MEDIA INFORMATICS

RWTH AACHEN UNIVERSITY

Master Thesis

Bottom-Up Multiple Person Multi-Camera 3D Human Pose Estimation

Wei Wang

MEDIA INFORMATICS

RWTH AACHEN UNIVERSITY

Master Thesis

Bottom-Up Multiple Person Multi-Camera 3D Human Pose Estimation

Author:	Wei Wang
Supervisor:	Supervisor
Advisor:	Advisor
Submission Date:	Submission date

I confirm that this master thesis is my own work and I have documented all sources and material used.

Munich, Submission date Wei Wang

Acknowledgments

Abstract

Contents

Acknowledgments **iii**

Abstract **iv**

1 Introduction **1**
1.1 Motivation 1
1.2 Problem Definition and Challenges 2
1.3 Applications 3
1.4 Thesis Outline 4

2 Theoretical Backgrounds **5**
2.1 Camera Model 5
2.1.1 Camera Intrinsic Matrix 5
2.1.2 Camera Extrinsic Matrix 6
2.2 Two-view Geometry 7
2.2.1 Epipolar Geometry 7
2.2.2 Fundamental Matrix 8
2.2.3 Triangulation 8
2.3 Supervised Learning with Deep Convolutional Networks 9
2.3.1 Deep Convolutional Networks 9
2.3.2 Residual Learning and ResNets 12
2.3.3 3D U-Net 14
2.3.4 Transfer Learning 14

3 Related Works **17**
3.1 Top-down Multi-person 3D Pose Estimation 17
3.1.1 Single View 17
3.1.2 Multiple Views 18
3.2 Bottom-up Multi-person 3D Pose Estimation 19
3.2.1 Single View 19
3.2.2 Multiple Views 21
3.3 Temporal Fusion 21
3.4 Brief Conclusion 22

4 Datasets **23**
4.1 COCO 2017 Keypoint Dataset 23
4.1.1 Annotation Format 23
4.1.2 Joint Heatmaps 28
4.2 CMU Panoptic Dataset 28
4.2.1 Annotation Format 28
4.3 Data Augmentation 29
4.3.1 Image Augmentation 29
4.3.2 Joint Augmentation 30

5 Fusion Network and 3D Pose Estimation **33**
5.1 2D Backbone Module: ResNets 33
5.2 Sampler Module: Epipolar Sampler 35
5.3 View Fusion 35
5.3.1 View Fusion Model 35
5.4 Temporal Fusion Model 37
5.5 3D Pose Estimations 39
5.5.1 Extract Poses from 3D Points 39
5.6 Summary 41

6 Experiments **44**
6.1 Quantitative Results 44
6.2 Qualitative Results 46
6.3 Discussion 46

7 Conclusion **50**

List of Figures **51**

List of Tables **55**

Bibliography **56**

1 Introduction

Computer vision has been an important role in the automation of processes in many different areas. A well-studied subject within the field of computer vision is humans. In detecting and tracking humans, humans are usually localized, by means of bounding boxes, within a single or multiple views. In pose estimation, a human body is usually represented, by a skeleton composed of a set of joints. The objective in pose estimation is to recover the joint locations of human bodies in 2D images or 3D space. While 2D single person detection and tracking of humans has been well addressed in real-world environments, 3D multi-person pose estimation remains an open problem, due to the complex scenarios that it has to consider. In this thesis, we study the problem from multiple views. We investigate the approach that aggregates multiple view information using epiporlar geometry. Our ultimate goal is to estimate 3D human pose from multiple 2D views.

1.1 Motivation

Estimation 3D multi-person human poses have been addressed, by using a single view [1, 2, 3] or multi-view camera [4, 5, 6, 7, 8, 9]. Regardless of single view or multi-view approaches, the first step is usually generating 2D likelihood maps of joint locations by a backbone network trained on a 2D pose estimation task. However, the accuracy of the 2D pose estimation backbone degrades significantly when joints being occluded [10], resulting single-view methods to produce unrealistic 3D poses. Since occlusions are a common issue in multi-person and in-the-wild environments, leveraging multi-view information becomes a natural choice in 3D multi-person pose estimation. Some previous studies adopt 3D pictorial structure (3DPS) [7, 4, 5], where the 3D poses are recovered from 2D joints in a discretized 3-space, with predefined geometric constraint of human bodies. However, a severe problem of 3DPS is the expensive computational cost due to the huge state space with multiple people in multiple views. Other approaches rely on deep networks that operate on volumetric grids [6, 3] to combine features coming from different views in accordance with epipolar geometry. Unfortunately, it requires a discretized space that covers the whole environment, which is memory demanding and, more importantly, impractical in outdoor environments.

1.2 Problem Definition and Challenges

In this work, we address the 3D multi-person pose estimation task from multiple views using epipolar geometry. Inspired by the lightweight approaches that utilize multiview geometry [4, 11, 12], we use a lightweight 2D pose estimator trained on the COCO dataset as backbone and aggregate multi-view information using epipolar geometry. Our goal is to automatically estimate the pose of multiple individuals in 3D space, given a set of images from a calibrated multi-view camera system. However, the transition from 2D to 3D space and from single to multiple human pose estimation is a challenging task.

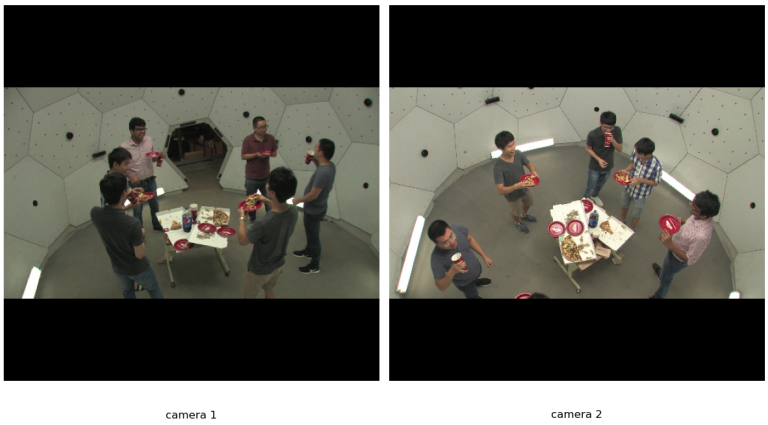


Figure 1.1: Estimating multi-person 3D human poses without using computational heavy operations is the task we address in this work.

First of all, learning 2D human bodies from image data is a hard task due to its articulation and large deformation it can go through. Thus, 2D human pose estimation researchers usually use a rigid skeleton compose of a various number of joints, such as elbow, pelvis, etc., shown in 1.2. Since the 2D models are built from images, we have to compensate the missing dimension when estimating 3D poses.

Secondly, the appearance of a joint varies significantly from one view to another. Most 2D human pose datasets that based on natural images provide only frontal and horizontal views. Yet it is common in multi-person environments that most of the joints are occluded in frontal view and can only be revealed at a novel view, see Fig.1.1. Occlusions cause a serious problem in 2D multi-person human pose estimation in single view images [10]; occlusions can not only caused by objects but by oneself and

Figure 1.2: A human pose is approximated by a rigid skeletons, composed by various joints. Performing human pose estimation, however, from images can be a difficult task due to the body pose and appearance high variation.

other humans as well, shown in Fig. 1.1.

Moreover, recent works [5, 4, 8, 13] cast multi-person 3D human poses estimation as associating 2D poses in all camera views. This is complicated problem in multiple human 3D pose estimation when the identity of individuals is unknown. An association between the individuals across all views is required to avoid mixing the body parts of different individuals. For instance, a left hand of one person in one view will have multiple left hand candidates in other camera views coming not only from the same person, but also from other individuals and potential false positive detections. In practice, this will create incorrect body part hypotheses that can lead to fake body poses in the 3D space. In addition, matching 2D poses between all pairs of views still makes the computational complexity explode as the number of cameras increases [9] .

To address these challenges, we propose to fuse features from multiple views using epipolar geometry and a deep convolution network. Our intuition is that a joint that being occluded in one view could be visible in other views and a deep convolution network can learn how to see through occlusions from the fused features.

1.3 Applications

A framework for multi-person human pose estimation from multiple views has a wide range applications, such as motion capture, surveillance, sports, and human activity recognition, etc.. We provide some examples that our framework could be applied on. Motion capture systems haven been beneficial in film industry, especially for CG characters. The current technology for estimating reliable 3D human poses is based on maker-based solutions, which work only in a studio environment. On

the contrary, our approach is marker-less and can be adapted to any unconstrained environments. Another useful scenario for our framework is sport science or analysis, where athletes can not wear MOCAP suits. For example, estimating the poses of basketball players, captured from different view, supports game analysis. In addition, body pose estimation can be used to study the tactic of opponents or to improve the training of teams. Furthermore, our framework can be used in the public surveillance systems since there are usually multiple cameras installed in public areas. As this work was done during the COVID-19 pandemic period, a crowd control system, combining our work and activity recognition, for detecting citizen gathering in a large group would be beneficial to the health of public. Lastly, our frame work can be applied on multi-pedestrian path prediction for autonomous driving. Lastly, our frame can add to human action recognition, where human poses are one of the strong cues. In general, our frame benefits various real-world applications because it is less computation demanding and suits for unconstraint environments.

1.4 Thesis Outline

We provide an overview for each chapter of the thesis

- **Chapter 2** We present the theoretical background of our work.
- **Chapter 3** In this chapter, we provide an overview of the current researches in the field of 3D human pose estimation and compare the difference between our work and related researches.
- **Chapter 4** We explain the design of our feature fusion modules and the network architectures.
- **Chapter 5** In this chapter, we give an overlook of two different datasets we used in this work and explain metrics being used by the datasets to evaluate the performance of our model. Futher, we introduce the data augmentation methods.
- **Chapter 6** We evaluate the performance of our work and give an ablation study
- **Chapter 7** We conclude our work by presenting our findings, the limitations of the proposed methods and our directions for future work.

2 Theoretical Backgrounds

2.1 Camera Model

2.1.1 Camera Intrinsic Matrix

A camera can be consider as a linear mapping from 3D spaces to a 2D images when we ignore the lens distortion. Let us denote a point in 3D space as $\mathbf{X_{cam}} \in \mathbb{R}^{4\times 1}$; the subscript *cam* mean the point is located in camera coordinate frame. The 3D point $\mathbf{X_{cam}}$ is imaged in a camera view, at $\mathbf{x} = \mathbf{PX_{cam}}$, where $\mathbf{x} \in \mathbb{R}^{3\times 1}$ represent 2D points in images and $\mathbf{P} \in \mathbb{R}^{3\times 4}$ is a *camera projection matrix*. Fig. 2.1 [14] illustrate the geometry relationship from 3D spaces to image spaces. Note that, for the sake of simplicity, we use homogeneous coordinate and column vector to represent a point.

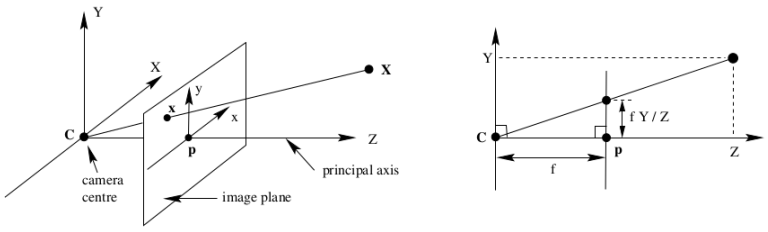


Figure 2.1: **C** is the camera centre and **p** is the center of the image plane or called *principal point*. The camera center is here placed at the origin of world coordinate frame. Note the image plane is placed in front of the camera center.

we can express a **basic pinhole camera model**, $\mathbf{x} = \mathbf{PX_{cam}}$, in the following equation

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} fX \\ fY \\ Z \end{pmatrix} = \begin{bmatrix} f & & & 0 \\ & f & & 0 \\ & & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{2.1}$$

Expression 2.1 assumes that the principle point is located at the center of image plane. Conventionally, we use the top-left or bottom-left corner of the image plane as origin.

If the principle point is located at (p_x, p_y) respective to a conventionally defined origin of image plane, this expression becomes

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} fX + Zp_x \\ fY + Zp_y \\ Z \end{pmatrix} = \begin{bmatrix} f & & p_x & 0 \\ & f & p_y & 0 \\ & & 1 & 0 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{2.2}$$

Now, writing

$$\mathbf{K} = \begin{bmatrix} f & & p_x \\ & f & p_y \\ & & 1 \end{bmatrix} \tag{2.3}$$

then 2.2 has the concise form

$$\mathbf{x} = \mathbf{K}[\mathbf{I}|\mathbf{0}]\mathbf{X}_{\mathbf{cam}} \tag{2.4}$$

The matrix $\mathbf{K}$ is called the *camera calibration matrix* or *camera intrinsic matrix* because it describe the projection from camera coordinate frame to image plane and independent from external coordinate frame, that is world coordinate frame.

2.1.2 Camera Extrinsic Matrix

In previous section, we describe a 3D point in camera coordinate frame. However, in 3D human pose datasets, 3D points $\mathbf{X}$ are mostly likely given in world coordinate frame. The two coordinate frames are related via a rotation and a translation. See Fig. 2.2.

If $\widetilde{\mathbf{X}}$ is an inhomogeneous 3-vector representing the coordinates of a point in the world, and $\widetilde{\mathbf{X}}_{cam}$ represents the same point in the camera coordinate frame, then we may write $\widetilde{\mathbf{X}}_{cam} = \mathbf{R}(\widetilde{\mathbf{X}} - \widetilde{\mathbf{C}})$, where $\widetilde{\mathbf{C}}$ represents the coordinates of the camera center in the world coordinate frame, and $\mathbf{R}$ is 3×3 rotation matrix representing the orientation of the camera coordinate frame. This equation may be written in homogeneous coordinates as

$$\mathbf{X}_{cam} = \begin{bmatrix} \mathbf{R} & -\mathbf{R}\widetilde{\mathbf{C}} \\ 0 & 1 \end{bmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \begin{bmatrix} \mathbf{R} & -\mathbf{R}\widetilde{\mathbf{C}} \\ 0 & 1 \end{bmatrix} \mathbf{X} \tag{2.5}$$

Putting this together with (2.2) leads to the formula

$$\mathbf{x} = \mathbf{K}\mathbf{R}[\mathbf{I}| - \widetilde{\mathbf{C}}]\mathbf{X} \tag{2.6}$$

It is often convenient not to make the camera center explicit, and instead to represent he world to image transformation as $\widetilde{\mathbf{X}}_{cam} = \mathbf{R}\widetilde{\mathbf{X}} + \mathbf{t}$ In this case the camera project matrix is simply

$$\mathbf{P} = \mathbf{K}[\mathbf{R}|\mathbf{t}] \tag{2.7}$$

, where from (2.6) $\mathbf{t} = -\mathbf{R}\widetilde{\mathbf{C}}$.

The concatenated matrix $[\mathbf{R}|\mathbf{t}]$ in (2.7) is called *camera extrinsic matrix*, respect to camera intrinsic matrix. It is transformation from world coordinate frame to camera coordinate frame or called the *pose* of a camera.

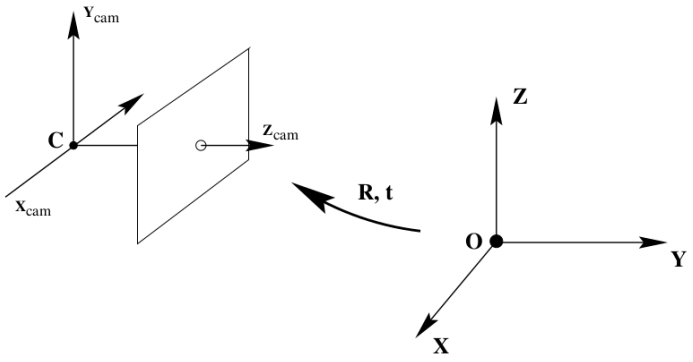


Figure 2.2: The Euclidean transformation between the world and camera coordinate frames.

2.2 Two-view Geometry

2.2.1 Epipolar Geometry

Let us denote a 3D point space as $\mathbf{X} \in \mathbb{R}^{4\times1}$ as shown in Fig. 2.3. The 3D point is imaged in two camera view, at $\mathbf{x} = \mathbf{P}\mathbf{X}$ by the first camera, and $\mathbf{x}' = \mathbf{P}'\mathbf{X}$ by the second camera, where $\mathbf{x}$ and $\mathbf{x}' \in \mathbb{R}^{3\times1}$ represent 2D points in images, $\mathbf{P}$ and $\mathbf{P}' \in \mathbb{R}^{3\times4}$ are the camera intrinsic matrix (2.3). Since both points, $\mathbf{x}$ and $\mathbf{x}'$, have the same semantic meanings because they are the same point in 3D, we can, for example, fuse their features such that each view benefits from the other view.

The epipolar geometry [14] between two views is essentially the geometry of the intersection of the image planes with the pencil of planes having the baseline as axis. The baseline is the line joining the camera centers $\mathtt{C}_1$ and $\mathtt{C}_2$. In particular, for each

location **x** in the first view, it helps us to determine the location of the corresponding point **x**′ in the second view without having to know X.

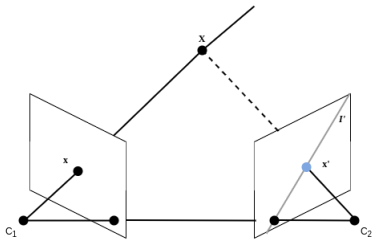


Figure 2.3: Illustration of the point-line correspondence in two views. For an arbitrary point x in one view, the corresponding point **x** in another view has to lie on the epipolar line $\mathbf{l}'$

2.2.2 Fundamental Matrix

In most cases, the 3D point **X** is unknown and, yet, we want to know where is the correspond point of **x** in the second view. We can naively search all locations in the second view, but this approach does not scale when we query more correspondent points from view 1 to view 2. Fortunately, we can compute a point-to-line relationship between two views to constrain the searching range.

The fundamental matrix is the algebraic representation of epipolar geometry. In this section, we'll only consider the case of two **calibrated** cameras. That is, the intrinsic matrices (2.3) and the poses (2.7) for both cameras are known. As illustrated in Fig. 2.3, assuming the camera centers do not overlap, the epipolar line $\mathbf{l}'$ corresponding to a given query pixel **x** = (x, y, 1) located on first view can be deterministically located on the other view as follows [14]

$$\mathbf{l}' = [\mathbf{P}'\mathbf{c}_1]_\times \mathbf{P}'\mathbf{P}^{-1}\mathbf{x} = \mathbf{F}\mathbf{x} \tag{2.8}$$

,where $[\cdot]_\times$ represents the skew symmetric matrix and **F** is the fundamental matrix.

x's corresponding point in the second view: $\mathbf{x}'$, should lie on the epipolar line: $\mathbf{l}'^T\mathbf{x}' = 0$.

2.2.3 Triangulation

To estimate the 3D location, we use the paired corresponding 2D points between two views and the camera projection matrices of each view to estimate the 3D location of a paired 2D points. The method is called **triangulation**.

Taking Fig. 2.3 as an example, we have a 3D points $\mathbf{X}$ being projected to a paired of corresponding points, $\mathbf{x}$ and $\mathbf{x}'$, by camera projection matrices $\mathbf{P}$ and $\mathbf{P}'$. Note that we are using homogeneous representation.The projections from $\mathbf{X}$ to $\mathbf{x}$ can be describe as

$$\mathbf{x} = \mathbf{PX} \tag{2.9}$$

Convert (2.9) from homogeneous to inhomogeneous equation (making the homogenous term equal to 1 in $\mathbf{x}$ and $\mathbf{X}$), we get

$$\mathbf{x} = \alpha\mathbf{PX} \tag{2.10}$$

Then, the homogeneous scale factor is eliminated by a cross product $\mathbf{x} \times (\mathbf{PX}) = 0$ because $\mathbf{x}$ and $\mathbf{PX}$ represent the same ray. Writing this out gives

$$\begin{aligned} x(\mathbf{p}^{3^T}\mathbf{X}) - (\mathbf{p}^{1^T}\mathbf{X}) = 0 \\ y(\mathbf{p}^{3^T}\mathbf{X}) - (\mathbf{p}^{2^T}\mathbf{X}) = 0 \\ x(\mathbf{p}^{2^T}\mathbf{X}) - y(\mathbf{p}^{1^T}\mathbf{X}) = 0 \end{aligned} \tag{2.11}$$

where two equations are linear independent and we have 3 unknown entries in $\mathbf{X}$ An equation of the form $\mathbf{AX} = 0$ can then be composed, with

$$\mathbf{A} = \begin{bmatrix} x(\mathbf{p}^{3^T} - \mathbf{p}^{1^T}) \\ y(\mathbf{p}^{3^T} - \mathbf{p}^{2^T}) \\ x'(\mathbf{p}'^{3^T} - \mathbf{p}'^{2^T}) \\ y'(\mathbf{p}'^{3^T} - \mathbf{p}'^{2^T}) \end{bmatrix} \tag{2.12}$$

Combining these equations we get an over determined homogeneous system of linear equations that we can solve with SVD.

2.3 Supervised Learning with Deep Convolutional Networks

2.3.1 Deep Convolutional Networks

Deep convolutional neural networks (DCNNs) have achieved state-of-the-art results in many computer vision tasks, such as image classification, object detection, semantic segmentation, human pose estimation, and so on. The strength is that DCNNs are able to learn richer representations than conventional hand-crafted representations.

Most recently-developed 2D pose estimation networks that able to reach state-of-art performance, including Convolutional Pose Machine [15], Stacked Hourglass Network [16], High-Resolution Network [17], adopt repeative convolutional modules in their network architectures. Convolution leverages three important ideas that can help improve a machinelearning system: **sparse interactions**, **parameter sharing** and **equivariant representations** [18].

sparse interactions

A conventional 2D covolution operation is define as

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n) K(m,n) \tag{2.13}$$

, where K refer to an 2D kernel and I refer to an 2D tensor. One can imaging applying 2D convolution operation is like sliding a fixed window (refer to the 2D kernel) through the 2D input tensor. See Fig. 2.4.

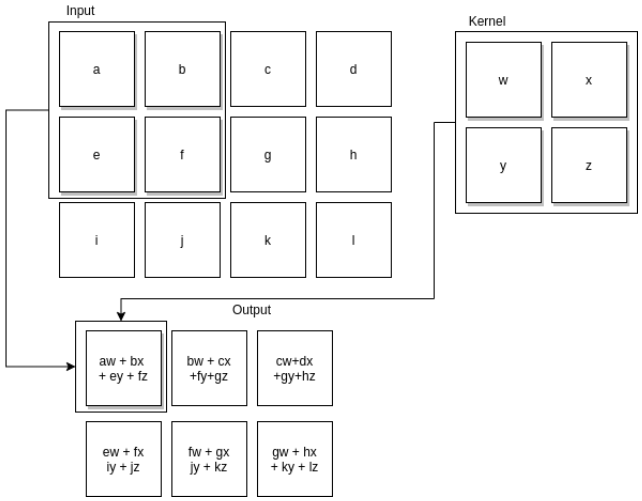


Figure 2.4: A 2D convolution. We draw boxes with arrows to indicate how the upper-left element ofthe output tensor is formed by applying the kernel to the corresponding upper-left region of the input tensor [18]

Convolutional networks typically have sparse interactions. This is accomplished by making the kernel smaller than the input. For example,when processing an image, the input image might have thousands or millions of pixels, but we can detect small, meaningful features such as edges with kernels that occupy only tens or hundreds of pixels. This means that we need to store fewer parameters, which both reduces the memory requirements of the model and improves its statistical efficiency. In a deep convolutional network, units in the deeper layers may indirectly interact with a larger portion of the input, as shown in Fig. 2.5. This allows then etwork to efficiently describe complicated interactions between many variables byc onstructing such interactions from simple building blocks that each describe only sparse interactions.

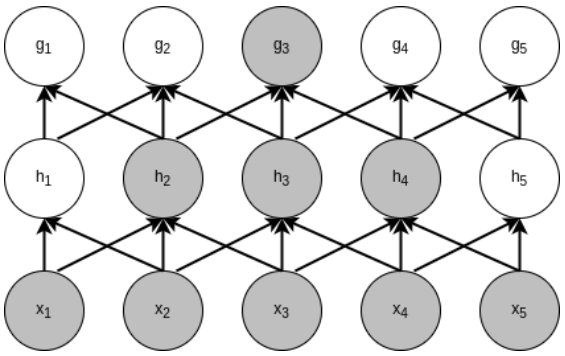


Figure 2.5: A griphical demotration of a simple 3-layer CNN with only convolutional layer. Layer **g** and **h** are formed by kernel size with width 3. The receptive field of the units in the deeper layers of a convolutional network is larger than the receptive field of the units in the shallow layers. This means that even though direct connections in a convolutional net are very sparse, units in the deeper layers can be indirectly connected to all or most of the input image [18]

.

parameter sharing

Parameter sharing refers to using the same parameter for more than onefunction in a model. This is a direct effect of the convolution, since we apply the same kernel at every location of the input tensor, shown in Fig. 2.4.

equivariant representations

In the case of convolution, the particular form of parameter sharing causes thelayer to have a property called equivariance to translation. To say a function is equivariant means that if the input changes, the output changes in the same way. Specifically, a function $f(x)$ is equivariant to a function g if $f(g(x)) = g(f(x))$. In the case of convolution, if we let g be any function that translates the input, that is, shifts it, then the convolution function is equivariant to g. In images, convolution creates a 2D map of where certain features appear in the input. If we move the object in the input, its representation will move the same amount in the output. This is useful for when we know that some function of a small number of neighboring pixels is useful when applied to multiple input locations.

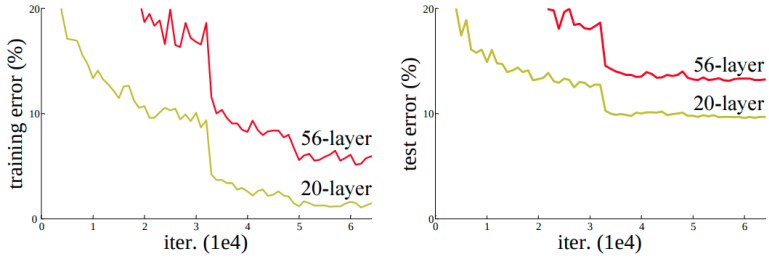


Figure 2.6: Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks (that simply stack layers). The deeper network has higher training error, and thus test error [19].

2.3.2 Residual Learning and ResNets

As deeper CNNs achieve better performance on large scale datasets, researchers observe that when deeper networks are able to start converging, a degradation problem shown in has been exposed: with the network depth increasing, accuracy gets saturated (which might be unsurprising) and then degrades rapidly. Unexpectedly, such degradation is *not caused by overfitting*, and adding more plain layers to a suitably deep model leads to higher training error, shown in Fig. 2.6.

Let us $\mathbf{H}(\mathbf{x})$ as an underlying mapping to be fit by a few stacked layer and $\mathbf{H}(\mathbf{x})$ can approximate the desired underlying function. *He et al.* [19] propose that rather than expect stacked layers to approximate $\mathbf{H}(\mathbf{x})$, they explicitly let these layers approximate a residual function $\mathbf{F} := \mathbf{H}(\mathbf{x}) - \mathbf{x}$. The layer that learns such function $\mathbf{F}$ becomes a residual learning block Fig. 2.7.The original function thus becomes $\mathbf{F}(\mathbf{x}) + \mathbf{x}$. They argue that although both forms should be able to asymptotically approximate the desired functions, but the form of residual learning might be easier and conduct empirical experiments supporting the argument.

Residual Networks or ResNets, shown in Fig. 2.8, are able to reach a deeper depth and continuously improve their accuracy compare to plain networks without residual learning. Moreover, deeper ResNets outperform the previously state-of-art VGG nets [20] with significantly less parameters [19]. One reason that explain the improved learning capabilities of ResNet might be given by a research [21], that observes residual connections promote flat minimizers, shown in Fig. 2.9, and prevent the transition to chaotic behavior, which helps explain why skip connections are necessary for training extremely deep networks.

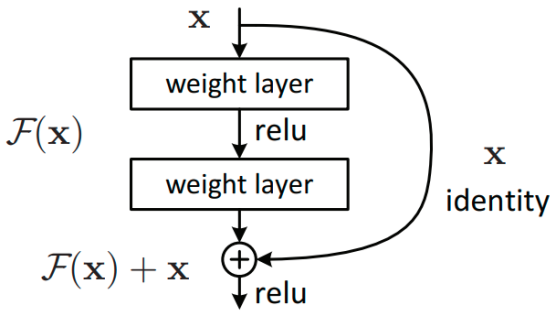


Figure 2.7: An illustration of residual block: the task for the stacked layer is to learn the residual representation $\mathbf{F}(\mathbf{x})$ [19]. The curved connection is commonly called *residual connections* or *skip connections* interchangeably nowadays.

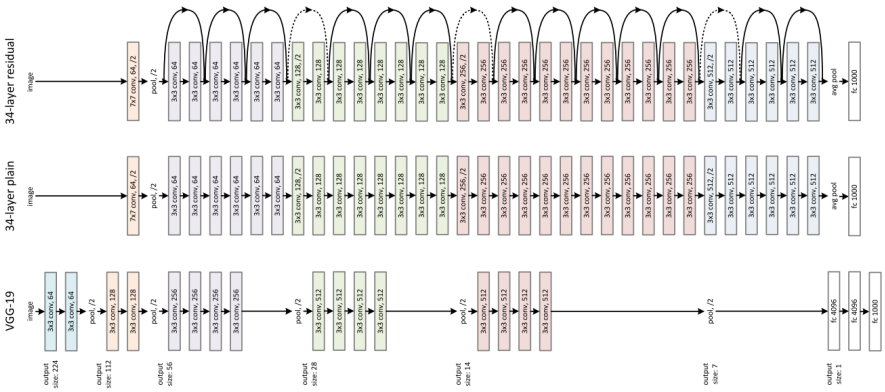


Figure 2.8: Example network architectures for ImageNet. Top: a residual network with 34 parameter layers (3.6 billion FLOPs). The dotted shortcuts increase dimensions. Middle: a plain network with 34 parameter layers (3.6 billion FLOPs). Bottom: the VGG-19 model [20] (19.6 billion FLOPs) as a reference. [19]

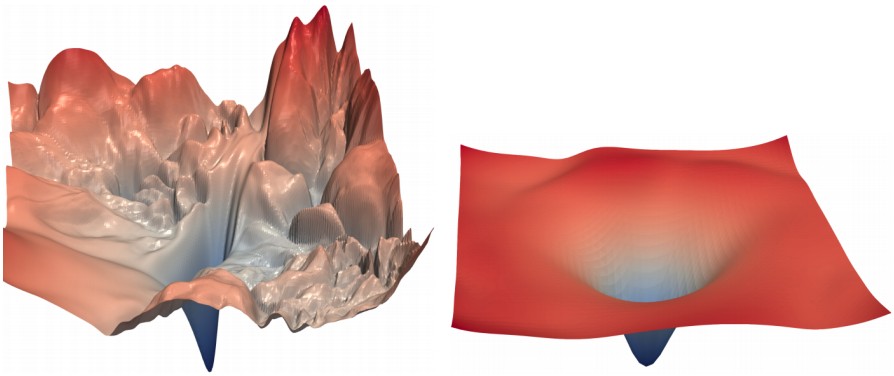


Figure 2.9: The loss surfaces of ResNet-56 (left) without residual connections (right) with residual connections [21].

2.3.3 3D U-Net

3D U-Net [22] is a network for volumetric segmentation that learns from sparsely annotated volumetric images. See Fig. 2.10 The network is based on its predecessor that takes 2D images as input, U-Net [23], but takes 3D volumes as input and processes them with corresponding 3D operations, in particular, 3D convolutions, 3D max pooling, and 3D up-convolutional layers. Using 3D kernels allow the network to learn dependecies across the volumetric structures in data.

The network is composed by two stages: contracting stage and a expansive stage (or an analysis and a synthesis path in [22]). In contracting stage, channel size of the input is double, but the width and height are shrink to half by mas pooling, leading the receptive field of output unit to grow fast. In expansive stage, $2 \times 2 \times 2$ transpose convolution kernels with stride two are used to double the dimiension before proceeding to the next layer. In addition, shortcut connections from layers of equal resolution in the contracting stage provide the essential high-resolution features to the synthesis stage.

2.3.4 Transfer Learning

DCNNs proof to be successful in demonstrating high accuracy in various tasks. In practice, however, very few people train an entire DCNN from scratch (with random initialization) using the specific dataset for the task, because it is relatively rare to have a dataset of sufficient size. Instead, it is common to pretrain a DCNN on a very large dataset (e.g. ImageNet, which contains 1.2 million images with 1000 categories), and then use the DCNN either as an initialization or a fixed feature extractor for the task of

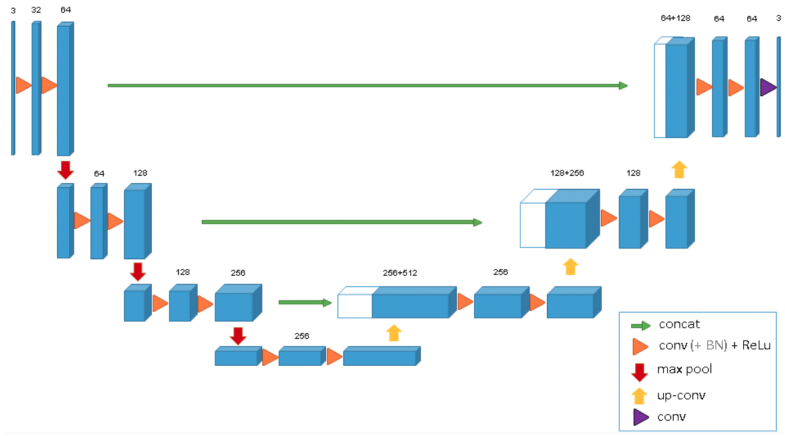


Figure 2.10: 3D U-Net architecture

interest. This is called *transfer learning*

There several tricks in conducting transfer learning:

- **DCNN as fixed feature extractor** Take a DCNN pretrained on ImageNet, remove the last fully-connected layer (this layer's outputs are the 1000 class scores for a different task like ImageNet), then treat the rest of the DCNN as a fixed feature extractor for the new dataset. In an AlexNet, this would compute a 4096-D vector for every image that contains the activations of the hidden layer immediately before the classifier. We call these features CNN codes. It is important for performance that these codes are ReLUd (i.e. thresholded at zero) if they were also thresholded during the training of the DCNN on ImageNet (as is usually the case). Once you extract the 4096-D codes for all images, train a linear classifier (e.g. Linear SVM or Softmax classifier) for the new dataset.

- **Fine-tuning the DCNN** The second strategy is to not only replace and retrain the classifier on top of the DCNN on the new dataset, but to also fine-tune the weights of the pretrained network by continuing the backpropagation. It is possible to fine-tune all the layers of the DCNN, or it's possible to keep some of the earlier layers fixed (due to overfitting concerns) and only fine-tune some higher-level portion of the network. This is motivated by the observation that the earlier features of a DCNN contain more generic features (e.g. edge detectors or color blob detectors) that should be useful to many tasks, but later layers of the DCNN becomes progressively more specific to the details of the classes contained in the original dataset. In case of ImageNet for example, which contains many dog breeds, a significant portion of the representational power of the DCNN may

be devoted to features that are specific to differentiating between dog breeds.

• **Pretrained models** Since modern DCNNs take 2-3 weeks to train across multiple GPUs on ImageNet, it is common to see people release their final DCNN checkpoints for the benefit of others who can use the networks for fine-tuning. For example, the Caffe library has a Model Zoo where people share their network weights.

3 Related Works

In 3D multi-person human pose estimation, there are two challenging association problems in this task. First, it needs to associate the joints of the same person by either top-down or bottom-up strategies. Second, it needs to associate the 2D poses of the same person in different views based on appearance features which are unstable when people are occluded. We will show the pros and cons of both top-down and bottom-up approaches that tackle the mentioned association problems.

3.1 Top-down Multi-person 3D Pose Estimation

In top-down methods, humans are first detected, through bounding boxes, in each view and 2D pose is estimated for each bounding box. Then, bounding boxes are associated with individuals across multiple views. After the association, estimated 2D poses are lifted to hypothesized 3D poses, where each individual might have more than one hypothesized 3D poses that need to be refined later. Finally, hypothesized 3D poses are refine, based on heuristic, 3DPS etc., to a single 3D pose for each individual.

3.1.1 Single View

Rogez *et al.* [2] estimate multiple 3D humans poses with single-view natural images using a learning-based framework. They cast the task of estimating multiple 3D human poses to a combination of classification and a regression problem. In their work, they store a set of anchor 2D-3D poses, denoted by $(p, \mathbf{P})$, that collected from the MoCap dataset. In their framework, an image is first fed into a localization network that outputs a set of bounding boxes, where each bounding box B contain a fixed set of predefined anchor poses with labels $c_B \in \{0 \dots K\}$ (0 is the background class). The scale of anchor poses is normalize according to the size of bounding boxes, making the regression independent of the scale and position of the person in the image. After the localization network, an RoI pooling layer is attached and it is connected by two consecutive fully-connected networks: (1) *pose classification network* and (2) *pose regression network*. *The classification network* aims at predicting the closest anchor-pose, i.e., the correct label, for each bounding box B. In other words, each bounding box B is assigned with distribution u among $K+1$ anchor poses. *The regression network* contains $K+1$

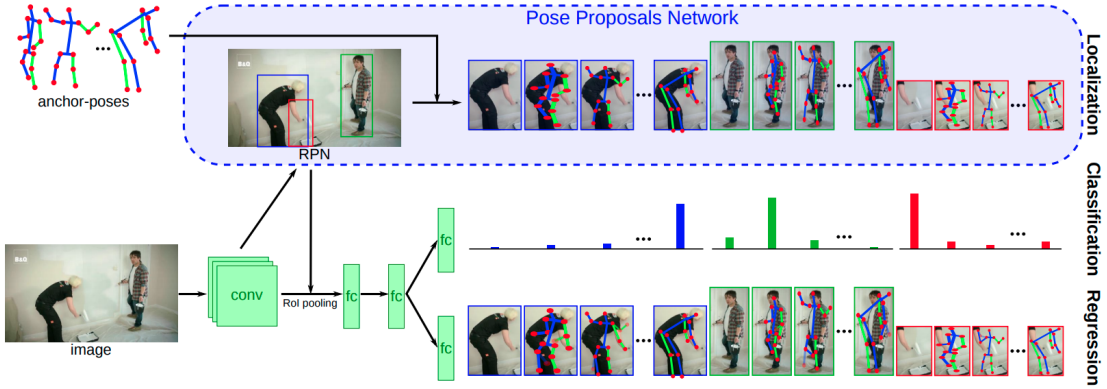


Figure 3.1: Overview of the [2] approach. It first extract candidate regions using a Region Proposal Network (RPN) and obtain pose proposals by placing a fixed set of anchor-poses into these boxes (top). These pose proposals are then scored by a classification branch and refined using class-specific regressors, learned independently for each anchor-pose.

regressors, learned independently for each anchor pose. The regression outputs v, that has a dimension of $5 \times J \times (K+1)$ (5 because of 2D + 3D coordinates) for each B in one forward pass. See Fig. 3.1 for the pipeline. The final 2D and 3D poses are the weighted averages on v using the probability of anchor poses u.

The advantage of this work is the network can outputs full 2D and 3D poses even when the humans are partially occluded because anchor poses are full human body skeletons. However, the diversity of estimated human poses depending on the number of anchor poses. In addition, the optimization for reliable 2D and 3D poses from anchor poses at pose regression is not convex.

3.1.2 Multiple Views

Dong *et al.* [8] first detects p_i number of bounding boxes in *i-th* view from an off-the-shelf 2D human pose detector [24]. To associate bounding boxes across views, they formulate the task as an optimization problem, that looks for an optimal permutation matrix $\mathbf{P}$ maximizes the corresponding affinities $\mathbf{A}$. Entries of the affinity matrix $\mathbf{A_{ij}} \in \mathbb{R}^{p_i \times p_j}$ are scores that composed by the appearance similarity and the geometric compatibility between bounding boxes. Specifically, the appearance scores are calculated from the euclidean distance between a pair of feature vectors, that generated by feeding the bounding boxes into a pre-trained re-ID model [25]. Besides appearance, another important cue to associate two bounding boxes is that their associated 2D poses should

be geometric consistent. More specifically, the corresponding 2D joint locations should satisfy the epiplolar constraint (2.8), i.e. a pair of corresponding joints should lie on the epipiolar lines associated with one and the other. Thus, the distance of joint 2D location $\mathbf{x_j}$, from *j-th* view, should be as close as possible to the epipolar line L_{ij} associated with $\mathbf{x_i}$ from the *i-th* view and vice versa. While solving the optimal permutation matrix $\mathbf{P_{ij}}$ for maximizing $\langle \mathbf{P_{ij}}, \mathbf{A_{ij}} \rangle$ between a pair of view can be done by Hungarian algorithm, when there are multiple views, solving the matching problem separately for each pair of views ignores the cycle-consistency constraint and may lead to inconsistent results. The all-view permutation matrix $\mathbf{P}$, composed by $\mathbf{P_{ij}}$, is an optimal solution of $\langle P_{ij}, A_{ij} \rangle$ with cycle consistency condition [26]. Finally, they solved 3D poses, from multiple hypothesized 3D poses estimated from paired 2D poses in different pairs of views, with 3DPS. As the number of views grows, however, the time complexity of using 3DPS to estimate 3D pose increases exponentially [9].

Chen *et al.* [9] formulate 3D human poses estimation as a tracking task across multiple views and timestamps. The approach has two stages: initialization and tracking stage. In tracking stage, given a pair of target (associated 2D and 3D poses in previous timestamps) and detection at the different time frames, they combine 2D and 3D geometric correspondences. For 2D correspondences, they use the euclidean distance between the 2D pose of target and detection as affinity measurements. For 3D correspondences, they back-project detection's 2D joint location to rays in 3-space and measures point-to-ray euclidean distance between target and detection. See Fig. 3.3. When initialization, they use [8] approach. To estimate 3D poses, they use a weighted triangulation with a penalty rate depending on timestamps. Since they use only triangulation for estimating 3D poses, the approach can reach realtime and time complexity only grow linearly with the number of cameras.

3.2 Bottom-up Multi-person 3D Pose Estimation

In bottom-up methods, joints are estimated first and ,then, individuals are associated through some graph algorithms.

3.2.1 Single View

Metha *et al.* [1] regress the 3D coordinates of joints directly from a single-view image with DCNN. To tackle occlusion, they treat the rigid skeleton of the human body as a kinematic tree and embed 2D locations of sibling joints and root joint to the entries of feature maps called OPRM. Thus, joints are implicitly chained with siblings and the root joint. With the embedding, the DCNN learns not only how to predict 3D coordinate of joint but also the relation between joints while training on OPRM. To

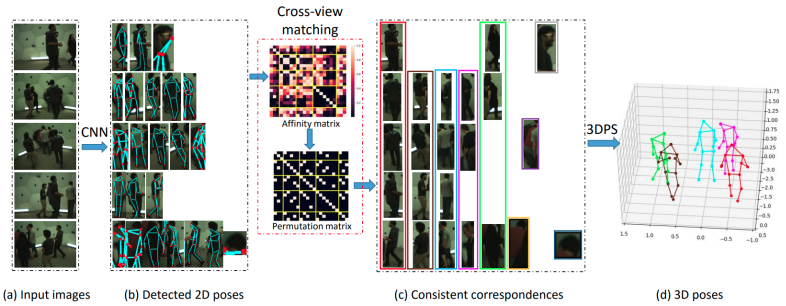
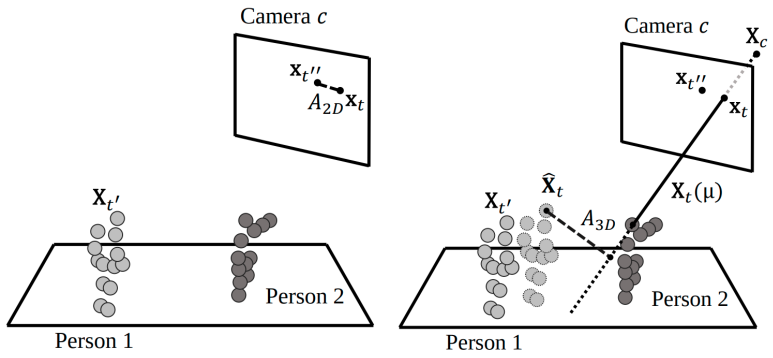


Figure 3.2: Overview of the [8] approach. (a), an off-the-shelf human pose detector is used to produce 2D bounding boxes and associated 2D poses in each view, which may be inaccurate and incomplete (b). Then, the detected bounding boxes are clustered. Each resulting cluster includes the bounding boxes of the same person in different views (c). The isolated bounding boxes that have no matches in other views are regarded as false detections and discarded. Finally, the 3D pose of each person is reconstructed from the corresponding bounding boxes and associated 2D poses (d).



(a) 2D correspondence (b) 3D correspondence

Figure 3.3: 2D and 3D correspondence affinity measurments in [9]. (a) 2D correspondence is computed within the same camera. (b) 3D correspondence is measured between the predicted location and the projected line in 3-space.

associate individuals, the DCNN predict part affinity field and joint heatmaps as well. To predict 3D coordinate of joints, they traverse the kinematic tree from root node, to get the base 3D pose, and from edge node, to get a more versatile 3D poses. The approach is able to reach real-time performance but the 3D poses suffers from error accumulation. In addition, the approach is not robust to occlusions , predicting unreliable 3D poses when two edge joints from different individuals are in proximity or occluded.

3.2.2 Multiple Views

[3, 6] project 2D joint locations in all camera view into a common voxelized 3-space, where a projected 2D point become a ray in the voxelized 3-space. [3, 6] use 3D CNN to aggregate rays projected by same type of joints but from different views. Further, Tu *et al.* [6] use 3D CNN to predict cuboids that coarsely localize people, which can be consider as associating joints with individuals in 3-space with a learning-based method. See Fig. 3.4. The voxel-based approaches reach state-of-the-art, but these approaches require a fixed 3D grid that covers the whole environment. Thus, it only suits studio environments.

On one hand, approaches [4, 11] propose to fuse joint heatmaps from multiple source views to a single reference view, solving occlusions issue by aggregate 2D joint predictions from other views and localize 2D joints as accurate as possible. Comparing with [3, 6], these approaches does not work as good as voxel-based approach but are less computational demanding.

3.3 Temporal Fusion

[27, 28] has shown that fusing 3D joints from multiple timestamps can help learning-based methods achieve better results. Since human motion is the result of both physical limitations (torque exerted by muscles, gravity, moment preservation)) and the intentions of subjects, a learning-based method is able to pick these human motion dynamics to improve the smoothness of pose prediction. Pavllo *et al.* [28] use a ResNet-like architecture to estimate single-person 3D pose in YouTube video. The convolutional operation at each convolutional layer operate at temporal domain of the input. Moreover, the convolution kernel has dilation that increase with the depth of layer, drastically increasing the receptive field of hidden units at each layer. The residual connection connects neighboring layers between their inputs, so later layers have access to a higher temporal resolution from earlier layers. With only 4 layers, the hidden unit at output has a receptive field of 273 frames, generating temporal coherent single-person 3D pose from videos.

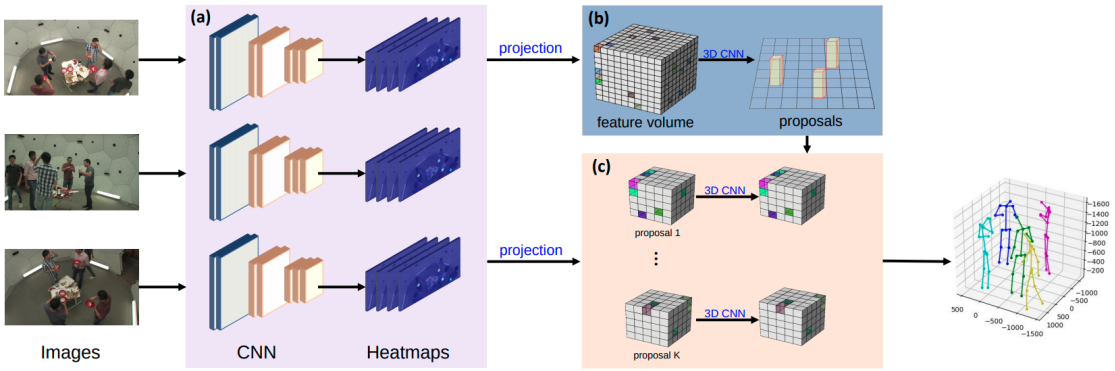


Figure 3.4: Overview of the [6] approach. (a) it first estimate 2D pose heatmaps for all views; then, (b) it warps the heatmaps to a common 3D space and construct a feature volume which is fed into a Cuboid Proposal Network to localize all people instances; (c) for each proposal, it construct a finer-grained feature volume and estimate a 3D pose.

3.4 Brief Conclusion

In top-down approaches, time and space complexity in estimating human poses grows as number of people and view increased, since each 2D poses must be estimated for each bounding box and in each view first. On the contrary, bottom-up approaches estimate 2D locations of joints, by means of feature maps, that has a fixed time and space complexity regardless number of people in each view. The complexity lies in the graph-based method or voxel-based method, which associating joints with individual across views. The voxel-based approaches suits only for constraint environments because it needs a dense 3D grid that cover the environments, refraining it from general applications. In our opinion, we prefer multi-view approaches over the single-view approaches because multi-view approaches are often more accurate and has less ambiguities in estimating 3D poses when multi-view data being available. In addition, we favor graph-based method that associate on 2D space for more general applications than voxel-based methods that suit for constraint environments.

To this end, we briefly introduce our framework for multi-person 3D pose estimation can be divided into two parts: in the first stage, we trained our own 2D joint heatmaps estimator that composed of several modules inspired by [11, 4] multi-view fusion, [28] for temporal fusion. All the modules are fully convolutional and can be trained jointly. In the second stage, we associate 2D pose using [29], a graph-based greedy method in associating 3D poses. Our framework is lightweight since it operate in 2D space and can be extended into an end-to-end training scheme.

4 Datasets

In this work, we use two different datasets to train and evaluate the performance of our framework. We use Microsoft Common Objects in Context (COCO) Dataset [30] for our 2D joints localization backbone and CMU Panoptic Dataset [31] for both the view-fusion and the temporal-fusion module.

4.1 COCO 2017 Keypoint Dataset

COCO is a large-scale object detection, segmentation, and captioning dataset. It is one of the best image datasets available, so it is widely used in cutting edge image recognition artificial intelligence research. It is used in open-source projects such as Facebook Research's Detectron. In our thesis, we use a subset of it from 2017 Keypoint Challenge, which contains masks, keypoint, and bounding boxes for the "person" class. See Fig. 4.1 for an overview of the most relevant part of the dataset to us

4.1.1 Annotation Format

The COCO dataset is formatted in JSON and is a collection of "info", "licenses", "images", "annotations", "categories" (in most cases), and "segment info" (in one case).

Listing 4.1: Overview of JSON file in COCO 2017 Dataset

```
{
"info": {...},
"licenses": [...],
"images": [...],
"annotations": [...],
"categories": [...], <-- Not in Captions annotations,
"segment_info": [...] <-- Only in Panoptic annotations
}
```

We'll explain *annotation* in detail, since we only use this for our work (in addition to the images).

Figure 4.1: Overview of 2017 COCO Keypoint. (a) Images with human (b) un-occluded (colored)/occluded (white) joint locations and limb (colored line), where green means left part of the body and red means right part of the body (c) masks for individuals

Person Annotation

The annotation section contains a list of object with annotation infomation, see Listing 4.2.

Listing 4.2: Overview of Person Class Annotation

```
"annotations": [
{
"segmentation": [[204.01,306.23,...206.53,307.95]],
"num_keypoints": 15,
"area": 5463.6864,
"iscrowd": 0,
"keypoints": [229,256,2,...,223,369,2],
"image_id": 289343,
"bbox": [204.01,235.08,60.84,177.36],
"category_id": 1,
"id": 201376
}
]
```

The most important part for us are the iscrowd, segmentation, keypoints. We'll explain the meaning of the field:

- **segmentation** , which are usually a list of polygon vertices around the object, but can also be a run-length-encoded (RLE) bit mask. Typically, RLE is used for groups of objects (like a large stack of books). We convert mask from a common polygon representation into a binary image $\mathbf{M}$ shown in Fig. 4.1 (c)
- **iscrowd** specifies whether the segmentation is for a single object or a group/cluster of objects. In our work, we do not include images with "person" class and iscrowd with value 1.
- **keypoints** are a list containing predefined 2D joint locations on the image of the i-th individual. A list of keypoints has a length of 51, where every 3 elements can be grouped into a vector $\mathbf{x}_k^i \in \mathbb{R}^3$, specifically

$$\mathbf{x}_k^i = [x, y, v]^T, v = \begin{cases} 0 \text{ , } not\ annotated \\ 1 \text{ , } annotated\ but\ occluded \\ 2 \text{ , } annotated\ and\ visible \end{cases} \tag{4.1}$$

, where *not annotated* points have x and y set to 0. See Fig. 4.1 (b) for examples of $v = 1$ and $v = 2$

Table 4.1: (left) COCO Keypoint and (right) limb definition.

index	Joint Name	Definition
1	nose	-
2	eye_l	left eye
3	eye_r	right eye
4	ear_l	left ear
5	ear_r	right ear
6	sho_l	left shoulder
7	sho_r	right shoulder
8	elb_l	left elbow
9	elb_r	right elbow
10	wri_l	left wrist
11	wri_r	right wrist
12	hip_l	left hip
13	hip_r	right hip
14	kne_l	left knee
15	kne_r	right knee
16	ank_l	left ankle
17	ank_r	right ankle

index	limb
1	(ank_l, kne_left)
2	(kne_l, hip_l)
3	(ank_r, kne_r)
4	(kne_r, hip_r)
5	(hip_l, hip_r)
6	(sho_l, hip_l)
7	(sho_r, hip_r)
8	(sho_l, sho_r)
9	(sho_l, elb_l)
10	(sho_r, elb_r)
11	(elb_l, wri_l)
12	(elb_r, wri_r)
13	(eye_l, eye_r)
14	(nose, eye_l)
15	(nose, eye_r)
16	(eye_l, ear_l)
17	(eye_r, ear_r)
18	(ear_l, sho_l)
19	(ear_r, sho_r)

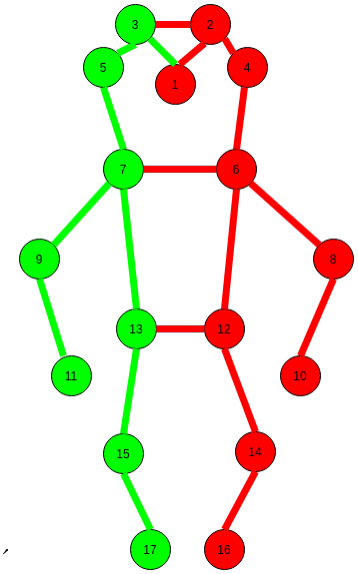


Figure 4.2: COCO provide a reference skeleton (facing toward the reader) that connecting the joints. For our convenience in visualization, we define joints that belong to the left part of the body, colored with red, and to the right part of the body, colored with green.

Figure 4.3: (left column) The image with a person class label in COCO annotation and (right column) the joint heatmap $\mathbf{H}_k$. Note that, we merge all K channels of joint heatmap to a single channel, in order to visuaulize compactly

4.1.2 Joint Heatmaps

To estimate joint locations, we need a target that can feed into the CNN. Like many previous works, we use 2D heatmaps that indicating joint locations on images. For each joint $\mathbf{x}_k$, defined in Table. 4.3, we define a correspond heatmap $\mathbf{H}_k$, where k denotes the k-th joint.

Recall that an individual's 2D joint $\mathbf{x}_k^i$ in (4.2) is defined as a 3-element vector with the first two elements indicating x and y coordinates, and the last element indicating visibility. The heatmap of an individual $\mathbf{H}_k^i$ is generated by placing a 2D Gaussian distribution at the joint location

$$\mathbf{H}_k^i(\mathbf{x} \mid \mathbf{x}_k^i) = \begin{cases} \exp\left(-\frac{(x-x_k^i)^2+(y-y_k^i)^2}{2\pi\sigma^2}\right), & \text{if} \quad x_k^i - 3\sigma \le x \le x_k^i + 3\sigma \cap \\ & \quad y_k^i - 3\sigma \le y \le y_k^i + 3\sigma \cap \\ & \quad v_k^i > 0 \\ 0, & \text{otherwise} \end{cases} \tag{4.2}$$

Finally, a joint heatmap $\mathbf{H}_k$ that given N individuals in an image is

$$\mathbf{H}_k(\mathbf{x} \mid (\mathbf{x}_k^1, \ldots, \mathbf{x}_k^N)) = \max\left(\mathbf{H}_k^1(\mathbf{x} \mid \mathbf{x}_k^1), \ldots, \mathbf{H}_k^N(\mathbf{x} \mid \mathbf{x}_k^N)\right) \tag{4.3}$$

, shown in Fig. 4.3.

4.2 CMU Panoptic Dataset

CMU Panoptic [31] is a dataset containing images with several people performing different scenarios (playing an instrument, dancing, etc.) in a dome where several cameras are placed. The dataset provides synchronized videos from 480 VGA views and 31 HD views. All the sequences are annotated with estimated 3D joint locations and skeletons from using off-the-shelf 2D pose estimators and consensus voting method across view [31]. Even **without ground-truth annotations** of dataset, many state-of-art multi-person 3D pose researches [6, 9, 5, 3] are able to train and evaluate because of the large number of views.

4.2.1 Annotation Format

For each multi-view sequence, the dataset provides extrinsic matrices $\mathbf{K}$ and intrinsic matrices $[\mathbf{R} \mid \mathbf{t}]$ of both VGA and HD cameras. They also provide 3D joint locations $\mathbf{x}_{k_{3D}}^i$ for every individual with COCO-style annotation. One can project 3D joint location

$\mathbf{x}^i_{k_{3D}}$ to 2D joint location $\mathbf{x}^i_k$, using the provided camera matrices with (2.6). However, the dataset does not provide any 2D annotation for occluded joints in a different view and, thus, one is not able to know if $\mathbf{x}^i_k$ being visible on the image or not, shown in Fig. 4.4

Listing 4.3: Overview of Camera Annotation

```
{
"calibDataSource": "160223_calib_tent13_norm",
"cameras": [
{
"name": "01_01",
"type": "vga",
"resolution": [640,480],
"panel": 1,
"node": 1,
"K": [...],
"distCoef": [...],
"R": [...],
"t": [...]
},...
]
}
```

Table 4.2: (left) COCO Keypoint and (right) limb definition.

4.3 Data Augmentation

Data augmentation has been a common practice in training deep neural network. In this work, we adopt image argumentation for COCO Dataset and augmentation on intrinsic camera matrices for CMU Panoptic Dataset.

4.3.1 Image Augmentation

A set of image augmentations are applied on both COCO, see Fig. 4.5, and CMU Panoptic. For augmentating CMU dataset, we must update camera projection matrix **P**

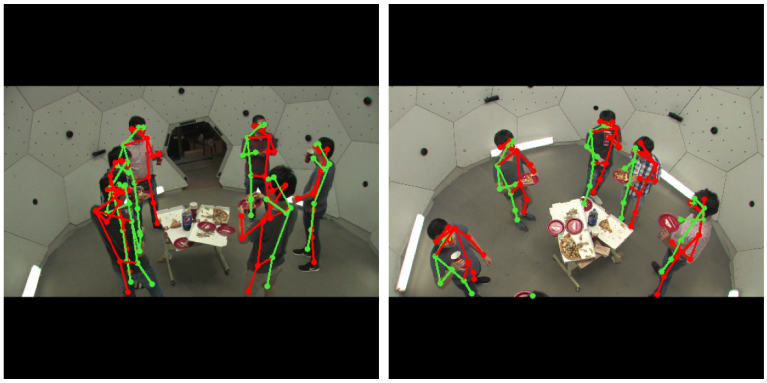


Figure 4.4: CMU Panoptic Overview. Note that it does not provide 2D annotation about the visibility of projected 2D joints.

(2.7) with an affine transformation parameterized by $\mathbf{A} \in \mathtt{R}^{2\times 2}$ and $\mathbf{b} \in \mathtt{R}^2$.

$$\mathbf{P} := \begin{bmatrix} \mathbf{A} & \mathbf{b} \\ \mathbf{0^T} & 1 \end{bmatrix} \mathbf{P} \tag{4.4}$$

For example, scaling of a projection matrices, suppose the input image is spatially down-sampled s_x and s_y times along the x-axis and y-axis, the projection matrix $\mathbf{P}$ is updated as follows

$$\mathbf{P} := \begin{bmatrix} 1/s_x & 0 & (1-s_x)/2s_x \\ 0 & 1/s_y & (1-s_y)/2s_y \\ 0 & 0 & 1 \end{bmatrix} \mathbf{P} \tag{4.5}$$

The coordinates are aligned with the center of pixels rather than the top-left corners. We can verify if we update the camera projection matrix correctly by observing the whether corresponding epiploar lines passing through the keypoint from the same person, see Fig. 4.6.

4.3.2 Joint Augmentation

We increased the total number of joints from 17 to 55, by interpolating two intermediate joints on each limb defined in 4.3. The intuition behind this is to use redundancy of joints to mitigate the effect of occlusion. These intermediate joints are placed at locations that are $0.1\times$*limb length* away from both ends of the limb.

Table 4.3: Augmentations used in training.

Dataset	rotation	translation	resizing	left-and-right flip
COCO	$[-60, 60]$	$[-50px, 50px]$	$[0.7\times, 1.3\times]$	50%
CMU Panoptic	$[-60, 60]$	$[-50px, 50px]$	$[0.7\times, 1.3\times]$	-

Figure 4.5: Data augmentation for COCO Dataset

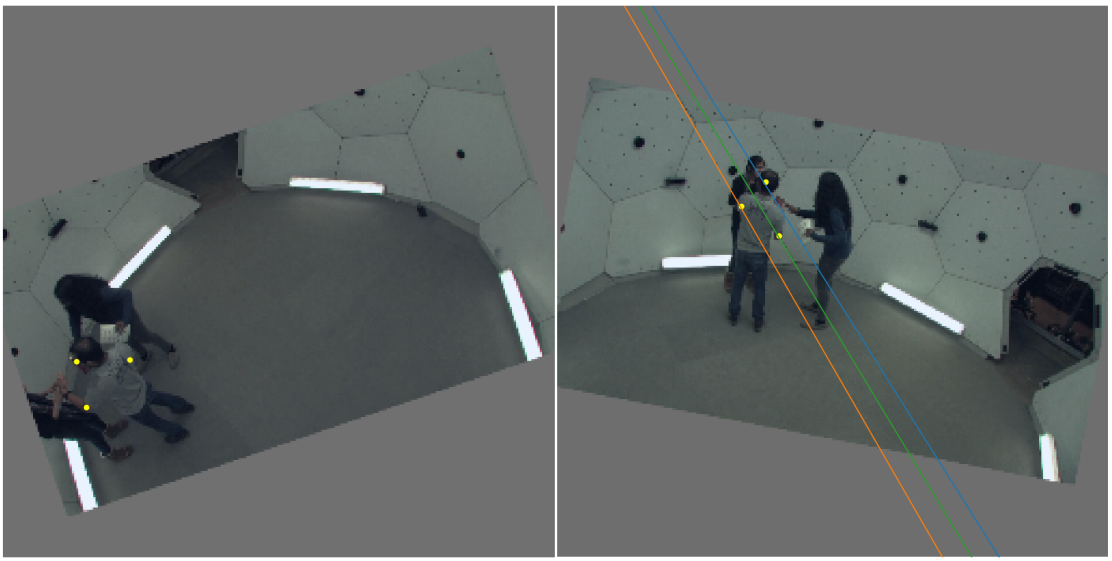


Figure 4.6: (left) keypoints from one view (right) correspond epiploar lines and the same keypoints in other view. Note that epiploar lines pass through the same keypoints in different views, verifying the camera projection matrix is updated correctly.

5 Fusion Network and 3D Pose Estimation

In this chapter, we introduce our learning-based framework, that is fully constitutional, for bottom-up 3D human pose estimation.

5.1 2D Backbone Module: ResNets

Our 2D backbone is a fully convolution network (FCN) based on ResNet50 that pretrain on COCO for the task of segmentation [32]. Similar to [33], we retrain the last convolutional layer ***C*4** of the ResNet50 backbone and connect it with a deconvolution head ***D***, shown in Fig. 5.1, to increase the resolution and reduce the number of channels form pretrained model. For each convolutional operation, a ReLU and a batch normalization operator are followed.

We used Adam optimizer and train on COCO dataset. We fixed the pretrained model, except ***C*4**. ***C*4** and ***D*** are trained together but with different learning rate with 0.0001 and 0.01. The smaller learning rate for ***C*4** is given because we only want to fine tune this layer for the 2D joint heatmap regression task. In addition, we do not crop and feed the network with regions with person bounding boxes, like many other 2D backbone being used in many other works. However, joints usually occupy a relative small areas in the images and overwhelm with negative examples, such as background. To effectively train our 2D backbone, we feed the down-sampled images from COCO and calculated a weighted loss with high penalty rate on places with joints. Recall (5.1) and the weighted L2 Loss L is calculated as

$$L = \sum_{k=1}^{J} \left\| \mathbf{W}_k \odot \left(\mathbf{H}_k - \mathbf{H}_{k_{GT}} \right) \right\|^2 \tag{5.1}$$

where

$$\mathbf{W}_k(\mathbf{x}) = \begin{cases} 10, \text{ if } & \mathbf{H}_{k_{GT}}(\mathbf{x}) \geq \lambda \\ 1, & \text{otherwise} \end{cases} \tag{5.2}$$

We set λ to 0.5 empirically, since we observe better quality in prediction.

layer index	trans. conv2d	conv2d.
1	2048, 1024, (4, 4), (2, 2), (0, 0)	1024, 512, (3, 3), (1, 1), (0, 0)
2	512, 256, (4, 4), (2, 2), (0, 0)	256, 128, (3, 3), (1, 1), (0,0)
3	-	128, 64, (3, 3), (1, 1), (1, 1)
4	-	64, 55, (3, 3), (1, 1), (1, 1)

Table 5.1: Structure of deconvolution head. We list out the parameters of 2d convolutional operation in the following order: input channels, output channel, kernel size, stride, and padding

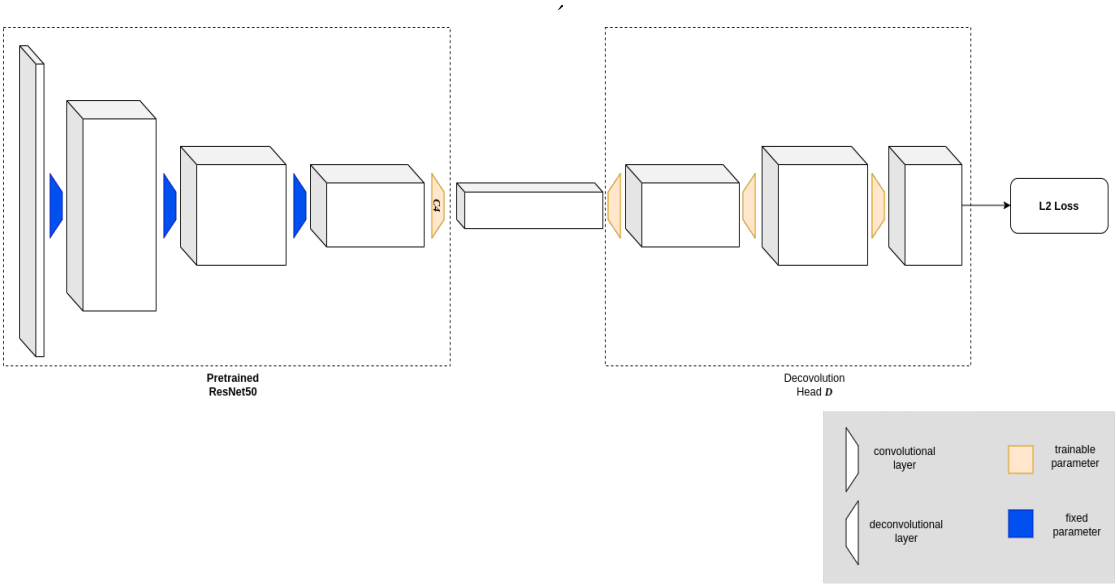


Figure 5.1: Overview of our 2D backbone. We train the last convolutional layer of the pretrained ResNet50 with the deconvolutional head together.

5.2 Sampler Module: Epipolar Sampler

A 2D joint represented by heatmap $\mathbf{H}_k^{(v)}(\mathbf{x}^{(v)})$ from view v might be occluded or has a low score due to lack of training examples (e.g. novel view angle). Is there a way to retrieve the heatmap score $\mathbf{H}_k^{(v')}(\mathbf{x}^{(v')})$ at the corresponding point $\mathbf{x}^{(v')}$ in other view v', which the joint is has a higher score? According to (2.8), we can estimate a point-to-line relationship for a pair of views if we know the extrinsic and intrinsic matrices of both views. If we sample enough points along the epipolar line on the joint heatmap $\mathbf{H}_k^{(v')}$, the sampled point with the highest score can the corresponding joint in another view, demonstrating Fig. 5.2 (a) and (b).

The epipolar sampler E outputs the maximum scores on the corresponding epipolar line of the joint heatmap. To explain in detail, given a heatmap score $\mathbf{H}_k^{(ref)}(\mathbf{x}^{(ref)})$ at pixel location $\mathbf{x}^{ref}$, where superscript ref means at *reference view* and subscript k means k-th joint index, and the projection matrices, $\mathbf{P}_{src}$ and $\mathbf{P}_{ref}$, of source and reference view. We can sample a set of N points $S^{(src)} = \{\mathbf{x}_1^{src}, \ldots, \mathbf{x}_N^{src}\}$ on the corresponding epipoloar line $\mathbf{l}$ at $\mathbf{H}_k^{(src)}$ from source view. We compare N scores $\mathbf{H}_k^{(src)}(S^{(src)}) = \left\{\mathbf{H}_k^{(src)}(\mathbf{x}_1^{src}), \ldots, \mathbf{H}_k^{(src)}(\mathbf{x}_N^{src})\right\}$ on the epipolar line and retrieve the biggest one, see Fig. 5.2 (b). Specifically,

$$E(\mathbf{x}_{ref}, \mathbf{P}_{ref}, \mathbf{P}_{src}) = \max\left(\left\{\mathbf{H}_k^{(src)}(\mathbf{x}_1^{src}), \ldots, \mathbf{H}_k^{(src)}(\mathbf{x}_N^{src})\right\}\right) \tag{5.3}$$

We can use E for every pixel on the reference view and create a heatmap $\mathbf{H}_{ep}(\mathbf{x}_{ref})$ in the image space of reference view but contains the information from source view, shown in Fig. 5.2 (c). Moreover, we can integrate more source views rather than just one, shown in Fig. 5.2 (d).

5.3 View Fusion

The architecture of view fusion network is based on UNet. A fusion network, where the convolutional layer is compose by 2D convolutional kernels and the input is the heatmap $\mathbf{H}_{ep}$ gernerated by epipolar sampler E .

5.3.1 View Fusion Model

The view fusion model takes heatmap $\mathbf{H}_{ep}$ and concatenating with heatmap $\mathbf{H}^{ref}$ from reference view, thus the number of channels at the input is 110 (channels from reference view 55 plus 55 channels from epipolar sampler). In additon, the number of parameters

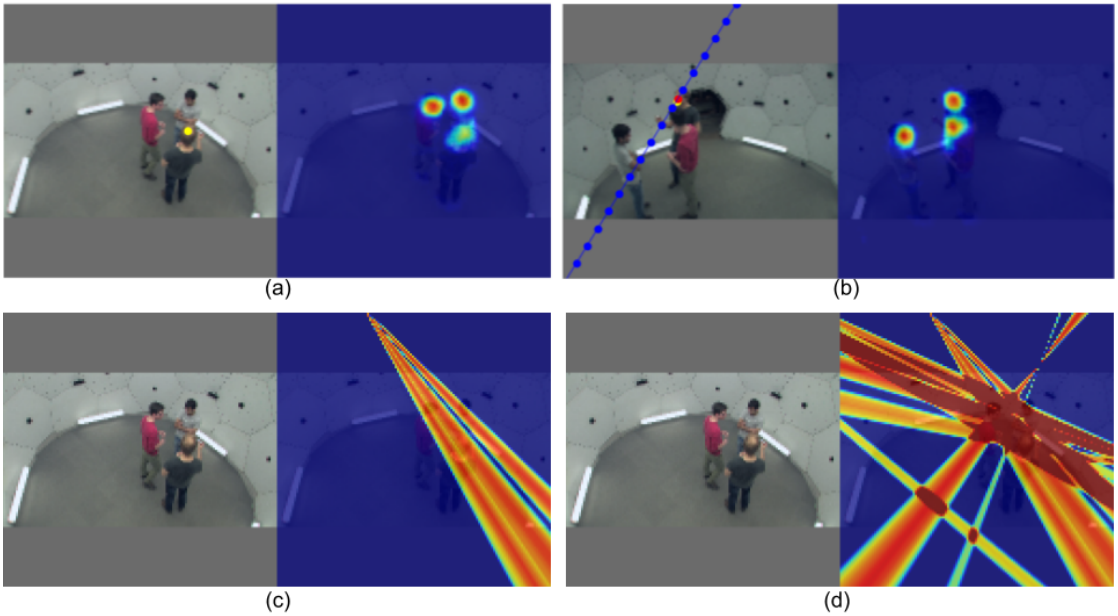


Figure 5.2: Illustration of Epipolar Sampler E. Image from a view is on the left and the joint heatmap is on the right side. Only nose joint is visualize (a) An occluded nose joint (yellow circle) has low score in view v (b) The corresponding epipolar line in the other view v'. The same nose joint (yellow circle) is near the maximum (red circle) of all sampled points (blue circle) on the epipolar line (c) retrieved maximum for every point on view v (d) Output of Epipolar Sampler E, by integrate 4 different views.

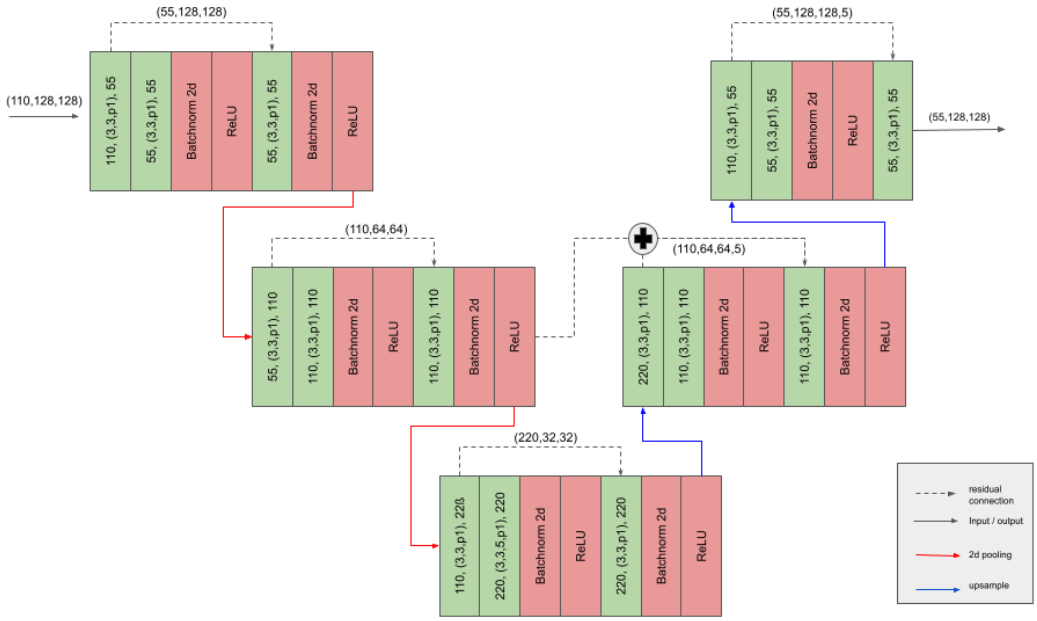


Figure 5.3: Overview of the fusion model, where the green blocks are 2d convolutional kernels. The with input channel, (width, height, 2d padding), and output channel.

between baseline and fusion model is the same. Although fusion model has more layer, it only use 2d convolution kernels.

5.4 Temporal Fusion Model

In temporal fusion model, we use 3d convolutional kernels like baseline model but the third dimension operates on time domain. The time resolution decrease because we do not use padding at time domain during convolution operation in the contractive path. In the expansive path, we feed the output of contractive path to the input of convolution layer, since the contractive path has higher definition in time domain.

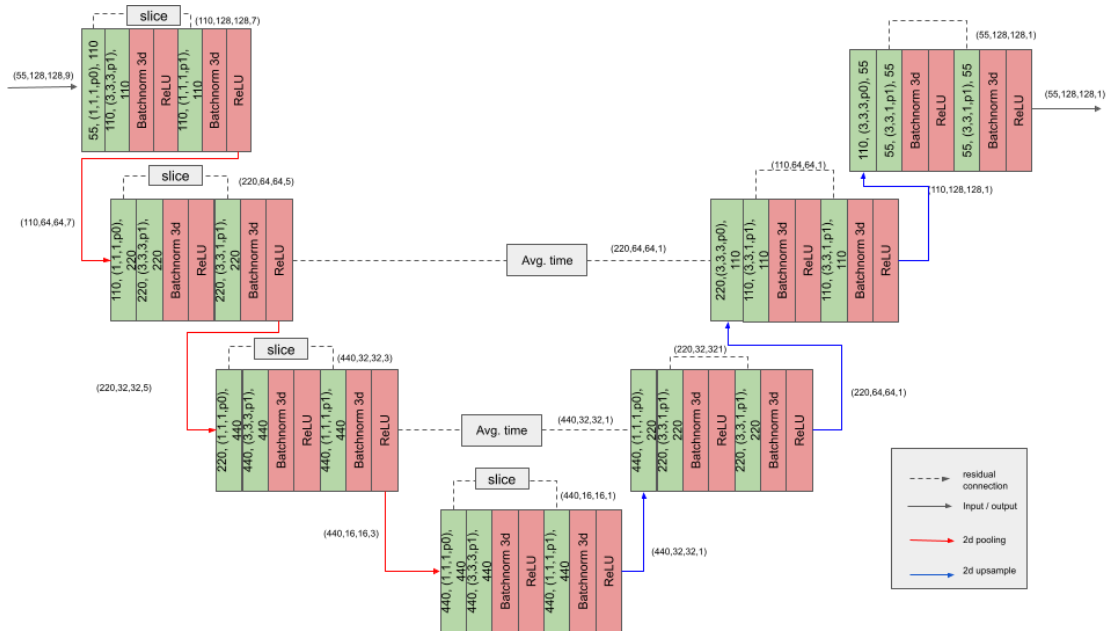


Figure 5.4: Overview of the fusion model, where the green blocks are 2d convolutional kernels with input channel, (width, height, depth, 2d padding), and output channel.

5.5 3D Pose Estimations

After estimate 2D heatmaps **H** form all view using our framework, we use triangulation to find the 3D locations. Our 3D pose estimation, shown in alg (2), has two major parts: *generate_3d_points* and *extract_pose*.

Algorithm 1: Overview of 3D pose estimation algorithm

Input: List of heatmaps $HMs = [\mathbf{H_1}, \mathbf{H_2}, \ldots, \mathbf{H_N}]$ and camera projection matrices $P_{cameras} = [\mathbf{P_1}, \mathbf{P_2}, \ldots, \mathbf{P_N}]$
Output: List of 3D poses $poses3D = [\mathbf{Person}_1, \mathbf{Person}_2, \ldots, \mathbf{Person}_M]$
$joints3D$ = [ ];
for $j \in [0, \ldots, NumJoints]$ **do**
  HMS_{joint} = [ ];
  foreach $\mathbf{H_i} \in HMs$ **do**
    HMS_{joint} append $\mathbf{H_i}$ [: , : , j]
  end
  $points3D$ = generate_3d_points (HMS_{joint}, $P_{cameras}$);
  $points3D$ = cluster ($points3D$);
  $joints3D$ append $points3D$
end
$poses3D$ = extract_poses($joints3D$);

The algorithm for *generate_3d_points* first detect peaks in the heatmaps for each view pairs and consider peaks that are within 1 pixel away from the epipolar line in the other view as correspond points. Then, 3D locations of paired 2D points between two views are estimate using Direct Linear Transformation (2.12). Then, the estimated 3D points are clustered using mean shift and aggregate by *extract_poses*

5.5.1 Extract Poses from 3D Points

In our method, a 3D pose is composed a set of limbs $L = \{l_1, l_2, \ldots, l_{12}\}$ shown in Table 5.2. A limb is the connection between joints defined in COCO dataset. In addition, we interpolate two intermediate joints between both end of a limb. These intermediate koints are predicted by our 2D backbone as well as COCO joints, and act as a redundancy to mitigate the effect of occlusion. In our work, a pose is only compose by the limbs 5.2 instead of full COCO skeleton.

Algorithm 2: extract poses

Input: A List of lists that containing 3D points of specific joints
$X = [[\mathbf{X_{1_1}}, \mathbf{X_{1_2}}, \ldots, \mathbf{X_{1_N}}], \ldots, [\mathbf{X_{55_1}}, \mathbf{X_{55_2}}, \ldots, \mathbf{X_{55_M}}]]$
Output: List of 3D poses $final_poses3D = [\mathbf{Person}_1, \mathbf{Person}_2, \ldots, \mathbf{Person}_M]$

```
poses3D = [ ] ;
foreach limb type ∈ [1, ..., 12] do
    LimbList_i = extract_limb(X, limb_type, limb_type.min_length,
     limb_type.max_length);
    # where LimbList_i is a list of i-th type limbs
end
foreach limb ∈ LimbList_lower_arm_left do
    pose = create_new_pose();
    pose.l_arm_left = limb;
    poses3D append pose
end
foreach limb_type do
    foreach limb ∈ LimbList_limb_type if limb_type ≠ lower_arm_left do
        pt_1 = limb.end_joint;
        distances = [ ];
        foreach pose ∈ pose3D do
            pt_2 = pose.l_arm_left.start_joint ;
            d = la.norm(pt_1 − pt_2) ;
            distances append d ;
        end
        best = arg min(distances);
        best_val = distances[best];
        if merge_distance_mm > best_val then
            best_pose = poses[best] ;
            best_pose.u_arm_left = limb ;
        end
    end
end
final_poses3D = merge_poses(poses3D)
```

Table 5.2: Our 3D Pose Limb Definiotn.

Limb Type	Start Joint	End Joint	Mid Joint 1	Mid Joint 2	Min Length	Max Length
upper_arm_left	sho_l	elb_l	(0.1*sho_l, 0.9*elb_l)	(0.9*sho_l, 0.1*elb_l)	150mm	400mm
lower_arm_left	elb_l	wri_l	(0.1*elb_l, 0.9*wri_l)	(0.9*elb_l, 0.1*wri_l)	150mm	350mm
upper_arm_right	sho_r	elb_r	(0.1*sho_r, 0.9*elb_r)	(0.9*sho_r, 0.1*elb_r)	150mm	400mm
lower_arm_right	elb_r	wri_r	(0.1*elb_r, 0.9*wri_r)	(0.9*elb_r, 0.1*wri_r)	150mm	350mm
left_side	sho_l	hip_l	(0.1*sho_l, 0.9*hip_l)	(0.9*sho_l, 0.1*hip_l)	200mm	600mm
right_side	sho_r	hip_r	(0.1*sho_r, 0.9*hip_r)	(0.9*sho_r, 0.1*hip_r)	200mm	600mm
hip_lr	hip_l	hip_r	(0.1*hip_l, 0.9*hip_r)	(0.9*hip_l, 0.1*hip_r)	150mm	400mm
shoulder_lr	sho_l	sho_r	(0.1*hip_l, 0.9*hip_r)	(0.9*hip_l, 0.1*hip_r)	150mm	500mm
left_upper_leg	hip_l	kne_l	(0.1*hip_l, 0.9*kne_l)	(0.9*hip_l, 0.1*kne_l)	100mm	450mm
right_upper_leg	hip_r	kne_r	(0.1*hip_r, 0.9*kne_r)	(0.9*hip_r, 0.1*kne_r)	100mm	450mm
left_lower_leg	kne_l	akl_l	(0.1*kne_l, 0.9*akl_l)	(0.9*kne_l, 0.1*akl_l)	100mm	600mm
right_lower_leg	kne_r	akl_r	(0.1*kne_r, 0.9*akl_r)	(0.9*kne_r, 0.1*akl_r)	100mm	600mm

5.6 Summary

Our full framework is composed of 3 models: a 2D joint estimation backbone, an epiploar sampler, a fusion model and a temporal fusion model. All the source views are merge at the epipolar sampler and refine at the view fusion model. Then, all the frames are merge into one at the temporal fusion model. The framework, shown in Fig. 5.5 capture both time and spatial information from multi-view video. Moreover, the 3D pose estimation algorithm conduct association at last stage of our pipeline, allowing the flexibility of an end-to-end training scheme.

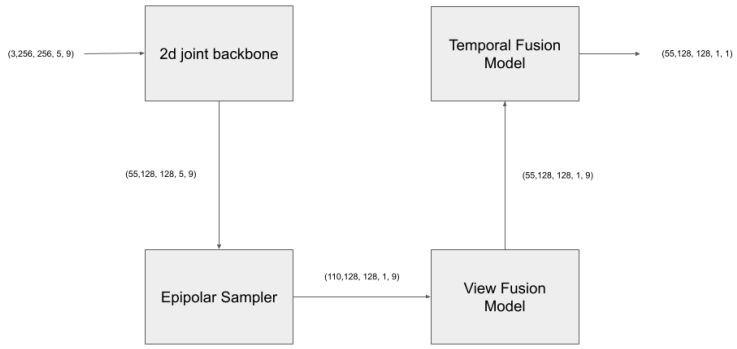


Figure 5.5: Our complete framework covers both time and spatial domain. The parenthesis describe the dimension of input and output tensor, it means (channels, height, width, camera view, number of frame).

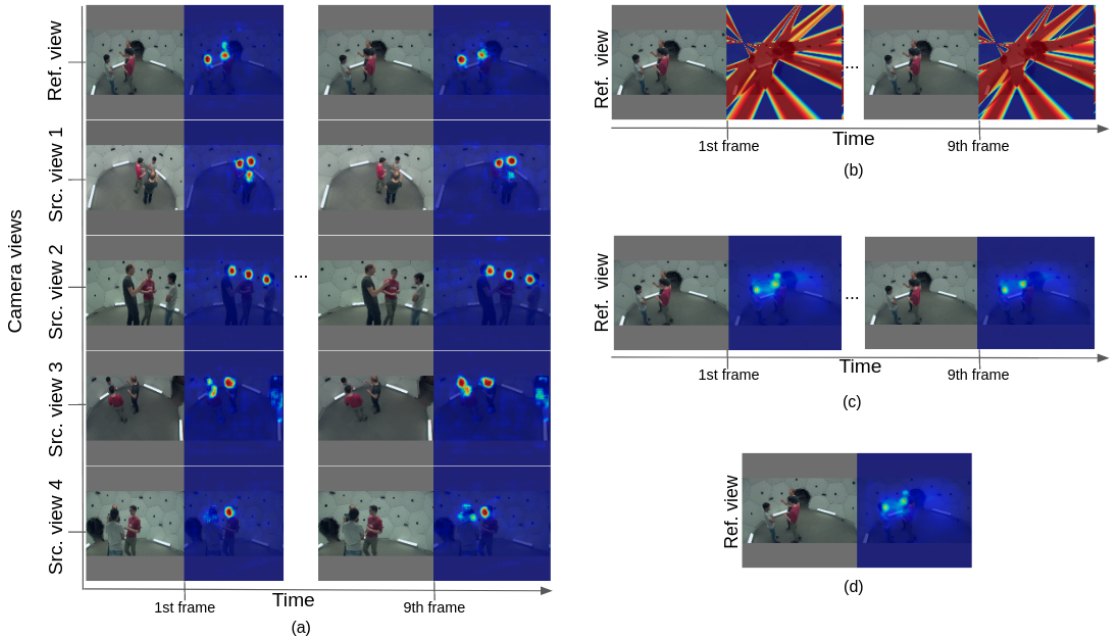


Figure 5.6: Predicted sample from the start (a) to the end (e) of our framework. (a) The output of 2D joint backbone at different frame and across 5 different views, one reference view and 4 other source views. (b) The output of epipolar sampler. All the pixels on reference view sample from 4 different source views using epipolar geometry. (c) The output of fusion net that takes the epipolar heatmap at (b) as input. (d) The output of temporal fusion net that merge multiple frames into one.

6 Experiments

We evaluate both the 2D and 3D results for our framework, since our model estimate 2D heatmaps and 3D poses. The training set contains sequences: *"170407_haggling_a1"*, *"160226_haggling1"*, *"161202_haggling1"* and we randomly sample 5 different camera views out of 26 camera views. We choose *"160906_pizza1"* as the validation sequence and 5 fixed views, *(0, 12), (0, 6), (0, 23), (0, 13), (0, 3)*, that do not exist in the training set.

To compare our framework, we train a baseline model, that has the same structure as the 2d backbone but is train on CMU dataset. Since the CMU dataset has only 3D skeletons, the project groundtruth 2D pose have all the joints annotated. Thus, the baseline model is trained with both occluded and unoccluded joints. Our intuition is the baseline should be able to learn to "see" through the occlusion.

We use Adam optimizer with a learning rate of 0.01 and train every modules, e.g. Baseline Model, 2D backbone, view Fusion, and temporal fusion module, for 20 epochs. During the training, we use the batch size of 32 for all the above mentioned modules except temporal modules due to memory limitation. For temporal module, we use a batch size of 1, where each sample is the tensor with dimension (55, 128, 128, 1, 9), meaning number of keypoints, height, width, number of view, and number of frames.

6.1 Quantitative Results

In 2D evaluation, we use ground truth bounding boxes to localize people in the view. Each bounding box are generated by taking the most top-right and bottom left corners of the 2D pose, projected from the ground-truth 3D pose. After localizing individual with the ground-truth bounding boxes, we take the maximum score within the bounding box for each channel of the heatmap as the joint location and calculate PCKh accordingly. In 3D evaluation, we assign ground truth 3D poses to the estimated 3D poses using Hungarian algorithm, where the cost is the mean distance of joints difference between the 3D poses of ground truth and predictions.

Model	**nose**	**eye_l**	**eye_r**	**ear_l**	**ear_r**	**sho_l**	**sho_r**	**elb_l**	**elb_r**
Baseline	0.29	0.08	0.23	0.09	0.17	0.22	0.23	0.10	0.15
ResNet	0.29	0.08	0.23	0.09	0.17	0.22	0.23	0.10	0.15
+ View Fusion	**0.34**	**0.35**	**0.39**	**0.37**	**0.40**	0.22	**0.31**	**0.13**	**0.19**
+ Temporal Fusion	0.27	0.35	0.36	0.35	0.31	0.20	0.29	0.10	0.18
Model	**wri_l**	**wri_r**	**hip_l**	**hip_r**	**kne_l**	**kne_r**	**ank_l**	**ank_r**	
Baseline	0.12	0.16	0.17	0.15	0.13	0.10	0.17	0.13	
ResNet	0.12	**0.16**	0.17	0.15	0.13	0.10	**0.16**	0.13	
+ View Fusion	**0.13**	0.15	**0.24**	**0.25**	**0.17**	**0.18**	0.13	0.13	
+ Temporal Fusion	0.11	0.09	0.20	0.19	0.12	0.13	0.13	0.12	

Table 6.1: 2D evaluation based on PCKh@0.5 with 5 views input.

Model	**sho_l**	**sho_r**	**elb_l**	**elb_r**	**wri_l**	**wri_r**	**hip_l**	**hip_r**	**kne_l**
Baseline	0.08	0.17	0.02	0.03	0.01	0.03	0.09	0.09	**0.02**
ResNet	0.07	0.18	0.03	0.06	0.02	0.03	0.09	0.08	0.01
+ View Fusion	**0.15**	**0.18**	**0.04**	**0.09**	**0.02**	0.03	**0.13**	**0.11**	0.01
+ Temporal Fusion	0.01	0.03	0	0.03	0	0	0.01	0	0
Model	**kne_r**	**ank_l**	**ank_r**						
Baseline	0.007	0.006	**0.006**						
ResNet	0.008	0.003	0.004						
+ View Fusion	**0.01**	**0.009**	0.004						
+ Temporal Fusion	0	0	0						

Table 6.2: 3D evaluation based on PCKh@0.5 and with 5 views input.

	sho_l			**sho_r**			**elb_l**			**elb_r**		
	3	4	5	3	4	5	3	4	5	3	4	5
ResNet	0.02	0.04	0.07	0.03	0.1	**0.18**	0.006	0.01	0.03	0.007	0.03	0.06
+ View Fusion	0.02	0.06	**0.15**	0.04	0.1	**0.18**	0.006	0.02	**0.04**	0.01	0.04	**0.09**
+ Temporal Fusion	0.02	0.02	0.01	0.05	0.09	0.03	0	0	0	0.02	0.04	0.03
	wri_l			**wri_r**			**hip_l**			**hip_r**		
	3	4	5	3	4	5	3	4	5	3	4	5
ResNet	0.003	0.01	**0.02**	0.005	0.02	**0.03**	0.02	0.05	0.09	0.01	0.04	0.08
+ View Fusion	0.004	0.02	**0.02**	0.004	0.02	**0.03**	0.02	0.06	**0.13**	0.02	0.06	**0.11**
+ Temporal Fusion	0.003	0.01	0	0.005	0.008	0	0.005	0.01	0.01	0.01	0.01	0
	kne_l			**kne_r**			**ank_l**			**ank_r**		
	3	4	5	3	4	5	3	4	5	3	4	5
ResNet	0.001	0.007	**0.01**	0.003	0.004	0.008	0.0003	0.0001	**0.003**	0.0006	0.0002	**0.004**
+ View Fusion	0.0003	0.008	**0.01**	0.0003	0.007	**0.01**	0	0.0003	0.0009	0.0003	0.001	**0.004**
+ Temporal Fusion	0	0.002	0	0	0	0	0	0	0	0	0	0

Table 6.3: 3D evaluation based on PCKh@0.5 with 3, 4 and 5 views input.

6.2 Qualitative Results

We show the 2D and 3D prediction of human poses in Fig. 6.1 and Fig. 6.2. Note that in 2D, the ground truth bounding boxes are used to localize people in each view.

6.3 Discussion

We first compare the baseline model with our 2D backbone. Observing on the results shown in Table 6.1 and 6.2, although baseline and 2D backbone each train on different datasets (one in indoor studio environment with all joints annotated and the other on natural image with some joints missing), both models have very similar performance. This means our 2D backbone generalize well on different dataset and the baseline model does not able to learn seeing through the occlusion.

We observe improvement at almost every type of joint when we combine 2d backbone with view fusion modules in 2D and 3D pose estimation. Especially, combining 2d backbone and view fusion modules help our 3D pose estimation algorithm produce less but more accurate poses, shown in Fig. 6.2. In addition, we also conduct experiments regarding the effect of number of input view to the performance of 3D pose estimation, shown in Table 6.3. The results support our intuition that the performance of our complete framework can benefit from more views.

However, we observe that a degradation at the performance of 3d pose estimation when we add temporal fusion module. According to our observation, the temporal

Figure 6.1: 2d pose estimation results. Better view with PDF and zoom.

Figure 6.2: 3d pose estimation results. Better view with PDF and zoom.

module smooth out the peak in the heatmap quite a lot, making the score margin between background and heatmap much smaller. Since our 3D pose estimation algorithm does binary threshold to find peaks first and, then, does triangulation based on the location, the algorithm suffers from a low number of peak. Furthermore, we consider out estimated 3D pose with less than 5 limbs as invalid, so lots of pose are discard, leaving 0 PCKh in detection of some type of joints.

7 Conclusion

In this work, we propose a simple end-to-end fully convolutional framework that let a 2D joint estimator aware of multiple views. We aggregare 2d joints estimations from multiple views into a single reference view using epipolar geometry. Furthermore, a network based view fusion module to learn from the heatmap that aggregate multiple view, making the 2D joint estimator aware of multiple view. In addition, we incorporate a network fuse heatmaps from different frames into one, so we might be able to benefit from temporal information. Lastly, we propose a greedy algorithm that solve association problem in multi-person 3D pose estimation.

List of Figures

1.1 Estimating multi-person 3D human poses without using computational heavy operations is the task we address in this work. 2
1.2 A human pose is approximated by a rigid skeletons, composed by various joints. Performing human pose estimation, however, from images can be a difficult task due to the body pose and appearance high variation. . . 3

2.1 **C** is the camera centre and **p** is the center of the image plane or called *principal point*. The camera center is here placed at the origin of world coordinate frame. Note the image plane is placed in front of the camera center. 5
2.2 The Euclidean transformation between the world and camera coordinate frames. 7
2.3 Illustration of the point-line correspondence in two views. For an arbitrary point x in one view, the corresponding point **x** in another view has to lie on the epipolar line $\mathbf{l}'$ 8
2.4 A 2D convolution. We draw boxes with arrows to indicate how the upper-left element ofthe output tensor is formed by applying the kernel to the corresponding upper-left region of the input tensor [18] 10
2.5 A griphical demotration of a simple 3-layer CNN with only convolutional layer. Layer **g** and **h** are formed by kernel size with width 3. The receptive field of the units in the deeper layers of a convolutional network is larger than the receptive field of the units in the shallow layers. This means that even though direct connections in a convolutional net are very sparse, units in the deeper layers can be indirectly connected to all or most of the input image [18] 11
2.6 Training error (left) and test error (right) on CIFAR-10 with 20-layer and 56-layer "plain" networks (that simply stack layers). The deeper network has higher training error, and thus test error [19]. 12
2.7 An illustration of residual block: the task for the stacked layer is to learn the residual representation $\mathbf{F}(\mathbf{x})$ [19]. The curved connection is commonly called *residual connections* or *skip connections* interchangeably nowadays. 13

2.8 Example network architectures for ImageNet. Top: a residual network with 34 parameter layers (3.6 billion FLOPs). The dotted shortcuts increase dimensions. Middle: a plain network with 34 parameter layers (3.6 billion FLOPs). Bottom: the VGG-19 model [20] (19.6 billion FLOPs) as a reference. [19] 13
2.9 The loss surfaces of ResNet-56 (left) without residual connections (right) with residual connections [21]. 14
2.10 3D U-Net architecture 15

3.1 Overview of the [2] approach. It first extract candidate regions using a Region Proposal Network (RPN) and obtain pose proposals by placing a fixed set of anchor-poses into these boxes (top). These pose proposals are then scored by a classification branch and refined using class-specific regressors, learned independently for each anchor-pose. 18
3.2 Overview of the [8] approach. (a), an off-the-shelf human pose detector is used to produce 2D bounding boxes and associated 2D poses in each view, which may be inaccurate and incomplete (b). Then, the detected bounding boxes are clustered. Each resulting cluster includes the bounding boxes of the same person in different views (c). The isolated bounding boxes that have no matches in other views are regarded as false detections and discarded. Finally, the 3D pose of each person is reconstructed from the corresponding bounding boxes and associated 2D poses (d). 20
3.3 2D and 3D correspondence affinity measurments in [9]. (a) 2D correspondence is computed within the same camera. (b) 3D correspondence is measured between the predicted location and the projected line in 3-space. 20
3.4 Overview of the [6] approach. (a) it first estimate 2D pose heatmaps for all views; then, (b) it warps the heatmaps to a common 3D space and construct a feature volume which is fed into a Cuboid Proposal Network to localize all people instances; (c) for each proposal, it construct a finer-grained feature volume and estimate a 3D pose. 22

4.1 Overview of 2017 COCO Keypoint. (a) Images with human (b) un-occluded (colored)/occluded (white) joint locations and limb (colored line), where green means left part of the body and red means right part of the body (c) masks for individuals 24

4.2 COCO provide a reference skeleton (facing toward the reader) that connecting the joints. For our convenience in visualization, we define joints that belong to the left part of the body, colored with red, and to the right part of the body, colored with green. 27
4.3 (left column) The image with a person class label in COCO annotation and (right column) the joint heatmap $\mathbf{H}_k$. Note that, we merge all K channels of joint heatmap to a single channel, in order to visuaulize compactly 27
4.4 CMU Panoptic Overview. Note that it does not provide 2D annotation about the visibility of projected 2D joints. 30
4.5 Data augmentation for COCO Dataset 31
4.6 (left) keypoints from one view (right) correspond epiploar lines and the same keypoints in other view. Note that epiploar lines pass through the same keypoints in different views, verifying the camera projection matrix is updated correctly. 32

5.1 Overview of our 2D backbone. We train the last convolutional layer of the pretrained ResNet50 with the deconvolutional head together. 34
5.2 Illustration of Epipolar Sampler E. Image from a view is on the left and the joint heatmap is on the right side. Only nose joint is visualize (a) An occluded nose joint (yellow circle) has low score in view v (b) The corresponding epipolar line in the other view v'. The same nose joint (yellow circle) is near the maximum (red circle) of all sampled points (blue circle) on the epipolar line (c) retrieved maximum for every point on view v (d) Output of Epipolar Sampler E, by integrate 4 different views. 36
5.3 Overview of the fusion model, where the green blocks are 2d convolutional kernels. The with input channel, (width, height, 2d padding), and output channel. 37
5.4 Overview of the fusion model, where the green blocks are 2d convolutional kernels with input channel, (width, height, depth, 2d padding), and output channel. 38
5.5 Our complete framework covers both time and spatial domain. The parenthesis describe the dimension of input and output tensor, it means (channels, height, width, camera view, number of frame). 42

5.6 Predicted sample from the start (a) to the end (e) of our framework. (a) The output of 2D joint backbone at different frame and across 5 different views, one reference view and 4 other source views. (b) The output of epipolar sampler. All the pixels on reference view sample from 4 different source views using epipolar geometry. (c) The output of fusion net that takes the epipolar heatmap at (b) as input. (d) The output of temporal fusion net that merge multiple frames into one. 43

6.1 2d pose estimation results. Better view with PDF and zoom. 47
6.2 3d pose estimation results. Better view with PDF and zoom. 48

List of Tables

4.1 COCO keypoints and limb definition 26
4.2 COCO keypoints and limb definition 29
4.3 Augmentations used in training 31

5.1 Structure of deconvolution head 34
5.2 3D Pose Limb Definiotn 41

6.1 2D evaluation based on PCKh@0.5 with 5 views input. 45
6.2 3D evaluation based on PCKh@0.5 and with 5 views input. 45
6.3 3D evaluation based on PCKh@0.5 with 3, 4 and 5 views input. 46

Bibliography

[1] D. Mehta, O. Sotnychenko, F. Mueller, W. Xu, S. Sridhar, G. Pons-Moll, and C. Theobalt, "Single-shot multi-person 3d pose estimation from monocular rgb," in *3DV*, 2018.

[2] G. Rogez, P. Weinzaepfel, and C. Schmid, "Lcr-net++: Multi-person 2d and 3d pose detection in natural images," in *TPAMI*, 2019.

[3] K. Iskakov, E. Burkov, V. Lempitsky, and Y. Malkov, "Learnable triangulation of human pose," in *ICCV*, 2019.

[4] H. Qiu, C. Wang, J. Wang, N. Wang, and W. Zeng, "Cross view fusion for 3d human pose estimation," in *ICCV*, 2019.

[5] Y. Zhang, L. An, T. Yu, x. Li, K. Li, and Y. Liu, "4d association graph for realtime multi-person motion capture using multiple video cameras," in *CVPR*, 2020.

[6] H. Tu, C. Wang, and W. Zeng, "Voxelpose: Towards multi-camera 3d human pose estimation in wild environment," in *ECCV*, 2020.

[7] V. Belagiannis, S. Amin, M. Andriluka, B. Schiele, N. Navab, and S. Ilic, "3d pictorial structures revisited: Multiple human pose estimation," *TPAMI*, In Press.

[8] J. Dong, W. Jiang, Q. Huang, H. Bao, and X. Zhou, "Fast and robust multi-person 3d pose estimation from multiple views," *CVPR*, 2019.

[9] L. Chen, H. Ai, R. Chen, Z. Zhuang, and S. Liu, "Cross-view tracking for multi-human 3d pose estimation at over 100 fps," in *CVPR*, Jun. 2020.

[10] I. Sárándi, T. Linder, K. O. Arras, and B. Leibe, "How robust is 3d human pose estimation to occlusion?" In *IROS Workshop - Robotic Co-workers 4.0*, 2018.

[11] Y. He, R. Yan, K. Fragkiadaki, and S.-I. Yu, "Epipolar transformers," in *CVPR*, 2020.

[12] Z. Zhang, C. Wang, W. Qiu, W. Qin, and W. Zeng, "Adafuse: Adaptive multiview fusion for accurate human pose estimation in the wild," in *IJCV*, 2020.

[13] V. Belagiannis, S. Amin, M. Andriluka, B. Schiele, N. Navab, and S. Ilic, "3d pictorial structures for multiple human pose estimation," in *CVPR*, 2014.

[14] R. Hartley and A. Zisserman, *Multiple View Geometry in Computer Vision*, 2nd ed. USA: Cambridge University Press, 2003, ISBN: 0521540518.

[15] S.-E. Wei, V. Ramakrishna, T. Kanade, and Y. Sheikh, "Convolutional pose machines," in *CVPR*, 2016.

[16] A. Newell, K. Yang, and J. Deng, "Stacked hourglass networks for human pose estimation," in *ECCV*, 2016.

[17] K. Sun, B. Xiao, D. Liu, and J. Wang, "Deep high-resolution representation learning for human pose estimation," in *CVPR*, 2019.

[18] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, `http://www.deeplearningbook.org`.

[19] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, 2016.

[20] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *ICLR*, 2015.

[21] H. Li, Z. Xu, G. Taylor, C. Studer, and T. Goldstein, "Visualizing the loss landscape of neural nets," in *NIPS*, 2018.

[22] Ö. Çiçek, A. Abdulkadir, S. Lienkamp, T. Brox, and O. Ronneberger, "3d u-net: Learning dense volumetric segmentation from sparse annotation," in *MICCAI*, 2016.

[23] O. Ronneberger, P.Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *MICCAI*, 2015.

[24] Y. Chen, Z. Wang, Y. Peng, Z. Zhang, G. Yu, and J. Sun, "Cascaded Pyramid Network for Multi-Person Pose Estimation," 2018.

[25] Z. Zhong, L. Zheng, Z. Zheng, S. Li, and Y. Yang, "Camera style adaptation for person re-identification," in *CVPR*, 2018.

[26] Q.-X. Huang and L. Guibas, "Consistent shape maps via semidefinite programming," 2013.

[27] J. Martinez, M. J. Black, and J. Romero, "On human motion prediction using recurrent neural networks," in *CVPR*, 2017.

[28] D. Pavllo, C. Feichtenhofer, D. Grangier, and M. Auli, "3d human pose estimation in video with temporal convolutions and semi-supervised training," 2019.

[29] J. Tanke and J. Gall, "Iterative greedy matching for 3d human pose tracking from multiple views," in *German Conference on Pattern Recognition*, 2019.

[30] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. Zitnick, "Microsoft coco: Common objects in context," May 2014.

[31] H. Joo, H. Liu, L. Tan, L. Gui, B. Nabbe, I. Matthews, T. Kanade, S. Nobuhara, and Y. Sheikh, "Panoptic studio: A massively multiview system for social motion capture," in *ICCV*, 2015.

[32] E. Shelhamer, J. Long, and T. Darrell, "Fully convolutional networks for semantic segmentation," *TPAMI*, 2017.

[33] B. Xiao, H. Wu, and Y. Wei, "Simple baselines for human pose estimation and tracking," in *ECCV*, 2018.