 2.16, it can be inferred that approximately 97% of detected fingertips are within a threshold distance $\sigma = 0.2$ on the SCUT-Ego-Gesture dataset. Similarly, for the HGR dataset, this value is 83%.



(a) SCUT-Ego-Gesture [12] (b) HGR [16]

Figure 2.16: Performance comparison of anchor-based fingertips detection method in terms of PCK.

Table 2.3: **Performance comparison of the anchor-based fingertips detection method**

Model	Metric	SCUT-Ego-Gesture [12]	HGR [16]
YOLSE [12]	Precision	0.8412	0.8567
	Recall	0.8632	0.8666
	FScore	0.8521	0.8616
	PCK@ (0.2)	0.6987	0.4562
SRHandNet [14]	Precision	0.9654	0.9887
	Recall	0.9869	0.9816
	FScore	0.9761	0.9815
	PCK @(0.2)	0.9382	0.7235
FingID [101]	Precision	0.9132	0.9122
	Recall	0.9342	0.8910
	FScore	0.9235	0.9015
	PCK@ (0.2)	0.8302	0.6132
OpenPose [84]	Precision	0.9543	0.9096
	Recall	0.9633	0.9383
	FScore	0.9588	0.9237
	PCK@ (0.2)	0.9309	0.6653
Anchor-based	Precision	**0.9890**	**0.9927**
	Recall	**0.9912**	**0.9934**
	FScore	**0.9900**	**0.9930**
	PCK@ (0.2)	**0.9791**	**0.8321**

Some sample fingertips' detection is also shown in Figure 2.17 and Figure2.18 for the SCUT-Ego-Gesture and the HGR datasets, respectively. The ground-truth fingertips are shown with green markers while the red marker shows the estimated fingertips' position.

Additionally, the performance of the anchor-based algorithm was tested for rotation invariance. It was observed that the performance of the algorithm remains unaffected due to the rotation of the hand at various angles in the image. An example is presented in Figure 2.19.

2.5.5 Discussion

The experimental results show the feasibility of the anchor-based algorithm to estimate the fingertips' position from an RGB image. The proposed method is suitable for monocular vision and works for a high range of potential hand postures. It is also

(a) YOLSE [12]

(b) FingID [101]

(c) Anchor-Based

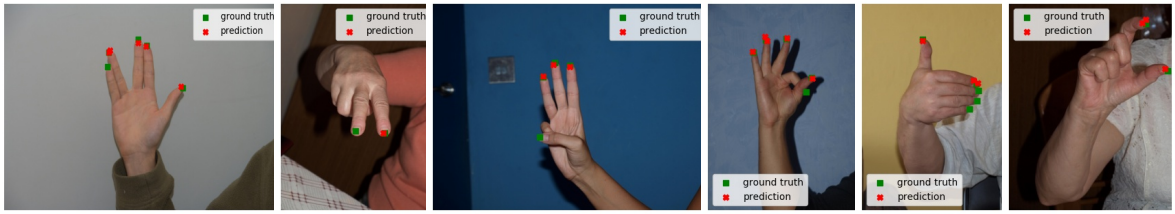
Figure 2.17: Sample fingertips detection results on the SCUT-Ego-Gesture dataset with the anchor-based method.

effective against variation in lighting conditions, skin color, the subject's gender or age, background, etc. The proposed anchor-based algorithm method performed satisfactorily in the fingertips position estimation of multiple hands present in the image given all hands are segmented by a separate detector. The fingertip position can be estimated for different hand sizes present in the image. However, the overall process follows a two-step approach. Hence, the performance of this fingertips detection algorithm is dependent on the performance of the hand localization algorithm. An algorithm that can perform hand keypoints detection without being dependent on hand localization is needed to port it to a large number of use cases.

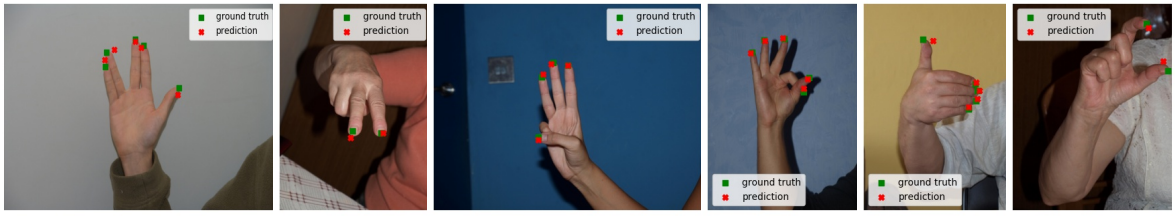
2.6 Nearest Neighbor Fingertips Detection

The nearest-neighbor fingertips detection method is designed to yield the fingertips' position of all the visible fingers from a monocular RGB image in a single stage. The same process works effectively for both single as well double-hand scenarios. The framework of the current process is shown in Figure 2.20. The process involves a CNN

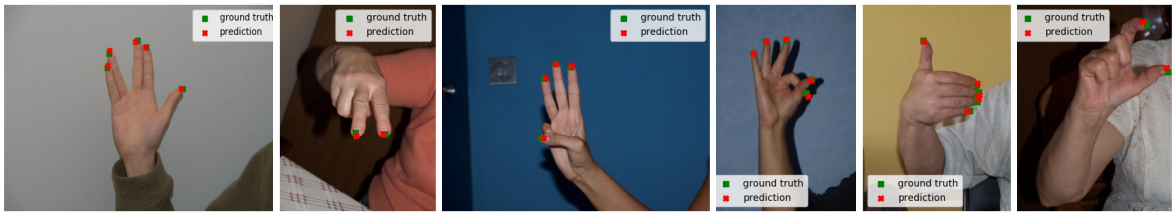
model that takes as input the full-size monocular RGB image. The output of the CNN model contains the fingertips' position in an encoded form. With the of a decoder and reference points known as *pose particle*, the fingertips' position is provided in the form of $p_n = (x_n, y_n)$. Here, pose particles are fixed reference points placed uniformly along



(a) YOLSE [12]



(b) FingID [101]



(c) Anchor-Based

Figure 2.18: Sample fingertips detection results on HGR dataset with the anchor-based method.

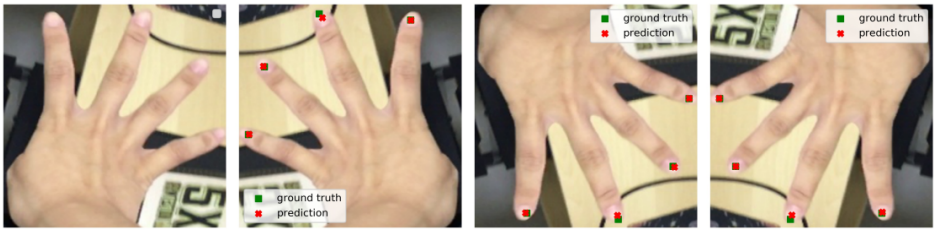


Figure 2.19: Demonstrating the effectiveness of rotation invariance of anchor-based fingertips' detection algorithm.

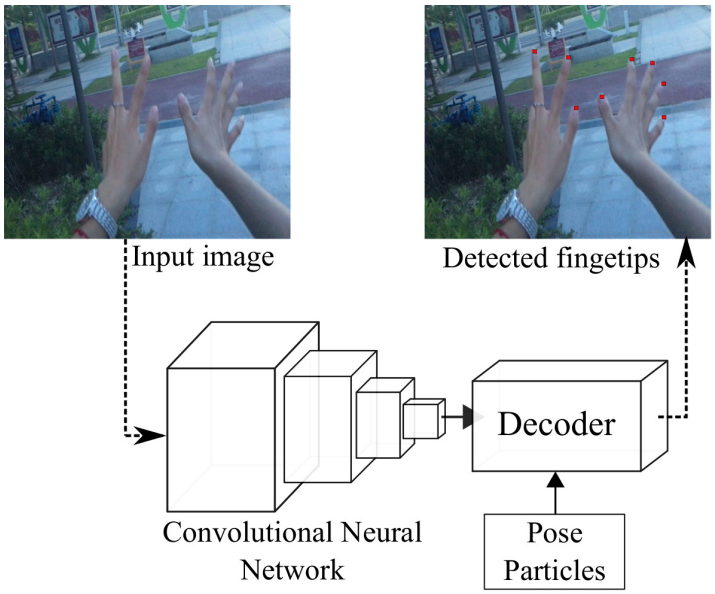


Figure 2.20: The framework for nearest-neighbor fingertips detection process.

the spatial dimension of the image. They probabilistically indicate the presence of a fingertip in their neighborhood. A set of N such high probability reference points is selected in the proximity of a fingertip to return its position. Along with pose particles, the position vector originating from the nearest reference point points toward the closest fingertip. The centroid of all the final points suggested by the position vector is taken as the estimated position of a fingertip.

2.6.1 Working principle

As earlier described , $p_n = x_n, y_n \in \mathbb{R}^2$ for $n \in [1, K]$ represents the location of a fingertip in a RGB image $I \in \mathbb{R}^{w \times h \times 3}$. The value of K is 10, which signifies the maximum number of fingers in both hands. The pose particles are represented as $Z = z_{mm \in M}$, where $z_m = (x_m, y_m) \in \mathbb{R}^2$ and $M = U \times V$. Here, U and V represented the number of rows and the numbers of columns of the grid formed by the pose particles. The grids formed by the pose particle are shown in Figure 2.21. Since the RGB image is resized to a square form before being used for fingertip detection with the help of a CNN model, therefore the number of rows is equal to the number of columns, that is, $U = V$.

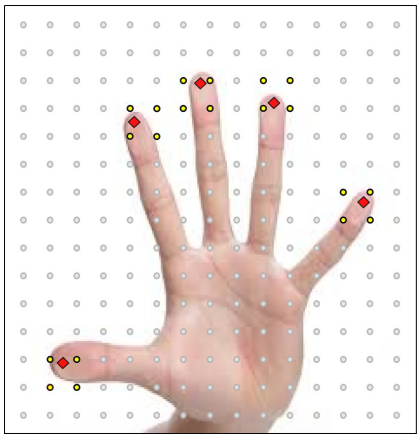


Figure 2.21: The grids formed by the pose particles and the 4-nearest pose particles to all visible fingertips.

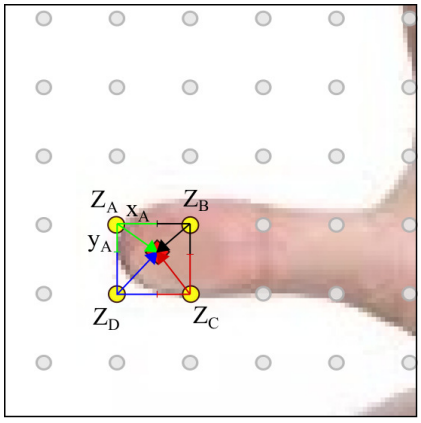


Figure 2.22: The 4-nearest pose particle to the thumb's tip and the position vectors from the nearest pose particles

Given the aforementioned pose particle arrangement, a fingertip will always be in the proximity of a few of these particles. Moreover, since the positions of pose particles are known, therefore a fingertip position can be predicted if we can predict the position of the pose particle close to it. Here, we used the N number of the nearest pose particles, referred to as $N-nearestneighbors$, to estimate a fingertip position in the given image. A pose particle is considered nearest to fingertips if the Euclidean distance between them is the minimum. That is, the m-th particle is the nearest to n-th fingertips for $\min d(z_m, p_n)$.

Instead of using just a single pose particle to estimate the position of fingertips

in the image a set of N-nearest pose particles are used. For example, as shown in Figure 2.22, the four points namely $Z_A, Z_B, Z_C,$ and Z_D are 4-nearest neighbors to the Thumb's tip position. After being able to locate the nearest pose particle to a fingertip, the next task is two get its final position. The nearest pose particles are at some offset distance from its closest fingertip as shown in Figure 2.22, and do not give its exact location. Therefore, to estimate the final position of the fingertip, we calculate position vectors from m-th pose particle to n-th fingertip. $\mathbf{R} = \{\vec{r}_i\}_{i \in N}$ represent position vectors $\vec{r}_i = \langle x_i, y_i \rangle$ is a position vector from m-th pose particle to n-th fingertip. Here, x_i and y_i are the x- and y- components of the position vector $\vec{r}_i$ and they are calculated as given in (2.13) and (2.14), respectively.

$$x_i = x_n - x_m, \tag{2.13}$$

$$y_i = y_n - y_m. \tag{2.14}$$

A position vector links a fingertip to its nearest pose particle. For example, there are four different position vectors linking the points $Z_A, Z_B, Z_C,$ and Z_D to the tip of the Thumb. Therefore, there are four different position vectors starting from points $Z_A, Z_B, Z_C,$ and Z_D namely $\vec{r}_A = \langle x_A, y_A \rangle$, $\vec{r}_B = \langle x_B, y_B \rangle$, $\vec{r}_C = \langle x_C, y_C \rangle$, and $\vec{r}_D = \langle x_D, y_D \rangle$, respectively.

In the ideal computational cases, all the position vectors should point to a fixed position, that is, a fingertip location. However, the value of position vector components x_i and y_i are computed using the regression technique. Given the errors associated with the regression technique, the estimated position with one position vector may not be the same as estimated by another position vector. That is,

$$x_j + x_l \neq x_i + x_m, \tag{2.15}$$

$$y_j + x_l \neq y_i + x_m. \tag{2.16}$$

where, $\vec{r_j} = \langle x_j, y_j \rangle$ and $\vec{r_i} = \langle x_i, y_i \rangle$ are position vector from the pose particle points Z_l and Z_m, respectively. Therefore, we calculate the centroid of all endpoints given by position vectors. This centroid is the estimated position of a fingertip. Mathematically, the estimated position of the n-th fingertip, $\bar{p_n} = (\bar{x_n}, \bar{y_n})$, is calculated as

$$\bar{x_n} = \frac{1}{N} \sum_{i,j}^{N,M} (x_i + x_j), \tag{2.17}$$

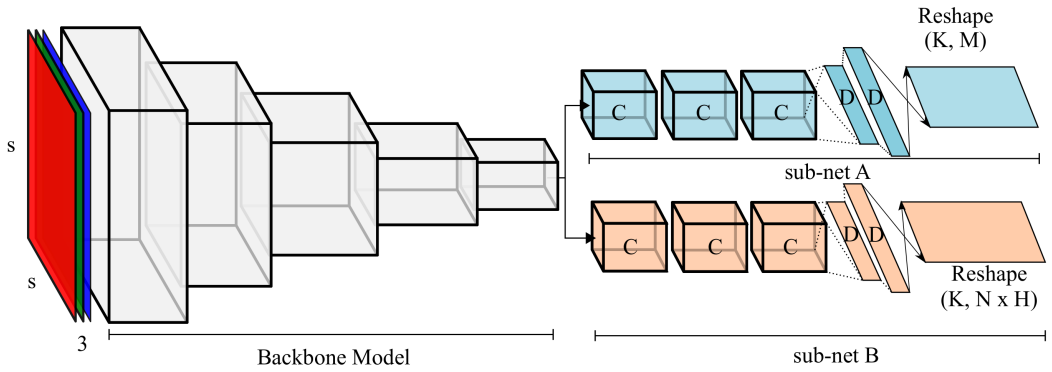
$$\bar{y_n} = \frac{1}{N} \sum_{i,j}^{N,M} (y_i + y_j). \tag{2.18}$$

. Here, x_i and y_i are the components of position vector $\vec{r_i}$. $Z_j = (x_j, y_j)$ is the pose particle coordinates from which the position vector $\vec{r_i}$ originates.

The N-nearest pose particles and corresponding N position vectors are obtained using a 2D CNN architecture. The CNN architecture assigns probability values of all pose particles ($U \times V$). The pose particles having fingertips in their proximity will have probability values higher as compared to others. Using a threshold δ_N, this high-probability pose particles are identified. The position vector components from these pose particles are computed using regression. The CNN architecture details are provided next.

2.6.2 The CNN architecture

The architecture of the CNN model used in the process called nearest-neighbor-based fingertips detection is shown in Figure 2.23. An RGB image of size $S \times S \times 3$ is passed as input for fingertips detection. The architecture of the model partially uses another



C: Convolutional layer; **D**: Dense Layer; **K**: no. of fingertips; **M**: no. of pose particles; **H**: no. of position vector's component

Figure 2.23: The CNN model architecture used in nearest-neighbor based fingertips detection process.

CNN architecture as its backbone. This backbone model is used as its main feature extractor. The output stride of the backbone is 32, that is, for input of spatial dimension $S \times S$, it produces a feature map, $\mathbf{F}$, of spatial dimension $\frac{S}{32} \times \frac{S}{32}$. This reduction in the spatial dimension of the input image helps in keeping the computational cost low. In the CNN model architecture shown in Figure 2.23, the DenseNet architecture [112] is used. The DenseNet architecture allows maximum flow of information and it requires less number of parameters than the other CNN architectures [112].

Next, in the architecture, there are two sub-network that perform individual tasks of identifying $N - nearestposeparticle$ of each visible fingertip in the input image and calculating the values for position vector components for each pose particle. The input to these sub-networks is the feature map $\mathbf{F}$ obtained from the backbone model. Both of these sub-network are almost identical in their design except for the last few output layers. In each of them, there are three convolutional blocks represented as **C** in Figure 2.23. Each convolutional block has 3×3 kernel and 256 learnable filters. The *sub-network A* in the architecture shown in Figure 2.23 computes the probability of each pose particle being in the neighborhood of fingertips. It has a dense layer with $K \times M$ neurons. The value of K represents the number of fingertips which is equal to 10 and M is the number of pose particles. The value of M is equal to $U \times V$. The

sub-network B is designed to compute the values of the position vector's components. It calculates two values of each pose particle. So, if **H** is the number of position vector's components for each fingertip, there are $N \times H$ values computed. In total, there are $K \times N \times H$ neurons in the output layer of the sub-network B that estimates the values of each position vector's components. While the backbone model weights are initialized with weights trained on ImageNet dataset [37], the weights of other layers in the model are initialized with He Normal initialization technique [113].

2.6.3 Decoding

There are two arrays obtained from the output of the CNN model shown in Figure 2.23. The first one is a $K \times M$ matrix $M_{k,m}$ and another is a $K \times N \times H$ matrix $(G_{k,n,h})$. These output forms do not directly represent the position of fingertips. The position $p_n = x_n, y_n$ is obtained by decoding the values contained in these matrices. The inference of sub-network A, provides the coordinates of N-nearest neighbors pose particles. The value at column M of the matrix $M_{k,m}$ contains the probability N-pose particle in the proximity of k-th fingertips (represented by the row index of the matrix). The index in the k-th row having probability, $P_M \geq \delta_N$, represents a pose particle in the neighborhood of fingertips p_n. The coordinates of these pose particles are calculated from the k-th row as

$$x_m = (P + 0.5)\frac{S}{U}, \tag{2.19}$$

$$y_m = (Q + 0.5)\frac{S}{U}. \tag{2.20}$$

where,

$$P = m \mod U, \tag{2.21}$$

$$Q = (m - P)/U. \tag{2.22}$$

. Here, m is the index value at which $P_M \geq \delta_N$ in the K-th row of the matrix $M_{k,m}$. Having obtained the value of $z_m = (x_m, y_m)$ from (2.19) and (2.20), the point $p_n = (x_n, y_n)$ is calculated using (2.13) and (2.14).

2.6.4 Model optimization

The total loss calculated from the model's output is composed of two loss components. One is the loss from sub-network A and another from sub-network B. The total loss can be given as

$$loss = loss_A + loss_B \tag{2.23}$$

. Here,

$$loss_A = \|\bar{G_A} - G_A\|_2, \tag{2.24}$$

and

$$loss_B = \|\bar{G_B} - G_B\|_2, \tag{2.25}$$

are losses of sub-network A and sub-network B, respectively. G_A and G_B are ground-truth values for sub-network A and sub-network B, respectively. These two ground-truth values are in the encoded form obtained from 2-D pixel ground-truth coordinates. For encoding the pre-defined pose particles coordinates are used. The training dataset from which samples are taken is SCUT-Ego-Gesture [12] and Rendered Hand Pose (RHD) [20]. These two datasets contain samples for single as well the double hands.

2.6.5 Experimental Analysis

Performance Evaluation Metric: The PCK metric defined in Section 2.5.4 is used for performance evaluation of the nearest-neighbor-based fingertips' detection algorithm.

Performance with respect to different backbone model: The backbone model is the main feature extractor in the CNN model architecture shown in Figure 2.23. In this design three different variations of DenseNet [112] architectures are used. These archi-

tectures can be represented as DenseNetX, where X represents the number of layers in the architecture. The three variations of the DenseNet model that are used in the experiments are DenseNet12, DenseNet169, and DenseNet201. The larger the number of layers in the model, the slower it gets as compared to a lesser number of models. The DenseNet201 model has 1.8 times more parameters in comparison to the DenseNet121 architecture. The performance of these three architectures was tested on three datasets namely OneHand, SCUT-Ego-Gesture, and RHD datasets. The results are furnished in Table 2.4. The best performance is achieved with DenseNet201 architecture when used in the backbone of the model shown in Figure 2.23. However, there is only incremental gain in the performance as compared to DenseNet169 and DenseNet121. Therefore, the choice of backbone will be based on the application requirement.

Table 2.4: Effect of different parameters viz. backbone model, N-nearest neighbor, and grid size on the performance of the proposed model tested on three different datasets.

Datasets			Backbone			Nearest Neighbor (N)					Grid Size (U or V)				
			DenseNet121	DenseNet169	DenseNet201	1	2	4	8	16	7	9	11	13	15
		Parameters	16.15 M	23.23 M	29.50 M										
		Time (ms)	29.85	34.79	39.67										
OneHand10K [13]		PCK ($\sigma = 0.2$)	0.6221	0.6598	0.6698	0.4826	0.6051	0.6698	0.6097	0.5956	0.6186	0.6238	0.6291	0.6303	0.6698
		mean	0.7498	0.7591	0.7683	0.6602	0.7332	0.7683	0.7361	0.7286	0.7381	0.7368	0.7462	0.7491	0.7683
SCUT-Ego-Gesture [12]	Single hand	PCK ($\sigma = 0.2$)	0.9903	0.9929	0.9919	0.9876	0.9910	0.9913	0.9855	0.9900	0.9889	0.9915	0.9919	0.9907	0.9913
		mean	0.9541	0.9574	0.9619	0.9507	0.9584	0.9619	0.9513	0.9522	0.9556	0.9586	0.9591	0.9575	0.9619
	Double hands	PCK ($\sigma = 0.2$)	0.9910	0.9932	0.9940	0.9767	0.9928	0.9939	0.9921	0.9929	0.9905	0.9919	0.9928	0.9931	0.9939
		mean	0.9495	0.9513	0.9617	0.9448	0.9575	0.9627	0.9519	0.9501	0.9535	0.9553	0.9581	0.9577	0.9617
RHD [20]		PCK ($\sigma = 0.2$)	0.6545	0.6652	0.6851	0.6280	0.6545	0.6851	0.6780	0.6633	0.6009	0.6397	0.6633	0.6780	0.6851
		mean	0.7883	0.7909	0.7976	0.7759	0.7899	0.7976	0.7880	0.7840	0.7432	0.7656	0.7795	0.7863	0.7976

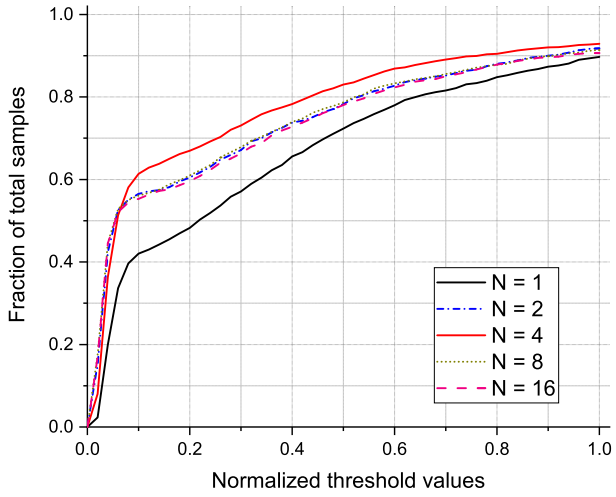


Figure 2.24: Performance comparison at different numbers of nearest-neighbor pose particles

Effect of the number of nearest neighbors pose particles: Experiments were conducted to determine the number of neighborhood pose particles required to achieve the best performance on fingertips' detection. N denotes the number of nearest pose particles for used fingertips detection. The value of N is chosen for a set of $\{1, 2, 4, 8, 16\}$. The performance at each of these values of N is furnished in Table 2.4. The comparison in terms of PCK value for different thresholds is also shown in Figure 2.24. The performance when $N = 1$ is relatively inferior as compared to when $N \geq 2$. But, as we kept increasing the value of N further there was no such gain in the performance. The best performance is achieved when $N = 4$. This dip in performance for a higher value of N is due to the inaccurate estimation of the position vector's components. This resulted in the final centroid deviating more from the actual fingertip position.

Effect of the number of pose particles: A grid structure formed by the pose particle is used for fingertips' detection. With U rows and V columns in the grids, the number of pose particles is given by $M = U \times V$. As the value of M will be higher, the two pose particles will be closer to each other. And, the closer the pose particles to each other

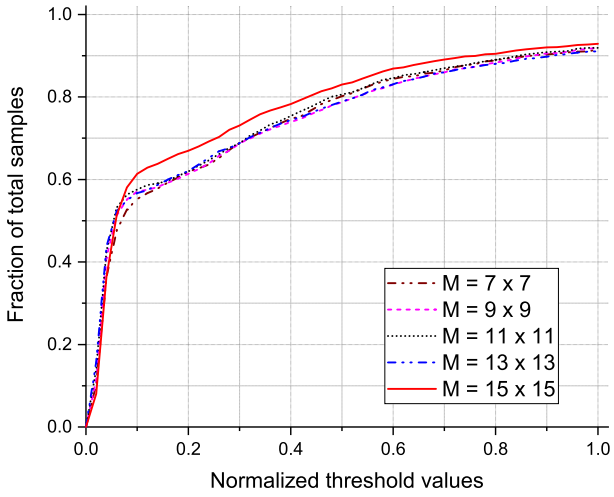


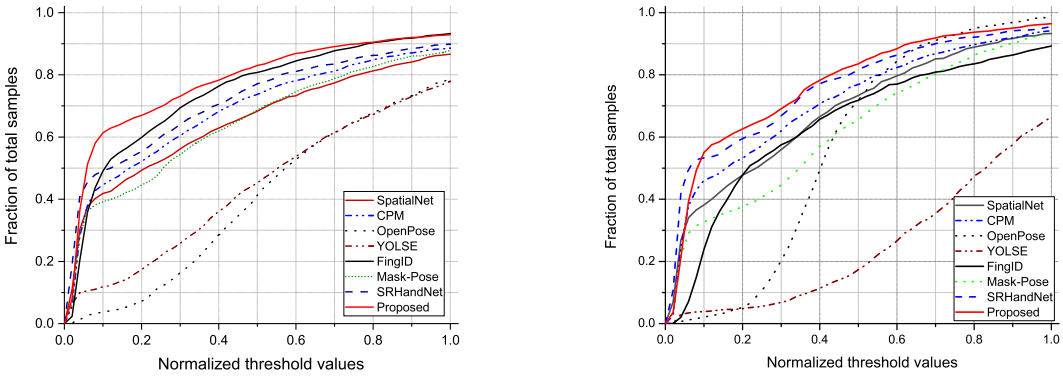
Figure 2.25: Performance comparison for different grid sizes formed by the pose particles

the better is the accuracy of the proposed method in fingertips detection. Five different values of U or V viz. $7, 9, 11, 13, 15$ are tested to identify the best possible number of pose particles to use. The experimental results are furnished in Table 2.4. The comparison in terms of PCK value for different thresholds is also shown in Figure 2.25. The best performance is achieved when $M = 15 \times 15$, that is, $M = 225$. The same pattern was observed on three different datasets used in the experiment.

The grid's size has a negligible effect on the model's prediction time as it does not increase the number of trainable parameters in the last dense layer of sub-net A of the model shown in Figure 2.23. The model's prediction time increases by only about 3% for $M = 225$ compared to $M = 49$.

Comparative analysis

The nearest-neighbor algorithm performance is tested on six different datasets namely OneHand10K, HGR, STB, FPHA, RHD, and SCUT-Ego-Gesture. Multiple CNN-based models were selected to compare the results on these datasets. The comparison results are furnished in Table 2.5. The comparison in terms of PCK value at different thresh-



(a) OneHand10K [13] (b) HGR [16]

Figure 2.26: Comparative analysis in terms of PCK on two different datasets.

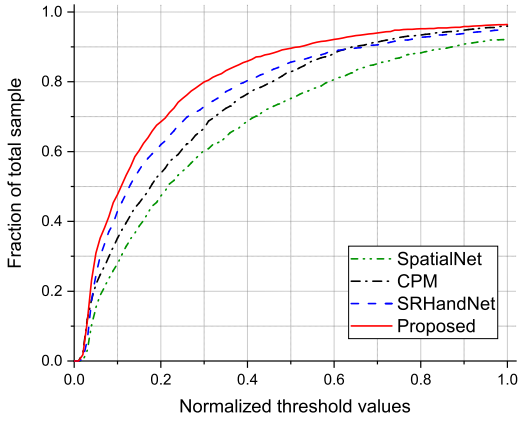
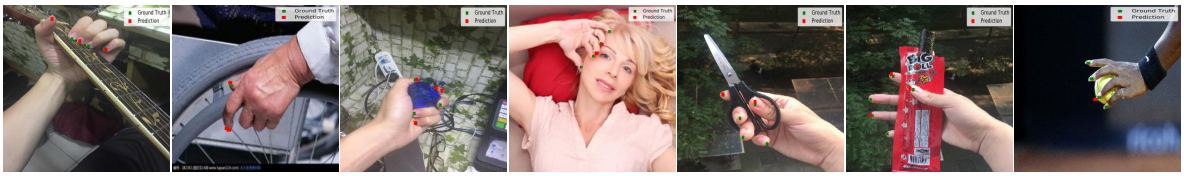
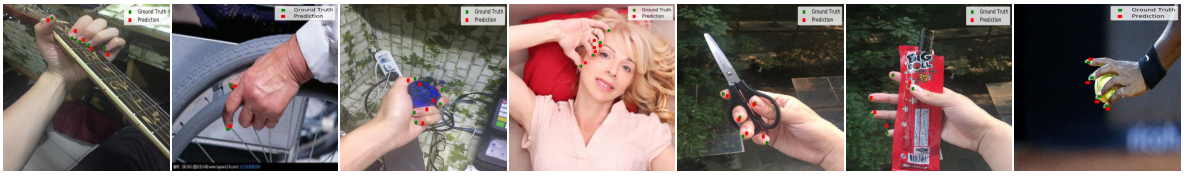


Figure 2.27: Comparative analysis in terms of PCK on RHD dataset

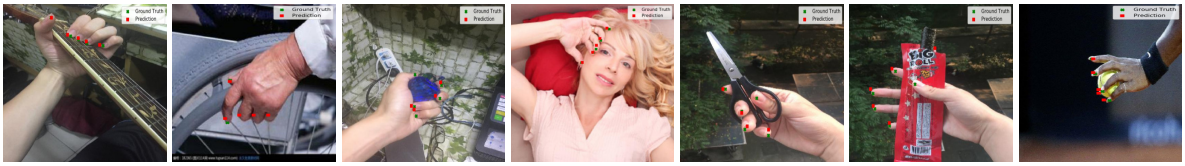
olds on OneHand10K and HGR datasets are shown in Figure 2.26. The PCK curves RHD dataset is shown in Figure 2.27. The red curve with the legend as 'proposed' shows the performance of the nearest-neighbor algorithm. The qualitative comparison of a few methods on the OneHand10K dataset is shown in Figure 2.28. The ground truth and estimated fingertips are represented with red and blue markers, respectively.



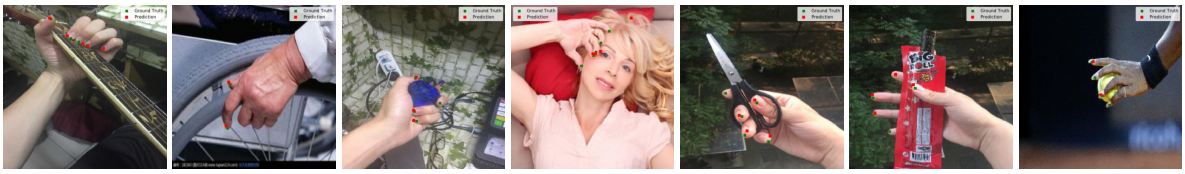
(a) YOLSE [12]



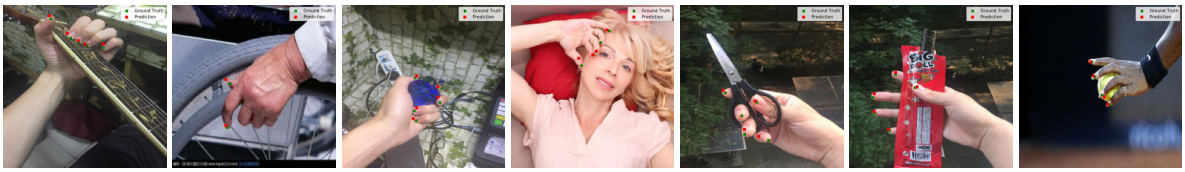
(b) FingID [101]



(c) OpenPose [84]



(d) SpatialNet [3]



(e) Mask-Pose [13]

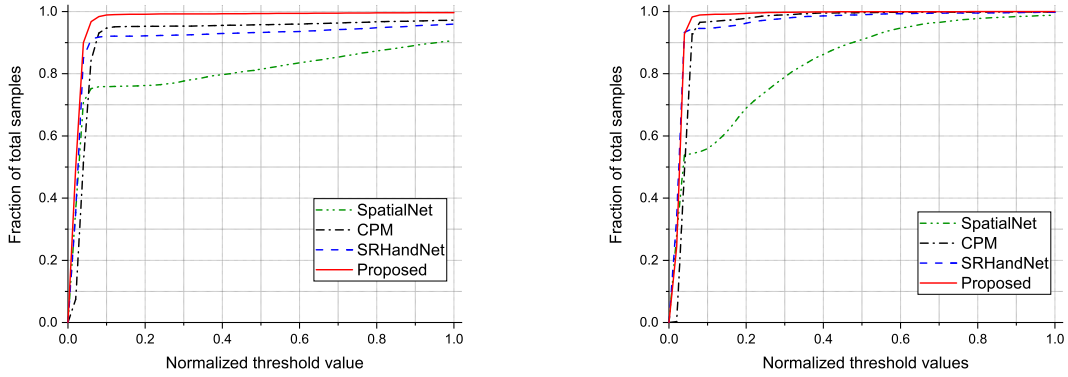
(f) Nearest-Neighbor

Figure 2.28: Qualitative comparison of fingertips' detection on OneHand10K dataset samples with various methods

Table 2.5: **Comparison Results of the proposed method for fingertips detection with different methods on the various datasets**

Models	OneHand10K [13]		HGR [16]		STB [21]		FPHA [22]		RHD [20]		SCUT-EgoGesture [12]			
											Single		Double	
	PCK ($\sigma = 0.2$)	mean	PCK ($\sigma = 0.2$)	mean	PCK ($\sigma = 0.2$)	mean	PCK ($\sigma = 0.2$)	mean	PCK ($\sigma = 0.2$)	mean	PCK ($\sigma = 0.2$)	mean	PCK ($\sigma = 0.2$)	mean
SpatialNet [3]	0.4932	0.6415	0.4771	0.6753	0.3813	0.6685	0.9288	0.9046	0.4726	0.6691	0.7619	0.7957	0.6892	0.8272
CPM [83]	0.5215	0.6759	0.5313	0.7056	0.6542	0.7445	0.9482	0.9046	0.5379	0.7315	0.9523	0.9119	0.9779	0.9419
OpenPose [84]	0.0718	0.3918	0.0519	0.5645	0.1986	0.6814	0.7425	0.7934	-	-	-	-	-	-
YOLSE [12]	0.1736	0.4329	0.0478	0.2465	0.2195	0.6005	0.6526	0.6595	-	-	-	-	-	-
FingID [101]	0.5986	0.7328	0.4781	0.6331	0.2486	0.6569	0.8602	0.8224	-	-	-	-	-	-
Mask-Pose [13]	0.4457	0.6369	0.3774	0.6164	0.6167	0.7391	0.9118	0.8975	-	-	-	-	-	-
SRHandNet [14]	0.5544	0.7068	0.5952	0.7422	0.7390	0.8215	0.9546	0.9186	0.6198	0.7562	0.9216	0.9039	0.9622	0.9504
Liu *et al.* [114]	0.5415	0.7286	0.6062	0.7446	0.6963	0.7272	0.9248	0.9412	0.6352	0.7781	0.9032	0.9142	0.9578	0.9351
Yang *et al.* [115]	0.5661	0.7458	0.6112	0.7231	0.6851	0.7976	0.9374	0.9080	0.5992	0.7746	0.9317	0.8863	0.9351	0.9378
Fan *et al.* [116]	0.5392	0.7196	0.6238	0.7155	0.6726	0.8003	0.9118	0.8974	0.5438	0.7618	0.9483	0.9417	0.9231	0.9356
Nearest Neighbors	**0.6698**	**0.7683**	**0.6256**	**0.7619**	**0.8698**	**0.8994**	**0.9691**	**0.9567**	**0.6851**	**0.7976**	**0.9913**	**0.9619**	**0.9939**	**0.9627**

The model is shown in Figure 2.23 designed for single-hand fingertips detection. But the same model can be extended for the detection of fingertips from both hands. After, increasing the number of output neurons, the model was tested to detect the fingertips of double hand, samples for which are provided in the SCUT-Ego-Gesture dataset. The model's like SpatialNet, CPM, and SRHandNet are modified and retrained on the same dataset for performance comparison. All the model's performance was compared on about 12,163 image samples of a single hand and 5,497 image samples of a double hand from the SCUT-Ego-Gesture dataset. The PCK curves for single and double hands are shown in Figure 2.29. The nearest-neighbor-based algorithm is the best-performing algorithm among the models used for comparison. Some sample results on double-hand fingertips detection are shown in Figure 2.30.



(a) Single hand (b) Double hand

Figure 2.29: Performance comparison for single and double hand.

2.6.6 Discussion

The experiment conducted to validate the nearest-neighbor-based fingertips detection algorithm shows its feasibility in the estimation of fingertips' position from a monocular RGB image. The design works without the need for a hand localization step. Additionally, the results showed that the fingertips can be estimated for a high range of potential valid hand postures. The challenges of lighting variations, skin color, background, etc. are handled effectively in the proposed approach. In addition, the

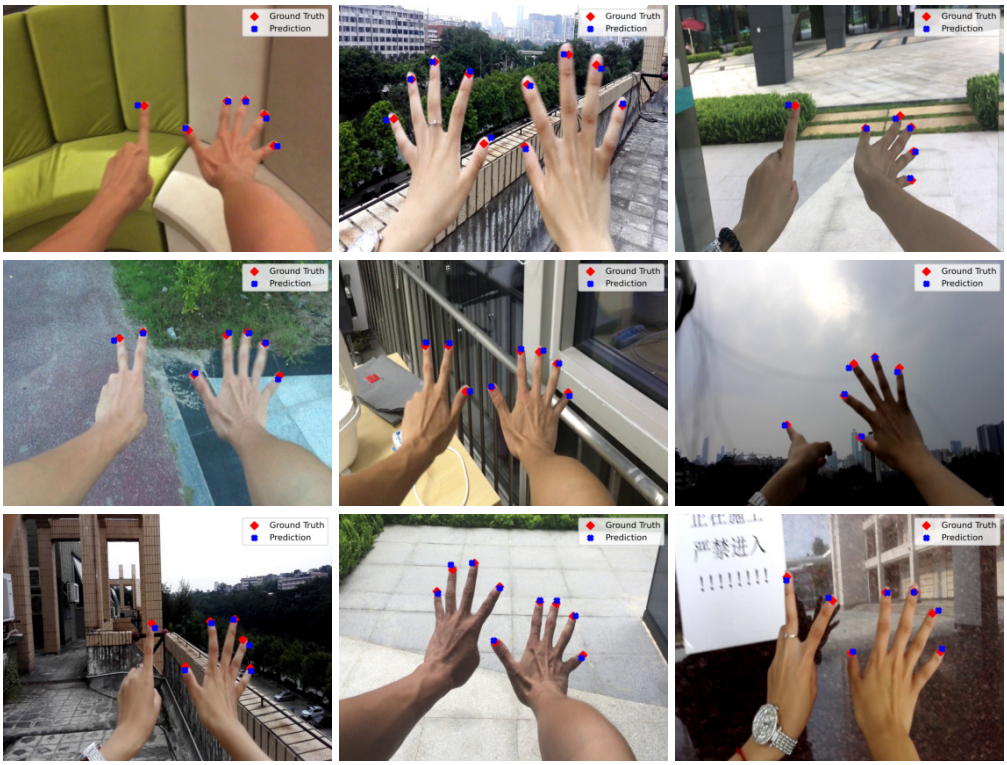


Figure 2.30: Sample results on double hand fingertips detection.

results showed that the algorithm designed for single-hand fingertips detection can be extended for the estimation of fingertips from both hands. It was also observed that the accuracy improved when the number of neighborhood pose particles involved in the estimation of fingertips location is more than one. The nearest-neighbor-based fingertips detection algorithm can be used in gesture-based interactive applications like virtual typing, airwriting, etc.

Concluding Remarks

In this chapter, the topic of partial hand keypoints detection was covered. The partial keypoints detection process is limited to the detection of sub-sets of complete hand keypoints like fingertips, wrist-point, palm center, etc. The detection of these subsets of keypoints is important for certain standalone applications like air-writing, virtual typing, and others. Traditionally, hand keypoints detection or any other keypoint detection of the human body follows a two-step approach. First, the subject is localized,

and then using an independent algorithm hand keypoint is detected. With time the process of keypoint detection has moved on from feature engineering to the use of the deep neural network for automatic feature extraction. The use of heatmap regression is the most common approach used in keypoints detection. In this chapter, we described the working of three different approaches used for fingertips detection from a monocular RGB image. We described the challenges associated with RGB images when used for a computer-vision application and how to take care of those while estimating the fingertips' position. The first approach described in this chapter solves the problem of simultaneous multiple fingertips detection using a multi-label object classification approach. In the second approach, based on the observation of the distribution of fingertips' position in the cropped image, an algorithm based on local regression was proposed. Eventually, taking hints from the anchors-based approach, a single-stage algorithm for fingertips' estimation was designed. This approach made the overall process independent of hand localization and improved the performance compared to the heatmap regression-based approaches. However, all these discussed approaches for the estimation of fingertips' position were limited to 2D pose. Nevertheless, there are multiple approaches available that can take 2D keypoints positions and can estimate the 3D positions.