ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

KHOA CÔNG NGHỆ THÔNG TIN

COURSE PROJECT

Digital Image & Video Processing

Hand detection and Finger Pose Estimation

Giảng viên hướng dẫn:

PGS. TS. Lý Quốc Ngọc

Người thực hiện:

Dương Thị An – 20120240

Phan Đình Anh Quân – 20120635

Phạm Gia Thông – 20120201

Hoàng Đức Nhật Minh – 20120328

2022 – TP. Hồ Chí Minh

Table of contents

I. Introduction

1. Motivation

Gesture recognition techniques have been used to understand human gestures and body language, thereby building a bridge between humans and machines. Hand gestures is a unique and significant component of human action since the information hand gestures convey is sophisticated. In comparison to previous generations, the

current generation is more reliant on these computer-based devices for day-to-day activities. As our dependence on electronic devices grow, there is a need to advance technology that connects humans and machines. One of such uses which has received great recognition and interests recently, is hand gesture recognition.

Gestures are expressive, meaningful body motions that involve physical movements of the fingers, hands, arms, head, face, or torso with the goal of communicating or engaging with the surroundings. Gestures are complementary to words in human-human interaction for those who can speak, while gestures are the only methods of communicating to other humans for those who do not have the ability to speak. Thus, hand gesture recognition makes it easier to regulate human-human or human-machine communication.

2. Application

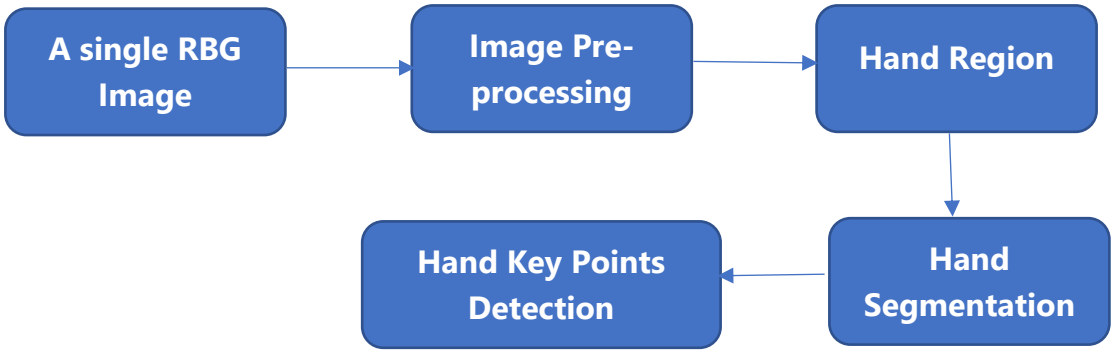
Hand gesture recognition has been used in many fields of study including recognition of sign language, which is beneficial for people with disabilities, and in medicine to comprehend a user's movements in real time and modify items in a medical data visualization environment. One study explored the application of this approach in 3D gaming, specifically the Candy Crush game. Another study used a real-time hand gesture to control a robot to facilitate human-robot interaction. In another study, gesture control is used to control TV operations and interact with a mobile device without the need of sensors attached to the human body. Finally, one research looks at how gesture interaction technology may be used in virtual reality (VR), along with discussion of the present challenges with gesture interaction.

3. Problem Statement

Our problem is to recognize the hand and the key points and then apply it to recognize its actions.

The input of the problem is an image which contains a single hand. The output of the problem is the analyzed action of that hand.

The framework of the system



4. Challenges in Dataset Construction

Task formulation and algorithms for estimating 3D hand poses are outlined. To train an estimator, a large amount of training data with diverse hand poses, viewpoints, and backgrounds is necessary. However, obtaining such massive hand datasets with an accurate annotation of 3D hand poses is challenging for the following reasons.

Although numerous works seek to estimate 3D hand poses from a single RGB image, annotating the 3D position of hand joints from a single RGB image is inherently impossible due to an ill-posed condition.

To assign accurate hand pose labels, handmarker-based annotation using magnetic sensors, motion capture systems, or hand gloves has been studied. They provide 6- DoF information (i.e., location and orientation) about hand joints. However, these methods are unsuitable for visual learning based on the RGB modality because they change hand appearance drastically.

On the contrary, depth sensors (e.g., RealSense) or multi-view camera studios make it possible to obtain depth information near hand regions. Given 2D keypoints for an image, these setups enable annotation of 3D hand poses by measuring the depth distance at each 2D keypoint. However, these annotation methods do not always produce an accurate annotation e.g., due to an occlusion problem. The depth images are significantly affected by the sensor noise and become inaccurate especially when the hands are far from the sensor and appear to be tiny. Depth measurement by triangulation in a multi-camera setting is unreliable when the cameras are not densely arranged.

II. Related works

The latest comprehensive review work on hand pose estimation and hand gesture was published in 2007, the foucus of paper was primarily vision based hand pose estimations.

They divided pose estimation into categories:
1. Appearance based: inferring the hand gesture or pose directly from the observed visual data without the help of a model. This usually implemented using machine learning (ML) which require a large training 3 data or with Inverse Kinematic (IK) or a hybrid approach.

2. Model based: this approach generate multiple hand model usually called hypothesis and it tries to find the model that best fit the observed data. Pretty much it convert the problem into a high dimension optimization problem, finding the model that minimizing a certain cost function.

There are a lot of progress and improvements since 2007 in vision based articulated hand pose estimation, especially from depth sensor. This paper will focus primarily on the state of the art works done post 2007.

A more recent paper in 2015 focused on comparing 13 hand pose estimation algorithms from a single depth frame and evaluated their performance on various publicly available dataset. Furthermore, created a new hand training dataset that is more diverse and complex to existing one. focus primarily on comparing the quality of each of the 13 algorithms using a common training dataset, this paper focus on reviewing the latest state-of-the-art hand pose estimation algorithms.

There are also older reviews for hand gesture; however, those reviews focused mainly on gesture recognition and not pose estimation which is a more challenging problem.

Paper	Year	Solutions	Perform-ance	Datas et	Advantages	Disadvantage s
Hands Deep in Deep Learning for Hand Pose Estimation	2016	Hand detection with LRF. Predicting hand keypoints with CNN.	AUC = 0.79	NYU	Does not need the depth information. Good accurancy.	Need labeled dataset and 3D locations for training. When the pixel in the image is not available due to the brightness effect, the result will be inaccurate.
Learning to Estimate 3D Hand Pose from Single RBG Images	2017	Hand segmentation with HandSegNet Keypoint score maps with PoseNet 3D hand pose with the PosePrior network	AUC = 0.724	RHD	Does not need the depth information	The performance seem mostly limited by the lack of an annotated large scale dataset with real world images and diverse pose statistics
3D Convolut-ional Neural Netwo rks for Efficient and Robust Hand Pose Estimation f-rom Single Depth Images	2017	Estimates 3D hand pose from single depth images. The hand depth image is encoded by a volumetric representation which is the input of our proposed 3D CNN.	90% accuracy	NYU	Fast result for real time application. Can be applied for a variety of data with different hand size by enhancing 3d data in the training set	The computational complexity increases with the image resolution

3D Hand Shape and Pose Estimation from a Single RGB Image	2019	Extracting feature maps and 2D heat-maps: a two stacked hourglass network Graph CNN to infer the 3D coordinates of mesh vertices.	AUC = 0.974	STB	Can achieve superior 3D hand pose estimation accuracy when compared with SOTA methods	Hard to self-implement due to large dataset and need robust devices
Hand Gesture Recognition Based on Auto Landmark Localization and Reweighted Genetic Algorithm for Healthcare Muscle Activities	2021	Hand segmentation and hand detection Landmark detection: Geodesic Distance Features extraction: using hand key points and full hand Feature Optimization: gray wolf optimization Classification: Reweighted Genetic Algorithm	90% accuracy	Sign word, Dexter1, STB, NYU	Take advantage of hand landwmark for feature extraction and gesture recognition	The efficiency is not too high compared to the above methods

III. Research Methods

1. Datasets

To train the MediaPipe models, three different data sets were chosen. The datasets used to train the model are described below.

• In-the-wild dataset: This dataset comprises six thousand images with a wide range of characteristics, such as geographical diversity, lighting conditions, and hand appearance. The dataset's limitation is that it does not include complicated hand articulation.

• Googles in house collected dataset: This collection contains ten thousand images that depict all physically conceivable hand movements from various perspectives. The dataset's shortcoming is that it was compiled from only 30 people with little variance in their backgrounds. The in-the-wild and in-house datasets are excellent complements for improving robustness.

• Synthetic dataset: This dataset maps a high-quality synthetic hand model to the corresponding 3D coordinates after rendering it over diverse backdrops. A commercial 3D hand model with 24 bones and 36 blendshapes was used to regulate the thickness of the fingers and palm. The hand model also includes five textures with various skin tones. Transformation video sequences between hand positions were constructed and hundred thousand images were sampled from the videos. Each posture was rendered using three separate cameras and a random high-dynamic-range lighting environment.

In-the-wild dataset was used for the palm detector since it is sufficient for hand localization and has the most variability in appearance. All datasets, however, are utilized to train the hand landmark model.

2. Palm Detection Model

Detection of hands is a complex task compared to others as it has to work across variety of hand sizes and be able to detect occluded part of hands. The palm detector is trained because estimating bounding boxes around rigid objects like the palm and fist is a much easier task. Encoder decoder feature is used for a larger scene context awareness similar to FPN and finally focal loss is minimized to support large scale variance.

The object detection module currently consists of the following steps: Hypothesize bounding boxes, resample features for each box and apply a high-quality classifier. There are several object detection algorithms that are currently being used such as Faster RCNN, SSD, You only look once (YOLO) detector. But they all differ on speed and accuracy. The speed of object detection is measured in frames per second (FPS) and accuracy is often determined by calculating Mean Average Precision (mAP). There's a tradeoff between speed and accuracy and usage scenarios vary depending on the problem at hand. Faster R-CNN has high accuracy but only runs at 7 frames per second. It could be applicable in a scenario where accuracy is of utmost importance. For example, it is not fast enough for automated Driving as speed and accuracy both matter and faster RCNN is very slow. Likewise YOLO is known for its speed, however, it suffers in terms of accuracy. Unlike other deep neural network based object detectors, SSD forgoes the resampling of pixels and features for bounding box hypothesis. SSD is much faster and more accurate (59 FPS with mAP 74.3%) than YOLO(45 FPS with mAP 63.4%) and Faster RCNN (7 FPS with 73.2% mAP) for the VOC2007 test. (Liu 2). SSD produces high accuracy even on low resolution input images which improves the speed vs accuracy trade off.

2.1. Single shot detector (SSD)

The process for SSD includes creating the bounding box around each object in the image. The whole image is divided into a feature map of diffirent sizes. Default boxes of different sizes are evaluated at each square of the feature map. These default boxes consists of two information: the location ($center_x$, $center_y$, width, height) and the confidence for all object categories (c_1, c_2, …, c_p). The shape offsets and confidences for all object categories should be predicted for each default box. The default boxes should be matched to the ground truth boxes during training.

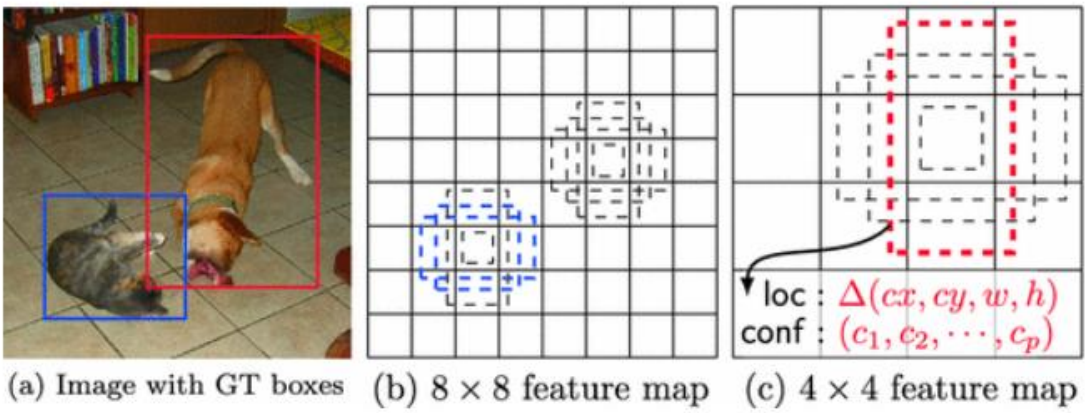


Figure 2.7: Single Shot Detector (SSD) framework, loc is location of the bounding box, conf is the confidence of all object categories, Figure from [16].

The boxes that match the default box should be considered positve, while the rest should be considered negative. The model loss should be calculated as the weighted sum between localization loss (Smooth L1) and confidence loss. In Figure 2.7, the example of an image with cats and dogs are given. The feature map chosen is of size 8 × 8 and 4 × 4. The default boxes that match up with the Ground truth box of cats and dogs are treated as positives and the rest as negatives. SSD Model is based on the feed forward convolutional network that produces a fixed size collection of bounding boxes and gives confidence value for the object that exists in the bounding box.

2.2. Feature Pyramid Network (FPN)

FPN is used as neck to the backbone of SSD. FPN is used to compute the multi-feature representation of the image. The featured pyramid are built upon the image pyramid and are scale inavariant. These are used to capture the various sizes of objects. Figure 2.8 (a) image is downsampled and feature maps are computed at each level. This is denoted by the blue outline which gets thicker for stronger features. Figure 2.8 (b) is a typical convolution neural network type structure that downsamples the image and then predicts at the end. This is expected to be fast as it only uses single scale features as

compared to (a). Figure 2.8 (c) is used by SSD. This architecture uses pyramidal feature hierarchy computed by convolutional neural network as if it were featurized image pyramid. Architecture in Figure 2.8 (d) is proposed in [14] as it is as fast as (b) and (c) but more accurate. This architecture first downsamples the image to it's lowest resolution and then upsamples them. The features are then combined by the lateral connection which is $1 * 1$ convolution. This combines the features from highest resolution to lowest resolution so it makes it more accurate than (b) and (c) due to availability of more information.

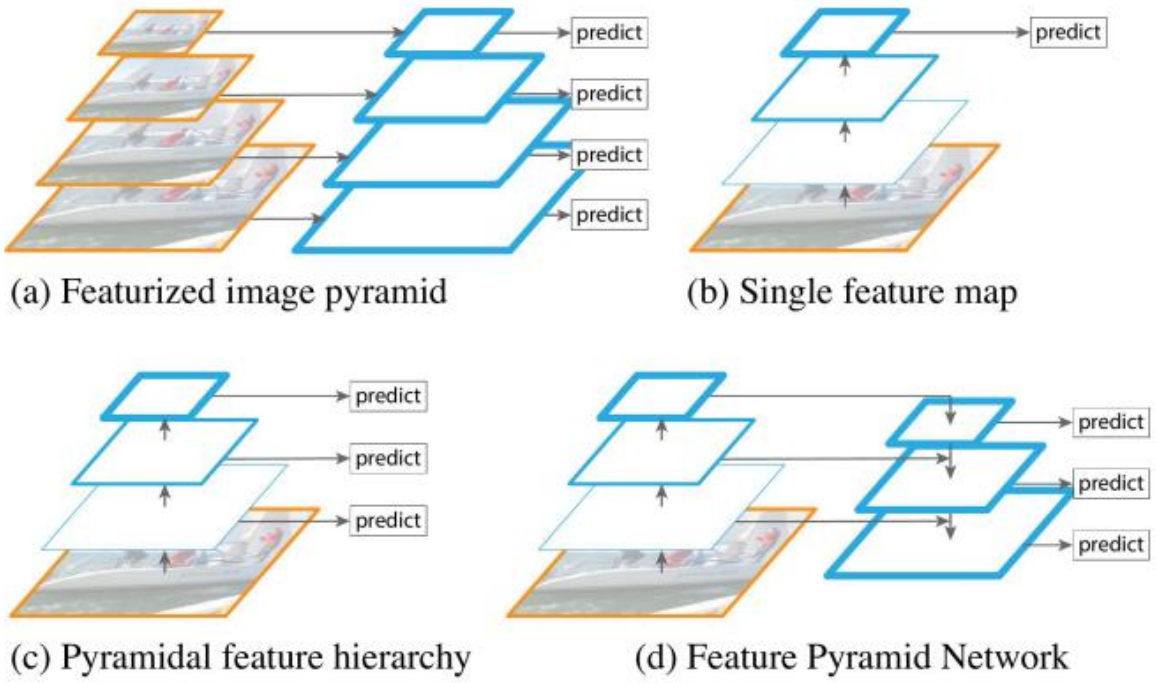


Figure 2.8: Feature Pyramid Network (FPN), Figure from [14].

2.3. Minimising focal loss

Focal loss addresses the problem of one stage detector where there is an imbalance between the foreground mage and the background image. With cross entropy loss, even easy examples that are easily classified incur a loss that is greater than expected.

And if these small losses are summed over a large number of easy examples, then the added loss can overwhelm the harder examples. The cross entropy loss is overwhelmed by the huge class imbalance experienced during dense detectors. The majority is made up of easily recognize negatives, which dominate the gradient. Balanced cross entropy addresses the class imbalance by inroducing a weighting factor alpha. This helps to distinguish between the negatives and positives but does not help

with easy and hard examples. Easy examples are those examples that have a probability of ground truth class (p_t >> 0.5). The equation for balanced cross entropy is given in () where CE is cross entropy loss.

$$CE(p_t) = -\alpha \log(p_t)$$

Focal loss reshapes the loss function to down weight the easy examples and focuses the training on hard negatives. The equation for focal loss FL is given in equation where y is the smoothing parameter.

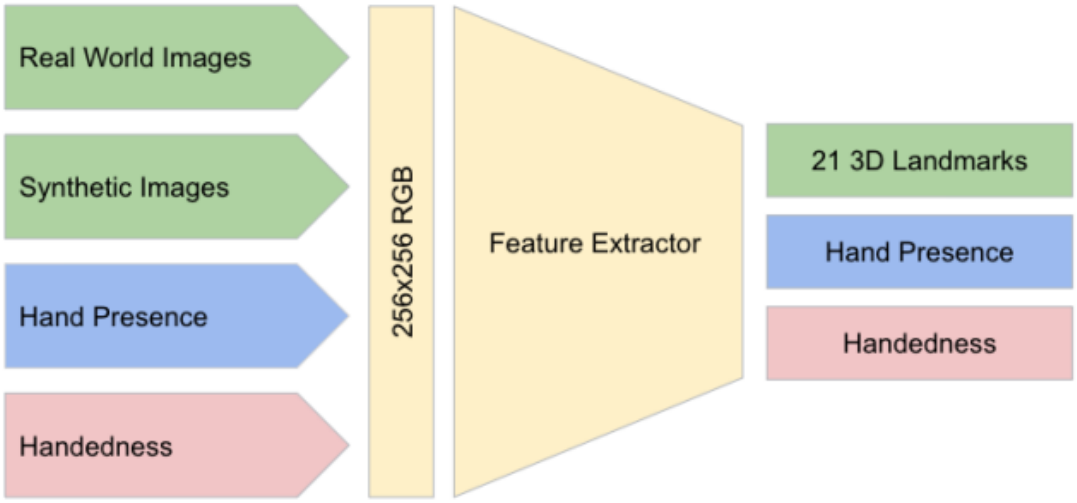
$$FL(p_t) = -(1 - p_t)^y \log(p_t)$$

In equation (), misclassified hard example has small pt, the modulating factor $1 - p_t$ is close to 1. Therefore the loss function is unaffected. But for easy examples, p_1 goes to 1, modulating factor goes to 0, the loss for well classified easy examples is down weighted. The focusing parameter y is used to smoothly adjust the rate at which easy examples are down weighted. In practice, alpha variant focal loss is used for better accurancy over Equation.

$$FL(p_t) = -\alpha_t(1 - p_t)^y \log(p_t)$$

3. Hand landmark Model

Following palm detection throughout the whole picture, our hand landmark model uses regression to conduct exact land-mark localization of 21 2.5D coordinates inside the identified hand areas. The is represented in Figure … Even with partially visible hands and self-occlusions, the model develops a consistent internal hand posture representation. The output given by the hand landmark model are the 21 3D landmarks, hand flag to know the probability of hand presence and the handedness to know which hand is present left or right. This architecture is given in 2.10. The topology similar to Multiview bootstraping is used to detect 21 landmarks specifically to boost the performance of a keypoint detector. This helps the hand landmark model by estimating the keypoints of the occluded hands. And to recover from the tracking failure a model is generated. This is done in order to increase the likelihood that a suitably aligned hand is present in the given crop.



3.1. Multiview Bootstrapping

Even if there is severe occlusion in a single picture of the hand, there is usually an unobstructed view. Multiview bootstrapping systematizes the concept, resulting in a more effective han detector that generalizes beyond the capture scenario. A weak detector trained on a small, annotated dataset can localize subsets of a key-points in good views and filter out incorrect detection by using 3D triangulation.

A keypoint detector d(.) maps a cropped input image patch $\boldsymbol{I} \in \mathbb{R}^{w \times h \times 3}$to P keypoints location $x_p \in \mathbb{R}^2$, each with an associated detection confidence c_p:

$$d(I) \mapsto \{(x_p, c_p) \text{ for } p \in [1 \dots P]\}$$

Each point p corresponds to a different landmark and we assume that only a single instance of the object is visible in **I**. The detector is trained on images with correspoding keypoint annotations,$(I^f, \{y_p^f\})$, where f denotes a particular image frame, and the set $\{y_p^f \in \mathbb{R}^2\}$ includes all labeled keypoints for image I^f. An initial training set $\mathcal{T}_0$ having N_0training pairs,

$$\mathcal{T}_0 := \{(I^f, \{y_p^f\}) \text{ for } f \in [1 \dots N_0]\}$$

Is used to train an initial detector d_0 with stochastic gradient descent

$$d_0 \leftarrow train(\mathcal{T}_0)$$

Given the initial keypoint detector d_0 and a dataset of unlabeled multiview images, our objective is to use the detector to generate a set of labeled images, $\mathcal{T}_1$,which can be used to train an improved detector, d_1, using all avaibable data:

$$d_1 \leftarrow train(\mathcal{T}_0 \cup \mathcal{T}_1)$$

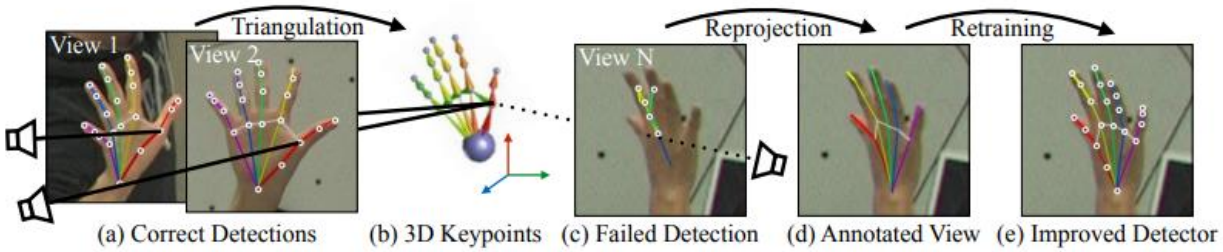


Figure 3: Multiview Bootstrapping. (a) A multiview system provides views of the hand where keypoint detection is easy, which are used to triangulate (b) the 3D position of the keypoints. Difficult views with (c) failed detections can be (d) annotated using the reprojected 3D keypoints, and used to retrain (e) an improved detector that now works on difficult views.

To improve upon the detector d0, we need an external source of supervision to ensure $\mathcal{T}_1$ contains information not already present in $\mathcal{T}_0$. We propose to use verification via multiview geometry as this source. The key here is that detection is easier in some views than others: if a point is successfully localized in at the least two views, the triangluted 3D position can be reprojected onto other images, providing a new 2D annotaion for the views on which detection failed. This process is show in Figure, where the detector succeeds on easy views but fails on more challenging views. However, by triangulating the correctly detected viewpoints, we can generate training data particularly for those views in which the detector is currently failing.

To overall procedure for multiview bootstrapping is described in Algorithm 1, where we denote by $\{I_v^f : v \in [1 \dots V], f \in [1 \dots F]\}$ the input set of unlabeled multiviews image frames, with v iterating over the V camera views, and f iterating over F distinct frames. There are three main parts to the process detailed in the following subsections: (1) for every for every frame, the algorithm first runs the current detector on every camera view independently (Fig. 3a,c) and robustly triangulates the point detections (Fig. 3b); (2) the set of frames is then sorted according to a score to select only correctly triangulated examples, and (3) the N-best frames are used to train a new detector by reprojecting the correctly triangulated points onto all views (Fig. 3d), producing approximately V training images for each of the N selected frames. The entire process can then be iterated with the newly trained detector (Fig. 3e).

Triangulating Keypoints from Weak Detections

Algorithm 1 Multiview Bootstrapping

Inputs:

- Unlabeled images: $\{\mathbf{I}_v^f$ for $v \in$ views, $f \in$ frames$\}$
- Keypoint detector: $d_0(\mathbf{I}) \mapsto \{(\mathbf{x}_p, c_p)$ for $p \in$ points$\}$
- Labeled training data: $\mathcal{T}_0$

for iteration i in 0 to K:

1. Triangulate keypoints from weak detections
   for every frame f:
   (a) Run detector $d_i(\mathbf{I}_v^f)$ on all views v (Eq. (5))
   (b) Robustly triangulate keypoints (Eq. (6))
2. Score and sort triangulated frames (Eq. (7))
3. Retrain with N-best reprojections (Eq. (8))
   $d_{i+1} \leftarrow \text{train}(\mathcal{T}_0 \cup \mathcal{T}_{i+1})$

Outputs: Improved detector $d_K(\cdot)$ and training set $\mathcal{T}_K$

Given V views of an object in a particular frame f, we run the current detector d_i (trained on set $\mathcal{T}_i$) one each image, yielding a set D of 2D location candidates:

$$\mathcal{D} \leftarrow \{d_i(I_v^f)\ for\ v\ \in [1 \ldots V]\}$$

For each keypoint p, we have V detections (x_p^v, c_p^v), where x_p^v is the detected location of point p in view v and $c_p^v \in [0, 1]$ is a confidence measure (we omit the frame index for clarity). To robustly triangulate each point p into a 3D location, we use RANSAC on points in $\mathcal{D}$ with confidence above a detection threhold λ. Additionally, we use a $\sigma = 4$ pixel reprojection error to accept RANSAC inliers. With this set of inlier views for point p, we minimize the reprojection error to obtain the final triangulated position,

$$X_p^f = \arg\min_X \sum_{v \in \mathcal{L}_p^f} \left\| P_v(X) - x_p^v \right\|_2^2$$

Where $\mathcal{L}_p^f$ is the inlier set, with $X_p^f \in \mathbb{R}^3$ the 3D triangulated keypoint p in frame f, and $P_v(X) \in \mathbb{R}^2$ denotes projection of 3D point X into view v. Given calibrated cameras, this 3D point can be reprojected into any view and serve as a new training label.

To improve robustness specifically for hands, we reconstruct entire fingers simultaneously. We triangulate all landmarks of each finger at a time, and use the average reprojection error of all 4 points to determine RANSAC inliers. This procedure is more robust because errors in finger detections are correlated: … if the knuckle is incorrectly localized, then dependent joints in the kinematic chain the inter-phalangeal joints and finger tip are unlikely to be correct. This reduces the number of triangulated keypoints but it futher reduces the number of false positives, which is more important so that we do not train with incorrect labels.

Scoring and Sorting Triangulated Frames

It is crucial that we do not include erroneously labeled frames as training data, especially if we iterate the procedure, as subsequent iterations will fail in a geometrically consistent way across views – a failure which cannot be detected using multi-view constraints. We therefore conservatively pick a small number of reliable triangulations.

Our input is video and consecutive frames are therefore highly correlated. Instead of uniform temporal subsampling, we pick the "best" frame for every window of W frames, defining the "best" as that frame with maximum sum of detection confidences for the inliers

$$score(\{X_p^f\}) = \sum_{p \in [1 \ldots P]} \sum_{v \in \mathcal{L}_p^f} c_p^v$$

We sort all the remaining frames in descending order according to their score, to obtain an ordered sequence of frames, $[s_1, s_2, \ldots s_{F'}]$, where F' is the number of subsampled frames and si is the ordered frame index.

We manually verify that there are no obvious errors in the frames to be used as training data, and train a new detector. While visual inspection of the training set may seem onerous, in our experience this is the least time consuming part of the training process. It typically take us one or two minutes to inspect the top 100 frames. Crowdsourcing such a verification step for continuos label generation is an interesting future direction, as verification is easier than annotation. In practice, we find that this manual effort can be almost eliminated by automatically removing questionable triangulations using a number of heuristics: (1) average number of inliers, (2) average detection confidence, (3)

difference of per-point velocity with median velocity between two video frames, (4) anthropomorphic limits on joint lengths1 , and (5) complete occlusion, as determined by camera ray intersection with body joints. Additionally, we require at least 3 inliers for any point to be valid.

Retraining with N-best Reprojections

We use the N-best frames according to this order to define a new set a training image-keypoint pairs for the next iteration i + 1 detector,

$$\mathcal{T}_{i+1} = \left\{\left(I_v^{S_n}, \left\{P_v\left(X_p^{S_n}\right): v \in [1 \dots V], p \in [1 \dots P]\right\}\right) for\ n \in [1 \dots N]\right\}$$

where $P_v\left(X_p^{S_n}\right)$ denotes projection of point p for frame index s_n into view v, and we aim for about N = 100 frames for every 3 minutes of video. Note that 100 frames yields roughly $100 . \frac{V}{2} \cong 1500$ training samples, one for each unoccluded viewpoint. Finally, we train a new detector using the expanded training set as $d_{i+1} \leftarrow train(\mathcal{T}_0 \cup \mathcal{T}_{i+1})$

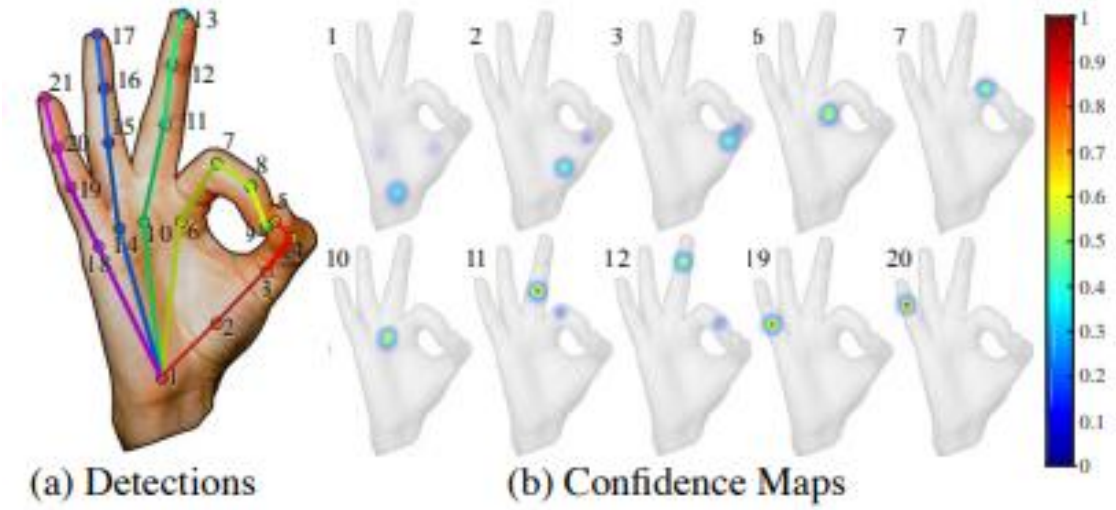


Figure 4: (a) Input image with 21 detected keypoints. (b) Selected confidence maps produced by our detector, visualized as a "jet" colormap overlaid on the input.

Detection Architecture

For the detectors di, we follow the architecture of Convoluntional Pose Machines (CPMs), with some modification. CPMs predict a confidence map for each keypoint, representing the keypoint's location as a Gaussian centered at the true position. The

predicted confidence map corresponds to the size of the input image patch, and the final position for each keypoint is obtained by finding the maximum peak in each confidence map.

Keypoint Detection via Confidence Maps.

In contrast to Convolutional Pose Machines, we use the convolutional stages of a pre-initialized VGG-19 network up to conv4_4 as a feature extractor, with two additional convolutions producing 128-channel features F. For an input image patch of size w×h, the resulting size of the feature map F is $w' \times h' \times 128$, with $w' = \frac{w}{8}$ and $h' = \frac{h}{8}$. There are no additional pooling or downsampling stages, so the final stride of the network is also 8. This feature map extraction is followed by a prediction stage that produces a set of P confidence or score maps, $S^1 = \{S_1^1 \dots S_p^1\}$, one score map $S_p^1 \in \mathbb{R}^{w' \times h'}$ for each keypoint p. Each stage after the first takes as input the score maps from the previous stage, S^{t-1}, concatenated with the image features F, and produces P new score maps S^t, one for each keypoint. We use 6 sequential prediction stages, taking the output at the final stage, S^6. We resize these maps to the original patch size (w × h) using bicubic resampling and extract each keypoint location as the pixel with maximum confidence in its respective map. We also modify the loss function in CPMs to be a weighted L_2 loss to handle missing data, where the weights are set to zero if annotations for a keypoint are missing (e.g., if triangulation for that point fails).

Hand Bounding Box Detection.

Our keypoint detector assumes that the input image patch $\boldsymbol{I} \in \mathbb{R}^{w \times h \times 3}$ to P is a crop around the right hand. This is an important detail: to use the keypoint detector in any practical situation, we need a way to generate this bounding box. We directly use the body pose estimation models from [29] and [4], and use the wrist and elbow position to approximate the hand location, assuming the hand extends 0.15 times the length of the forearm in the same direction. During training, we crop a square patch of size 2.2B, where B is the maximum dimension of the tightest bounding box enclosing all hand joints (see Fig. 8 for example crops of this size). At test time, we approximate B = 0.7H where H is the length of the head "joint" (top of head to bottom). This square patch is resized to w = 368 and h = 368, which serves as input to the network. To process left hands, we flip the image left-to-right and treat it as a right hand.

3.2. Recover tracking failure

The algorithm to recover tracking failure for face detection is given in Real-time facial surface geometry from monocular video on mobile gpus. MediaPipe uses a similar

method to make sure that the hand is present in the cropped image. We will breifly look at the image processing pipeline given in Real-time facial surface geometry for the face mesh prediction and generalize it to hand scenario. A lightweight hand detector can be used to process the image from the camera input, producing hand bounding rectangles and a number of landmarks for the wrists, and fingers. The landmarks can be used to rotate a hand rectangle such that, the line linking the wrist center is aligned with the rectangle's horizontal axis. Original image is cropped according to the rectangle obtained in the previous step and resized to provide the input to the mesh prediction neural network (varying in size from 256×256 pixels in the full model to 128×128 pixels in the smallest one). 3D landmark coordinate vector is generated by the model and it is transferred back into the original image coordinate system. The probability of a reasonably aligned hand being present in the supplied crop is produced via a distinct scalar network output.

IV. Conclusion and future works

We defined the hand pose estimation problem and explained the major methods of solving this problem in detail. We also reviewed some of the recent applications of this field. Since every data driven method needs sufficient data in the first place, we talked about major datasets and listed all the datasets in this field with their most important properties. We showed how this field have grown in just a few years, from completely controlled situations with color gloves to 3D hand pose estimation using a single RGB image. Although the papers discussed here show good results on these dataset they do not get satisfactory results in the real world problems. Most importantly, the result of most of these systems is worst than a simple nearest-neighbor baseline. However, because of the interests of big technology companies in this field, perhaps in the near future we see much bigger and more generalized datasets and therefore very well performing models even on a single RGB image. If we reach this technology, using an AR/VR device as our new PC, typing on the air and control objects in the display with our fingers will not be out of reach to.