UKRAINIAN CATHOLIC UNIVERSITY

MASTER THESIS

3D Hand Pose Estimation from Single RGB Camera

Author:
Olha CHERNYTSKA
olha.chernytska@gmail.com

Supervisor:
Dmitry PRANCHUK
dpranchuk@wanna.by

A thesis submitted in fulfillment of the requirements for the degree of Master of Science

in the

Department of Computer Sciences
Faculty of Applied Sciences

Lviv 2019

Declaration of Authorship

I, Olha CHERNYTSKA, declare that this thesis titled, "3D Hand Pose Estimation from Single RGB Camera" and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed:

Date:

UKRAINIAN CATHOLIC UNIVERSITY

Abstract

Faculty of Applied Sciences

Master of Science

3D Hand Pose Estimation from Single RGB Camera

by Olha CHERNYTSKA

With the increase of popularity VR/AR applications, 3D hand pose estimation task has become very popular. 3D hand pose estimation from single RGB camera has great potential, because RGB cameras are cheap and already available on most mobile devices. In this thesis we work on improving pipeline for 3D hand pose estimation from RGB camera. We dealt with two challenges - sophisticated algorithmic task and absence of good datasets. We trained several convolutional neural networks and showed that direct heatmaps method is the best approach for 2D pose estimation and vector representation - for 3D pose. We demonstrated that adding data augmentations even for synthetic dataset increases performance on real data. For 2D hand pose estimation, we proved that it is possible to train neural network on large scale synthetic dataset and finetune it on small partly labeled real dataset to receive adequate results, even when only small part of keypoint labels is available. With no real 3D labels available, model trained on synthetic data still could correctly predict 3D keypoint locations for simple poses. All code and pre-trained models will be publicly available.

Acknowledgements

I would like to express my deep gratitude to Oleksii Molchanovskyi and Ukrainian Catholic University for creating practical, challenging and inspiring Data Science program.

I would also like to extend my thanks to Grammarly, which provided me with scholarship and made my study possible here.

Special thanks should be given Dmitry Pranchuk, my research supervisor, for his patient guidance and valuable recommendations.

Contents

Chapter 1

Indroduction

With the increase of popularity of Virtual Reality and Augmented Reality, 3D hand pose estimation task has become very popular. Having algorithm that can predict hand pose with high accuracy will eliminate need of highly expensive and cumbersome wired gloves and make VR/AR experience more natural and affordable. 3D Hand pose estimation can also become a part of pipeline for such important tasks as gesture recognition and sign language recognition.

3D Hand Pose Estimation refers to set of approaches that return 3D locations of hand keypoints, from which hand pose can be correctly understood. Hand pose estimation is a challenging task, because of huge variety of possible hand poses, severe self-occlusion, and nearly identical finger appearance.

Hand Pose Estimation as a problem can be approached from different viewpoints, depending on data available, financial status of the project, required prediction accuracy or any other limitations. These viewpoints are - wired gloves, depth camera, stereo camera, mono RGB camera.

3D Hand pose estimation from depth image is the most popular approach as of now, because of its high accuracy and existence of non-expensive depth cameras. However, depth cameras have limitations. Depth sensors are not applicable for outdoor environment, because they are very sensitive to active light sources; depth cameras require relatively high power consumption, so they cannot be used on mobile devices.

The least researched approach in 3D hand pose estimation is using single RGB camera. This approach shows relatively lower performance comparing to depth-based algorithms. However, in recent years neural networks are proved to be highly accurate for most of the machine learning tasks and 3D hand pose estimation is not an exception. 3D hand pose estimation from single RGB camera could significantly increase its performance by introducing recent neural network best practices. RGB camera is also preferable because it can be used in many applications without any special setup required, including usage on mobile devices, outdoor usage and hand pose estimation from community videos (such as Youtube). RGB cameras are more widespread than depth cameras; relatively good quality RGB cameras are available on most mobile devices and laptops. So there is a large field for application of RGB methods.

3D hand pose estimation from single RGB camera is a complicated task not only because sophisticated algorithms are needed, but also because good quality large-scale datasets are not available open-source and are extremely expensive to create.

In this works we will focus on improving pipeline for 3D hand pose estimation from single RGB camera. Whole pipeline includes hand localization, left-right hand classification, normalized 3D hand pose estimation, absolute 3D pose hand pose restoration and tracking steps. Hand localization and left-right hand classification

are already solved tasks in Computer Vision, restoring absolute 3D pose requires introduction of prior assumptions and cannot be restored with high accuracy any way, and tracking is more an algorithmic task. So we are going to restrict our attention to normalized 3D hand pose estimation, because it is main part of the pipeline and the least researched topic at the same time. And improving normalized 3D hand pose estimation will result in boost of performance for 3D hand pose estimation from RGB image in general.

We are going to try several neural network architectures and provide recommendations of what architecture to use in case good dataset is available. We will go further and investigate possible solutions to use, when no good dataset is available. We will work a lot with synthetic large-scale datasets, extensive data augmentations and partly labeled real datasets. All the code and pre-trained neural networks will be available on GitHub.

Chapter 2

Background and Related Work

This chapter contains comprehensive review of the available datasets and latest approaches for 3D hand pose estimation from single RGB image. We will also review relevant approaches for hand pose estimation from depth, stereo and RGB-D cameras, as some ideas may be reused for our task. Additionally, we review papers on 3D human pose estimation, as this task is similar and some techniques may be reused as well.

All current state of the art approaches for 3D hand pose estimation, reviewed in this chapter, use some modifications of convolutional neural network architectures. We will go deeper on why this happened.

In 2012 AlexNet [1] won ImageNet challenge. Network was able to classify images into 1000 classes with 16.4% top-5 error rate, and significantly outperformed previous winners – classic machine learning approaches [2]. This event proved that deep learning is worth attention and raised interest in both academia and business communities. The more people were involved in deep learning researches, the more impressive results were achieved.

Today, in 2018, after 6 years passed, deep learning dominates classic machine learning for most of the problem types in computer vision, natural language processing and reinforcement learning. Deep learning approaches are more accurate, generalize well to unseen data and easy to implement (using PyTorch and Tensorflow libraries).

Convolutional neural network (CNN) is the best technique for solving image and video processing tasks; and for some tasks, image classification with 1000 classes in particular, CNNs outperform humans – 5.1% top-5 error for human [3] and 3.5% for residual neural network [2] on ImageNet Challenge. Researchers from Stanford Vision Lab [3] compared human and CNN performance and concluded, that CNNs fail more often than humans when object for classification is small/thin, image filters were applied, object is represented in abstract form (3D rendered object, drawing,..) and object photo is from atypical viewpoint; besides, three last fail reasons can be easily solved by adding/augmenting training data. Human performance is worse comparing to CNNs for fine-grained recognition (such as dog breeds) and because of class unawareness; additionally, human training requires much more time than that for CNN. As for hand pose estimation, human performance is expected to be high for 2D keypoint localization, but obviously poor for 3D, comparing to CNNs, because of problems with depth estimation.

Convolutional neural network (and deep learning in general) is a black-box algorithm. During training millions of parameters are learnt; and it is impossible not to understand but even to perceive such complicated architecture. But some attempts were made, like visualization of the first layer weights [4], saliency maps [5], experiments with occlusion [6] and intermediate feature visualization [6, 7]. Still it is not

enough to fully understand how CNNs work. Nevertheless, this does not prevent convolutional neural networks from being so widely used in image processing tasks and in 3D hand pose estimation, in particular.

Now we will explain how 3D hand pose estimation task can be solved, because single RGB image processing is not the only option.

3D hand pose estimation refers to set of problems that estimate 3D locations of hand joints. Hand pose can be accurately represented by locations of 21 joints - 4 joints per finger and location of wrist (sometimes hand center or palm). 3D locations are estimated in absolute coordinates - in the camera coordinate system, or relative (normalized). Hand pose normalization can be performed in different ways, one approach is to put middle finger metacarpophalangeal (MCP) joint in the center of the coordinate system and make distance between wrist and middle finger MCP joint to be of length 1, as in [8].

Hand pose estimation is a challenging problem, because of huge variety of possible hand poses, severe self-occlusion, and nearly identical finger appearance.

3D hand pose estimation task can be approached from different perspectives, such as:

- wired gloves;
- color gloves;
- depth camera;
- RGB-D camera;
- stereo RGB cameras;
- mono RGB camera.

Engineering solution, that does not require any machine learning algorithms, is wired gloves. Dozens of different sensors are used to track global hand position as well as hand pose [9]. Wired gloves are accurate, however, expensive and cumbersome to use, so they did not become widely used.

Color gloves approach combines commodity (ordinary cloth glove) with machine learning algorithms. Glove is imprinted with color pattern which significantly simplifies hand pose estimation problem. Approach described in [10] estimates 3D hand pose from single RGB frame using nearest-neighbor algorithm. Input image is transformed into normalized tiny image and compared to the same tiny normalized images in the database using Hausdorff-like distance; after nearest-neighbor image is found, using 2D constraints associated with this image and inverse kinematics, 3D pose is estimated. Color gloves are less expensive, comparing to wired gloves, but still far from natural interaction with objects. Color gloves approach has potential when combined with deep learning techniques, because standardized gloves (in terms of color and size) will make it possible to produce accurate estimates both for normalized 3D hand pose and scale. Unfortunately, no such research was found.

Hand pose estimation from depth image is the most popular approach as of now. Hand pose is estimated from single depth image which is one-channel image where every pixel contains depth value. Depth images has all the required data for 3D hand pose estimation – both 2D joint locations on the image plane and depth values, so estimation from depth image is an accurate approach. Paper [11] contains comprehensive summary and comparison of existing approaches. Based on [11] the best result for single frame 3D pose estimation is shown by V2V-PoseNet [12], which was able to estimate pose with error somewhere between 6.2mm per frame (averaged among joints here and later) for seen visible hand poses (best case) and 14.6mm per frame

for unseen occluded hand poses (worst case) on average on BigHand2.2M and First-Person Hand Action datasets. V2V-PoseNet uses encoder-decoder architecture, that takes 3D voxelized depth map as an input and produces 3D heatmaps for each joint. This network needs a lot of computations, so cannot be used for tracking. Among approaches, suitable for tracking, [13] is the best one, which estimates 3D hand pose with 29.2mm error per frame on average. Approach [13] divides hand pose estimation problem into subtasks by joint type; for tracking, previous frame is used as a guide to predict the hand pose in the current frame. However, depth sensors are not applicable for outdoor environment, because they are very sensitive to active light sources; depth cameras require relatively high power consumption, so they cannot be used on mobile devices.

More recent approaches estimate 3D hand pose using RGB-D camera; these approaches use both RGB and depth data, so 3D estimations are assumed to be more accurate comparing to depth-only approaches. Algorithm described in recent paper [14] is able to estimate 3D hand pose with about 30mm error per frame on average on EgoDexter dataset; this algorithm can be used in real time for hand tracking. This algorithm uses two separate convolutional neural networks - for hand localization and for hand pose estimation (which estimates 2D heatmaps and 3D coordinate vector); then kinematic pose fitting is applied to improve normalized 3D hand pose prediction and produce absolute coordinates.

Stereo based hand pose estimation uses pair of images from cameras situated on some known distance from one another. Idea of the algorithm is based on the natural human ability to estimate distance using eyes. This approach is not as popular as depth-based approaches, because 3D hand pose estimation strongly rely on depth values; and with stereo approach depth is estimated from a pair of images and is noisy therefore. But with some recent researches, this method is proved to be almost as accurate as depth-based. The typical pipeline for stereo-based method is described in [15]; it starts with recovering disparity of image pair and performing hand segmentation; combining hand segmentation and disparity map, depth map of the hand is received; after that usual depth-based approaches are applied to estimate hand pose. Stereo-based approach contains more estimation steps than depth-based, so errors are higher. However, with accurate hand segmentation and disparity recovering, approach in [15] resulted in accuracy almost as high as for depth-based approaches – 20-50mm per frame. And with even more recent approach [16], error of only 30.8mm per frame was achieved. This approach is stable against errors in depth estimation, because of simultaneous depth and hand pose optimization given stereo image pair.

The last, least accurate and therefore least popular approach is 3D hand pose estimation from single RGB image. These methods are preferable as they can be used in many applications without any special setup required, including usage on mobile devices, outdoor usage and hand pose estimation from community videos (such as Youtube). However, this task is complicated because of scale and depth ambiguities. The most recent approaches are [8] and [17]. These algorithms estimate normalized 3D hand pose; and when transformed into absolute coordinates, prior assumptions are made. Framework in [8] achieved 50mm error on Dexter+Object dataset; such results were achieved by introducing projection layer to CNN architecture and extensive training data augmentation with synthetic images and images turned to 'real' by generative adversarial network. This framework is suitable for real-time tracking. Approach in [17] uses encoder-decoder CNN architecture with skip-connections and 2.5D heatmaps; with 2.5D heatmaps not only 2D keypoint locations are predicted, but also depth maps are estimated from image, which makes

3D recovery much easier than from usual 2D heatmaps. These improvements resulted in 35mm error per frame on Stereo dataset and Rendered hand pose dataset.

To be mentioned, human pose estimation is very similar problem and in the same time very popular research field. Hand pose estimation is more complicated problem because of more ambiguities, much stronger articulation and heavier self-occlusion, compared to human pose estimation. But still some general ideas and minor tricks may be reused for hand pose estimation.

We restrict our discussion to 3D hand pose estimation from single RGB, but also review depth, stereo, RGB-D hand pose estimation and human pose estimation approaches that can be successfully implemented for solving our task.

2.1 Approach

The pipeline for 3D hand pose estimation on videos from single RGB camera includes the following steps: hand localization, single frame hand pose estimation, and tracking. At the first step hand has to be localized in the image and cropped. At the second step from cropped hand image normalized 3D hand pose is estimated; and absolute 3D hand pose is derived from normalized. And finally, tracking techniques are applied to produce smooth hand pose estimation for consecutive image frames. Additionally, tracking techniques help correct errors occurred during single frame estimation, using information from previous and next frames. We will go through these steps in details.

2.1.1 Hand Localization

Hand localization is the first step in hand pose estimation and cannot be avoided. This step is crucial, as errors made during hand localization will be propagated through the whole hand pose estimation pipeline.

Assuming that hand pose is estimated from single image and there is no previous frame that may give us a clue where hand is located. The general idea is to train some CNN that outputs bounding box, or some other information, that makes it possible to crop original image and receive cropped hand image that will be used as an input for hand pose estimation network. In paper [18] hand localization is approached as segmentation problem, and they trained convolutional neural network to predict hand mask. Based on the segmentation mask, cropped hand image is received. In paper [14] hand localization is done using CNN and output is 2D heatmap where each entry encodes probability of the hand root. In work [14] knuckle of the middle finger was used as root. This approach was developed for RGB-D images, and uses depth information to calculate size of bounding box. So this technique cannot be applied for hand localization in RGB image, because hand may be of any size. One good idea, as in [17], is to use standardized object detector, YOLO detector [19] in particular, to get bounding box for the hand, and then crop image appropriately.

Right-left hand issue does not receive enough attention. In paper [18] this problem is not solved, but rather assumed that it is known whether is it right or left hand. Hand pose estimation network is trained for left hand, so right hand image is flipped before inputting in the network. Paper [8] simply ignores this problem and their approach works only for left hands. There may be two approaches to solve this issue.

First one is to use data augmentations (horizontal flipping) and train hand pose estimation network to work for both hands. Second one is to add left/right hand classifier – introduce separate CNN for classification or combine left/right hand classification with hand localization to output both bounding box and 1/0 whether hand is left/right. In this case, hand pose estimation network should be trained only for left hand, for instance, and all the cropped images of right hand have to be flipped horizontally before inputting into the network.

2.1.2 Normalized 3D Hand Pose Estimation

At the beginning of this step we have cropped hand image, which is an input for all approaches described below. Additionally, there are 2D locations for N keypoints (xy coordinates for each keypoint), cropped appropriately to the hand image, and normalized 3D locations for N keypoints (xyz coordinates for each keypoint). Our task is to estimate normalized 3D keypoint locations. For neural network it is easier to predict normalized coordinated as they have less unknowns – no distance from camera to hand and no actual hand size. As was mentioned earlier, normalization can be performed by putting middle finger metacarpophalangeal (MCP) joint in the center of the coordinate system and making distance between wrist and middle finger MCP joint to be of length 1 as in [8].

There exist two general ideas for 3D hand pose estimation. One idea is to estimate 3D keypoints directly from neural network, another one – to estimate 2D keypoints first and then using optimization algorithms (or separate network) calculate 3D keypoints.

Pipeline explained in [18] includes two separate consecutive networks for hand pose estimation. First network estimated N heatmaps for 2D locations from cropped hand image. Second network has two parallel streams – separately for 3D hand pose estimation in canonical frame and separately for viewpoint relative to this coordinate system; both streams use 2D heatmaps as an input. Canonical coordinates are calculated from normalized coordinates using rotation 3x3 matrix. After transformation all hands are aligned in the same way, for instance, all hand poses are from egocentric viewpoint and palms are up. Transformation to canonical frame makes hand pose estimation even easier than that for normalized coordinates, as viewpoint (another degree of freedom) is fixed. Canonical coordinates are estimated as vector applying L2 loss, viewpoint is estimated as 3x3 rotation matrix (from normalized to canonical) applying L2 loss as well. To calculate normalized coordinates, canonical coordinates and rotation matrix are multiplied.

Approach in [17] includes two-stages as well, but with some modifications; as of now, it is one of state-of-the-art approaches. On the first stage, CNN with encoder-decoder architecture is trained to simultaneously produce N latent heatmaps and N latent depth maps from cropped hand image. Latent heatmaps encode 2D locations for each keypoint and latent depth maps – scale normalized and root-relative depth values for each keypoint; 'latent' means that model learns appropriate spread representation for a heatmap, but not optimizes for hand-crafted Gaussian distribution. Loss is constructed to be fully differentiable and consists of 2D location L2 loss and depth L2 loss. Differentiability is ensured using trick – latent heatmaps are converted to probability maps applying softmax and actual locations are calculated as weighted average of 2D pixel coordinates; and normalized depth values are calculated by summing dot products of latent depth maps and probability maps. 3D hand pose reconstruction is much easier from both depth values and 2D locations

that are available for all keypoints. Approach does not recover scaled and root-normalized hand pose, and directly jumps into recovering absolute 3D pose from 2.5D heatmaps; this idea will be explained in the next section.

Another state of the art approach, described in [8], estimates normalized 3D hand pose directly from cropped hand image. Also network is trained to outputs 2D heatmaps. Authors proposed to use ResNet architecture with additional convolutional and projection modules. Projection module is used to better coalesce 2D and 3D predictions; ResNet part outputs vector of intermediate 3D positions from which intermediate 2D heatmaps are created performing orthographic projection. Convolution module is applied to produce final 2D heatmaps and vector of 3D locations from intermediate 2D heatmaps. Loss that is used to train this network is sum of L2 loss for intermediate 3D estimates, L2 loss for final 2D heatmaps and L2 loss for final 3D estimates.

Second main difference among reviewed approaches is how to encode keypoint locations – as heatmaps or as numeric vectors. Convolutional networks are invented to work well with images, they find features and understand their spatial relations; so it is more natural for convolutional networks to produce heatmaps.

Heatmap representation is widely used for 2D keypoint locations. Heatmap is a one-channel image, the same size as hand image, where each pixel represents some kind of "probability" that keypoint is present here. Ground truth heatmaps are constructed by putting 1 in the position of actual keypoint location and 0 otherwise, and applying Gaussian blur to prevent neural network from overfitting. Softmax function may be applied to original heatmap to produce new heatmap where each pixel is true probability and all pixel values summed up to one. When recovering 2D location – x and y coordinates – we take position of the pixel with maximum value; or calculate average for x and y coordinates separately, weighting all x and y coordinates by heatmap pixel values. Here regression loss (L1 or L2) or intersection over union loss may be used.

However, for 3D locations vector representations are usually used, because 3D heatmaps are hard to construct and require a lot of computations. Vector is produced by concatenation all xyz coordinates for N keypoints and has 3N dimensions. Regression loss (L1 or L2) is used when training network.

There are some other interesting ideas of how to represent locations when training network, which may be used for 3D hand pose. Vector representation minimizes prediction error separately for each joint, and internal structure of the pose is ignored. Techniques below (bone representation and volumetric representations) is assumed to have higher performance over direct regression.

Approach from paper [20] uses regression representation with simple reparametrization, so pose structure is considered. Now bone locations are predicted, but not joints; bones are calculated as difference between two nearest joints. Bone locations are more stable than joint locations; and geometric constraints are expressed easier. Loss is calculated as sum of L1 losses for all bones. Authors propose to use all possible pairs of joints as "bones" so whole pose structure is exploited. This approach was not used in reviewed papers on hand pose estimation; but it has a potential in 3D hand pose estimation, when 3D locations are recovered directly from image.

Approach [21] uses volumetric representation for 3D pose. Such representation considers spatial relations among keypoints, the same as 2D heatmaps do, and more "natural" for convolutional networks to work with. Each joint location is represented as discretized volume of size $w \times h \times d$; there are N such volumes in total. Ground truth values are constructed by applying 3D Gaussian blur, the same as for 2D heatmaps. Network architecture consists of consecutive encoder-decoder

block, each of them inputs image concatenated with previous block output and produces 3D voxel. To decrease computations, authors propose to output voxel of size $w \times d \times 1$ from first block and gradually increase depth resolution, so that final block outputs voxel of size $w \times h \times d$. This approach is unlikely to be applied to real time tracking as it requires a lot of computations.

2.1.3 Restoring Absolute 3D Hand Pose

Absolute 3D pose recovery is a hard task, because of both depth and scale ambiguities; and scale and depth are interconnected parameters. It is impossible to recover scale and depth from a single image: we cannot be sure whether is it a small object just in front of the camera or a large object far away from camera. So most of restoring techniques are highly dependant on prior assumptions.

In paper [8] absolute 3D hand pose is restored from normalized hand pose using Kinematic hand model fitting. The idea of such approach is to introduce several additional constraints that simultaneously make hand pose more natural (such as bone and angle constraints) and restore absolute 3D pose (such as correspondence between absolute 3D pose and its 2D projection, and user-specific characteristics). Hand model fitting is represented by a function that sums up all the discrepancies from constraints; optimization algorithms are used to find the optimal value.

Approach in [17] restores scale-normalized absolute 3D pose from 2.5D heatmaps under assumption of perspective projection and that intrinsic camera parameters are known. It uses perspective projection equation that interrelated 2D and 3D locations and depth of root joint, recovered from 2.5D heatmaps. To denormalize scale-normalized 3D pose and recover absolute 3D pose, scale prior assumptions are introduced.

To be mentioned, when hand image is cropped from original image, additional adjustments for absolute 3D hand pose have to be made, because, when cropping, absolute keypoint positions in camera coordinate system changes.

2.1.4 Tracking

Brute-force approach for developing tracking system is to combine Hand Localization, Normalized 3D Pose Estimation and Absolute 3D Pose Restoring steps into a single pipeline and run them in series for each frame in a sequence. However, such approach will two main disadvantages. It will require a lot of computations so it cannot be applied for a real-time tracking; and there will be a significant jitter, because predictions for consecutive image frames may differ.

Smartly constructed tracking techniques will not only deal with above mentioned disadvantages, but also are assumed to increase accuracy for a single frame estimation by using information from previous frames.

Hand localization step is simplified, because there is no need to look for hand location on the whole image; we know where hand was on the previous image frame and we may assume that hand position is somewhere nearby on the current frame. This idea was used in [17], where authors propose to generate bounding box from previous frame estimated hand pose. This approach works well if hand location is quite stable between consecutive image frames. During fast hand motions, Faster RCNN [22] may be used for re-initialization, as explained in one of the approaches in [11]. Probably, here can be used classic tracking techniques, such as Kernelized Correlation Filter [23]; but papers where it was implemented for hand tracking were not found.

3D Hand Pose Estimation from cropped hand image is going to be rerun for each frame. To prevent jitter and make tracking more stable to errors, authors in [8] and in [14] propose to use temporal smoothing that constraints magnitude of location changes that each keypoint can perform between consecutive frames.

Interesting idea described in paper [24] can be used for tracking either. Authors propose a brand new approach for human pose estimation from a single frame. They trained a convolutional neural networks that inputs image and pose estimated on the previous step, and outputs corrections that have to be made for pose. Initial pose on the first iteration is set to an average pose in the train dataset; magnitude of corrections is limited above. This idea can be reused for hand pose tracking: neural network may be trained in such way, that it takes as an input previous frame hand pose and current frame hand image and predicts corrections to be made from previous to current frame hand pose. This approach both uses information from previous frames and prevents jitter.

2.2 Datasets

This section contains description of those datasets that can be used for 3D hand pose estimation from RGB image. Perfect dataset should be large-scale dataset that contains different people and backgrounds and has accurate labeling for 21 keypoints per hand. Unfortunately, no such dataset exists, so 3D hand pose estimation problem includes not only algorithmical solution, but also approaches to train convolutional networks with datasets available. Table 2.1 contains summary for all 7 existing datasets; summary is similar to those in [25]; we will refer to this table several times in this section.

There no real large scale dataset exists, and there is a reason for that. 2D labeling is expensive, because human annotation is required, but at least it is feasible. Human annotator can label images with high accuracy for each of 21 joints when they are visible. Problem arises with 3D keypoint locations. Human are not able to accurately estimate neither absolute distance to camera, nor normalized hand pose from image only; so complex multi-camera setups or semi-automated methods, like one described in [26], are needed. Approach in paper [26] works for sequence of depth images the following way: algorithmically most appropriate frames are selected; human annotator is asked to annotate 2D keypoints and their relative depth (closer/farther) for only these frames; optimization problem is solved to find 3D hand poses. This approach can be used for RGB data annotations - by producing pairs of depth and RGB images from calibrated cameras; and estimating 3D annotations from depth images; 3D annotations for RGB images can be calculated knowing relative positions of depth and RGB cameras. However, automated annotating leads to an error, which is propagated while training network.

There exist only 4 real datasets and none of them can be used for training convolutional neural network from scratch (refer to Table 2.1 for details). The largest dataset, Stereo Tracking Benchmark [15], contains 18k images. These datasets are video sequences, so a lot of poses are similar; there are only several people were filmed and there are only several background available; this means that there is a lack of variety in these datasets. Another problem is with annotations. EgoDexter [14] and Dexter+Object [27] have only 5 keypoints annotated - fingertips; Dexter1 [28] provides 6 keypoints - 5 fingertips + palm center; and only Stereo Tracking Benchmark [15] has 21 keypoints, however, last keypoint is a palm center, which is inconsistent with synthetic datasets where last keypoint is a wrist.

To train neural network when large-scale datasets are absent, authors in [18] propose to pretrain network on large scale synthetic dataset and fine-tune it using real dataset. There exist 3 synthetic large scale datasets. SynthHands [14] was generated using Unity Game Engine; Rendered Hand Pose Dataset [18] was created using Mixamo (where 3D human models are available) and Blender (open source software for rendering images); GANerated dataset [8] was created by changing images from SynthHand to make them more realistic using Generative Adversarial Network. These datasets provide accurate 2D and 3D annotations for all 21 keypoints; while Rendered Hand Pose Dataset [18] additionally has hand segmentation mask, left/right hand labeling and occlusion tickets. Refer to Table 2.1 for more details.

TABLE 2.1: Existing datasets for Hand pose estimation.

Dataset Name	**Real or Synthetic**	**Dataset Size**	**Number of Keypoints**	**View point**	**Comment**
EgoDexter [14]	R	1.5k annotated, 1.7k with no annotations	5 fingertips	ego	4 sequences, 4 actors, interaction with objects, absolute 3d locations, camera data provided
Dexter+Object [27]	R	3.1k	5 fingertips	3rd	6 sequences, 2 actors, interaction with objects, absolute 3d locations, camera data provided
Dexter1 [28]	R	3.2k * 5 cameras	6 (5 fingertips + palm center)	3rd	1 actor, 1 background, absolute 3d locations, camera data provided
Stereo Tracking Benchmark [15]	R	18k	21 (last keypoints is palm center)	3rd	1 actor, 6 backgrounds, absolute 3d locations, camera data provided
SynthHands [14]	S	63.5k	21 (last keypoint is wrist)	ego	large variety of hand types, background augmentation supported, absolute 3d locations, camera data provided
Rendered Hand Pose Dataset [18]	S	41k training / 2.7k testing	42 (21 for each hand, last keypoint is wrist)	3rd	20 characters, both hands on the image, segmentation mask and occlusion ticket provided, absolute 3d locations, camera data provided
GANerated dataset [8]	S	330k	21 (las keypoint is wrist)	ego	large variety of hand types, only normalized 3D locations, no camera calibration and no denormalization coefficient provided

However, there is still a large domain gap between synthetic and real data, so model trained on synthetic data does not generalize well to real data. One possible solution for this problem is domain adaptation techniques. Method explained in [29] uses very simple idea that can be reused for almost any sequential architecture of neural network. Two datasets are needed: source domain dataset with correct labels (keypoint locations in our case) - synthetic dataset; and target domain dataset with no labels - unlabeled real dataset. Model architecture consists of two networks that have the same feature extractor (weights are shared); first network is being optimized to estimate keypoint locations for synthetic data; second network is being trained to distinguish synthetic data from real. Whole network is trained to minimize loss for keypoint locations for synthetic data; and simultaneously trained so that feature produced by feature extractor are the same for synthetic and real data, so network is unable to distinguish between real and synthetic data, so domain classification loss is maximized. Such approach decreases domain gap, by aligning features generated by feature extractor for synthetic and real data. However, it is unknown how this method will work for hand pose estimation, as no paper on hand pose estimation from reviewed ones used this idea.

This chapter covered state of the art approaches for 3D hand pose estimation from single RGB image, their current results and future challenges. Additionally, we outlined several interesting ideas from other fields that can be reused for our task. We provided comprehensive description of the existing datasets and gave examples of how data-related problems may be solved.

Chapter 3

Approach and Implementation Details

This chapter contains detailed description of proposed solution for 3D hand pose estimation, which includes data processing techniques, implementation details of several convolutional network architectures and losses, and data augmentation techniques. We developed several model architectures, that combine state of the art techniques with interesting and promising ideas, to select the best model.

Proposed solution does not contain whole hand pose estimation pipeline - hand localization, left-right hand classification, normalized 3D hand pose estimation, absolute 3D pose hand pose restoration and tracking. We focus our attention only on normalized 3D hand pose estimation from RGB hand image. Hand localization and left-right hand classification are already solved tasks in Computer Vision, restoring absolute 3D pose requires introduction of prior assumptions and cannot be restored with high accuracy any way, and tracking is more algorithmic task. Normalized 3D hand pose estimation is the main part of pipeline and the least researched topic in the same time; pipeline performance highly depends on the performance of normalized 3D hand pose estimation, so improving 3D hand pose estimation will result in boost of 3D hand pose estimation from RGB image in general.

Proposed pipeline for Normalized 3D hand pose estimation is shown in Figure 3.1. Cropped image of the hand (left hand) is inputted into encoder-decoder convolutional neural network to produce both 2D heatmaps and intermediate image features. Intermediate features and 2D heatmaps are concatenated, and after more convolutions applied, output normalized 3D hand pose. There are two losses used to train model - loss for 2D pose and loss for 3D pose.

3.1 Data Preprocessing and Preparation

To train a model, we need dataset that contains hand images with 2D and 3D coordinates. Hand images as well as keypoint coordinates have to be appropriately prepared before using for training and prediction.

The best option is to crop hand image as much as possible, so model will not overfit to background. If there are 21 keypoints provided, we may crop image based on them. As we need squared crop, we calculated pixel distance between most left and most right keypoints and distance between lowest and highest keypoints; we select maximum of them - that will be size of the crop; we add small intent in several pixels to make sure keypoints are visible. Unfortunately, for most real datasets only 5 keypoints are provided (see section 2.2), which means cropping is impossible.

To prevent model from overfitting to background we apply background augmentation when it is possible. For instance, SynthHands datasets [14] contains green

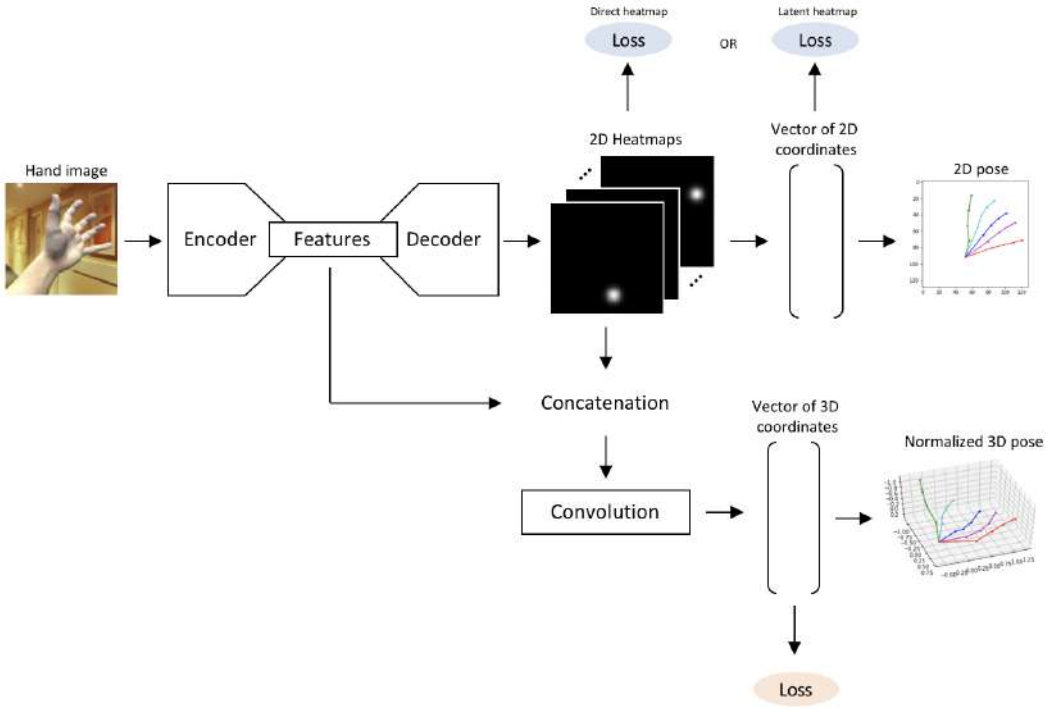

FIGURE 3.1: Pipeline for Normalized 3D Hand Pose Estimation.

background, so when forming a batch, we randomly select background image from ADE20K dataset [30] for each sample; the same hand images will have another background when taken second time. To perform background augmentation, for each hand image mask M is created; it is an array of the same size as hand image, when each entry contains 1 if this pixel belongs to hand and 0 otherwise. We crop / resize background image to be the same size as hand image. We convert images and mask to arrays and apply the following formula to created augmented image (multiplication is element wise here):

$$I_{augmented} = I_{original} * M + I_{background} * (1 - M)$$

After background multiplication is performed, new hand image is center cropped, so its height and width are of the same size; and then resized to 128x128. For hand pose estimation task such small image size is just fine, because hand and its pose in this resolution is still human recognizable. And finally, image is color normalized by subtracting channel means and dividing by channel standard deviations. Because some of the model architectures are based on pyTorch models pretrained on ImageNet, we use the same normalization coefficients:

$$mean = [0.485, 0.456, 0.406],\ std = [0.229, 0.224, 0.225]$$

When encoding keypoint locations as vectors it is crucially important to preserve the same order and to use the same keypoints among different datasets. Keypoints and their order that will be used in this work is the same as for SynthHands [14] and GANerated [8] datasets. We will use 21 keypoints - 4 for each finger and wrist joint; keypoints and their order are shown in the Figure 3.2. We assume that all keypoint are present in the image; it is a strict assumption that forces algorithm to output

locations somewhere in the image frame for all 21 keypoints, even for those that are outside image frame.

FIGURE 3.2: Keypoint Map.

2D keypoint locations are predicted using heatmaps, so vector of 2D keypoint coordinates is converted to heatmap set. Each keypoint location that initially was represented as (x,y) coordinate is transformed to separate heatmap. Ar first, after image cropped and resized, (x,y) coordinates are adjusted appropriately. To create heatmap, zero array of size 128×128 is created (the same as final hand size); value 1 is placed in the position of keypoint - its (x,y) coordinate. To prevent model from overfitting, heatmap is blurred using 2D Gaussian kernel with size 61 pixels and standard deviation of 3 pixels. Actual Gaussian kernel parameters do not matter, the only rule is not to make blurred keypoint neither too big nor too small. After applying blurring, heatmap has to be normalized, using formula below, so actual keypoint location is equal to 1 again:

$$H_{final} = H_{blurred} / max(H_{blurred})$$

There are 21 heatmaps in total; heatmap values are in the range [0,1]. Heatmap visualization is in Figure 3.3.

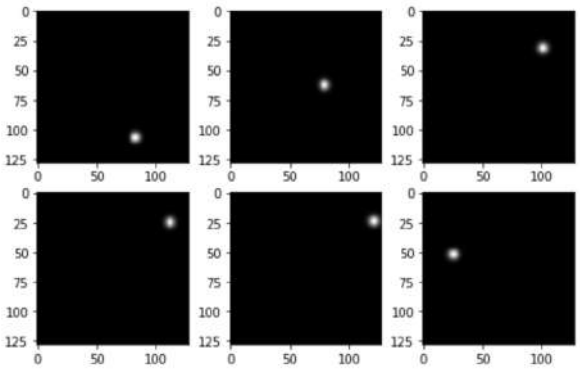

FIGURE 3.3: Heatmaps for a Random Sample.

Additionally, vector of 2D coordinates is returned; this representation will be used to train a model that outputs latent heatmaps. 2D pixel coordinates are normalized to be in range [0,1] by dividing by image size of 128; and flattened into a vector of size 2×21. Such normalization is needed to make loss scale independent from input image size, and to shift 2D loss closer to 3D loss in scale.

If sample does not have all 21 required keypoints, for those keypoints that are absent, zero heatmaps and zero vector entries will be imputed; keypoint order is preserved.

3D coordinates are predicted in normalized representation, because such representation has less degrees of freedom, so it is easier for neural network to generalize. Normalization is performed the same way as in [8]: middle finger metacarpophalangeal (MCP) joint (marked red in Figure 3.2) shifted to origin and distance between wrist and middle finger MCP set to 1. Normalization is performed in this way because of two reason: middle finger MCP position and distance between wrist and middle finger MCP are assumed to be the most stable in datasets; GANerated dataset [8] uses such normalization and provides no absolute 3D coordinates, so normalization cannot be performed any other way; and it is large scale dataset we are going to use for model training. 3D coordinates and returned as a vector of size 3×21.

And finally, we need an indicator whether keypoint is labeled or not. We are going to train model using real data as well, but real datasets do not contain all the 21 keypoints we are going to use. To make it possible to train a model, we for each sample we will create boolean vector of size 21 that has entry 1 if keypoint location is provided and 0 otherwise. Order of the keypoints is the same as in Figure 3.2.

3.2 Model Architectures

We are going to implement several approaches for predicting hand pose, and then test which one of them shows the best result. Approaches differ in two key area: 2D keypoints coordinates to predict through direct heatmaps or latent differentiable heatmaps; and 3D keypoints coordinates to predict through simple vector of size 3×21 or using tree representation of the hand. Approaches are built upon the same Encoder-Decoder convolutional neural network, where encoder part is pretrained ResNet feature extractor.

3.2.1 ResNet-based Encoder-Decoder network

For hand pose estimation we use encoder-decoder architecture, because such architectures are able to restore original image size, and that is exactly what we need as network for hand pose estimation has to return 2D heatmaps that are of the same size as input image - 128×128.

Among encoder-decoder architectures UNet [31] is very popular. It is widely used for segmentation tasks; because it is able to transform image to some feature space, process these features and then restore original image back from this feature space, so spatial relation between features preserved. The main feature of UNet is skip connections: when image is decoded from feature space, intermediate features from earlier convolutions are used to make such decoding precise. We use similar skip connections while constructing encoder-decoder architecture for hand pose estimation.

We use ResNet34 [32] feature extractor pretrained on ImageNet as an encoder in proposed architecture. It is always faster to adjust existing network for a new task than learn weights from scratch. It is also assumed that pretrained networks generalize better, because they 'saw' more data when trained on large scale datasets. We

selected ResNet34 because combination 'number of parameters - accuracy on Imagenet' is optimal and avalible in PyTorch deep learning library [33]. Feature extractor is created from ResNet by removing global average pooling layer and fully connected layer from the network. Detailed architecture of feature extractor is shown in Figure 3.4. When image is passed through feature extractor, we save intermediate feature maps, because they will be used for decoder part.

Decoder part consists of four Upsampling Blocks and 2D Pose Regressor that outputs 2D heatmaps. Figure 3.4 show architecture with direct heatmaps, architecture with latent heatmap is implemented with slight modifications explained in next section. Each Upsampling block doubles size of inputted feature map. Upsampling block inputs feature map of depth $f1$ from the previous step, upsamples it, concatenates with intermediate feature map of depth $f2$ from feature extractor, applies convolution and outputs new feature map of depth d that is twice large as initial (see detailed Upsampling Block architecture in Figure 3.5a).

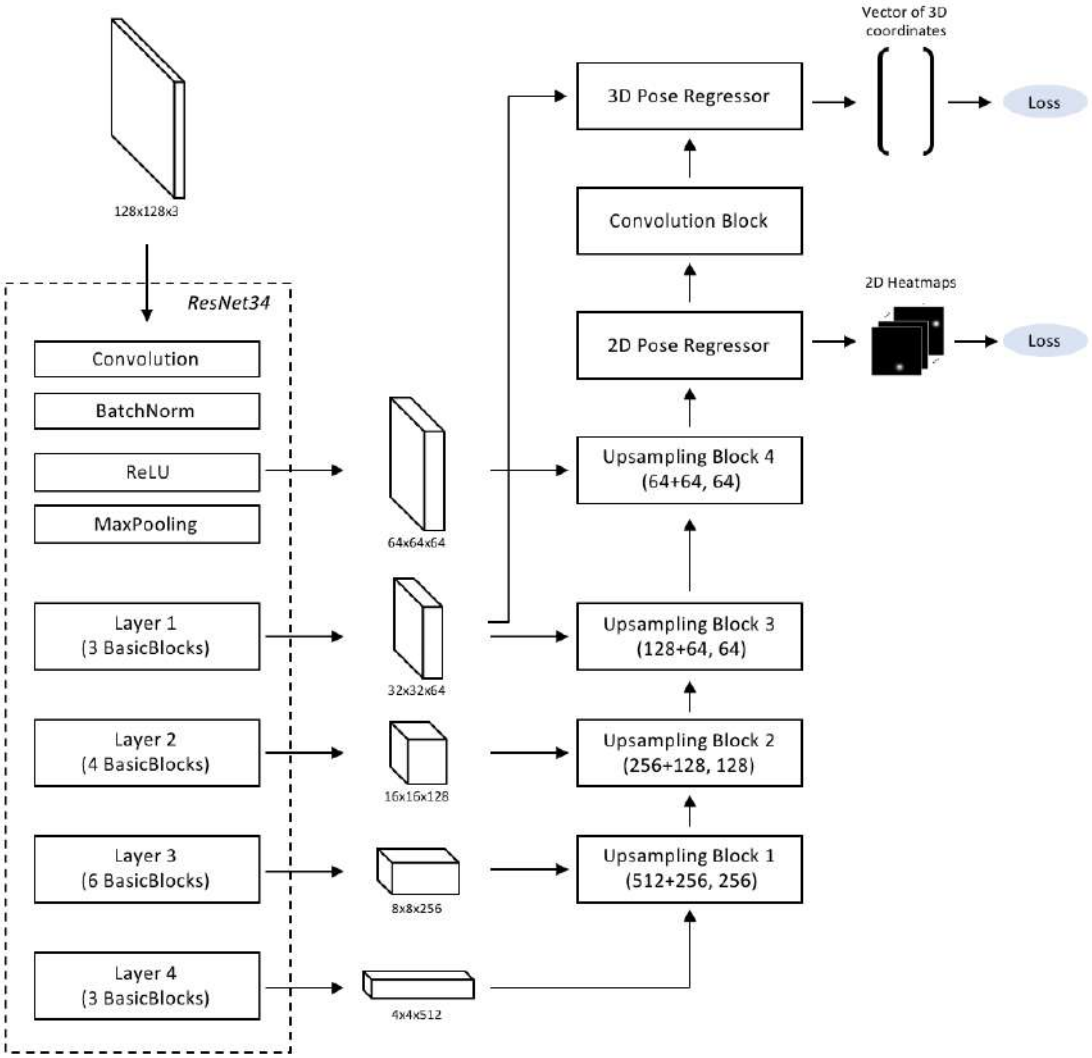

FIGURE 3.4: Encoder-Decoder Architecture.

Output of forth Upsampling block is feature map of size 64. This feature map is used as an input to 2D Pose Regressor, where feature map is upsampled and convolved; see Figure 3.5b for more details. 2D Pose Regressor outputs 2D heatmaps of size 128×128 - a heatmap for each keypoint.

Heatmaps are going to be used for predicting normalized 3D hand pose. Heatmaps are processed using Convolution Block (see its architecture in Figure 3.5c) and result is used as an input to 3D Pose Regressor. 3D Pose Regressor concatenates processed heatmaps and intermediate feature map from feature extractor and applied convolutions; final features are flattened and fully connected layers are used; output is vector of size 3×21. Detailed architecture for 3D Pose Regressor is shown in Figure 3.5d.

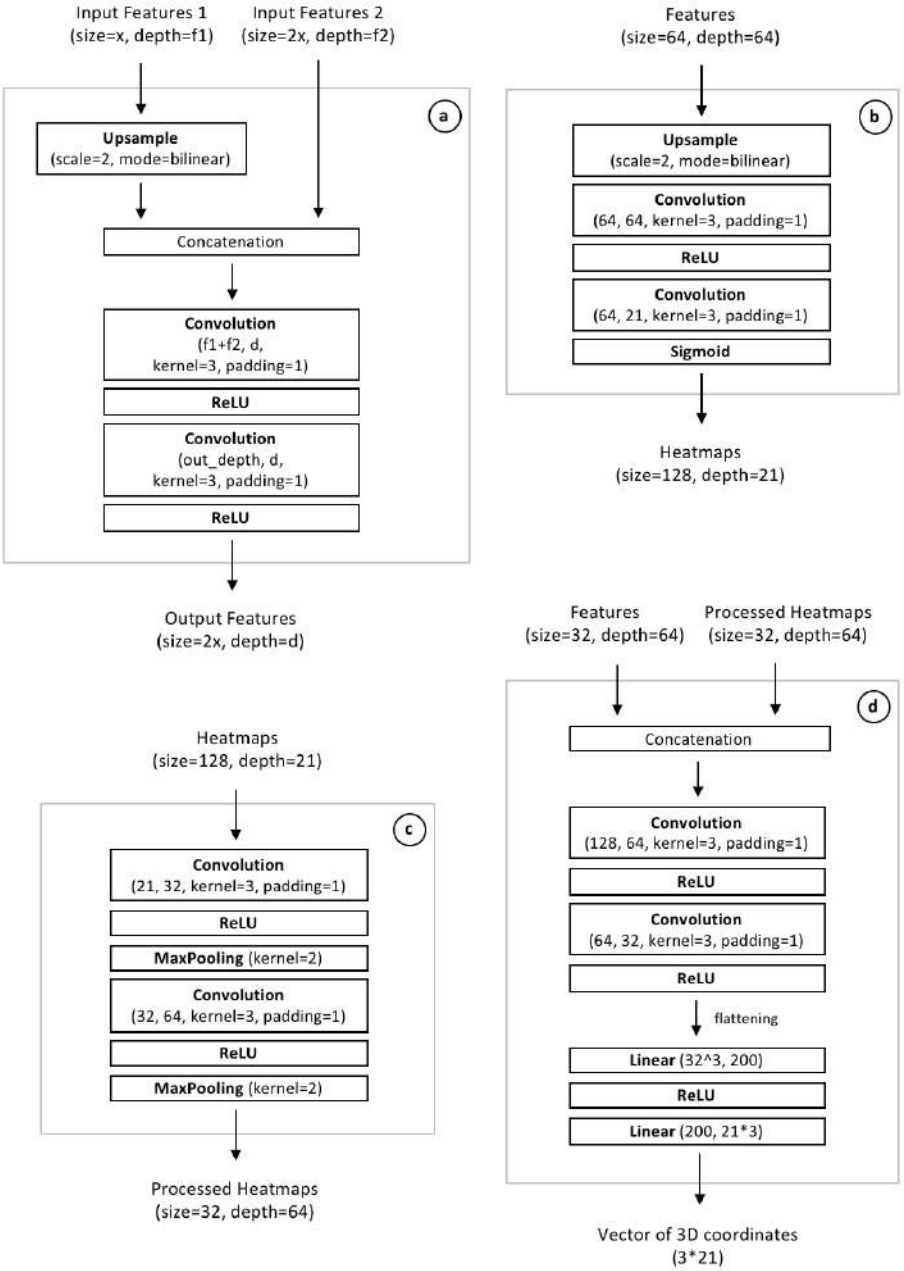

FIGURE 3.5: Block Architecture: a) Upsampling Block; b) 2D Pose Regressor; c) Convolution Block; and d) 3D Pose Regressor

Model is trained using two losses: for heatmaps and for vector of 3D coordinates (see section 3.3 for more details). Explained architecture predicts 2D pose using direct heatmaps and 3D pose using simple vector of coordinates. To predict 2D hand pose using latent heatmaps or 3D pose using tree representation, some modifications have to be introduced; modifications are explained in sections 3.2.2 and 3.2.3.

3.2.2 Direct vs Latent Heatmaps

2D keypoints are usually predicted using direct heatmap regression [8, 14, 18]. For each hand image, convolution neural network outputs heatmap set - tensor of size (N $\times$ image size $\times$ image size), where N - number of keypoints. Each entry of the heatmap array is in the range [0,1], because sigmoid function was applied to the heatmap. Network output is the same as ground truth heatmaps, created at the data preparation step (see section 3.1).

Then estimated heatmaps are converted to vector of 2D coordinates. There are two approaches to do it:

- Find (x,y) coordinates of the heatmap pixel that has the highest value. This is the simplest approach, but is not accurate, because it is sensitive to outliers.
- Normalize heatmap, so all heatmap values sum up to 1. This can be done using softmax function, but when values are in the range [0,1] softmax does not work, because it requires larger scale to distinguish difference between values; so we should divide each heatmap entry by the sum of all heatmap entries. Then weighted average separately for x and y coordinates is calculated, where weights are values in the normalized heatmap and x and y coordinates are in the range [0, image size). This approach is more robust, so we are going to use it.

Another option for 2D keypoints estimation is through latent heatmaps. This approach was used in paper [17]. Authors are motivated by the fact, that it is unknown what size of the kernel should be used when creating ground truth heatmaps, whether kernel size is the same for all keypoints (for sure, not) and whether kernel should be circular (for sure, not). Latent heatmaps approach lets neural network estimate heatmaps by itself. Neural network outputs vector of 2D coordinates; conversion from heatmaps to (x,y) coordinates is produced inside the network. Conversion is done almost the same way as we do it for simple heatmaps - normalizing and averaging all coordinates using normalized weights; but except sigmoid and then division by sum of heatmaps values, we use 2D softmax function for latent heatmap approach. Obviously, another big advantage of this approach is that ground truth heatmaps are not needed to be created at all.

In both cases predicted 2D keypoint coordinates are normalized to be in range [0,1] - this is done by dividing pixel coordinates by image size. As a result, predicted coordinates are the same as ground truth, that were mentioned in section 3.1.

3.2.3 Simple vs Tree-based Vector Representation for 3D Pose

The most popular method for 3D hand pose estimation (when 3D pose is estimated directly from image) is simply vector representation. There are N keypoints, each has (x,y,z) coordinates, so vector of coordinates will be of size $3 \times N$, or 3×21as in our case. Fully connected layers are used in the model architecture to output vector

of needed size. Output vector already contains normalized 3D keyoint locations so no additional processing needed.

However, this approach has a big disadvantage, each keypoint (even each keypoint coordinate) is predicted separately and independently from other keypoints. To deal with this problem, authors in [20] propose to consider human (or hand as in our case) as a tree and predict rather bones than joints, because bones are more stable. Bones are represented as geometric vectors of (x,y,z) coordinates and calculated as difference between coordinates of joints that form this bone according to selected tree structure. When coordinate of particular joint is calculated, all bones on the path are summed up. Additionally, such approach exploits geometric structure of a human/hand; simple vector representation does not do that.

3D Pose Regressor (see Figure 3.5d) is modified the way, that it outputs vector of size 20×3 (number of bones), but not of size 21×3 (number of keypoints). Figure 3.6 shows 20 bones that are outputted by model; they are represented as red geometric vectors and labeled with numbers. Bone representation is converted to keypoint representation; keypoint representation is simply vector of coordinates. We can apply regression loss to keypoint representation, because transition from bone representation to keypoint representation is fully differentiable.

Keypoint locations in the normalized 3D pose can be considered as geometric vectors, pointing from the origin; origin is middle finger MCP (origin is keypoint 10 in Figure 3.6; and keypoint locations are shown as black geometric vector). Transition from bone to vector representation is done in the following way:

- Middle finger MCP joint has location of (0,0,0).
- To calculate location of some keypoint, all vectors are summed up that along the path from middle finger MCP joint to this keypoint. For instance, location of keypoint 4 is calculated as (-bone 9 + bone 1 + bone 2 + bone 3). Bone 9 is subtracted, because we are moving in opposite direction. See Figure 3.6 to follow explanation.

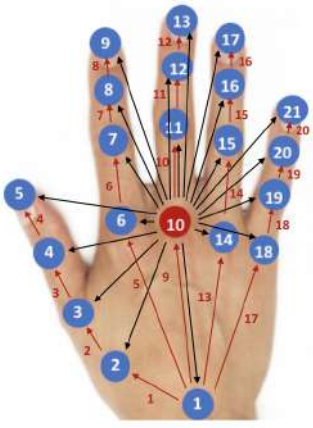

FIGURE 3.6: Tree-based Hand Representation.

3.3 Losses

There are two losses that will be used for training neural network: mean squared error loss for vectors of coordinates (when predicting 2D pose through latent heatmaps, for 3D pose estimation both thought simple vector and 'tree' vector) and intersection over union loss for heatmaps (direct heatmaps).

Mean squared error (MSE) loss is assumed to be best fit for regression tasks. One of the state of the art approaches for 3D hand pose estimation [8] uses it for both heatmaps and vectors of 3D coordinates. However, we will use MSE loss only for vector of 3D coordinates and vector of 2D coordinates, when 2D coordinates are predicted through latent heatmaps. MSE loss is calculated according to the formula:

$$L_{MSE} = \frac{1}{n}\sum_{i}(y_i - t_i)^2$$

As was explained in section 3.1, 2D coordinates are represented as vector of dimension 2×21 and 3D coordinates - as vector of dimension 3×21. So here n is a number of entries in vector of coordinates, y - predicted value of vector entry, and t - true value of vector entry. MSE loss for an image may be considered as "average" of all keypoint losses, however, it is not true average in mathematical sense.

Intersection over union loss (IoU) was not used in reviewed papers on hand pose estimation. This measure is widely used in object detection and segmentation problems, where predictions are represented as regions - as bounding boxes or set of pixels. IoU is defined as intersection of ground truth and prediction regions divided by their union. When regions are perfectly coincide, IoU equals to 1; when there is no intersection at all, IoU is 0; and when region intersection is close to region union, prediction is good. IoU measure will be used when optimizing heatmap predictions. We will calculate this measure using formulas from [34] with slight modifications for heatmaps; here yi and ti are predicted and target values for pixel in a heatmap:

$$IoU = \frac{I}{U}$$

$$I = \sum_{i}(y_i * t_i)$$

$$U = \sum_{i}(y_i * y_i) + \sum_{i}(y_i * t_i) - \sum_{i}(y_i * t_i)$$

IoU measure means the higher the better, so IoU loss is calculated the following way:

$$L_{IoU} = 1 - IoU$$

IoU loss for an image is calculated as an average for all keypoint heatmap losses.

Architectures described in the previous section have several losses outputted - for 2D, for 3D and some intermediate losses if any. Final loss that is used for training model is a weighted sum of all losses. Weighting is needed when loss scales are different, or one loss is more important than another. It is crucial, because those losses that are larger in scale affects final loss more, so during training all model weights will be optimized to decrease larger loss. This may lead to imbalanced training, when for instance, 3D coordinates are predicted correctly, but 2D coordinates predictions are far from ground truth.

Weighting coefficients are needed to equalize contribution of various losses to final loss. There are several ways to set up these coefficients:

- Those losses that are more important should receive higher weights, even if the scales are initially equal. This is usually done with intermediate (auxiliary) losses. Losses that outputted early in the model, should have lower weights. This idea was used when training GoogLeNet [35].
- When losses are equally important, their scales have to be equal as well. The easiest way to do it is to set weights based on initial gap in scales and programmatically or manually adjust weights when gap changes during training. That is how loss weightning was implemented in our model.

3.4 Data Augmentation

Data augmentation is a widely used technique to prevent overfitting. The idea of the data augmentation is to create additional data samples from existing ones and use this data samples to train a neural network. This techniques is used since the very beginning of the neural network training and was used to train AlexNet [1]. Collecting and labeling dataset is expensive and for some fields (including 3D hand pose estimation) even complicated. So we are going to use data augmentation: 1) to increase size of synthetic datasets even more, and 2) to increase size of small real datasets and make learning on real data possible.

For both synthetic datasets - SythHands [14] and GANerated [8] datasets - the following data augmentations are applied for each image:

- brightness is randomly changed by some percentage in the interval $[-25\%; 25\%]$;
- contrast is randomly changed by some percentage in the interval $[-25\%; 25\%]$;
- saturation is randomly changed by some percentage in the interval $[-25\%; 25\%]$;
- image is blurred using Gaussian blur with some random radius in range $[0, 0.8]$.

All these transformations are applied to each image in the same order. Gaussian blur is applied to already resized image to size 128×128. These augmentations only change hand image and leave 2D and 3D labels unchanged.

Additionally, we apply scaling and rotation to SynthHand dataset. Scaling changes both image and 2D pose; while rotation changes image, 2D and 3D pose. After applying scaling and rotation, we should be able to fix blank spaces that are created by transformation, so image should have some hand-background mask. To perform rotation of 3D pose, absolute coordinates of keypoints in camera system should be available. SynthHands dataset includes both hand-background masks and absolute 3D keypoint locations, but GANerated dataset does not have neither masks, nor absolute 3D locations (only normalized), that is the reason, why we perform scaling and rotation augmentation only for SynthHands dataset.

When performing scaling transformation, image is scaled by some random percentage in range $[-25\%, 15\%]$. Upscaling range is smaller than downscaling, because we do not want keypoints to be outside the image. After scaling image is centered-cropped to create image of the same size as original. 2D keypoint locations are changes according the formulas (x and y here are pixel coordinates in range [0, width] for x and [0, height] for y):

$$x_{new} = x_{old} * (1 + scale\ percent) - (w_{old} * scale\ percent)/2$$

$$y_{new} = y_{old} * (1 + scale\ percent) - (h_{old} * scale\ percent)/2$$

Rotation is performed for image and absolute 3D keypoint locations. To create new 2D keypoint locations, new 3D keypoint locations are projected onto the image plane using provided intrinsic matrix. New normalized 3D keypoint locations are created from new absolute 3D keypoint locations the same way as explained in section 3.1. Image is rotates clockwise around the image center by some random angle in range $[-\pi; \pi]$ radians. 3D coordinate system is rotated around z axis in the same clockwise direction using rotation matrix [36] (angle is with negative sign to make coordinate system rotate clockwise):

$$R_z(\theta) = \begin{bmatrix} cos(-\theta) & -sin(-\theta) & 0 \\ sin(-\theta) & cos(-\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

New absolute 3D location for a keypoint is calculated using formula:

$$\begin{bmatrix} x_{new} \\ y_{new} \\ z_{new} \end{bmatrix} = R_z(\theta) * \begin{bmatrix} x_{old} \\ y_{old} \\ z_{old} \end{bmatrix}$$

Scaling and rotation augmentations are applied before brightness, contrast, saturation and blurring augmentations, however, order does not matter.

We do not use horizontal and vertical flipping, because it may creates images of right hand, but we train neural network to work only for left hand. We do not use random cropping either, because cropping may remove some of the keypoints from the image hand, but according to our assumption all keypoint are present on the image.

Augmentation for real datasets will be explained in section 3.5, because real datasets have some issues.

All augmentation are performed 'on fly' using PyTorch library [33]: existing image is loaded from the dataset, transformed and then used for training; it is not stored on the drive. When the same sample will be used next time, different transformations will be applied because of randomness. This means that dataset on infinite size will be created and used for training.

3.5 Adding Real Data

For model to perform well on real dataset, it has to be trained on the real data. Distribution of training data should correspond or be close to distribution of the data that model will be tested/ run on. For this reason we train convolutional neural network for real dataset as well.

However, as were explained in section 2.2 existing real datasets are far from perfect, so extensive preprocessing and data augmentations are needed for these datasets to be usable for model training. We are going to use two datasets - Dexter+Object [27] and Stereo Tracking Benchmark [15]. These datasets were chosen because it was possible to get bounding boxes for hands.

Proposed solution for 3D hand pose estimation takes hand image as an input, so whole image has to be cropped. Stereo Tracking Benchmark dataset contains locations for 21 keypoints, so hand was cropped based on these location with boundary added. Dexter+Object dataset contains only 5 keypoints (or even less) per image,

so cropping based on keypoint locations was impossible. So we decided to semi-automatically create bounding boxes for hands using tracking techniques.

To create bounding boxes for a frame sequence, we used implementation of Kernelized Correlation Filter [23] from OpenCV library [37]. We manually selected hand bounding box for the first frame in the sequence and initialized tracker using this bounding box. We saved all bounding boxes created by tracker, checked them and manually changed bounding boxes if it does not correspond to hand position in the image. Kernelized Correlation Filter Tracker performed well and amount of manual labeling was very low.

As was mentioned in section 2.2, Dexter+Object and Stereo datasets do not contain 21 keypoints that are needed. Dexter+Object contains only 5 keypoints (fingertips), when keypoint is occluded, its location is not provided. Stereo Tracking Benchmark does not contain wrist location, but palm center location instead. As was explained in section 3.1, we are going to have a $1/0$ vector of size 21 that indicates whether required keypoints are provided. Each keypoint Loss (refer to section 3.3) will be multiplied by indicator whether keypoint is present or not, for absent keypoint loss is going to be zero and no network weights updates are going to occur. Convolutional neural network weights are going to be updates only for present keypoint locations and neural network will be trained to predict only these keypoints correctly. Absence of all keypoints means that normalized 3D hand pose cannot be derived for both Dexter+Object and Stereo datasets, so model will be trained only for 2D locations. 3D Loss will not be calculated at all, so updates of model weights for 3D locations will not occur.

We exploit augmentation techniques to increase dataset size of Dexter+Object and Stereo Tracking Benchmark dataset. For both datasets the following data augmentations are applied for each image (the same as for synthetic datasets):

- brightness is randomly changed by some percentage in the interval $[-25\%; 25\%]$;
- contrast is randomly changed by some percentage in the interval $[-25\%; 25\%]$;
- saturation is randomly changed by some percentage in the interval $[-25\%; 25\%]$;
- image is blurred using Gaussian blur with some random radius in range $[0, 0.8]$ (applied after resizing of image to 128×128 pixels).

Additionally, we implemented random scaling and rotation for these datasets. Scaling was implemented by randomly varying boundaries of the bounding boxes. Rotation was implemented by rotating image around image center through some random angle in range $[-\pi; \pi]$ radians and rotating 2D locations through the same angle. Rotation was performed using 2D rotation matrix [36] (angle has negative sign to make rotation clockwise):

$$R(\theta) = \begin{bmatrix} cos(-\theta) & -sin(-\theta) \\ sin(-\theta) & cos(-\theta) \end{bmatrix}$$

New 2D location for a keypoint is calculated using formula (subtraction and addition of half-width/ half-height are needed to make 2D locations rotate around image center):

$$\begin{bmatrix} x_{new} \\ y_{new} \end{bmatrix} = R(\theta) * \begin{bmatrix} x_{old} - w/2 \\ y_{old} - h/2 \end{bmatrix} + \begin{bmatrix} w/2 \\ h/2 \end{bmatrix}$$

This chapter explained in details pipeline of proposed solution for Normalized 3D hand pose estimation from hand image, that consists of several data preprocessing techniques, model architectures and data augmentations. We developed several solution pipelines to evaluate and select the best one.

Chapter 4

Results and Evaluation

This chapter contains comprehensive evaluation of proposed architectures as well as evaluation of data augmentations and addition of partially labeled real data. We will show what architecture is better when trained and evaluated on synthetic datasets - with direct or latent heatmaps for 2D pose, or with vector or tree representation for 3D pose. Then we will further train best architecture adding synthetic data augmentations and investigate its impact on performance for synthetic data (validation sets) and for real data. We will fine-tune model on real partly labeled 2D data and analyze its performance for 2D and 3D hand pose estimation on real data. And finally, we will compare our results with state-of-the-art approaches for hand pose estimation from single RGB image.

4.1 Architectures Evaluation

This section contains comparison and evaluation of 3 convolutional network architectures, so we can select best one to further work with. These architectures are:

- encoder-decoder model that estimates 2D locations though direct heatmaps and 3D locations though simple vector representation (we will refer to this model as Model 1);
- encoder-decoder model that estimates 2D locations though latent heatmaps and 3D locations though simple vector representation (Model 2);
- encoder-decoder model that estimates 2D locations though direct heatmaps and 3D locations though tree-based vector representation (Model 3).

See section 3.2 for actual architecture details.

4.1.1 Training details

Each model was trained on mixture of SynthHands [14] and GANerated [8] datasets with no data augmentations. From both datasets 5000 images were selected for valuation and 5000 - for testing; selection was performed in random way. All other images were used for training, which is 321k from GANerated dataset and 176k from SynthHands dataset; they were randomly mixed while training. We trained each model using two losses - one for 2D locations and one for 3D locations. Model 1 and Model 3 were trained with IoU loss for 2D locations and MSE loss for 3D locations. Model 2 was trained with MSE loss for both 2D and 3D locations. We used weighting coefficients for losses in Model 2; MSE losses for 2D and 3D vector are different in scales, so loss for 2D vector was multiplied by 10. Model 1 and 3 had no loss weighting (coefficients are equal to 1).

Each model was trained separately during 1000 epochs, with 250 random batches per epoch and batch size of 64. We used stochastic gradient descent optimization

algorithm with learning rate of 0.005, momentum of 0.9 and ReduceLROnPlateau scheduler from PyTorch [33], which decreases learning rate when loss reaches plateau and does not decrease any more.

Training was performed only for decoder part; decoder weights were randomly initialized. Weights in ResNet feature extractor were freezed during training for all models; weights were initialized to be ResNet34 weights pretrained on ImageNet from PyTorch. This is assumed to make models more generalizable for other datasets including real data.

4.1.2 Direct vs Latent Heatmaps for 2D hand pose estimation

As 2D keypoint locations will be used for estimation of 3D locations, we want 2D estimates to be accurate. There are 2 ways to estimate 2D keypoint locations - through direct heatmaps and through latent heatmaps (refer to 3.2.2 for more details on that).

We trained two models - Model 1 with direct heatmaps and Model 2 with latent heatmaps. Models differ only in how 2D keypoint locations are estimated; architecture of other parts and training details are the same. Such isolation makes it possible to understand how each 2D hand pose estimation approach works and prevent us from being distracted by other implementation details.

Table 4.1 contains comparison of Model 1 and Model 2 performances for 2D hand pose estimation on SynthHands and GANerated datasets separately (valuation parts of the datasets). Numeric values in table are roots of mean squared error, averaged among validation set samples. 2D locations are estimated to be in range [0,1] for both x and y coordinates, so roots of mean squared error is approximately a percentage for how predicted location is far away from ground truth. For instance, Model 1 makes on average 2% error for a keypoint on SynthHands dataset. We calculated not only average error among all keypoints, but also investigated per keypoint performance. In Table 4.1 we grouped per joint performance by finger and by joint type. This means that we averaged all joint error in the finger when calculated per finger performance, and averaged error among all fingers when calculated joint type performance. For instance, thumb error means that we averaged errors for thumb MCP, PIP, DIP joints and fingertip; when calculating MCP joint errors we averaged MCP joint error for thumb, index, middle, ring and little fingers. Total error means averaged error among all fingers and all joint types (among all keypoints); wrist error is average error for wrist keypoint only.

Table 4.2 contains average keypoint error (total error as in Table 4.1) for hand images with and without objects from SynthHands and GANerated datasets (valuation parts). Both datasets contain approximately half of images with object and half - without; so each entry in the table was calculated as average among 2500 images.

TABLE 4.1: Keypoint Performance of Direct Heatmaps (Model 1) vs Latent Heatmaps (Model 2) for 2D Hand Pose Estimation.

		SynthHands		Ganerated	
		Model 1	Model 2	Model 1	Model 2
wrist		0.008	0.018	0.027	0.036
joint	MCP	0.009	0.017	0.022	0.029
	PIP	0.017	0.024	0.029	0.037
	DIP	0.024	0.031	0.036	0.042
	fingertip	0.032	0.039	0.050	0.052
finger	thumb	0.014	0.023	0.027	0.036
	index	0.019	0.026	0.032	0.037
	middle	0.021	0.027	0.035	0.040
	ring	0.025	0.032	0.040	0.045
	little	0.023	0.030	0.037	0.042
total		0.020	0.027	0.034	0.040

TABLE 4.2: General Performance of Direct Heatmaps (Model 1) vs Latent Heatmaps (Model 2) for 2D Hand Pose Estimation for images with object and without.

	SynthHands		Ganerated	
	Model 1	Model 2	Model 1	Model 2
no object	0.017	0.024	0.028	0.033
object	0.022	0.030	0.039	0.046

Analyzing performance of Model 1 and Model 2, we came up to the following conclusions (refer to Tables 4.1 and 4.2 for details):

- Direct heatmaps approach (Model 1) has lower keypoint error for both datasets, all keypoints and with/without object images.
- Error for GANerated dataset is higher than for SynthHands dataset, even though training dataset contains twice as many images from GANerated dataset than from SynthHands. This can be explained by the fact that GANerated dataset contains much more images that look strange - unrealistic hand appearances and unrealistic poses.
- For MCP joints error is the lowest and for fingertips - the highest.
- For hand images with no objects performance is higher. This conclusion is obvious, because object may occlude several keypoints, so location estimates cannot be accurate for these keypoints.

We are going to use direct heatmap approach for 2D hand pose estimation in final model architecture, because it shows better performance.

4.1.3 Simple vector vs Tree-based vector for 3D hand pose estimation

We also evaluated two approaches for 3D hand pose estimation - simple vector representation and tree-based vector representation. We trained two models - Model 1, that estimates 3D pose through simple vector representation, and Model 3, that uses tree-based hand representation (refer to section 3.2.3 for implementation details). All

other architecture details are the same for Model 1 and Model 3. Both models predict 2D keypoint locations using direct approach heatmap method which is a better option according to conclusions in section 4.1.2.

Tables 4.3 and 4.4 are similar to Tables 4.1 and 4.2. Numeric values in Tables 4.3 and 4.4 are roots of mean squared error for normalized 3D location. So these values are approximately "distance" deviations of predicted locations from ground truth in normalized coordinate system, where distance between wrist and middle finger MCP joint is equal to 1.

TABLE 4.3: Keypoint Performance of Vector Representation (Model 1) vs Tree-Based Representation (Model 3) for 3D Hand Pose Estimation.

		SynthHands		Ganerated	
		Model 1	Model 3	Model 1	Model 3
wrist		0.098	0.098	0.142	0.143
joint	MCP	0.047	0.047	0.064	0.062
	PIP	0.113	0.114	0.127	0.126
	DIP	0.169	0.169	0.182	0.180
	fingertip	0.229	0.230	0.243	0.240
finger	thumb	0.133	0.140	0.182	0.181
	index	0.134	0.136	0.141	0.141
	middle	0.130	0.124	0.137	0.129
	ring	0.149	0.151	0.150	0.150
	little	0.151	0.150	0.160	0.159
total		0.137	0.138	0.153	0.151

TABLE 4.4: General Performance of Vector Representation (Model 1) vs Tree-Based Representation (Model 3) for 3D Hand Pose Estimation for images with object and without.

	SynthHands		Ganerated	
	Model 1	Model 3	Model 1	Model 3
no object	0.114	0.116	0.136	0.134
object	0.161	0.161	0.167	0.164

Based on data in Table 4.3 and Table 4.4 the following conclusions are made:

- Both vector (Model 1) and tree-based (Model 3) approaches have the same performance on GANerated and SynthHands datasets.
- And the same as for 2D hand pose estimation, performance of models in higher for SynthHands than for GANerated dataset; for hand images with no objects performance is higher; and for MCP joints error is the lowest and for fingertips - the highest.

Even though, performance of both methods is the same, we are going to use vector approach for 3D hand pose estimation in final model, because it has simpler implementation.

4.1.4 Best Architecture Evaluation

Based on conclusions in sections 4.1.2 and 4.1.3, final model architecture (Model 1) estimates 2D hand pose using direct heatmaps and 3D hand pose using simple vector representation.

As was mentioned before, Model 1 has 0.02 error on SynthHands dataset and 0.034 error on GANerated dataset for 2D hand pose; and 0.137 error on SynthHands dataset and 0.153 error for GANerated dataset for normalized 3D hand pose.

We further qualitatively investigated cases for these dataset when Model performs well and when fails. Figure 4.1 shows some of the images from SynthHands and Ganerated dataset that have high and low error for both 2D and 3D hand pose. For both datasets performance is good when hand has natural appearance and pose and when keypoints are visible. Performance is poor for hands that are highly occluded by objects, and when hand and its pose are unnatural and indistinguishable even for humans. So Model 1 performance would be even higher if all bad images were removed from datasets.

FIGURE 4.1: Some images from SynthHands and GANerated dataset with highest and lowest prediction errors.

Model 1 was also evaluated on two real datasets - Stereo Tracking Benchmark [15] and Dexter+Object [27]. As was explained in sections 2.2, Stereo dataset has 20 needed keypoints available, and Dexter+Object provides only 5 keypoints and frame may have less than 5 keypoints available. Also, Stereo Tracking Benchmark and Dexter+Object do not have enough data to derive normalize 3D hand pose, so 3D locations are not available at all. To evaluate Model 1 performance for these datasets, we provide numeric values for 2D estimation errors for available keypoints and visual evaluation for 3D hand pose.

Table 4.5 is similar to Table 4.1 and contains percentage errors for available keypoints. Performance is poor, because there is a big domain gap between synthetic data that were used for training and real data. Average error for Stereo Tracking Benchmark (B1Counting and B1Random sequences) is 12% and for Dexter+Object (Grasp2 sequence) is 32%.

TABLE 4.5: Keypoint Performance of Best Model for 2D Hand Pose Estimation on Real Datasets.

		Stereo (B1Counting, B1Random)	Dexter+Object (Grasp2)
wrist		–	–
joint	MCP	0.088	–
	PIP	0.102	–
	DIP	0.120	–
	fingertip	0.152	0.322
finger	thumb	0.119	–
	index	0.089	–
	middle	0.109	–
	ring	0.130	–
	little	0.131	–
total		0.115	0.322

Stereo dataset has lower error because of large variety of hand poses, some of them are easy to predict. Dexter+Object dataset (Grasp2 sequence) has similar hand poses on all image frames, and these poses are not typical for synthetic training set - in Dexter+Object hand is in inverted position (upside down), and there is no such images in GANerated and SynthHands datasets.

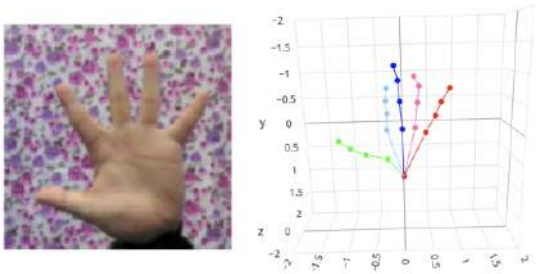

FIGURE 4.2: Example of Success Case from Stereo Tracking Benchmark dataset (thumb, index, middle ring and little finger are shown in green, light blue, blue, pink and red color, respectively).

Based on the result of manual visual evaluation, for Stereo dataset model fails to predict hand pose for almost all poses except simplest ones. We provided an example of success case in Figure 4.2. For Dexted+Object dataset success cases were not found. By manual evaluation we are not able to say whether model predicts distances accurately, but rather we can tell whether model can "understand" general hand pose.

It seems that model learned distribution from the synthetic datasets, so in most cases it outputs realistic poses of left hand with correct finger order and natural angles, even when these poses are not correct (see Figure 4.3 for these cases).

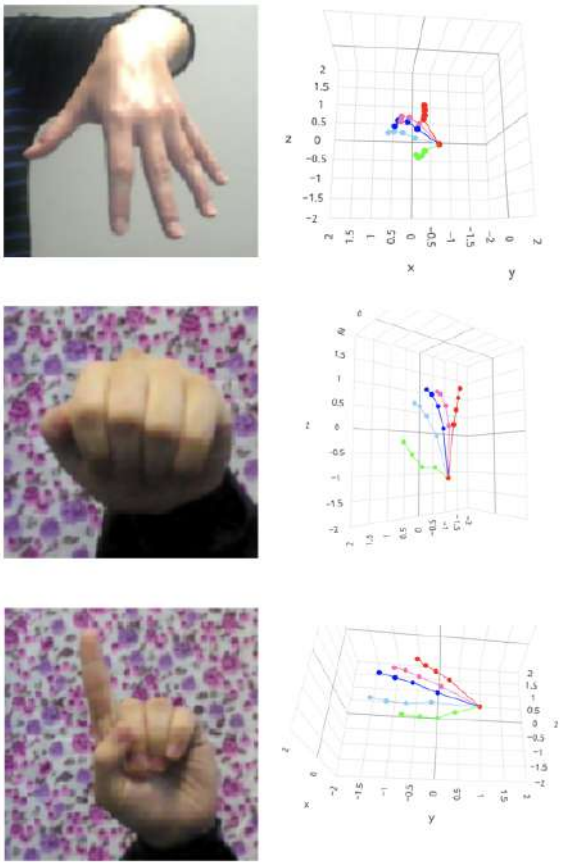

FIGURE 4.3: Failure Cases from Stereo and Dexter+Object Datasets (thumb, index, middle ring and little finger are shown in green, light blue, blue, pink and red color, respectively; 3D plots were arbitrarily rotated to show hand pose better).

4.2 Data Augmentations and Adding real data

Best model architecture, selected in section 4.1 is going to be trained further on augmented synthetic data and with addition of real data in order to increase performance.

4.2.1 Data Augmentations

Training Details

Model 1 was continued to train on mixture of SynthHands [14] and GANerated [8] datasets with data augmentations explained in section 3.4. Datasets were mixed in the way, that probability to get instance from SynthHands is 0.66 and from GANerated dataset is 0.33. The reason, why model now is trained more on SynthHands dataset is because this dataset has much more augmentations than GANerated.

Model was trained during additional 400 epochs, with 250 random batches per epoch and batch size of 64. Weights in ResNet-based encoder were unfreezed and fine-tuned, because model is unlikely to overfit with such extensive data augmentations.

We used stochastic gradient descent optimization algorithm with learning rate of 0.003 (lower than in initial training), momentum of 0.9 and ReduceLROnPlateau scheduler from PyTorch [33].

Evaluation

We compared this model with Model 1 from section 4.1, that was trained for 1000 epochs on non-augmented synthetic data and with feature extractor freezed. Such comparison makes it possible to understand impact of data augmentations and fine-tuned feature extractor.

Initially, we evaluated model on validation parts of SynthHands and GANerated datasets. Surprisingly, 3D error for SynthHands increased (refer to Table 4.6a). To further investigate reasons for that, model was evaluated on the same validation parts of SynthHands and GANerated, but with random data augmentations added; augmentations are the same as were used for model training. And now we see improvement in performance for both datasets - significant for SynthHands and slight for GANerated (refer to Table 4.6b).

TABLE 4.6: Model Performance on validation sets before and after trained with data augmentations.

	Before Augmentation		After Augmentation	
	2D error	3D error	2D error	3D error
dataset				
a) non-augmented:				
SynthHands	0.020	0.137	0.018	0.148
GANerated	0.034	0.153	0.026	0.144
b) augmented:				
SynthHands	0.119	0.697	0.020	0.165
GANerated	0.035	0.156	0.027	0.145

Below are possible explanations of evaluation results:

- Train/Validation/Test split for SynthHands and GANerated datasets was random. However, these datasets contain similar images that slightly differ in viewpoints. We were unable to correctly split datasets, because no information on these similar images was provided in dataset description. So similar images may be included both in train and validation part. As a result, model overfitted for such cases and showed high performance on validation datasets. But when augmentations for validation set were added, such overfit did not influence model evaluation.
- SynthHands dataset was augmented more heavily than GANerated, because it additionally includes rotation and scaling augmentations. So distribution gap between SynthHands with augmentation and SynthHands without augmentation is larger than between GANerated dataset with augmentation and GANerated without augmentation. This explains why model trained with no augmentation performed so much poorly on SynthHands dataset with augmentations.

To be mentioned, evaluation on validation datasets with augmentation is not fully correct, because of randomness in augmentations. So "before" and "after" models were evaluated on different image sets. We rerun evaluation several times to be sure, that such tendencies were not received by chance. After each run, numbers were slightly different, but general tendency is the same.

We evaluated model performance on real datasets - Stereo Dataset Benchmark (B1Counting and B1Random sequences) and Dexter+Object (Grasp2 sequence) datasets. We conducted the same comparison as in Table 4.6a - model performance before training on augmented synthetic data and after training.

Table 4.7 shows that augmentations lead to slight improvement on Stereo dataset and more significant improvement on Dexter+Object dataset. The reason, why augmentations lead to such significant improvement for Dexter+Object dataset is that initially synthetic datasets have no hand images in inverse position (upside down), but data augmentations (rotation, in particular) changed it. Figure 4.4 show model predictions before and after data augmentations applied. Initial model cannot predict that hand is inverted and tries to fit some pose with wrist at the bottom, but after augmentations added, at least position of wrist and hand direction are predicted correctly.

TABLE 4.7: 2D errors on Real Datasets before and after trained with Data Augmentations.

	Before Augmentation	After Augmentation
Stereo (B1Counting, B1Random)	0.115	0.099
Dexter+Object (Grasp2)	0.322	0.214

Performance of model for 3D hand pose estimation is still poor and visually looks the same as before data augmentations applied - only simple poses are predicted correctly.

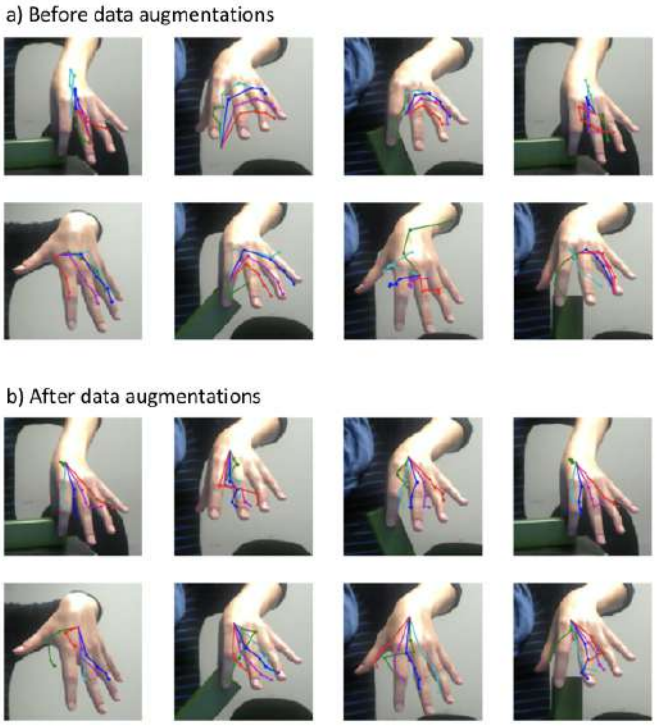

FIGURE 4.4: Visual comparison of model performance for 2D hand pose estimation on Dexter+Object dataset before and after data augmentations applied.

4.2.2 Adding Real Data

We fine-tuned model from section 4.2.1 separately for two real datasets - Stereo Tracking Benchmark [15] and Dexter+Object [27]. We evaluated model performance on these datasets.

Stereo Tracking Benchmark

We split Stereo Tracking Benchmark into train and validation sets. Train set includes video sequences B2-B7 both Random and Counting (15k frames); validation set includes sequences B1Random and B1Counting (3k frames). Stereo Tracking Benchmark has 20 keypoints (no wrist location provided) and only 2D locations.

Model was fine-tuned for 100 epochs with 250 batches per epoch and batch size of 32 (batch size was decreased from 64 because of technical reasons only - to fit into computer memory). We used stochastic gradient descent optimization algorithm with learning rate of 0.001 (lower than during training with augmentations), momentum of 0.9 and ReduceLROnPlateau scheduler from PyTorch. Feature extractor was finetunes as well. Model was trained using augmented train part of Stereo dataset (see section 3.5 for details on augmentations).

Average 2D error for Stereo dataset (validation part) is 0.028, which means 2.8% deviation of predicted values from ground truth. And model now performs much better than when trained only on synthetic data. See Table 4.8 for details.

2D labeling for Stereo dataset is different from SynthHands and GANerated datasets - while synthetic datasets contain exact joint labels, Stereo dataset has labels for the center of the bone. And model during training was able to learn new labeling (see Figure 4.5).

TABLE 4.8: Model Performance on Real Datasets (validation) for 2D Hand Pose Estimation.

	Synthetic Data Only	Fine-tuning with Real Data
Stereo (B1Counting, B1Random)	0.099	0.028
Dexter (Grasp2)	0.214	0.058
Dexter (Rotate)	–	0.043

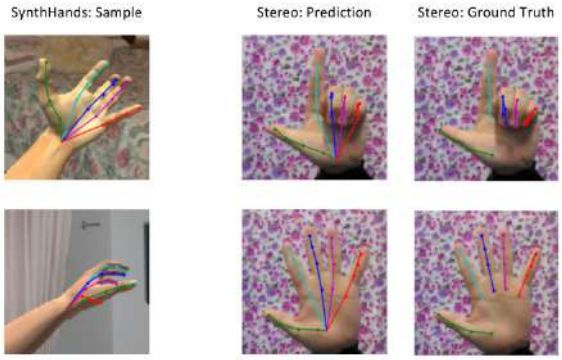

FIGURE 4.5: Comparison of labeling for SynthHands and Stereo datasets.

Prediction for 3D pose are still poor, and were not improved by correct 2D pose estimation. For 3D hand pose, model trained only on synthetic data could not generalize to real data.

Dexter+Object dataset

We split Dexter+Object dataset into train and validation sets. Train set includes sequences Grasp1, Occlusion, Rigid, Pinch and Rotate (2500 frames); validation set includes sequence Grasp2 (650 frames). Dexter+Object dataset has at most 5 keypoints per frame (fingertips locations if they are not occluded) and only 2D locations.

Model was fine-tuned for 40 epochs with 250 batches per epoch and batch size of 64. We used stochastic gradient descent optimization algorithm with learning rate of 0.001 (lower than during training with augmentations), momentum of 0.9 and ReduceLROnPlateau scheduler from PyTorch. Feature extractor was finetunes as well. Model was trained using augmented train part of Dexter+Object dataset (see section 3.5 for details on augmentations).

Performance for Dexter+Object (validation set) was improved with adding real data - 2D error dropped from 0.214 to 0.058 (see Table 4.8). However, visually, 2D predictions are not accurate (see Figure 4.6a). Such difference between quantitative and qualitative evaluation is because there are only locations for non-occluded fingertips and fingertips predictions on average are close to true locations.

We tested different train/validation splits. Now validations set contains Rotate sequence (800 frames) and train set - all other sequences (2350 frames). Training details are the same. For Rotate sequence 2D error is lower - 0.043, and visually performance looks good (see Table 4.8 and Figure 4.6b).

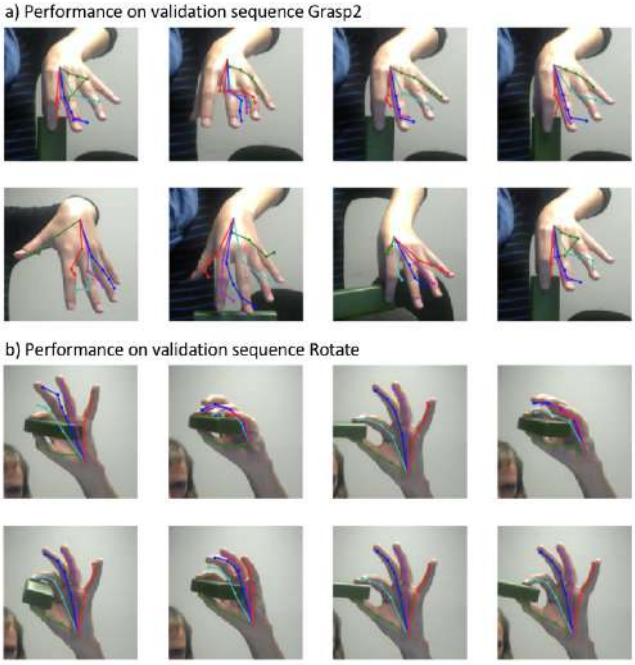

FIGURE 4.6: 2D Predictions for Dexter+Object on different Validation Sequences.

Predictions for 3D pose are still poor for Grasp2 sequence and better for Rotate sequence (see Figure 4.7). However, for these poses in Rotate sequence predictions of model trained on augmented synthetics data were also good. So tuning 2D estimation on real data does not help. Model is unable to generalize and 3D ground truth locations are required to improve model performance.

Interestingly, that model was trained on data with some (and even most) keypoins are missing, but could correctly predict keypoints for which there is no ground truth locations at all - wrist in Stereo dataset and mostly all keypoints (except fingertips) for Dexter+Object. This means, that model when trained on synthetic data could learn spacial relations among hand joints. And when some keypoint locations were adjusted because of training on real data, model was able to adjust keypoint locations for which data is not provided as well.

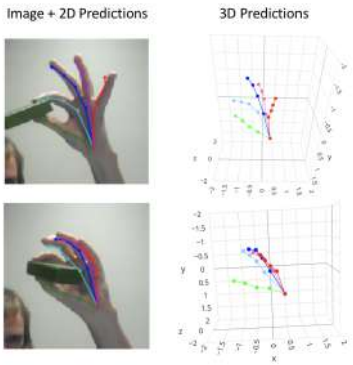

FIGURE 4.7: Example of success cases of 3D Predictions for Rotate sequence from Dexter+Object dataset (thumb, index, middle ring and little finger are shown in green, light blue, blue, pink and red color, respectively).

4.3 Comparison with SOTA approaches

We compared performance of our model for 2D hand pose estimation with existing state-of-the-art approaches for the hand pose estimation from RGB image. 3D comparison was not conducted, because our model produces normalized 3D hand poses, but not absolute locations; besides, without having ground truth 3D locations for training, model performance is poor.

For comparison we selected metric - mean squared error per keypoint in pixels. Comparison is approximate, because authors used different datasets to train models, different train/validation set splits, as well as different assumptions to model evaluation. We are providing numeric values (see Table 4.9) as well as authors' explanation, how these values were received.

SynthHands and GANerated datasests. Train/validation/test split was random, and as a result, similar hand images belong to different train/validation/test, which is not fully correct. Correct split could not be performed, because no information on similar images was provided in dataset descriptions. Model was trained on SynthHands and GANerated datasests with data augmentations applied. Model performance was reported for test sets, because validation sets were used to select best model architecture. Pixel error was calculated by multiplying percent error by original image size, which is 256 pixels for GANerated dataset and 480 pixels for SynthHands dataset (initially, image in SynthHands dataset was of rectangle size 640×480, it was centered cropped to be of size 480×480). Error for SynthHands dataset is higher than for GANerated because image size in SynthHands dataset is larger as well. No other authors reported performance on these datasets.

TABLE 4.9: Our Model Performance for 2D Hand Pose Estimation comparing to State-of-the-art Approaches (metric - mean squared error per keypoint in pixels).

	Our Model	Zimmermann and Brox [18]	Mueller and others [8]	Iqbal and others [17]
SynthHands [14]	8.760	–	–	–
GANerated [8]	6.708	–	–	–
Stereo TB [15]	6.078	5.522	–	–
Dexter+Object [27]	11.360	25.160	19.263	16.660

Stereo Tracking Benchmark. We fine-tuned model using B2-B6 sequences and evaluated using B1 seqs. Hand bounding boxes were provided. Pixel error is calculated as percent error multiplied by bounding box size (bounding box sizes may differ for different frames).

Pipeline by Zimmermann and Brox [18] has two stages - 1) localize hand on image; and 2) predict hand pose from cropped hand image. Authors reported performance of the second stage, providing cropped hand images with ground truth bounding boxes as an input for hand pose estimation model. Model was trained on 15k images and evaluated on 3k; but what image sequences were used for training and evaluation is unknown. Our model has higher performance.

Dexter+Object. We fine-tuned model using Grasp1, Grasp2, Occlusion, Rigid and Pinch sequences; and evaluated on Rotate sequence. Bounding boxes are assumed to be known. Pixel error is calculated as percent error multiplied by bounding box size (bounding box sizes may differ for different frames).

Zimmermann and Brox in [18] reported accuracy on the whole dataset; no data from this dataset was used for training and no hand bounding boxes were provided. Mueller and others in [8] did not train model on Dexter+Object, no bounding boxes were used either. Pixel error was not reported by authors directly, we calculated it using AUC ration between [18] and [8]; AUC metric was reported in both papers:

$$Pixel\ Error_{model\ 1} = Pixel\ Error_{model\ 2} * AUC_{model\ 2} / AUC_{model\ 1}$$

Iqbal and others in [17] trained model with adding data from Dexter+Object; train/validation split is unknown. Hand bounding boxes were not provided. Pixel error was not reported by authors directly, we calculated it using AUC ratio between [18] and [17] (the same as for [8]).

Unfortunately, experiment setups are very different between SOTA approaches and our approach, so we cannot compare performance on Dexter+Object dataset.

In this chapter we conducted comprehensive evaluation of the experiments taken. We evaluated several architectures and discovered what is the best way to predict 2D pose and 3D pose, assuming that good dataset is available. We trained best model on synthetic data with extensive data augmentations and evaluated its performance on real datasets. We fine-tuned model on partially labeled 2D real data and reported, whether we were able to increase model performance for 2D and 3D hand pose estimation. Finally, we compared our results with state-of-the art approaches for hand pose estimation from single RGB image.

Chapter 5

Discussion and Conclusions

In this thesis we have worked on pipeline improvement for 3D hand pose estimation from single RGB camera. Among pipeline steps - hand localization, left-right hand classification, normalized 3D hand pose estimation, absolute 3D pose hand pose restoration and tracking - we focused our attention to normalized 3D hand pose estimation from cropped hand image, because improving this step will result in performance boost for general pipeline.

We trained and evaluated two approaches for 2D pose estimation (latent and direct heatmaps) and two approaches for 3D pose estimation (vector representation and tree-based representations); and concluded that direct heatmap is the best approach for 2D pose and vector representation - for 3D.

We trained best model on augmented synthetic datasets and evaluated that model mean keypoint error on synthetic datasets (validation part) is about 2% for 2D and 0.14 for normalized 3D. We assume that at least the same performance could be achieved if good large-scale real dataset is available. This means that performance of Hand pose estimation from RGB image is limited by absence of good datasets.

We tried to deal with this problem by pre-training model on synthetic datasets and fine-tuning it using real datasets available. Real datasets do not have all required keypoints. Also, because of the absence of some keypoints, normalized pose cannot be derived either. So among real data available we had 2D locations for some keypoints and no 3D normalized locations at all.

With partly available 2D labeling we were able to train model that adequately predicts whole 2D hand pose even when only 5 keypoint locations were available. Model when pre-trained on synthetic data could learn spacial relations among hand joints. And when some keypoint locations were adjusted because of training on real data, model was able to adjust keypoint locations for which data is not provided. For 2D hand pose estimation, we could outperform state-of-the-act approach for Hand pose estimation from RGB image.

With no real 3D data available, model performance is still unsatisfied. It correctly predicts only the simplest hand poses and even after 2D real data were added it is unable to generalize. However, model learnt hand pose distribution from synthetic datasets, so in most cases it outputs realistic poses of left hand with correct finger order and natural angles, even when these poses are not correct. We assume, that fine-tuning model with some real 3D data will result in significant performance boost.

All code and pre-trained models will be opened to research community for evaluation and reuse for their own researches. See GitHub project page.

Bibliography

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, pp. 1097–1105, 2012.

[2] R. Prabhu, "Cnn architecutes - lenet, alexnet, vgg, googlenet and resnet," in *Medium*, 2018.

[3] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, *et al.*, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[4] S. Y. Fei-Fei Li, Justin Johnson, "Cs231n: Convolutional neural networks for visual recognition," in *Stanford University*, http://cs231n.stanford.edu/, 2017.

[5] K. Simonyan, A. Vedaldi, and A. Zisserman, "Deep inside convolutional networks: Visualising image classification models and saliency maps," *arXiv preprint arXiv:1312.6034*, 2013.

[6] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," in *European conference on computer vision*, pp. 818–833, Springer, 2014.

[7] J. Yosinski, J. Clune, A. Nguyen, T. Fuchs, and H. Lipson, "Understanding neural networks through deep visualization," *arXiv preprint arXiv:1506.06579*, 2015.

[8] F. Mueller, F. Bernard, O. Sotnychenko, D. Mehta, S. Sridhar, D. Casas, and C. Theobalt, "Ganerated hands for real-time 3d hand tracking from monocular rgb," in *Proceedings of Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[9] https://en.wikipedia.org/wiki/Wired_glove. Wikipedia article on wired glove.

[10] R. Y. Wang and J. Popović, "Real-time hand-tracking with a color glove," *ACM transactions on graphics (TOG)*, vol. 28, no. 3, p. 63, 2009.

[11] S. Yuan, G. Garcia-Hernando, B. Stenger, G. Moon, J. Yong Chang, K. Mu Lee, P. Molchanov, J. Kautz, S. Honari, L. Ge, *et al.*, "Depth-based 3d hand pose estimation: From current achievements to future goals," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[12] G. Moon, J. Y. Chang, and K. M. Lee, "V2v-posenet: Voxel-to-voxel prediction network for accurate 3d hand and human pose estimation from a single depth map," in *CVPR*, vol. 2, 2018.

[13] X. Chen, G. Wang, H. Guo, and C. Zhang, "Pose guided structured region ensemble network for cascaded hand pose estimation," *arXiv preprint arXiv:1708.03416*, 2017.

[14] F. Mueller, D. Mehta, O. Sotnychenko, S. Sridhar, D. Casas, and C. Theobalt, "Real-time hand tracking under occlusion from an egocentric rgb-d sensor," in *Proceedings of International Conference on Computer Vision (ICCV)*, vol. 10, 2017.

[15] J. Zhang, J. Jiao, M. Chen, L. Qu, X. Xu, and Q. Yang, "3d hand pose tracking and estimation using stereo matching," *arXiv preprint arXiv:1610.07214*, 2016.

[16] R. R. Basaru, C. Child, E. Alonso, and G. G. Slabaugh, "Hand pose estimation using deep stereovision and markov-chain monte carlo.," in *ICCV Workshops*, pp. 595–603, 2017.

[17] U. Iqbal, P. Molchanov, T. Breuel, J. Gall, and J. Kautz, "Hand pose estimation via latent 2.5 d heatmap regression," *arXiv preprint arXiv:1804.09534*, 2018.

[18] C. Zimmermann and T. Brox, "Learning to estimate 3d hand pose from single rgb images," in *International Conference on Computer Vision*, vol. 1, p. 3, 2017.

[19] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pp. 779–788, 2016.

[20] X. Sun, J. Shang, S. Liang, and Y. Wei, "Compositional human pose regression," in *The IEEE International Conference on Computer Vision (ICCV)*, vol. 2, p. 7, 2017.

[21] G. Pavlakos, X. Zhou, K. G. Derpanis, and K. Daniilidis, "Coarse-to-fine volumetric prediction for single-image 3d human pose," in *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pp. 1263–1272, IEEE, 2017.

[22] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: towards real-time object detection with region proposal networks," *IEEE Transactions on Pattern Analysis & Machine Intelligence*, no. 6, pp. 1137–1149, 2017.

[23] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista, "High-speed tracking with kernelized correlation filters," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 3, pp. 583–596, 2015.

[24] J. Carreira, P. Agrawal, K. Fragkiadaki, and J. Malik, "Human pose estimation with iterative error feedback," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4733–4742, 2016.

[25] X. Chen and H. Guo, "Awesome work on hand pose estimation," in *GitHub*, https://github.com/xinghaochen/awesome-hand-pose-estimation.

[26] M. Oberweger, G. Riegler, P. Wohlhart, and V. Lepetit, "Efficiently creating 3d training data for fine hand pose estimation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4957–4965, 2016.

[27] S. Sridhar, F. Mueller, M. Zollhöfer, D. Casas, A. Oulasvirta, and C. Theobalt, "Real-time joint tracking of a hand manipulating an object from rgb-d input," in *European Conference on Computer Vision*, pp. 294–310, Springer, 2016.

[28] S. Sridhar, A. Oulasvirta, and C. Theobalt, "Interactive markerless articulated hand motion tracking using rgb and depth data," in *Proceedings of the IEEE international conference on computer vision*, pp. 2456–2463, 2013.

[29] Y. Ganin and V. Lempitsky, "Unsupervised domain adaptation by backpropagation," *arXiv preprint arXiv:1409.7495*, 2014.

[30] B. Zhou, H. Zhao, X. Puig, S. Fidler, A. Barriuso, and A. Torralba, "Scene parsing through ade20k dataset," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, vol. 1, p. 4, IEEE, 2017.

[31] O. Ronneberger, P.Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Medical Image Computing and Computer-Assisted Intervention (MICCAI)*, vol. 9351 of *LNCS*, pp. 234–241, Springer, 2015.

[32] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pp. 770–778, 2016.

[33] https://pytorch.org. PyTorch: Open source deep learning platform.

[34] M. A. Rahman and Y. Wang, "Optimizing intersection-over-union in deep neural networks for image segmentation," in *International Symposium on Visual Computing*, pp. 234–244, Springer, 2016.

[35] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1–9, 2015.

[36] https://en.wikipedia.org/wiki/Rotation_matrix. Wikipedia article on rotation matrix.

[37] https://opencv.org. OpenCV: Open Source Computer Vision Library.